POLITECNICO DI TORINO

MASTER's Degree in MECHATRONIC ENGINEERING

MASTER's Degree Thesis

Automatic Sorting with Manipulator using Moveit2 and ROS2

Supervisors

Prof. MARCELLO CHIABERGE

Dr. CLAUDIO CHIEPPA

Candidate

GABRIELLA MARINO GAMMAZZA

October 2024

Abstract

In recent years, robotic sorting technologies have rapidly expanded across various industries, revolutionizing the way objects are arranged, classified, and organized. This study addresses the growing need for efficient and adaptable sorting solutions by developing a flexible robotic system capable of handling different sorting tasks with high accuracy in different environments. The research utilizes a six-degree-of-freedom robotic arm, the Ufactory850, integrated with ROS2, an open-source framework for robot software development, along with MoveIt 2, a motion planning library. The system is further enhanced using computer vision techniques for object recognition, localization, and feature selection.

Initially, a static Pick and Place task is implemented using the MoveIt Task Constructor framework, which simplifies planning through a modular, staged approach. In this first phase, the planning scene is static, incorporating a priori informations such as object positions and geometric dimensions. In order to dynamically identify and localize objects through a camera, the ROS "Find Object 2D" package is employed, utilizing a feature matching algorithm based on data from the camera sensors. Once objects are recognized, Find Object 2D attaches a frame to their center point, allowing their position and orientation to be determined relative to the camera. A frame transformation is then computed with respect to the robot's base link frame, enabling dynamic localization. Lastly, for object dimension computation, the Point Cloud Library is used to segment and process cylindrical and box-shaped objects, following a series of pre-processing operations on the point cloud data. The system is integrated through ROS services, with a single client managing three dedicated servers: one for object localization, one for dimension computation, and one for the Pick and Place task. This architecture enables the dynamic construction of the planning scene, ensuring real-time updates of object names, positions, orientations, and dimensions.

To validate the system, four sorting policies were implemented and tested, evaluating both speed and accuracy in parameters detection. The system demonstrates both robustness and speed, sorting approximately 2–3 objects per minute, while maintaining object localization error typically below 7%. Cylinder segmentation is highly reliable, though box segmentation performs less effectively, particularly in complex environments. Handling very low objects occasionally presents challenges, but the system's robustness is evident as it automatically resubmits requests if recognition or localization fails, ensuring continuous operation. Simulation and visualization are supported using Gazebo, a 3D robotics simulator and Rviz, a 3D visualization tool for ROS, while Docker ensures a consistent and portable development environment across platforms. To verify scalability, the system was successfully adapted and tested on another manipulator with real hardware, demonstrating flexibility across platforms.

Developed with Concept Engineering Reply at "Area42" in Turin, the system provides a robust and adaptable solution for industrial applications, achieving a sorting success rate of about 90%.

Table of Contents

List of Tables

List of Figures

Acronyms

MTC
Moveit Task Constructor

HSV
Hue Saturation Value

OpenCv
Open Computer Vision

ROS
Robot Operating System

PCL
Point Cloud Library

SIFT
Scale-Invariant Feature Transform

FAST
Features from Accelerated Segment Test

SURF
Speeded Up Robust Features

BRIEF
Binary Robust Independent Elementary Feature

ORB
Oriented FAST and Rotated BRIEF

UF850

UFactory 850

API

Application Programming Interface

CLI

Command Line Interface

SRDF

Semantic Robot Description Format

URDF

Unified Robot Description Format

TF

Transform Frame

OMPL

Open Motion Planning Library

CHOMP

Covariant Hamiltonian Optimization for Motion Planning

GUI

Graphical User Interface

SDF

Simulation Description Format

ID

Identifier

RANSAC

RANdom SAmple Consensus

AABB

Axis-Aligned Bounding Box

RRT

Rapidly-exploring Random Tree

IP

Internet Protocol

RGB

Red Green Blu

Chapter 1

Introduction

Sorting in robotics is the process of arranging, classifying, and organizing objects based on specific criteria, such as size, shape, colour, or function.
Sorting robots are a type of stationary or mobile robots that enable fast and efficient sorting of goods/parcels based on end destination, categorisation and more. These robots use a combination of sensors, cameras, actuators, and other mechanical components to detect, characterize, and sort objects into the correct bin. Their sorting process is tailored to meet the specific needs and requirements of each business and is adapted to the type of objects/parcels being sorted [1].

1.1 Background of the study

Sorting robots have become increasingly important in recent years due to the benefits they offer over manual and conventional sorting methods, including higher efficiency, accuracy, and speed. They are used across various industries, including e-commerce, logistics, manufacturing, agriculture, and recycling [1]. Traditionally, these processes were manual, requiring extensive labor, time, and resources. Subsequently, they became reliant on basic mechanical systems, such as conveyor belts with rudimentary sensors. These early systems could only distinguish between limited parameters like size or weight, often resulting in inefficiencies and human intervention. Nowadays, with advancements in robotics, it is possible to perform complex sorting tasks, reducing human error and enhancing operational efficiency. This study focuses on exploring the integration of robotic systems into sorting operations, analyzing both the technological aspects and the potential for improvements in various sectors. In addition to the robotic component, computer vision algorithms play a critical role in enhancing the system's ability to accurately identify and categorize objects based on various features, such as shape, size, color, and texture. Using advanced image processing techniques, these algorithms enable the robotic system to perform

complex sorting tasks with greater precision and adaptability, making it suitable for a wide range of applications beyond traditional industrial environments. While there are clear advantages to using these technologies, such as the ability to work in variable environments and the speed at which objects can be identified, there are also potential limitations, such as the need for adequate lighting and the dependence on the quality of the images captured.

1.2 Problem Statement

Manual object sorting in industry faces significant challenges in terms of efficiency and accuracy. Manual processes are inherently limited by the human ability to handle large volumes of products consistently and without errors. This limitation results in higher costs, longer production times, and variable product quality. The urgent need for effective and accurate automation solutions becomes evident in this context. Numerous studies and success stories support the idea that automation has revolutionized entire industries, improving efficiency, reducing costs and raising product quality [2].
However, there exist a factor that slowing down the process of robotisation, namely the lack of flexibility of existing robotic solutions [3]. Many automated solutions are highly specialized, tailored to handle a particular type of object or to function within a specific environment, such as a manufacturing line or a recycling facility. This specialization restricts their potential applications, making them unsuitable for different contexts where a more flexible approach is required. Additionally, developing new systems for different industries or tasks can be costly and time-consuming, as each system must be customized to meet the unique demands of the application.

1.3 Objectives of the research

This research seeks to address the following problem: How can a generic robotic sorting system be designed to provide the flexibility and adaptability needed to operate effectively across a wide range of applications, both industrial and non-industrial? A solution that is capable of performing sorting tasks with minimal reconfiguration has the potential to transform industries beyond manufacturing, such as healthcare, retail, and even domestic environments. Such a system would not only improve operational efficiency but also reduce the need for specialized equipment, offering a cost-effective and scalable solution for diverse sorting challenges.

1.4 Research questions/hypothesis

To realize this project, several research questions and hypotheses were formulated. The primary objective was to develop a sorting system that utilizes a six-degree-of-freedom manipulator, where the central function is the robotic pick-and-place operation. This functionality is fundamental in any sorting system, as it directly influences the system's efficiency and accuracy. Consequently, the research began with the goal of identifying a robust solution for pick-and-place operations, with a particular focus on high processing speed and optimal motion planning. The first question that arose was: What is the most effective approach to achieve efficient pick-and-place tasks? The solution was found in MoveIt, the robotic manipulation platform for ROS.

Another critical question emerged regarding motion planning: Which algorithm is most suitable for generating an efficient path for the robot, from its initial position to its target location? This led to a hypothesis centered around the trade-off between speed and optimality. Given that the goal of the system is to sort objects efficiently, the hypothesis posited that processing speed is more critical than absolute optimality in motion planning. For this specific application, it was theorized that a faster algorithm, even if slightly less optimal in terms of path precision, would be preferable to achieve the desired throughput.

Once the pick-and-place operation was defined, a new question naturally emerged: How can the system dynamically identify and localize objects using a camera? At this stage, the importance of computer vision became evident. The task of recognizing and localizing objects in real-time required advanced vision algorithms. This challenge was compounded by the fact that the entire system was built using ROS2 and MoveIt2, both of which are relatively new in the field of robotics, adding complexity to the integration of computer vision techniques. Moreover, as previously mentioned, one of the key objectives of this thesis was to develop a flexible sorting system. In this regard, the computer vision system needed to support this goal, ensuring that the system could adapt to different tasks and environments with minimal reconfiguration. To address this, initial research on ROS tools was conducted, which led to the implementation of a ROS package (Find Object 2D) for object detection and a ROS library (PCL) for point cloud segmentation. As a consequence, another question naturally arose: what tools from these frameworks would best support the goals of the project, ensuring that the system could effectively handle both geometric segmentation and feature-based object recognition.

Finally, once the tools were selected, it became clear that a robust communication mechanism was necessary to integrate and manage the various system components effectively. For this purpose, the ROS2 client/server architecture was chosen. This architecture was selected to enhance communication and coordination among the

components, ensuring modularity and flexibility. By utilizing the ROS2 architecture, the system benefits from seamless interaction between motion planning, object recognition, and real-time control, thus improving overall system performance and adaptability.
In addition to focusing on the pick-and-place operation and computer vision integration, this project also utilized Gazebo for simulation and RViz for real-time visualization of the robot's movements. These tools were essential in testing the robotic system in a controlled environment before moving to physical deployment, ensuring accurate motion planning and object recognition.

1.5 Industrial Contribution

This research represents a significant contribution not only to the academic field but also to industry, particularly through its development within the research laboratory of Reply, namely "Area42" situated in Turin. In particular, the project was carried out in collaboration with Concept Engineering Reply, with real-world applicability in mind, making it practical for immediate integration into different operational contexts.
Reply is an Italian IT consulting, outsourcing, digital services applications company specialising in the design, implementation and maintenance of Internet and social network-based solutions. The company focuses on emerging technologies like AI, cloud computing, and IoT. In particular, Concept Engineering Reply plays a key role in IoT solutions, covering the software development process from the Device, through the Gateway, to the Cloud or Smartphone App. It guides customers through the design process of their application, specify the ideal Cloud solution architecture, develop in agile teams. Together with partners for hardware and telecommunication it provides to the customers an individual software solution based on IoT requirements. Furthermore, in recent years, it collaborates to the implementation of research projects oriented to robotics within the "Industrial Iot" area of Area42.

1.6 Scope and Limitation

The research's goal was to develop a robotic sorting system adaptable to any task. The entire system was developed using Docker, ensuring portability across different platforms and providing a consistent environment for development and testing. This approach allowed for the seamless integration of various ROS2 and MoveIt2 components, regardless of the underlying hardware or operating system.
The first step in the project was the implementation of the Pick-and-Place function, utilizing the MoveIt Task Constructor (MTC) framework. MTC's stage-based

architecture made easier flexible task planning, enabling the system to manage modular sorting tasks efficiently. In addition, computer vision techniques were introduced to enable both geometric and semantic recognition techniques, which play a crucial role in achieving a comprehensive understanding of the objects to be sorted. For geometric recognition, PCL was used to segment objects and distinguish between fundamental shapes like cylinders and boxes. This was complemented by SIFT-based semantic recognition, which enabled the identification and localization of objects based on their visual features through the "Find Object 2d" ROS package. This combination of geometric segmentation and semantic recognition was a key factor in enhancing the system's flexibility and precision. As it was already mentioned in the section 1.4, integrating these various components was facilitated by the ROS2 client/server architecture, which enabled effective communication and coordination between motion planning, object recognition, and real-time control. However, while the integration of geometric and semantic recognition techniques proved effective, there are limitations. The system's performance in more complex or visually cluttered environments is restricted by the current techniques. The use of deep learning or neural networks could greatly enhance both the geometric and semantic capabilities, allowing the system to interpret objects in a more holistic and adaptive manner. Future research should focus on strengthening this dual recognition approach to further improve sorting accuracy across a variety of applications.

Chapter 2

State of the Art

Sorting technologies have been rapidly advancing in recent years, driven by the increasing demand for efficiency and precision in various industries. This chapter provides a comprehensive review of the state of the art in sorting systems, with a particular focus on the latest developments and methodologies. Furthermore, it will be given a general overview about Computer Vision and Pick and Place, focusing only on the aspects covered in this thesis.

2.1 Sorting in different applications

Sorting is an important process in industrial applications, including manufacturing, agriculture, logistics and recycling. Usually, sorting applications are based on sensors able to take decisions. In these applications, the materials or the object are fed in the system by means of a conveyor mechanism, and during the transportation phase the sensor data acquisition takes place. The sensor data are processed with the goal of detecting and classifying individual particles in the material stream. The classification result serves as the basis for the sorting decision, which is executed by means of actuators [4].
Sorting objects presents specific challenges, such as the variability of shapes, sizes and materials, as well as the need to handle constantly changing products. Robotics offers a more effective solution than manual processes by addressing these challenges with precision and speed. In particular, the ability of robots to perform repetitive tasks with high precision makes them ideal for object classification in industrial environments [2]. In addition to sensors, computer vision also plays an important role in object classification, providing accurate information about the location, shape and characteristics of products. While these technologies offer clear advantages, such as the ability to operate in variable environments and to identify objects quickly, they also have potential limitations, such as the need for proper

lighting and the dependence on the quality of the images captured. There are various state-of-the-art sorting solutions based on different tools, but with the same goal in mind.
For example [2] proposes a MELFA RV-2SDB robotic manipulator join with two cameras (the first to identify the pieces by their geometric shape and the second to detect people within the robot's work area). The images are processed in real time to send control commands to the arm. The central point of each pieces is determined and sent to the robot that will pick it up. The recognition operations are performed through OpenCV while the manipulation actions by means of Matlab, computing the direct mathematical model of the robot based on Denavit-Hartenberg method and the inverse kinematics through the Newton-Raphson iterative method. The two tools are connected through ID Qt to send the coordinate of the objects computed by OpenCv to Matlab through a TCP/IP communication protocol.
There exist also sorting solutions based on colour recognition by means of OpenCv, like the one proposed by Nikita V. Belov and Andrey G. Vovik [3], in which the segmentation is done in the HSV colour space (colour changes when moving around a cylinder circle). The connection between OpenCv and OpenShow Var module (a Java open-source cross-platform communication interface to Kuka robots that allows for reading and writing variables and data structures of the controlled manipulators) is realized by means of C3 Bridge Interface Server Software.
Another solution proposed by the International Journal of Computer Science and Information Technology [5] is based again on the colour classification but the manipulation actions are performed through Moveit. In this case the robot chosen is not an anthropomorphic manipulator but an XYZ manipulator because is more precise and can achieve millimeter level positioning accuracy fastly.

One of the most important sector in which sorting is advancing is the logistics, as a way to improve efficiency. In the current literature the aim is always to increase the adaptability and flexibility of industrial robot that sort objects in complex environments by equipping them with a camera and to perform autonomuous identification, localization, grasping and other controls [6].
Another important field for sorting applications is the agriculture sector, in which robotics is advancing more and more as a consequence of the lack of youth interest in this sector. For instance, this papar [7] presents a machine-controlled fruit sorting able to effectively distinguish between a good quality and a bad quality fruit. This solution is implemented again through Matlab, while the dispensing part is completed on the conveyor belt arrangement with the assistance of Arduino which takes decisions about the separation of fruit through mechanical separators. Regarding the recognition of the fruit, three features are extracted: Edge Features applying different kind of filters like sobel, Color Features through HSV colour space and Texture Features by means of entropy filter (that highlights edges by

brightening pixels which have dissimilar neighbors) and standard deviation filter. For Classification, the fusion of the three feature map images is used.
In the recent years, deep learning is advancing even more. Although it has taken over in the field of computer vision, sorting applications still tend to use traditional vision algorithms. However, it is being considered to adopt it in sorting applications for agriculture. For example, this innovative paper [8] presents a solution based on a deep learning model, implemented through an oriented bounding box label Software called OBBLabel. A multi-label recognition model YOLO-MLD conducts quality grading and posture perception on individual target with 93.4 % mean accuracy, so that precise position and quality information for all objects can be obtained in near real-time for subsequent suction cup-based sorting operations.

2.2 Computer Vision Overview

Computer vision is an interdisciplinary scientific field that focuses on how computers can acquire a high-level understanding from digital images or videos. It can be used to transform the tasks of engineering and management in construction by enabling the acquisition, processing, analysis of digital images, and the extraction of high-dimensional data from the real world to produce information to improve decision-making [9]. A vision system, essentially, takes images as input and provide their description. It is characterized by a hierarchical organization, from perception to interpretation. In particular [10]:

- Perception is the process that provides a computer image.
- Pre-processing deals with noise reduction and details improvement.
- Segmentation divides the image in objects of interest.
- Description compute characteristics (such as dimensions and shapes), useful to differentiate one object from another.
- Recognition is the process that identifies such objects.
- Interpretation gives a meaning to the recognized objects.

Computer vision is a wide field, employed in many applications but it is not the focus of this thesis. For this reason, only the parts applied in the project will be explained, namely the techniques of feature detection, description and image matching, and the concept of segmentation.

2.2.1 Feature Detection, Description and Image Matching

Within the broader field of computer vision, feature detection, description and image matching are critical components that facilitate effective object recognition. In particular, Feature Detection is the process of computing the abstraction of the image information and making a local decision at every image point to see if there is an image feature of the given type existing in that point [11]. An ideal feature detection technique should be robust to image transformations such as rotation, scale, illumination, noise and affine transformations.
In order to establish correspondences among a collection of images, where feature correspondences between two or more images are needed, it is necessary to identify a set of salient points in each image [12]. So, the process begins with detecting interest regions or key-points that are consistent across different transformations. For each detected region, an invariant feature vector, or descriptor, is created, capturing the image data around the key-point. These descriptors can be based on various methods, including second-order statistics, parametric models, or coefficients from image transforms.
In the context of classification, feature descriptors of a query image are matched with all the pre-trained images features, and the image with the highest correspondence is considered the best match. This matching process often relies on distance measures such as Euclidean or Mahalanobis to determine similarity.
Regarding the image registration, it is necessary to spatially align two or more images of a scene captured by different sensors at different times.

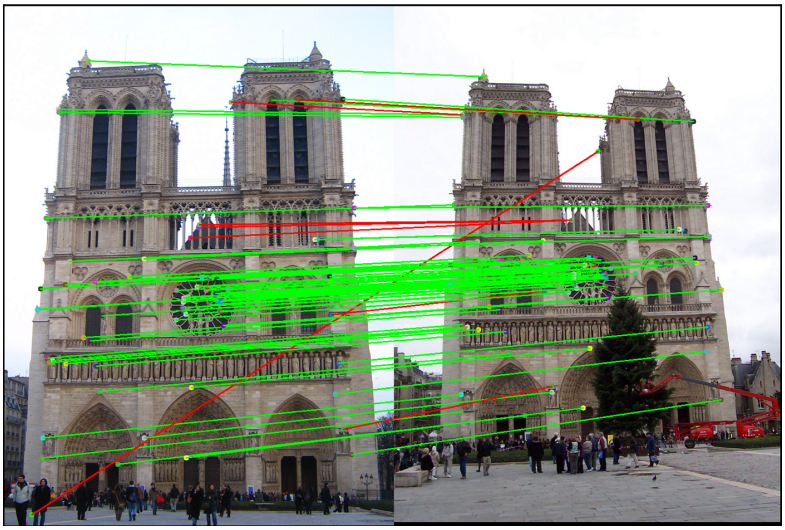

Figure 2.1: Feature Detection and Matching

Image features can be categorized into global and local features. The first ones (e.g., color and texture) describe the overall properties of an image, involving all pixels. Local features, on the other hand, aim to detect key-points or interest regions in an image and describe them. In this context, if the local feature algorithm detects n key-points in the image, there are n vectors describing each one's shape, color, orientation, texture and more. The use of global colour and texture features are proven surprisingly successful for finding similar images in a database, while the local structure oriented features are considered adequate for object classification or finding other occurrences of the same object or scene [12].
In literature, there exist different types of feature detectors and descriptors. Some of them will be described below:

- SIFT (Scale-Invariant Feature Transform): it is one of the most widely used algorithms for feature detectors and descriptors. It is designed to be invariant to scale, rotation, illumination, and viewpoint changes, making it highly effective at locating key points even in noisy, cluttered, and occluded environments. It identifies interest points in two steps. First, it generates a scale space by progressively smoothing the original image using Gaussian filters, which ensures scale invariance. Then, the original image is resized to half size, and again smoothing is performed using a Gaussian filter. This process is repeated to form an image pyramid, with the reference image at the bottom (level 1). The algorithm then computes the Difference of Gaussian (DoG) by subtracting two consecutive scales in the pyramid. In the second stage, key points are identified by examining the local extrema in the $3 \times 3 \times 3$ neighborhood of each pixel in the DoG pyramid. A pixel is considered a key point if its DoG value is an extremum (maximum or minimum) relative to its neighboring pixels across scales and space. SIFT also incorporates a descriptor extraction algorithm. This descriptor is a histogram that captures the orientations of local image gradients around each key point, encoding both the spatial and directional properties of the key point for robust matching. [13].

- FAST (Features from Accelerated Segment Test): it is a corner detector that identifies corners by first selecting a set of training images and then applying this algorithm. To determine the optimal detection criteria, namely the rules for deciding whether a pixel qualifies as a corner, machine learning is used. A decision tree is constructed to accurately classify all corners. This decision tree is then converted into C code, which is subsequently employed as a corner detector. While the FAST corner detector is well-suited for real-time applications due to its speed, it lacks robustness in the presence of significant noise. [13]

- SURF (Speeded Up Robust Features): it is a fast and robust method for feature detection and extraction in images. The key advantage of SURF lies

in its ability to perform rapid calculations using box filters, making it suitable for real-time applications. For interest point detection, SURF employs an approximation of the Hessian matrix, which offers a good balance between computation time and accuracy. The process of creating a SURF descriptor involves two main phases: first, determining a repeatable orientation based on data from a circular region around the key point; then, extracting the descriptor from a square area aligned with this orientation. [13]

- BRIEF (Binary Robust Independent Elementary Features): it is a descriptor based on binary strings. First, binary strings are computed from image patches, then the individual bits are generated by comparing the brightness of pairs of points along the same lines. It is fast compared with SIFT and SURF. [13]

- ORB (Oriented FAST and Rotated BRIEF): it is both a detector and descriptor and a good alternative to SIFT and SURF. ORB is a combination of the FAST detector and the BRIEF descriptor, with several enhancements to improve the performance. It adds a fast and accurate orientation component to the FAST algorithm and uses an image pyramid to generate multiscale features. For the descriptor, ORB adjusts BRIEF based on the orientation of the key points, making it rotation-invariant. [13]

- KAZE: it utilizes nonlinear diffusion filtering to detect and describe features in a nonlinear scale space because traditional (Gaussian scale space-based) methods attenuate both characteristics and noise in the same way, causing the decrease of accuracy and distinctive features of localization. The nonlinear scale space is constructed using efficient Additive Operator Splitting (AOS) methods and variable conductance diffusion. This approach allows blurring to be locally adapted to the image data, effectively reducing noise and enhancing localization accuracy and distinctiveness. [13]

- AKAZE: it is a multi-scale feature detection and description method, faster than KAZE. Previous methods for detecting and describing features in nonlinear scale space were time-consuming due to the high computational cost of constructing the nonlinear scale space. AKAZE addresses this by using the Fast Explicit Diffusion (FED) scheme within a pyramidal structure, significantly accelerating feature detection in nonlinear scale space. By employing FED techniques, nonlinear scale space may be created quicker than any other discretization technique. Additionally, AKAZE employs the Modified-Local-Difference Binary (M-LDB) descriptor, which is efficient, scale and rotation-invariant, and memory-efficient. [13]

2.2.2 Segmentation

Image segmentation refers to the process of dividing an image into meaningful and non-overlapping regions, and forms the basis of pattern recognition and image understanding. Each region is characterized by different features and are going to be extracted as region of interest (ROI) [14]. There are two difficulties in image segmentation:

1. How to define "meaningful regions", as the uncertainty of visual perception and the diversity of human comprehension lead to a lack of a clear definition of the objects, it makes image segmentation an ill-posed problem.

2. how to effectively represent the objects in an image. Digital images are made up of pixels, that can be grouped together to make up larger sets based on their color, texture, and other information. These are referred to as "pixel sets" or "superpixels". These low-level features reflect the local attributes of the image, but it is difficult to obtain global information (e.g., shape and position) through these local attributes.

There exist different algorithms in literature, that can be joined in three different methods:

- Classic Segmentation methods
- Co-Segmentation methods
- Semantic Segmentation Based on Deep Learning

Classic Segmentation methods

The classic segmentation algorithms were proposed for gray-scale images, which mainly consider gray-level similarity in the same region and gray-level discontinuity in different regions. In general, edge detection is based on gray-level discontinuity that is identified through the derivative or differential of the gray level. It is sensitive to noise, so it can be used only for images with low noise and complexity. For this reason, it was introduced the Canny operator characterized by strong denoising ability and also processes the segmentation of lines well with continuity, fineness, and straightness. However, it is more complex and takes longer to execute. Another edge detection method is the serial boundary technique, that concatenates points of edges to form a closed boundary. Usually, this method is based on graph search algorithm (in which the points are represented by a graph and the path with minimum cost is searched within the graph), and dynamic programming algorithm (that use heuristic rules to reduce the search computation).
In general, the region division strategy includes serial region division and parallel

region division. Thresholding is a typical parallel region division algorithm. The threshold is generally defined by the trough value in a gray histogram with some processing to make the troughs in the histogram deeper or to convert the troughs into peaks. [14]. The serial region technique involves sequential processes like region growing, where regions expand from seeds by merging similar pixels, and region merging, where adjacent regions are combined based on their gray value similarity.
Another classic method is the one based on graph theory, which maps an image onto a graph, which represents pixels or regions as vertices of the graph and the similarity between vertices as weights of edges [14]. There exist different techniques that optimize segmentation by globally analyzing these graphs.
Finally, K-means clustering is a special thresholding segmentation algorithm that is proposed based on the Lloyd algorithm. The algorithm operates as follows: (i) initialize K points as clustering centers; (ii) calculate the distance between each point K in the image and K cluster centers, and select the minimum distance as the classification k; (iii) average the points of each category (the centroid) and move the cluster center to the centroid; and (iv) repeat steps (ii) and (iii) until algorithm convergence.

Co-Segmentation Methods

The Collaborative Segmentation or Co-Segmentation methods involves extracting the common foreground regions from multiple images with no human intervention, to obtain prior knowledge. After this, it is possible to utilize the prior knowledge to process a set of images containing the same or similar objects. The extended model can be expressed as follows:

$$E = E_s + E_g \tag{2.1}$$

where E_s represents the energy function of seed image segmentation, that describes the difference between the foreground and background of the image and the smoothness of the image, and E_g represents the energy function of co-segmentation, that describes the similarity between foregrounds in a set of images. To achieve a good co-segmentation effect, segmentation energy E should be minimized. There exist two possible methods:

- improving the classic segmentation method to minimize E_s
- optimizing the unsupervised learning method to learn good representations in image sets to minimize E_g

For the classic segmentation model, E_s is the energy function derived using the Markov Random Field (MRF) method, and spitted in:

$$E_s^{MRF} = E_u^{MRF} + E_p^{MRF} \tag{2.2}$$

where E_u^{MRF} is the unary potential that measures the properties of the pixel itself, while E_p^{MRF} is the pairwise potential that measures itself in relation with other pixels [14]. Actually, the unitary potential represents the probability of a pixel belonging to class x_i when a feature of the pixel is y_i, while the pairwise potential represents the probability that two adjacent pixels belong to the same category. The co-segmentation term E_g penalizes inconsistencies in the foreground color histograms across multiple images.

Semantic Segmentation Based on Deep Learning

With the continuous development of image acquisition technology, the complexity of image details and the diversity of objects (e.g., scale, posture) have greatly increased. Low-level features such as color, brightness, and texture are no longer sufficient for achieving high-quality segmentation results. Feature extraction methods based on manual or heuristic rules cannot meet the complex demands of contemporary image segmentation, which calls for models with higher generalization ability. In the recent years, deep learning algorithms have been increasingly applied to segmentation tasks, and the segmentation effect and performance have been significantly improved. Initially, the approach involved dividing the image into small patches to train a neural network, which then classified the pixels. This patch classification algorithm has been adopted because the fully connected layers of the neural network require fixed-size images. Later, it was proposed fully convolutional networks (FCNs) with convolution instead of full connection, that made it possible to input any image size. FCNs demonstrated that neural networks could perform end-to-end training for semantic segmentation, laying a solid foundation for deep neural networks in this field. Subsequent models have advanced further based on the FCN architecture [14].

2.3 Pick and Place Overview

A pick and place robot is an automated robot that has the purpose of picking up items and placing them in another location (a conveyor or a stationary table) [15]. The automation of this repetitive task increases the speed of production and throughput rates. Pick and place robots are usually mounted on a stand that looks over items and joined with vision systems and sensors. A pick and place robot is composed by different elements, that are:

- A Robotic Arm: This is the extension of the robot. It is made from cylindrical or spherical parts and can extend and retract to reach items.
- End Effector: this element performs the action, usually made for gripping.

- Actuators: they allow for motion in the robotic arm and end effectors. There are various types of actuators, such as servo motors or stepper motors.
- Sensors: They are the eyes of the robot, used to identify the items.
- Controllers: they synchronise and control the movement of the actuators of the robot.

There exist different type of pick and place robot, including: [16]

- Robotic arm: they are the most common type of pick and place robot, usually are 5-axis for standard pick and place or 6-axis for more complex.
- Cartesian: They work in multiple planes using Cartesian Coordinates. Typically, they have better position accuracy.
- Delta: They are used in applications where robots pick items in groups and places them in assembly patterns or containers. They are characterized by advanced vision technologies and they have usually three arms that operate on four axes.
- Fast Pick: They are used in medium and high volume applications.
- Collaborative: They augment the work of humans by leading associates to pick locations and guiding associates through each task. By optimizing routes in real-time and keeping associates on task, collaborative robots help associates work more efficiently.
- SCARA: this kind of robots are known for their speed, precision, and compact design. They operate within a two-dimensional space defined by the X and Y axes, with an additional rotational movement around the vertical Z axis. This allows them to perform tasks such as assembly, packaging, and material handling with a high degree of speed and precision. The ability to perform both linear and rotational movements allows them to handle objects with complex shapes and orientations [17]

One of the key concept of pick and place is the motion planning, essential for enhancing the efficiency and effectiveness of robotic systems. Motion planning involves determining a feasible path for a robot to move from one position to another while avoiding obstacles and adhering to constraints. The next section provides an overview of the main concepts and algorithms involved in motion planning in pick and place robotics.

2.3.1 Motion Planning

Motion planning is a key problem in robotics that is concerned with finding a path that satisfies a goal specification subject to constraints. In its simplest form, the solution to this problem consists of finding a path connecting two states, and the only constraint is to avoid collisions. Given its complexity, most planning algorithms forego completeness and optimality for slightly weaker notions such as resolution completeness, probabilistic completeness, and asymptotic optimality. [18] In the state of the art, there exist different types of planners. For each of them, the main issue is the trade-off between computational complexity and the quality of the final path. The more precise the paths and with higher quality, the greater their computational complexity and consequently the time required to generate them.
Motion planning can be divided into two categories: offline motion planning and online motion planning [19]. Offline motion planning operates using a "scan-plan-execute" approach. It first scans the environment to detect obstacles, often through CAD models or point clouds converted into octomaps. Then, it generates a collision-free path based on this static snapshot of the scene. However, if the environment changes during execution, the path won't adapt, potentially leading to collisions. On the other hand, online planners continuously updates the environment while executing the path, making it more suitable for dynamic settings. However, this comes with a higher computational cost since the planner must recompute paths at each step. To balance path update frequency and computation time, the scene can be updated at multiple intervals.
Another common distinction in the motion planning world is between deterministic and nondeterministic planners. The former generate the path uniquely, so given the same initial input conditions (initial state, goal state, and collision objects in the scene) the planned path will always be the same. The latter generate it randomly, so the planned paths may differ even with the same input.
Regarding the planner, the main subdivision is among graph search-based, sampling-based and optimized-based planners. The first two planners generate a collision-free path without considering kinematic constraints such as path smoothness or velocity, acceleration and jerk profiles [20]. On the other hand, optimized-based algorithms plan also the timing law of the final trajectory, optimizing its kinematic parameters. This causes the trajectory to have the main parameters considered optimal (path length, execution time, smoothness, kinematics profiles, and so on) with, however, generally higher computational costs and failure rates [21]. Furthermore, in recent years much research and efforts have focused on post-processing type algorithms: these algorithms provide a hybrid solution between sampling-based and optimized-based algorithms. In this type of planner, there is the generation of an initial collision-free path by a sampling-based planner and then the path is optimized

thanks to an optimized-based planner [22].

Graph search-based algorithms

Graph search-based planning uses graph search algorithms to compute discrete paths in a robot's state space [19]. It involves two main challenges: converting the problem into a graph and finding the best solution within that graph. The state space is discretized, limiting the number of possible positions. Advanced algorithms, like Dijkstra's, assign costs to nodes and arcs to guide the search for the optimal path. There are two main categories: depth-first (which explores one node deeply before backtracking) and breadth-first (which explores all nodes at each level before moving deeper). Breadth-first is more complex but guarantees the shortest path. The key advantage of graph search-based planners is completeness. If a solution exists, the planner will find it and provide the optimal path based on length. However, the discretization reduces accuracy, and building the graph for the entire state space is computationally heavy, especially in complex environments. This leads to unnecessary planning time. For this reason these kind of planners are not longer used in industrial applications, in favour of Sampling-based planners.

Sampling-based algorithms

Sampling-based planners are the most common planners, because they are very quick and quite effective even in complex environments [23] [20] [19] [24]. These planners randomly sample the configuration space and check if each sampled point is collision-free. If it is, the point is added to the graph of possible configurations, connecting it to others when possible. The algorithm then finds a collision-free path from the start to the goal. The success rate of these planners depends strongly on the complexity of the scene and the maximum planning time set.
An important distinction of this type of random-sampling planners is the one between single-query planners and multiple-query planners. In single-query planners, the important concept is the speed in computing a single path to be executed. If another path is then to be planned, it restarts from the beginning and it is necessary to re-initialize everything, since nothing is stored of the previously planned paths. In multiple-query planners, on the other hand, it is assumed in principle that there will be many motion planning problems to be solved in the same environment, so it is essential to save the information of each path in order to speed up subsequent computations, having no need to re-initialize the algorithm. The main drawback of sampling-based planners is their non-optimality. The resulting paths may be inefficient, with abrupt changes in direction, leading to higher energy consumption and less smooth, potentially unsafe paths for the robot.

Optimized-based algorithms

These kind of algorithms have been developed to overcome the limitations of the sampling-based algorithms. They compute the final trajectory as the solution of an optimization problem, considering a cost function with various constraints (e.g., maximize the distance to obstacles in the environment, consider limits in the velocity, acceleration and jerk profiles, and so on). The main disadvantage in optimized-based planners turns out to be the computational complexity. It grows considerably compared to sampling-based planners, thus resulting in longer planning times and making them unsuitable for online planning. Another major disadvantage is the dependence on initial conditions. This leads them to trap into local minima rather than global minima, thus causing an increase in failure rates compared to sampling-based planners [21].

Chapter 3

System Design

The sorting application requires a planning framework to execute the pick and place action. In this thesis, the focus is on the MoveIt 2 planning framework which is ROS2 Humble based, deployed within a Docker container. Furthermore, computer vision libraries was added for object detection and sorting. The principal actor of this job is the manipulator UF850. This chapter provides a detailed explanation of the frameworks and tools used for the simulation and implementation of the sorting system. It begins with an overview of ROS2, explaining its significance in robotic applications and the advantages of using Docker for creating a consistent and portable development environment. It then introduces MoveIt 2, RViz, and Gazebo. Following this, the chapter will explore the specific characteristics of the UF850 manipulator and its integration into the ROS2 and MoveIt 2 ecosystem.

3.1 ROS2

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications [25]. It is open source and provides a communication middleware that allows nodes (independent execution units) to exchange messages through an anonymous publish/subscribe mechanism. The ROS graph is the heart of ROS2 since it represents the network of nodes and the way in which they communicate. Thanks to its structure, ROS has a lot of advantages:

- Modularity: Each node is launched and executed independently of the others. This structure allows for the design, development, and testing of individual nodes, making them reusable for other purposes since each node has a specific function.
- Portability: Due to the structure of independent nodes, each performing a specific function, nodes can be easily carried and used for other applications.

- Effective Communication: The messaging system of ROS2 is simple and scalable. Nodes can communicate with each other via topics or services, enabling the transmission of sensor data, status, control information, or any other type of data.
- Flexibility: This derives from the previous features and the support for various programming languages, including the most common ones, Python and C++.
- Graphics: ROS2 supports simulation environments such as RViz and Gazebo, allowing for graphical simulations by loading models like URDFs (Unified Robot Description Format) that describe the geometry and kinematics of the robot.

As it was already mentioned, when implementing an application in ROS2, two key components for node communication are topics and services. Both enable data exchange between nodes, but they serve different purposes and operate in distinct ways. In particular:

- **Topics** should be used for asynchronous and continuous data streams, like sensor data, robot state, etc. A node can publish information on a topic, while other nodes can subscribe to that topic to receive data in real-time.

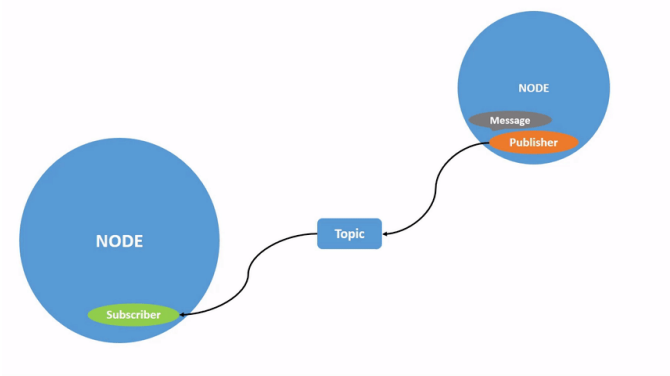

Figure 3.1: Topics Communication mechanism

- **Services** are designed for synchronous communication. They work like remote procedure calls, where a node can send a request to another node, which will perform a computation and return a result. A service consists of two main parts: a service client, which sends the request, and a service server, which processes the request and returns the result.

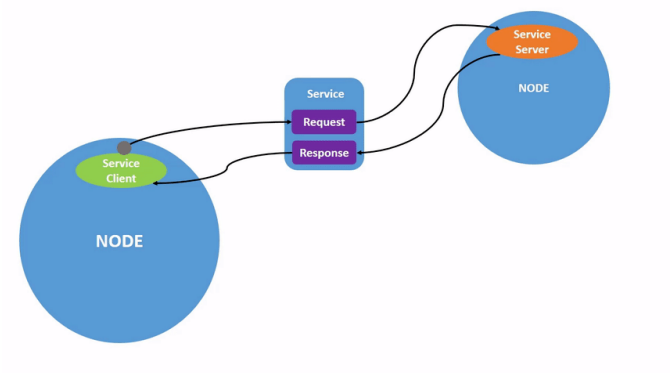

Figure 3.2: Services Communication mechanism

Nodes in ROS2 can be a complex combinations of publishers, subscribers, service servers, service clients, all at the same time. To manage multiple nodes efficiently, ROS2 provides a launch system that automates the execution of nodes with a single command. This allows users to define their system configuration — specifying which programs to run, their arguments, and how they interact — within a launch file written in Python, XML, or YAML.

3.2 Docker

Docker is an open platform for developing, shipping and running applications [26]. It allows to separate the applications from the infrastructure, thanks to an isolated environment called Container in which it is possible to run and package the applications. It is possible to run many containers simultaneously and each of them is lightweight and fast. Another important property is the portability since it can run on any system regardless of the underlying hardware or operating system. Docker uses a client-server architecture. The main elements are:

- The Docker client communicates with the Docker daemon, which handles the essential tasks of building, running, and distributing Docker containers. They can run on the same system or it is possible to connect a Docker client to a remote Docker daemon. It is a command-line interface (CLI) that allows users to interact with the Docker daemon through three main commands: docker run, docker build and docker pull. It can communicate with more than one Daemon.
- The docker daemon listens for Docker API requests and menages Docker object such as images, containers, networks and volumes.

- The Docker registry stores the Docker images, such as Docker Hub that is public and anyone can use to push their images to a registry or pull existing images to use in their projects. An image is a read only template with instructions to create a Container and can be based on another image like Ubuntu.

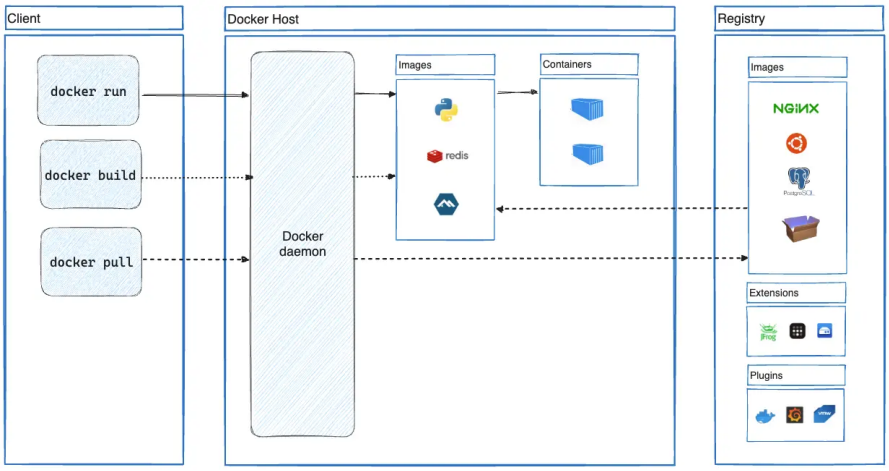

Figure 3.3: Docker architecture

Therefore, the container is a runnable instance of an image than can be created, started, stopped, moved or deleted using the Docker API or CLI.

3.3 MoveIt 2

MoveIt 2 is the robotic manipulation platform for ROS 2, and incorporates the latest advances in motion planning, manipulation, 3D perception, kinematics, control, and navigation. [27] It consists of a series of ROS libraries and implements functions perfectly integrated with the rest of the framework, it uses RViz visualization software and Gazebo simulation software; it also has modules for the definition of tasks and a module entirely dedicated to grasping.
MoveIt essentially allows to implement every functionality described above, from trajectory planning to collision avoidance and sensor functionality (for 3D perception), it also implements functionality for a further abstraction of the controllers, passing from the one already provided by ROS (ros control) by virtualizing a control to high level based not on strength or position but on the trajectory itself.
Nevertheless, it is essential to create a MoveIt package with the all the necessary configuration and launch file essential for controlling the robot, such as the SRDF(Semantic Robot Description Format) file and other configuration files that

are used in the MoveIt pipeline [28]. Typically, the Setup Assistant simplifies the generation of this package from the URDF(Unified Robot Description Format), but for the UF850 custom packages were available on GitHub.
In addition, sensors play a crucial role by providing important data such as the robot's joint state, point cloud information, and TF frames, all of which are essential for MoveIt's core functionality.

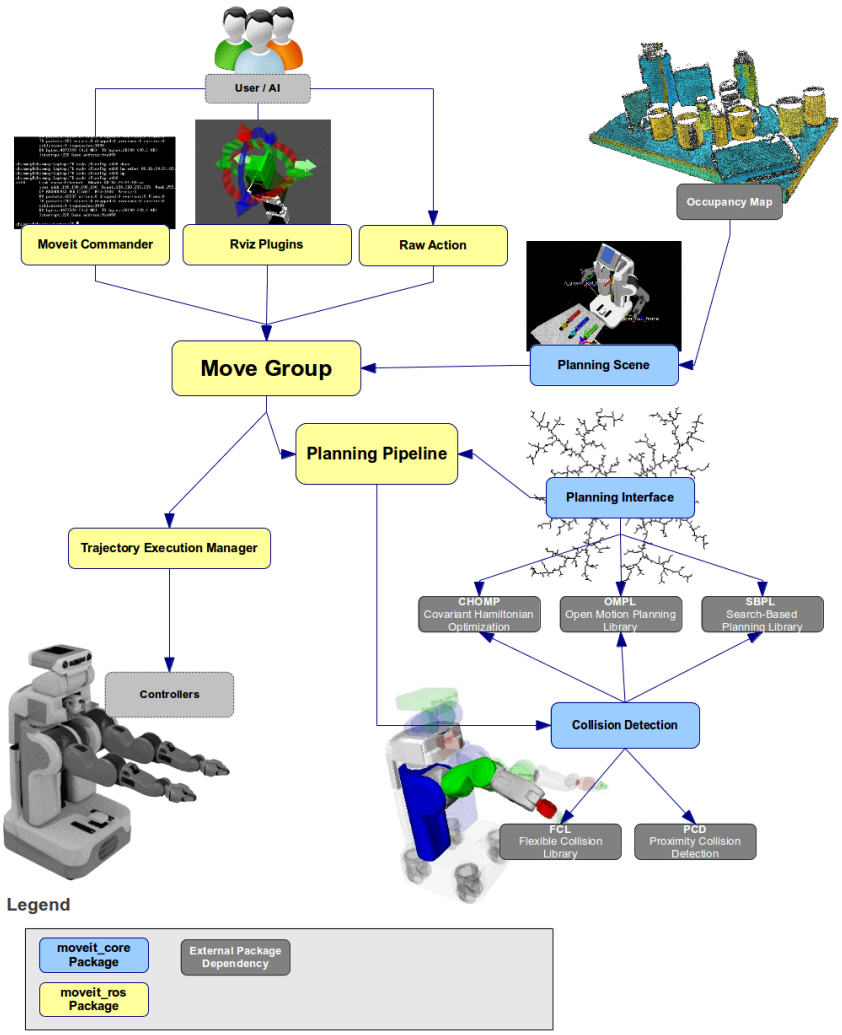

Figure 3.4: Moveit pipeline

The most important node provided by MoveIt is the move_group. The high-level system architecture for this node is shown in Figure 3.4, while the node structure is shown in Figure 3.5.

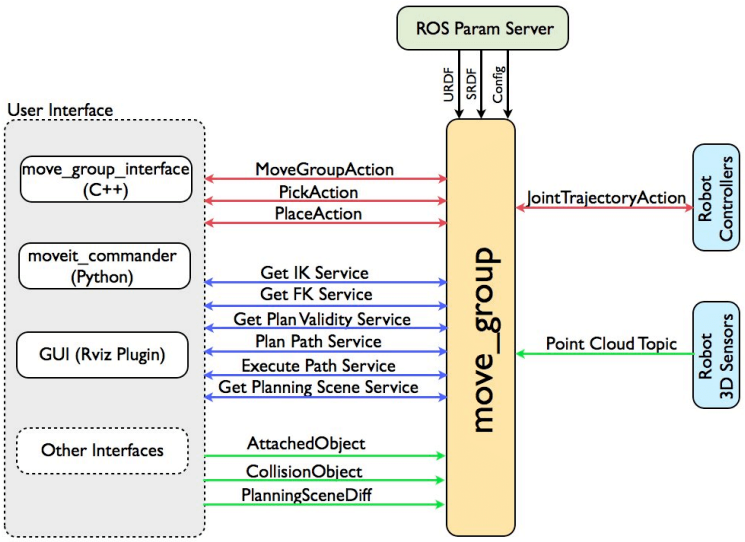

Figure 3.5: Move Group Architecture

This node serves as an integrator: pulling all the individual components together to provide a set of ROS actions and services for users to use [27]. For the user it is very easy to access the actions and services provided by move_group, since it can be configured using the ROS param server. From this server, the move_group ROS node gets three pieces of information:

- URDF: the robot description parameters.
- SRDF: the robot description semantic parameters.
- Moveit configuration: other data as joint limits, kinematics, motion planning and perceptions.

The move_group node comunicates with the robot through ROS topics and actions getting all the current state informations: the joint positions, the point clouds of the scene, the data coming from the other sensors and from the controllers on the robot, and so on.

3.3.1 Motion Planning in MoveIt

Motion planning is one of the most important problem in robotics that is related with finding a path that satisfies a goal specification subject to constraints, as it was already explained in the section 2.3.1.
MoveIt works with motion planners through a plugin interface, allowing users to select and utilize different planning algorithms based on their needs. This

interface to the motion planners is through a ROS action or service offered by the move_group node. This node acts as an interface for sending motion planning requests and receiving planned trajectories. Key aspects of this interaction include:

- Motion Plan Request: When a motion planning request is made, it is specified the desired goal (e.g., moving an arm to a new location or changing the pose of an end-effector) and the planner through the planning_pipeline and planner_id parameters. Constraints for the motion planner to consider are also provided.
- Trajectory Generation: The move_group node is responsible for generating trajectories based on the motion plan request. It translates the path provided by the Motion Planner (without timing information) into a complete trajectory. This trajectory takes into account the maximum velocity and acceleration limits for the robot's joints, as specified in a configuration file (e.g., joint_limits.yaml).
- Planning Request Adapters: MoveIt includes several trajectory processing algorithms and planning request adapters. These components work together to refine and optimize the generated paths, ensuring that they are time-parameterized and feasible under the robot's operational constraints.

MoveIt supports a variety of motion planners through its plugin interface. Below there is a list of most commonly used planners, in descending order of popularity/-support within MoveIt:

Open Motion Planning Library (OMPL) : it is an open-source motion planning library that consists of many state-of-the-art sampling-based motion planning algorithms [29] [27]. MoveIt integrates directly with OMPL and uses the motion planners from that library as its primary/default set of planners. The planners in OMPL are abstract, in fact OMPL has no concept of a robot.

Pliz Industrial Motion Planner : it is a deterministic generator for circular and linear motions. Additionally, it supports blending multiple motion segments together using a MoveIt capability.

Stochastic Trajectory Optimization for Motion Planning (STOMP) : it is an optimization-based motion planner partially supported by Moveit. It does not require gradients, and can thus optimize arbitrary terms in the cost function like motor efforts.

Search-Based Planning Library (SBPL) : it is a generic set of motion planners using search based planning that discretize the space. Integration into latest version of MoveIt is work in progress.

Covariant Hamiltonian Optimization for Motion Planning (CHOMP) : it is a novel gradient-based trajectory optimization procedure that makes many everyday motion planning problems both simple and trainable. This algorithm utilizes covariant gradient and functional gradient methods in the optimization phase to create a motion planning algorithm that is entirely based on trajectory optimization. Given an infeasible trajectory, CHOMP reacts to the surrounding environment to quickly pull the trajectory out of collision while simultaneously optimizing dynamical quantities such as joint velocities and accelerations. It rapidly converges to a smooth collision-free trajectory that can be executed efficiently on the robot. Integration into latest version of MoveIt is currently in progress.

Each of these planners has its strengths and is chosen based on the specific requirements of the motion planning task at hand. In summary, the integration of MoveIt with various motion planners and its use of trajectory processing algorithms enable complex and efficient motion planning in robotic systems. By leveraging these tools, it is possible to achieve precise and optimized robot movements, fulfilling the desired goals while adhering to system constraints.

3.3.2 Planning Scene Monitor

The planning scene stores the current state of the robot and the world around it. The internal state of the planning_scene object is typically maintained by a planning_scene_monitor. Particularly the planning scene monitor listens to:

- State Monitor: it tracks state of robot and attached object.
- Scene Monitor: it listens from an externally defined planning scene.
- World Geometry Monitor: it builds world geometry using information from the sensors on the robot such as depth cameras and from user input. It uses the occupancy map monitor described below to build a 3D representation of the environment around the robot and augments that with information on the planning_scene topic for adding object information. [27]

The occupancy map monitor handles 3D perception in Moveit using a plugin architecture. Specifically, MoveIt supports two types of sensor input:

1. Point Clouds
2. Depth Images

3.4 Rviz

RViz is a 3D visualization tool with a graphical user interface (GUI) natively supported by ROS [30]. It enables detailed viewing and analysis of any robotic system model or the environment in which the system operates. Additionally, RViz can acquire and reproduce data from any sensor for which a specific module has been implemented in ROS.
RViz is highly modular, allowing the integration of additional functionalities through appropriate plugins. For example, the MoveIt plugin enables operations and commands directly from the plugin interface, making RViz a tool with capabilities that extend far beyond simple visualization.

3.5 Gazebo

Gazebo is an open source 3D robotics simulator natively supported by ROS, able to reproduce accurately the environments that a robot may encounter, from gravity to wind as regards the environments, from the dynamics of the controllers to the behavior of the sensors (taking into account the noises) as regards the robots, and from the distribution of inertias to friction for any model in general [31]. Gazebo is a comprehensive simulator equipped with all necessary functionalities. For example, it uses its own format, the SDF (Simulation Description Format), for describing robot models and environments, with many models freely available. Additionally, it offers a wide variety of plugins and supports custom-developed plugins.

3.6 UFactory 850

Figure 3.6: UF850

The UFACTORY 850 Robotic Arm is a robotic arm manipulator with six degree of freedom. [32] It has a 5 kg payload and ±0.02 mm repeatability. It is able to reach 850mm along the horizontal axis and is characterized by a maximum speed equal to 1 m/s. The joints are limited physically about the maximum reached angles. In the tables below, the working range of each joint and the range of various motion parameters of the robotic arm.

Joint 1	±360°
Joint 2	$-132° \sim 132°$
Joint 3	$-242° \sim 3.5°$
Joint 4	±360°
Joint 5	$-124° \sim 124°$
Joint 6	±360°

Table 3.1: On the left the Joint name, on the right the working range for the relative joint

	TCP motion	Joint Motion
Speed	$0 \sim 1000$ mm/s	$0 \sim 180°/s$
Acceleration	$0 \sim 50000$ mm/s^2	$0 \sim 1145°/s^2$
Jerk	$0 \sim 100000$ mm/s^3	$0 \sim 28647°/s^3$

Table 3.2: In this table the physical limits of the robot in terms of speed, acceleration and jerk

Regarding the gripper, the value range opening and closing is: -10 to 850. The larger the value, the greater the stroke of the gripper, meaning the smaller the value, the smaller the stroke of the gripper. If the clamping is not tight, a negative value can be set until it is tightened. The speed of the gripper should be in 1000-5000mm/s. If a speed less than 1000 was set, the gripper may not work.

Figure 3.7: Gripper of UF850

The robot is equipped with a RealSense depth camera D435i, which combines the robust depth sensing capabilities of the D435 with the addition of an inertial measurement unit (IMU) [33]. The inertial measurement unit (IMU) is used for the detection of movements and rotations in 6 degrees of freedom (6DoF). An IMU combines a variety of sensors with gyroscopes to detect both rotation and movement in 3 axes, as well as pitch, yaw and roll. This allow a better Point Cloud alignment and high quality depth data, so it is optimal for object detection applications.

Figure 3.8: RealSense d435i

3.6.1 Software Integration and Repository

To simulate and control the UFactory 850 robotic arm within the ROS2 environment, the corresponding simulation models and control packages were installed from the official GitHub repository. This repository provides essential tools to enable motion planning and control of the UFactory xArm series in simulation environments such as Gazebo. The repository is structured into several key packages, including:

- xArm Description: This package provides the URDF files and 3D models necessary for simulating the xArm robots in various environments. These files define the physical properties of the robot, including its geometry, joints, and link structure.
- xArm Controller: This package includes tools and configurations to manage the control of the xArm robots. It contains control scripts that interface with ROS2 to send commands to the robot, allowing for precise control of its movements and real-time adjustments during operation.
- xArm Moveit Config: this package provides abilities for controlling xArm/Lite6 (simulated or real arm) by moveit. It includes the SRDF files, motion planning settings, controllers, and predefined planning groups that allow for seamless integration with MoveIt. By utilizing this package, advanced motion planning scenarios were tested, enabling efficient path generation and collision avoidance during pick-and-place operations.
- xArm Planner: It gives functions for controlling xArm (simulated or real arm) through Moveit API.
- xArm Gazebo: This package is specifically designed to work with Gazebo, providing simulation environments and configurations that allow for the virtual testing of the xArm manipulator. It includes plugins that enable realistic physics and sensor simulations, making it possible to evaluate the robot's performance before deploying it in real-world applications.
- xArm msgs: This package contains all interface definitions for xarm_ros2. In particular includes all the services and the messages.
- xArm Moveit Servo: it serves as a demo for jogging xArm with devices such as joystick.
- Third part: it provides the custom Real Sense Gazebo plugin.

Chapter 4

Implementation

To implement the sorting application, the project was divided into three main parts. First, a pick-and-place system was developed on a robotic manipulator (UF850) using the MoveIt Task Constructor Library. Next, computer vision techniques were employed to dynamically determine the position and orientation of the objects. Finally, the Point Cloud Library (PCL) in ROS2 was utilized to obtain the geometric dimensions of the objects. In the final phase, these three components were integrated using ROS2, enabling the sorting functionality.

4.1 Pick and Place

The Pick and Place constitutes the first phase of the project. For this aim, the official tutorial of Moveit2 is followed, adapted to the UF850 manipulator. In the ROS2 context, the application operates as a node that has to be launched by means of a launch file because it needs some parameters, which are the URDF, SRDF and planner parameters (the choice of the planner is explained in the section 5.1). In addition, to create the environment for the simulation, several nodes must be launched together, including move_group, ros2_control, static_tf, robot_state_publisher, and rviz.
Initially, the application was designed to handle a single object in the scene, but it was later extended to manage multiple objects within the same environment. The implementation was based on the MoveIt Task Constructor framework, which provided a flexible and modular approach to task planning and execution. Before explained how this particular package of MoveIt was used for the thesis work, it is important to give a general overview of its main functionalities.

4.1.1 Moveit Task Constructor

The Moveit Task Constructor is a framework of Moveit used to simplify complex planning tasks with multiple interdependent sub-tasks called stages. A MTC stage refers to a component or step in the task execution pipeline. Information from the subtasks are passed through the InterfaceState object. Stages can be arranged in any arbitrary order and their hierarchy is only limited by the individual stage types. The order in which stages can be arranged is restricted by the direction in which results are passed. [27]. There exist three possible kinds of stages:

- Generators
- Propagators
- Connectors

Moreover, there are different hierarchy types allowing to encapsulate subordinate stages:

- Wrapper
- Serial Containers
- Parallel Containers

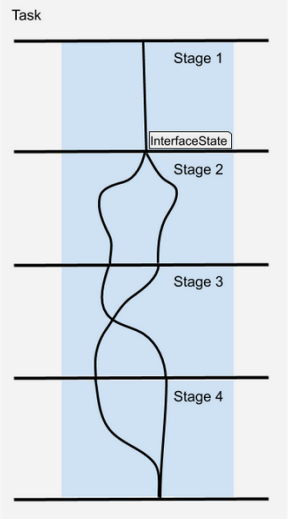

Figure 4.1: Moveit Task Constructor pipeline

In the following subsections, a definition of the most important types of stages in the MTC framework is presented.

Generator Stage

Generator stage do not take input from the near stages but just compute the results and pass them in both directions (forward and backward).

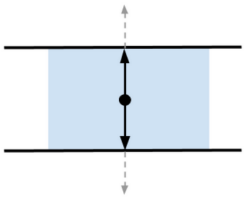

Figure 4.2: Generator Stage

The execution of an MTC task begins with the Generator stages. There are three types of Generator Stage, with the most important CurrentState which gets the robot's current state as the starting point for the planning pipeline. Moreover, there exist the Fixed State that spawns a pre-defined Planning Scene State as initial point and the FixedCartesianPose which is useful for planning based on a predefined future state without sampling and allow the simultaneous planning and execution. Finally, the Monitoring Generators which monitors the solution of another stage (not adjacent) and use it to generate something else.
Moreover, there are four kinds of Monitoring Generators:

1. The GeneratePose which generate pose based on a solution given by the monitored stage.

2. The GenerateGraspPose stage derives from GeneratePose, which monitors the CurrentState stage to find the location of the object to be grasped.

3. The GeneratePlacePose stage derives still from GeneratePose and generate pose for a place pipeline.

4. The GenerateRandomPose derives from GeneratePose and sets up Random-NumberDistribution sampler for a PoseDimension to randomize the pose.

Propagating Stage

Propagators receive solutions from one neighbor stage, elaborate those solution and spread them to the neighbor situated on the opposite side. This kind of stage can pass solution forward, backward or in both directions.

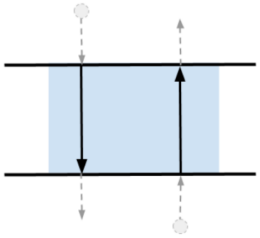

Figure 4.3: Propagating Stage

The MTC provides the following propagating stages:

- ModifyPlanningScene
- MoveRelative
- MoveTo
- FixCollisionObject

The ModifyPlanningScene applies modification to the PlanningScene without moving the robot, therefore the default cost term is equal to 0. The stage contains different functionality such as enable and disable collision checking between links or attach and detach objects to robot links.
The MoveRelative stage is useful to carry out a Cartesian motion with a default planning time equal to 1.0s. The default cost term depends on the length of the path.
The MoveTo stage is used to move to a joint state or Cartesian goal pose that can be specified in different formats. Again, the default planning time is equal to 1.0s and the default cost term is related to the path length.
The last type of Propagating Stages is the FixCollisionObject with a cost term constant at 0. It detects collision and resolves them if applicable.

Connecting Stage

Connectors do not produce results them-self; instead, try to connect the start and goal inputs provided by adjacent stages.

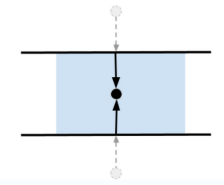

Figure 4.4: Connecting Stage

The MTC provides only one connecting stage called Connect stage, which connect two stages by finding a motion plan between two states. The default planning time is equal to 1.0s, while the cost depends on the path length.

Wrapper

Wrappers encapsulate another stage to modify or filter the results. The MTC provides three different kinds of Wrapper:

1. ComputeIK to compute the inverse kinematics for the poses in Cartesian space.
2. PredicateFilter used to filter generated solutions.
3. PassThrough, a generic wrapper used to set a custom CostTransform to change a solution's cost.

Serial and Parallel Containers

Serial Containers arrange stages sequentially, considering only end-to-end solutions as results. By default, an MTC Task is stored in a serial container.
Instead, Parallel Containers group multiple stages together, enabling the planning of alternate solutions. The MTC provides three kinds of stage:

1. Alternatives
2. Fallback
3. Merger

Alternative containers allow adding stage to be executed in parallel. All the solutions are collected at the end and ordered on the basis of the cost. The Fallback Container executes all the consequential stages until one of them return success or all of them return failure, while the Merger Containers combine multiple different problems and all the solutions are merged in a single one.

4.1.2 Pick and Place using MTC

The Pick and Place application realized for the UF850 is divided in three main parts:

- Setup of the Planing Scene
- Creation of the task
- Planning and execution of the task

The Setup of the Planning Scene is crucial because allows to the MoveIt planner to consider the objects when generating paths. The objects was defined as Collision Objects of Moveit that the robot avoids by default or can pick if it is specified in the Creation of the task (this will be clarified later). Each object is declared in the scene, specifying the type of object (for this work research, only primitive shapes are considered such as cylinders or box), the geometric dimensions of the object (for example radius and height for cylinders), a unique identifier (ID), the position and orientation of the object with respect a specified frame (for this application, the link base frame of the robot is considered). The objects are added to the scene through the PlanningSceneInterface that uses ROS interfaces to communicate changes to the planning scene to MoveGroup.
In order to create the task, three kinds of solvers was been defined with the aim of defining the type of robot motion:

1. PipelinePlanner uses MoveIt's planning pipeline, tipically OMPL. For this planner, it is important to specify the maximum goal joint tolerance.

2. JointInterpolation that interpolates between the start and the goal joint states, tipically used for simple motions because it is very quick.

3. CartesianPath that allow to move the end effector in a straight line in the Cartesian Space. Also in this case, some parameters are set, such us the maximum velocity, the maximum acceleration allowed and the step size.

Furthermore, it is important to set some initial properties, such as the Task name, load the robot model, define the names of some useful frames, and set those frame names as properties of the task. In this case, the joint group name of the arm, that of the gripper and the gripper's main link are specified. These elements will be useful in the stages.
Regarding the task, the stages mechanism of the MTC is used. In particular, the pipeline is composed by the following stages:

Current : a generator stage that starts the robot in its current state.

Open hand : a MoveTo stage (propagator) that uses the joint interpolation planner with the pose defined in the SRDF of the manipulator.

Move to Pick : a Connect Stage (Connector) to move the arm in a position where it can pick the object. The UF850 arm is specified as planning group and the PipelinePlanner as planner.

Pick Object : a serial container that contains all the stages related to the picking action. It contains the following stages:

- Approach Object: a MoveRelative stage (propagator) that uses a Cartesian Planner to find a solution. It is needed to set a forward direction with respect to the hand frame where the robot should move.
- Grasp Pose IK: it is a wrapper that encapsulates the Generate Grasp Pose stage to compute the inverse kinematics solutions for the grasping.

  Generate Grasp Pose: a monitoring generator stage that monitors the current state and determines the number of poses to generate. So, MTC will try to grasp the object from many different orientations specified by the angle delta.
- Allow Collision: a ModifyPlanningScene (propagator) that allows to disable a certain collision between a group of joints (in this case, the hand group joints of the robot) and a certain object that has to be pick.
- Close hand: Similar to the stage Open Hand, moving in the close position defined in the SRDF.
- Attach Object: a ModifyPlanningScene stage, that is used to attach the object to the hand of the manipulator.
- Lift Object: a MoveRelative stage, similar to Approach Object, it set the upward direction.

Move to Place : a Connect stage to connect the pick phase with the place; in this case, it is set the Pipeline Planner for the arm and the Joint Interpolation Planner for the hand.

Place Object : a serial container for place action. It contains the following stages:

- Place Pose IK: it is a wrapper that encapsulates the Generate Place Pose to compute as before the inverse kinematics solutions to reach the place pose.

  Generate Place Pose: a Monitoring Generator Stage that monitors the Attach Object Stage and set the pose where the object has to be placed.
- Open Hand: the same as before.
- Forbid Collision: Similar to Allow Collision stage, but in this case it is enabled the collision property of the object.
- Detach Object: a ModifyPlanningScene stage to detach the object from the arm.
- Retreat: it is a MoveRelative stage, similar to Lift Object Stage, it applies the Cartesian Planner and sets the retreat direction of the arm.

Return Home : it is a MoveTo stage that uses the Joint Interpolation Planner and passes it the goal pose "home" defined in the UF850 SRDF.

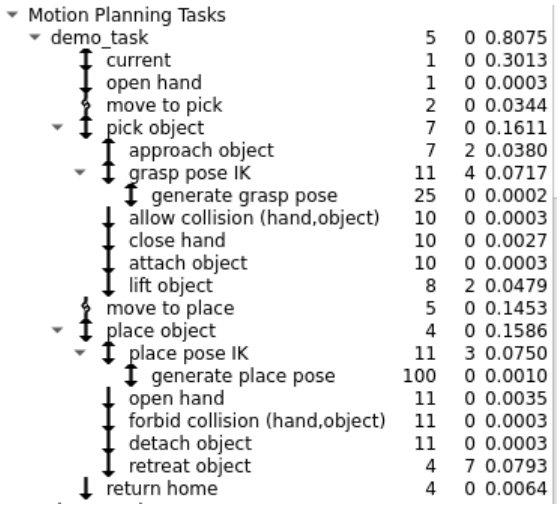

Figure 4.5: Stages of the Pick and Place application

Finally, task is planned, executed and visualized in Rviz. Execution occurs via an action server interface with the RViz plugin that is the Execute Task Solution Capability plugin provided by MTC. The plugin extends MoveGroupCapability, constructing a MotionPlanRequest from the MTC solution and uses MoveIt's PlanExecution to actuate the robot.

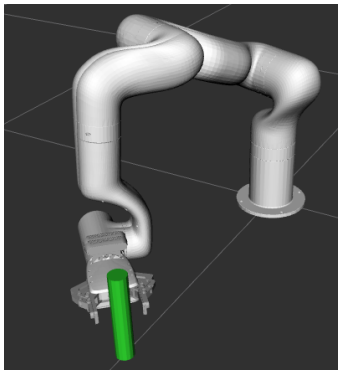

Figure 4.6: Robot during the Stage ApproachObject

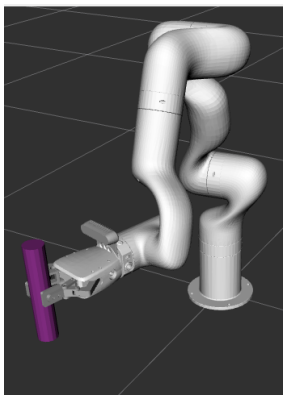

Figure 4.7: Robot during the Stage Lift Object

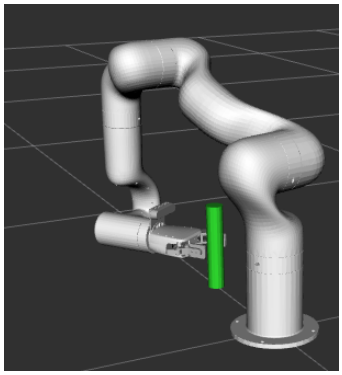

Figure 4.8: Robot during the Stage Detach Object

4.2 Object Recognition and Localization

In the second phase of the project, the concept of recognition is added with the aim to find dynamically the position and orientation of the objects. To accomplish this task, Find Object 2D is engaged, an open source object recognition package from ROS2 that uses feature detection algorithms. Once the objects are recognized, Find Object 2d is able to attach a frame on their central point and, as a consequence, to find their position and orientation with respect to the camera frame. After this, a transformation with respect to the link base was computed to get the position and orientation needed to build the planning scene. In this section, it is explained how Find Object 2d works, which algorithm is used and what it was implemented to localize the objects.

4.2.1 Object Recognition through Find Object 2D

Find Object 2D is a ROS package based on OpenCV library implementation of SIFT, SURF, FAST, BRIEF and other feature detectors and descriptors. Using a webcam, objects can be detected and published on a ROS topic (/info) with object ID, position (pixels in the image), homograph matrix and other information.
Object recognition processing consists of two steps. The first is training and should be done in a preliminary phase. During this stage, the object of interest is presented to the vision system, and the algorithm automatically collects a set of features, storing them sequentially as a pattern. The second stage is recognition, which is performed continuously while the robot is running. Each frame from the camera is processed, image features are extracted and compared with the data set in memory. If enough features match the pattern, the object is recognized [34].

Figure 4.9: Find Object 2D: Object Recognition

For this thesis, the SIFT algorithm (Scale Invariant Feature Transform) is adopted as detector and descriptor (the reason for which it is chosen will be explained in the section 5.2). Once the object is recognized, it is localized. As it was already mentioned, a message called Detection Info is published on a topic called "/info"

by the node "find object 2d" located in the homonymous package. This node has to be launched through a launch file, in which it is important to specify the path in which the objects used in the training phase are located and the topics in which the data related to the camera sensors acquisition are saved. In particular, three topics are specified:

- */color/image_raw*: it typically publishes raw color images from a camera sensor. This raw image data is crucial for tasks that require high-quality image input.
- */aligned_depth_to_color/image_raw*: it publishes raw depth images that have been aligned to the color camera's frame of reference. This means the depth data has been adjusted so that it corresponds directly to the color image, pixel by pixel. Such precise alignment is essential for applications requiring accurate depth and color data integration, including 3D reconstruction, object detection, and augmented reality.
- */color/camera_info*: it publishes the intrinsic and extrinsic parameters of the color camera. These parameters include the camera's resolution, focal length, principal point, distortion coefficients, and its position and orientation relative to the robot or world frame. This information is crucial for accurate image processing tasks, including 3D reconstruction, object detection, and camera calibration.

In order to localize the object a pnp algorithm is used, that estimate the object pose given a set of object points. The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame [35].
The computed position of the objects with respect to the camera frame (center of the object with rotation) are published over another topic called /tf, so a frame is attached to their central point.

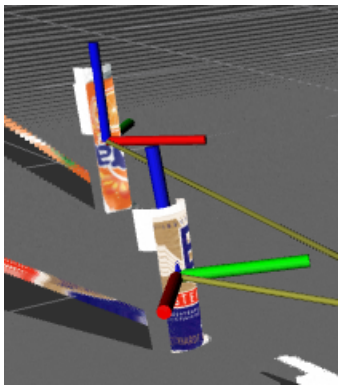

Figure 4.10: Object Localization

4.2.2 Object Localization

For the sorting purpose, it is necessary to get the position and orientation of the objects with respect to the base link frame. In order to do this, a subscriber was created for the topic /info to obtain the information stored in the Detection Info message, which are:

- Object ID
- Width in pixel
- Height in pixel
- File Path (where the images used for training are stored)
- Inliers
- Outliers
- Homography Matrix

This information is used to create a list of dictionaries, where each dictionary contains information about a recognized object. Each dictionary stores key-value pairs representing object attributes such as ID, width, height, file path, inliers, outliers and homography matrix.
From the file path, a function is implemented that is able to retrieve the name of the object as the last world in the path string after removing the (.png) extension. This will be useful in the integration part, as the object name will be used as an object ID in the construction of the planning scene.
To determine the object's position and orientation, the tf2 package is used to perform a transformation of the object's frame (whose name is always "object_" plus "ID of the object") with respect to the robot's base link. In this way, it is possible to send this information to the MoveIt planning scene in order to dynamically update the scene each time (further details on how this information is sent will be covered in section 4.4).

4.3 Object Dimensions from Point Cloud

In the third phase of the project, the concept of Segmentation was introduced to get the geometric dimensions of the objects needed to construct the MoveIt Planning Scene. For this purpose, the Point Cloud Library (PCL) of ROS2 was used. This section will explain what the PCL is, what kind of modules were used for the purpose of the thesis and how the dimensions of the objects were obtained.

4.3.1 Point Cloud Library

The Point Cloud Library (PCL) is a collection of state-of-the-art algorithms and tools to process 3D data. The 3D data are got through camera depth sensors that are usually cheap. In the ROS2 context, the information about these points is published in a topic (PointCoud2) through a message called sensor_msgs/PointCloud2.msg. The PCL core is structured into smaller libraries offering algorithms and tools for specific areas of 3D processing, which can be combined to efficiently solve common problems such as object recognition, point clouds registration, segmentation, and surface reconstruction, without the need of re-implementing all the parts of a system that are needed to solve these sub-tasks [36]. The aim of the use of this library is the necessity of extract some primitive objects (cylinders and box) with a segmentation to get the geometric dimensions. For this purpose, the following modules of PCL were used [37]:

- **Filters** refers to the elimination of the noise from a Point Cloud. In particular, it is based on the computation of the distribution of point to neighbor distances in the input dataset. For each point, the mean distance from it to all its neighbors is computed. By assuming that the resulting distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset.
- **Features** contains data structures and mechanisms for 3D feature estimation from point cloud data. 3D features are representations at certain 3D points, or positions, in space, which describe geometrical patterns based on the information available around the point. The data space selected around the query point is usually referred to as the k-neighborhood.
- **Kd-tree (k-dimensional tree)** is a space-partitioning data structure that stores a set of k-dimensional points in a tree structure, allowing for efficient range searches and nearest neighbor searches. Nearest neighbor searches are a core operation when working with point cloud data and can be used to find correspondences between groups of points or feature descriptors, or to define the local neighborhood around a point or points.

- **Segmentation** contains algorithms for segmenting a point cloud into distinct clusters. These algorithms are best suited for processing a point cloud that is composed of a number of spatially isolated regions.
- **Sample Consensus** holds Sample Consensus (SAC) methods like RANSAC and models like planes and cylinders. These can be combined freely in order to detect specific models and their parameters in point clouds.
- **Common** contains the common data structures and methods used by the majority of PCL libraries. The core data structures include the Point Cloud class and a multitude of point types that are used to represent points, surface normal, RGB color values, feature descriptors, etc. It also contains numerous functions for computing distances/norms, means and covariance, angular conversions, geometric transformations, and more.

4.3.2 How to get the Object Dimensions

In this phase, the ROS2 node that allows to obtain the object dimensions is just a subscriber that receives the Point Cloud information from the PointCloud2 topic (later in the integration part, it will also become a server). Once the Point Cloud data is obtained, the functions based on PCL are used to achieve the purpose of this work part. In particular:

PassThroughFilter : this function belongs to the Filters modules and performs a simple filtering along a specified dimension (z axis in this case) setting an acceptable interval of values (if the point are outside this interval, they are cut off).

ComputeNormals : it is part of Feature modul and it consists on estimating the normal of a plane tangent to the surface passing from a point in the cloud. This surface needs to be estimated from the surrounding point neighborhood of the point (also called k-neighborhood). For this specific application 50 neighbors are considered and, in order to find them faster, a Kd-tree is created.

RemovePlanSurface : this is a segmentation function to remove the plan surface from the point cloud using the RANSAC algorithm. The function applies also the ExtractIndices filter to isolate the points corresponding to the identified plane based on the indices provided by the segmentation process. As a segmentation function, it is necessary to set two key parameters: the distance threshold, which defines the maximum allowable distance between a point and the estimated plane for the point to be considered part of the surface, and the maximum iterations, which limits how many times the RANSAC algorithm will try to find the optimal model. Once the plane is identified,

both the planar surface and the remaining points (obtained by applying the ExtractIndices filter in negative mode) are saved into two separate .pcd files, enabling verification of the segmentation process.

CreateCluster : it is used to divide an unorganized point cloud model P into smaller parts (the parts represents the different objects). In this case, an Euclidian Cluster is adopted, that consists in a 3D grid subdivision of the space, which involves a volumetric representation of the occupied space. A Kd-tree structure is employed for efficient nearest neighbor searches, and a threshold is set for the maximum distance between points that belong to the same cluster. Furthermore, it is important to set the maximum and the minimum cluster size, to ensure that only relevant objects are considered, allowing for more efficient processing and analysis tailored to the specific application requirements. After extracting the clusters, the function iterates through the identified clusters to create individual point clouds for each one, setting their dimensions and density accordingly. Finally, it returns a vector containing these clustered point clouds, enabling further analysis or processing of the segmented data.

ExtractCylinder : this is another segmentation function to extract the cylinder from the point cloud, again using a RANSAC algorithm along with point normals. It configures a segmentation object to detect cylinders, setting key parameters such as the distance threshold, which defines the maximum distance for a point to be considered an inlier, the radius limits, which constrain the expected cylinder size, and the maximum iteration, which, as before, determines how many times the RANSAC algorithm will attempt to find the best-fitting cylinder model. Additionally, the normal distance weight influences how closely the point normals must align with the cylinder's axis to be considered inliers. The function processes the input point cloud and its normals to identify inliers belonging to the cylinder, capturing the geometric properties in coefficients_cylinder, including the radius and the direction vector along the cylinder's z-axis.

ExtractBox : since in PCL the segmentation function related to the box does not exist, it is adopted a method based on the segmentation of the planes. In particular, three orthogonal planes are iteratively searched from the point cloud. For each iteration, it configures the segmentation parameters such as normal distance weight, maximum iterations, and distance threshold. If a planar model is successfully estimated, it extracts the inliers corresponding to the identified plane and updates the filtered cloud by removing these inliers. Once the three planes are found, the function uses the pcl::MomentOfInertiaEstimation class (which belongs to the Features module) to compute the Axis-Aligned

Bounding Box (AABB) around the entire point cloud. It calculates the dimensions of the cube — length, width, and height —by determining the differences between the maximum and minimum points along each axis. It also calculates the geometric center of the box by taking the average of the maximum and minimum points on each axis.

The functions previously described are applied sequentially in the Callback function of the subscriber, starting from the PassThroughFilter to CreateCluster. For each cluster, normals are recalculated, and an attempt is made to extract cylindrical shapes first. If successful, the cylinder parameters are logged and stored. If cylinder extraction fails, the system tries to extract a box. If neither shape is found, it outputs a corresponding message. This structured approach enables effective identification and extraction of cylindrical and cubic objects from the point cloud data.
However, the aim of this implementation part is to get the dimensions of the objects present in the planning scene. Regarding the cylinders, the two needed parameters are the radius and the height. The former is obtained from the cylinder coefficients found through the segmentation, while the computation of the height needs the use of another function that belongs to the Common module.
This function is called "getMinMax3D" and, as the name suggests, it is able to compute the minimum and the maximum point of a 3D point cloud. This allows for determining the center of cylindrical objects by calculating the mean value between these two extreme points. To accurately compute the height along the principal axis of the cylinder, the directional vector of the axis is used. Then, the maximum and the minimum point are projected along this axis through the dot product. After this, the height is computed as the difference between the projections of the maximum and minimum points along the axis.
Regarding the box, the geometric dimensions are found in ExtractBox function through the AABB, as described before.
For both cylinders and boxes, the center coordinates are later utilized in the integration phase to merge the Segmentation algorithm (based on PCL) with the Localization algorithm (using Find Object 2D) in the planning scene. In order to do this, a transformation function is applied from the camera frame to the base link frame to obtain the coordinates with respect to the base link of the robot, as in the previous section.

4.4 Integration

To implement the sorting application, the three parts described above was joined together through a ROS2 architecture. The communication mechanism is composed by a unique client and three servers for three different services: the first server able to localize the objects thanks to Find Object 2D package, the second able to compute the dimensions of the objects through the PCL library and the third to execute pick and place tasks basing on MTC.

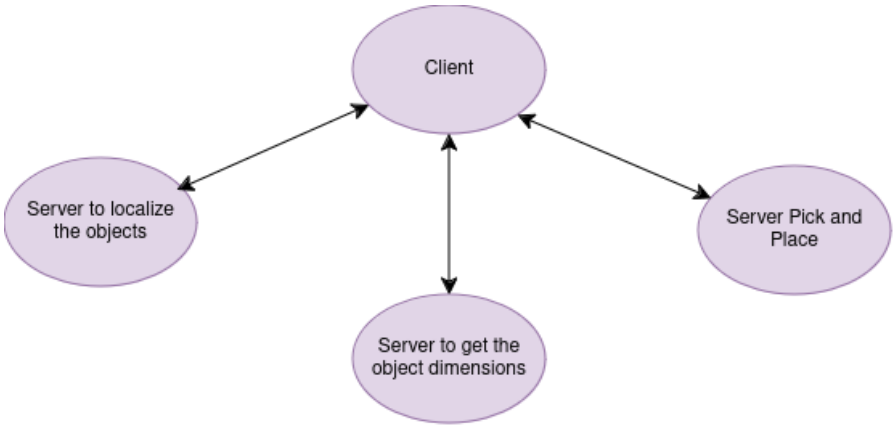

Figure 4.11: ROS2 architecture

The client is the core of the communication. Firstly, it sends two requests simultaneously: one to the localization sever and the other to the dimensions computation server. The first server responds with the position and orientation of the object contained in a geometry message, and with the name of the object, while the second responds with the dimensions of the object, the geometric type (cylinder or box) and the center position computed from the Point Cloud. The client performs a comparison between the position computed by the first server and the one computed by the second server. The aim of this comparison is to find the correspondence about the two different algorithms, to combine the position and orientation of the object found by the Find Object 2D with the geometric dimensions found by PCL of the same object. In this way, it is possible to define the third service with a Pick and Place request joined the data found before (position, orientation, name, geometric type and dimension of the object). The server Pick and Place receives these data from the client and builds the planning scene through this information. Once the planning scene is completed, the task of Pick and Place is built, planned and executed using the MTC described in the section 4.1. Therefore, a dynamic Pick and Place application is built since the needed information is computed each time the scene is updated because the client sends a new request of object detection

to the two servers described before.
Below is a summary diagram of the logical flow of information for the complete implementation, from the acquisition of the camera to the pick and place application.

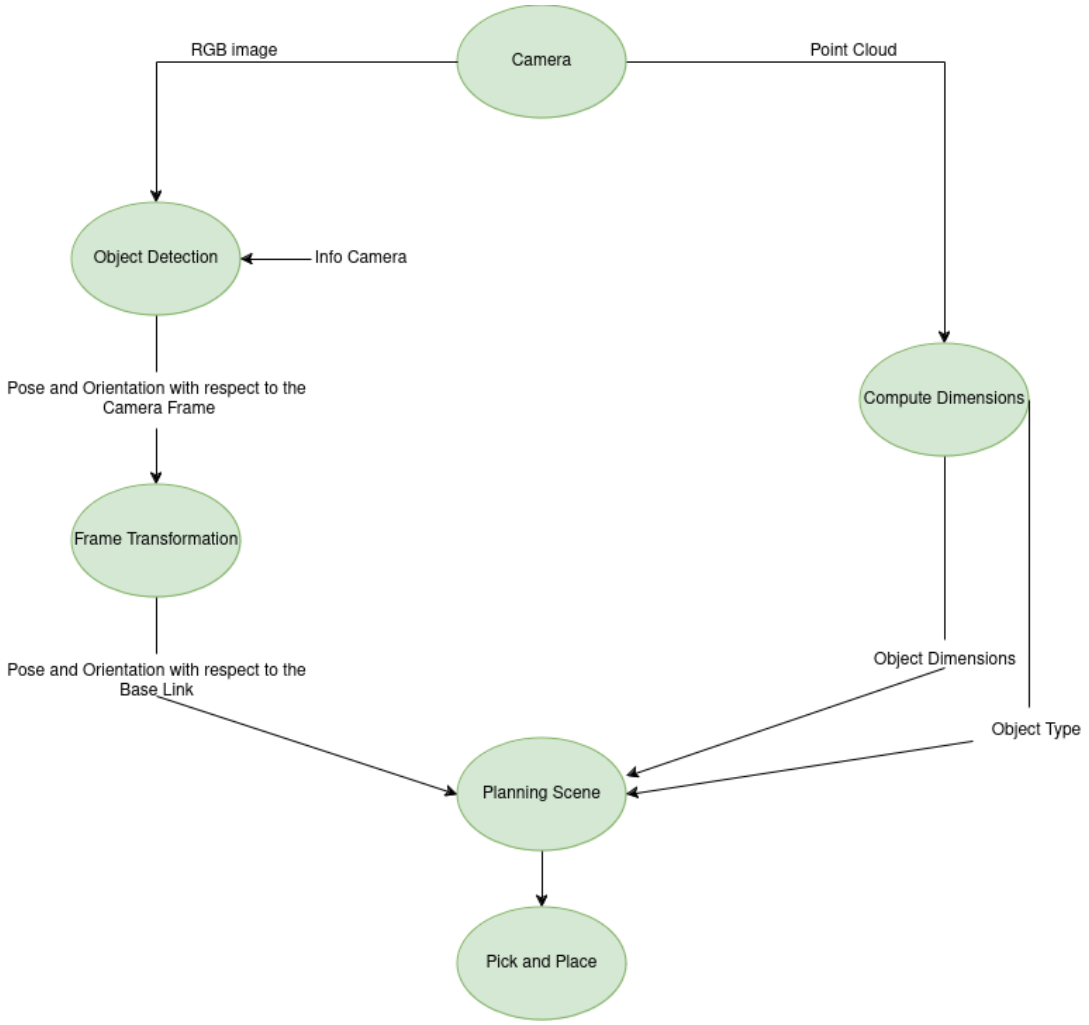

Figure 4.12: Summary Scheme

Finally, in order to obtain the sorting, different policies based on the size of the objects, the shape, the name and the type were defined in the .txt file. The decision about the kind of sorting policy to use is taken by the client and sent to the Pick and Place server in which it is implemented the sorting logic. In particular, on the basis of the policy kind decided by the client a different .txt file is opened and read. In each file, it is defined the position of place for each object kind. For example, supposed that the client requests a sorting policy based on the cylinder size and that in the file .txt are defined three kinds of objects: the ones with an

height equal to 10cm, 20cm and 30cm and for each of them are defined all the Pick and Place parameters, in particular the place position. Now, when the application will be started, the cylinders will be sort on the basis of their height, so the ones with 10cm of height will be put in a certain position of place that will be different with respect to the place position of the cylinder with 20cm of height. The same concept is applied to the other sorting policy that will be explained in detail in the following chapter to evaluate the performance of this application.

Chapter 5

Experiments and Results

During the development of the sorting system, several experiments were conducted to make choices aligned with the research objectives. Firstly, various motion planners available in MoveIt were tested to identify the most suitable one for the application. This was followed by an evaluation of different feature matching algorithms within the Find Object 2D package, under various environmental conditions, comparing multiple parameters such as the matching rate and the processing time. Finally, the entire system was tested to assess its overall performance by implementing different sorting policies. These experiments were conducted within the Gazebo simulation environment, with RViz used for real-time visualization of the robot's actions.
In the end, to further assess the system's scalability and real-world applicability, the Ufactory Lite6 manipulator was also tested using real hardware instead of simulation.
The experiments were performed using a Lenovo ThinkPad E14 Gen 2, which features an AMD® Ryzen 7 4700U processor with Radeon graphics, 16 GiB of RAM, and 512 GB of disk capacity. Since the system was developed in a ROS2 environment, a disk partition was created to install the Ubuntu 22.04.4 LTS (64-bit) operating system. The entire project was developed within a Docker container using a ROS Humble-based image, ensuring portability across different systems.

5.1 Planner Choice

Choosing the right motion planner is critical to the performance of robotic systems, especially in applications such as object sorting where efficiency and throughput are essential. The planning algorithm must achieve a balance between speed and accuracy, as the ability to quickly generate feasible paths has a direct impact on the overall performance of the system. Based on this, a hypothesis has been

developed: since the primary goal of the sorting system is to process a large number of objects in a limited amount of time, it was concluded that prioritizing processing speed would be more beneficial than aiming for absolute path accuracy. For this reason, optimisation-based planners such as CHOMP were excluded a priori, as they are designed to produce very smooth and accurate paths, but at the cost of significantly longer computation times. Instead, planners from the Open Motion Planning Library (OMPL) were considered, which use sampling-based algorithms that prioritize speed, making them a more suitable choice for the sorting application.

As it was explained in section 2.3.1, the sampling-based algorithms are distinguished into single query and multi query. In single query planners, the focus is on the speed of computing a single path. Each new path requires a new computation, as no information about previously planned paths is retained, requiring re-initialization. In contrast, multi query planners are advantageous when there are many motion planning problems to solve in the same environment, as they store information about each path to speed up subsequent computations. This could potentially benefit the sorting application. However, the planning function of the MTC framework only processes one task at a time, rather than a vector of tasks. In addition, testing was limited to a maximum of three objects within the same scene. As the application is event-based, once the three objects have been sorted, new objects must be introduced. By this time, any previously stored path information may have been cleared from memory, reducing the benefit of using multi query planners in this specific context. Consequently, these planners were excluded from consideration and the focus was shifted to the evaluation of single query sampling-based algorithms.

To assess the performance of various planners, ten experiments were conducted for each algorithm. Among the planners tested, Rapidly-exploring Random Tree (RRT) and RRT-Connect were selected for a detailed comparison. These two planners were chosen due to their distinctive approaches to path-finding and their relevance to the specific needs of the sorting application. RRT is known for its simplicity and effectiveness in exploring large, high-dimensional spaces. It builds a single tree from the initial configuration and incrementally expands towards the goal, making it suitable for scenarios where exploration is more critical than optimality. This characteristic makes RRT a good candidate for initial testing to understand its basic performance in this task. RRTConnect, on the other hand, enhances RRT by employing a bidirectional approach. It constructs two trees — one from the start configuration and one from the goal configuration —and attempts to connect them. This method often results in faster path-finding by reducing the search space and convergence time, which is particularly beneficial for applications requiring quick planning, such as the sorting task in this case.

The comparison was based on planning time, execution time, and the reliability of each algorithm. The table shows the average planning and execution times for each object. For each experiment, three objects were placed in front of the robot, and the planning and execution times were measured for each object. The times for these three objects were then averaged to obtain a single value per experiment. Finally, the results from all ten experiments were averaged to produce the values presented in the table.

Algorithm	Avarage Planning time (seconds)	Avarage Execution time (seconds)	Failure Rate
RRT	3.714	19.387	10%
RRTConnect	2.813	17.766	0%

Table 5.1: Comparison between RRT and RRTConnect in terms of avarage planning time, avarage execution time and failure rate

As shown in the table 5.1, RRTConnect consistently performed better, with shorter planning and execution times compared to RRT, and had a 0% failure rate, while RRT failed 10% of the time. These considerations led to the selection of the RRTConnect planner for this project.

5.2 Feature Matching Algorithm Selection

As it was already said, the ROS package "Find Object 2D" is based on OpenCv implementation of various feature detectors and descriptors algorithms.
Among the available algorithms, four were selected for evaluation: SIFT, ORB, KAZE, and AKAZE. These algorithms were chosen because they provide both detection and description capabilities. The evaluation of the feature detection and description algorithms focused on several key parameters. Firstly, the Number of key points detected by each algorithm was evaluated. This parameter indicates how many distinct points the algorithm identifies as features in the image. A higher number of key points typically suggests that the algorithm can detect more features, which can be crucial for accurate object recognition. Additionally, the Number of Key points specifically detected on the object itself was considered, as this reflects the algorithm's ability to focus on relevant features that belong to the object rather than the background or irrelevant parts of the scene. Next, the Number of Matches was examined. This measures how many feature matches were found between images. The ability to identify and match features across different views of the same object is essential for ensuring robust recognition, particularly in varied conditions. The Matching Rate was also considered, which is the ratio of correctly matched key points to the total number of key points detected. This

metric helps to evaluate the accuracy of the feature matching process, reflecting how well the algorithm performs in correlating features between images. Lastly, the Execution Time for each algorithm was measured. This parameter represents the time required to complete the feature detection and description process. Efficient algorithms that have shorter execution times are preferable in real-time applications, where processing speed is critical.
The evaluation involved testing the four algorithms under various conditions to assess their robustness. The same object was recognized in different scenarios, including:

- Baseline Condition: Same position and scale as during training, providing a reference for optimal performance.
- Rotation: The object was rotated by 180 degrees to assess performance with significant changes in object orientation.
- Scale Variation: Testing with a resized object to assess handling of scale changes.
- Illumination Reduction: Assessing performance in environments with reduced lighting, specifically at 25% and 50% less illumination compared to the baseline.

The different parameters described above are evaluated in these different scenario to identify the most suitable solution for this kind of application. The results are shown in the tables below.

	total key points	**object key points**	**n. of matches**	**matching rate**	**execution time (ms)**
SIFT	640	80	78	97.5%	300
ORB	500	28	23	82.1%	27
KAZE	750	64	63	98.4%	1000
AKAZE	680	41	41	100%	160

Table 5.2: Performance of the feature matching algorithms in the baseline condition.

	total key points	**object key points**	**n. of matches**	**matching rate**	**execution time (ms)**
SIFT	700	29	26	89.6%	313
ORB	500	too low	too low		30
KAZE	854	23	21	91.3%	1090
AKAZE	774	too low	too low		180

Table 5.3: Performance of the feature matching algorithms when the object is rotated by 180 degrees.

	total key points	object key points	n. of matches	matching rate	execution time (ms)
SIFT	638	20	17	85%	300
ORB	500	16	8	50%	25
KAZE	750	15	13	86.7%	1000
AKAZE	680	too low	too low		150

Table 5.4: Performance of the feature matching algorithms with a scale variation of the object.

	total key points	object key points	n. of matches	matching rate	execution time (ms)
SIFT	640	63	60	95.2%	319
ORB	500	43	35	81.4%	28
KAZE	750	49	47	95.9%	1000
AKAZE	680	20	15	75%	173

Table 5.5: Performance of the feature matching algorithms with illumination reduction of 25%.

	total key points	object key points	n. of matches	matching rate	execution time (ms)
SIFT	503	42	10	23.8%	336
ORB	500	too low	too low		25
KAZE	360	36	13	36.1%	1095
AKAZE	306	too low	too low		186

Table 5.6: Performance of the feature matching algorithms with illumination reduction of 50%.

The results show significant differences in performance between the algorithms. In general, SIFT demonstrates robust performance under all conditions. It consistently detects a high number of key points, resulting in reliable matching rates, even when the object is resized, rotated, or the lighting conditions changes. Despite slight drops in performance under extreme lighting changes, SIFT maintains a good balance between accuracy and execution time. In contrast, ORB has faster execution times but struggles in more challenging conditions. While it performs well in scenarios without significant variation, its lower accuracy in handling scale changes and rotations, combined with a tendency to produce fewer matches, limits its effectiveness in complex environments. KAZE, on the other hand, excels in key point detection and matching rates, often outperforming other algorithms in terms of accuracy. However, its relatively long execution times makes it less suitable for real-time applications where speed is a priority, despite its superior performance in recognizing objects under different conditions. Finally, AKAZE provides a

compromise between KAZE's accuracy and ORB's speed. While faster than KAZE, it struggles in certain conditions, particularly in scenarios involving scale variations, where it failed to find enough matches for reliable object recognition.
In conclusion, SIFT emerges as the most suitable choice for this application, offering a balance between accuracy and speed. While its performance is slightly lower than that of KAZE, it significantly outperforms KAZE in terms of execution time. Its reliable performance across a wide range of conditions makes it well-suited for real-time object recognition tasks, where both precision and efficiency are important.

5.3 Sorting Evaluation

In order to evaluate the performance of the implemented system, four different sorting policies were defined based on criteria such as object size, shape, name and type. These policies are implemented via configuration files in .txt format, which specify the sorting logic that will be used by the Pick and Place server. The client selects the required policy via a terminal interface, which is then sent to the Pick and Place server. It opens and reads the corresponding .txt file, which is structured into sections dedicated to different objects or criteria. Each section specifies some parameters for the handling of each object, including crucial details such as the object's placement position. The file format is simple:

```
key1: param_key1=value1,param_key2=value2,...
key2: param_key1=value1,param_key2=value2,...
```

In this format:

- `key` denotes a sorting criterion (e.g., object type or height range).
- `param_key` and `value` define parameters such as placement coordinates.

Based on the selected sorting policy, a key is defined that is specific to that policy - whether it's based on the object's name, size, type or other attributes. The server then checks for a match between the selected key and the keys defined in the file. While the method of determining the key is different for each policy, the process of checking the configuration file for a matching section is the same for all of them. Once the appropriate section is found, the system retrieves the relevant handling instructions and applies them to complete the sorting task.

All the servers required for the sorting system are initiated simultaneously from a single launch file, which loads the necessary parameters for the robot simulation and automatically opens RViz, Gazebo, and the Find Object 2D GUI. Before launching

the client and starting the application, the robot must be positioned in RViz in the predefined camera pose specified in the SRDF, with the arm lowered to ensure visibility of the objects placed in front of it. Once the robot is correctly positioned, the client is launched in a separate terminal, allowing the selection of the sorting policy and consequently the execution of the system.

To evaluate the realized application, ten experiments were carried out for each implemented policy on the UF850 manipulator, focusing on different key metrics: the accuracy of object localization, the precision of dimension computation, the reliability of the Pick and Place task, and the total time taken from client request to the completion of the task. The tests involved sorting three objects per scenario to assess the overall system performance. The following sections describe each sorting policy in detail, along with the results and analysis of the corresponding experiments.

5.3.1 Sorting Policy based on object's name

The first tested policy is based on the object's name. In this case, the primary criterion for sorting is the name of the object, which is extracted by the localization server from the object's file path. This name is then used as a key to access the relevant sorting instructions in a configuration file. The system checks the file for a matching section dedicated to that object name. Once the correct section is found, the object is sorted and managed according to the specific parameters outlined in the configuration, ensuring that each object is handled and positioned efficiently based on predefined rules.
For this purpose, a Gazebo simulation environment was constructed with three objects positioned in front of the robot. These objects were specifically selected to apply the implemented logic, and before the test, a training phase was conducted for the feature-matching algorithm used. This training process involved capturing images of the objects to allow the Find Object 2D ROS package to recognize and localize them based on their visual features, as explained in the previous chapter.

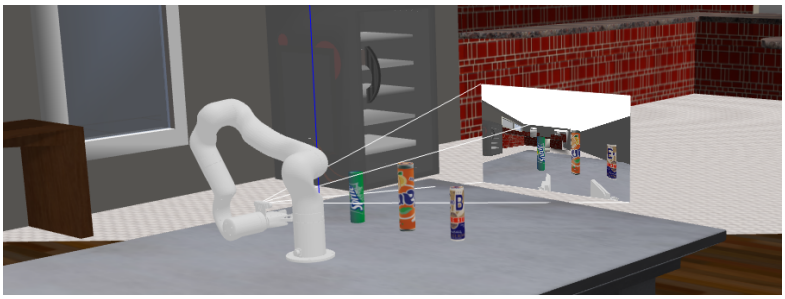

Figure 5.1: Gazebo environment for testing the name-based sorting policy.

From Gazebo, it is easy to determine the real position of the objects relative to the robot's base link. The robot's position in the world coordinates is known: $x = 0.1$, $y = -0.15$, $z = 1.0$, with a rotation of 1.571 radians around the y-axis. Using this information, along with the objects' known positions relative to the world, their real positions with respect to the robot can be calculated through a simple coordinate transformation. In addition, the geometric dimensions of each object are defined in the Gazebo SDF file. For this first set of experiments, the selected objects are cylindrical in shape, each with different textures and names. Specifically, three cans were chosen for these tests: fanta, beer and sprite. These names are present in the configuration file, each associated with different placement positions and instructions. The table below presents the known information for each object, which will be compared with the values derived from the servers.

Object	X position(m)	Y position (m)	Z position(m)	radius	height
fanta	0.65	0.1	0.15	0.035	0.3
beer	0.65	0.4	0.115	0.03	0.23
sprite	0.65	-0.2	0.115	0.032	0.23

Table 5.7: Real positions and geometric dimensions of the objects from Gazebo for the name-based policy.

For each test, the steps are always the same: the application is launched, the object are recognized, localized and segmented, as described in the previous sections. Based on the received information, the MoveIt Planning Scene is constructed, and the sorting task is executed.

Figure 5.2: Object Recognition using the name-based policy.

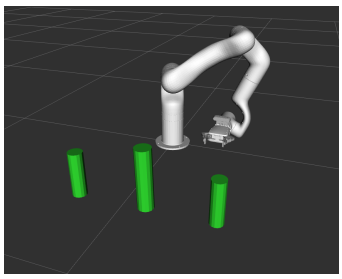

Figure 5.3: Building of the Planning Scene using the name-based policy.

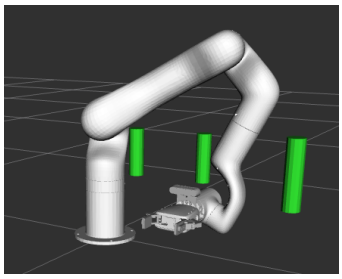

Figure 5.4: Objects after sorting using the name-based policy.

These experiments collected data on the object positions computed by both the Localization server and the Dimension Computation server (based on PCL). In addition, the geometric dimensions of the objects derived from the PCL segmentation

were recorded and the total time taken by the sorting application to process and sort three objects was measured. The tables below summarizes the average results obtained, including the relative errors compared to the exact values.

Object	**Type of measure**	**Exact Value (m)**	**Localization Server computation (m)**	**Localization Server (Error %)**	**PCL Computation (m)**	**PCL (Error %)**
Fanta	Position (x)	0.65	0.615	5.3	0.633	2.55
	Position (y)	0.1	0.104	4.17	0.105	5.58
	Position (z)	0.15	0.153	2.35	0.166	10.6
Beer	Position (x)	0.65	0.626	3.67	0.633	2.58
	Position (y)	0.4	0.398	0.49	0.398	0.45
	Position (z)	0.115	0.124	7.94	0.130	13.05
Sprite	Position (x)	0.65	0.619	4.76	0.640	1.58
	Position (y)	-0.2	-0.191	4.51	-0.194	2.82
	Position (z)	0.115	0.116	0.84	0.131	14.03

Table 5.8: Position measurements and errors for objects using the name-based policy.

Object	**Dimension Type**	**Exact Value**	**PCL (Measured)**	**Error %**
Fanta	Radius	0.035	0.0349	0.33
	height	0.3	0.2683	10.56
Beer	Radius	0.03	0.0299	0.33
	height	0.23	0.199	13.49
Sprite	Radius	0.032	0.0319	0.31
	height	0.23	0.198	13.96

Table 5.9: Dimension measurements and errors for objects using the name-based policy.

The position measurements show that both the Localization Server and the Point Cloud Library (PCL) provide accurate estimates overall. The errors across all axes remain within a reasonable range, with most deviations being below 5%. However, there are slightly larger errors in the z-axis, especially for PCL, which occasionally exceeded 10%. Despite this, both methods generally maintain accuracy, with PCL offering better performance in some cases, particularly in the x-axis. For geometric dimensions, the PCL system demonstrates high accuracy in radius calculations, with errors remaining below 1%. In contrast, the height measurements show larger errors, exceeding 10%. The reason for the increased errors in height is attributed to the RemovePlaneSurface function, which removes the plane under the objects before

applying cylinder segmentation to extract the shapes. This function inadvertently truncates approximately 3cm from the bottom, affecting both the z-component and the height measurements. Specifically, the height measurement is approximately 3 cm shorter than the true value, meaning the minimum detected point is elevated by 3 cm compared to the true minimum, while the z-component has an average deviation of 1.5cm. However, this truncation is a necessary compromise, since removing the plane improves the accuracy of the cylinder segmentation by preventing the algorithm from misidentifying part of the plane as part of the object. A lower threshold may reduce truncation, but may leave parts of the plane, especially in noisy or uneven data. In such scenarios, the threshold compensates for noise and data artefacts, ensuring complete removal of the plane but compromising object height. In this case, the objects used had an average height of over 20 cm, so the 3 cm truncation introduced by the removePlaneSurface function did not significantly affect the performance of the application. While the height was reduced, it remained within an acceptable range for the sorting task, and the system continued to recognize and manipulate the objects accurately. However, this parameter can be easily adjusted if higher precision is required for different applications or smaller objects. The flexibility of the implementation allows for fine-tuning based on specific needs without significant changes to the system.

During the experiments, the system encountered three cases out of ten where the matching process failed on the first attempt. Specifically, Find Object 2D was unable to identify all three objects, typically recognizing only two. In such cases, the client automatically sent a new request to both algorithms, as the experimental setup required a 100% match between the algorithms for all three objects. In the general application, the system is less restrictive, allowing a match of at least two out of three objects to proceed. However, the client still sends a new request to the algorithms when the task is complete or if the matching process fails, ensuring continuous operation.
For each experiment conducted, the total application time was measured. The average result for this set of tests was 80.93 seconds.

5.3.2 Sorting Policy based on Object's Shape

The second sorting policy tested is based on the shape of the object, which can be either a cylinder or a box. This shape information is provided by the PCL server. Similar to the name-based policy, the shape is used as a key to retrieve the corresponding sorting instructions from a configuration file. In particular, there are two possible sorting paths: if the object is identified as a cylinder, it will follow one set of sorting rules and be placed in a particular location, whereas if the object is a box, it will follow a different set of rules and be placed to a different location

accordingly.

Also in this case, a new Gazebo environment was built with three objects positioned in front of the robot, training the feature-matching algorithm. For simplicity, the new set of objects was positioned in the same location as in the previous experiments. The selected objects included a box, a Fanta can, and a wine bottle. It was hypothesized that the wine bottle could be approximated as a cylinder for sorting purposes. This approximation was validated during the experiments, confirming that treating the bottle as a cylinder was appropriate for the shape-based sorting policy.

Figure 5.5: Gazebo environment for testing the shape-based sorting policy.

The table below presents all the known information for each object, which will be compared with the values derived from the servers. Regarding the bottle, the radius considered refers to the larger radius at the bottom of the bottle, while the height corresponds to the total height of the bottle.

Object	X position(m)	Y position (m)	Z position(m)	radius	height	length	width
box	0.65	0.1	0.075		0.15	0.07	0.07
bottle	0.65	0.4	0.115	0.034	0.23		
Fanta	0.65	-0.2	0.115	0.03	0.23		

Table 5.10: Real position and geometric dimensions of the object from Gazebo using the shape-based policy.

Below, the representation of the most important steps followed by the application, from object recognition to placement.

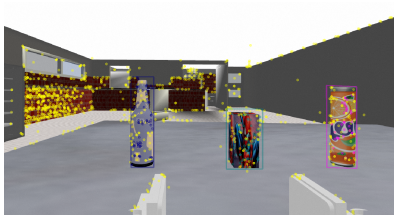

Figure 5.6: Object Recognition using the shape-based policy.

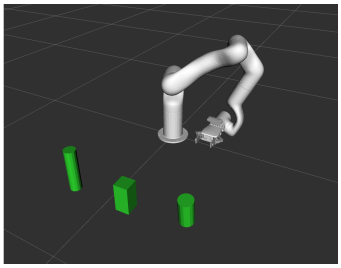

Figure 5.7: Building of the Planning Scene using the shape-based policy.

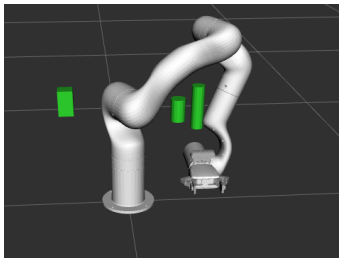

Figure 5.8: Objects after sorting using the shape-based policy.

To test the efficiency of the algorithms, the same parameters of the previous set are considered. The results are shown in the following tables.

Object	**Type of measure**	**Exact Value (m)**	**Localization Server computation (m)**	**Localization Server (Error %)**	**PCL Computation (m)**	**PCL (Error %)**
Box	Position (x)	0.65	0.614	5.5	0.642	1.26
	Position (y)	0.1	0.086	14.01	0.106	5.92
	Position (z)	0.075	0.088	17.39	0.091	21.8
Bottle	Position (x)	0.65	0.616	5.2	0.632	2.72
	Position (z)	0.115	0.135	17.68	0.083	28.1
Fanta	Position (x)	0.65	0.622	4.37	0.641	1.34
	Position (y)	-0.2	-0.194	3.06	-0.195	2.31
	Position (z)	0.115	0.117	1.50	0.131	14.15

Table 5.11: Position measurements and errors for objects using the shape-based policy.

Object	**Dimension Type**	**Exact Value**	**PCL (Measured)**	**Error %**
Box	length	0.07	0.0705	0.76
	width	0.07	0.0701	0.12
	height	0.15	0.11797	21.3
Bottle	Radius	0.034	0.0339	0.21
	height	0.23	0.102	55.57
Fanta	Radius	0.03	0.0299	0.23
	height	0.23	0.198	14

Table 5.12: Dimension measurements and errors for objects using the shape-based policy.

The results show that the system generally performs well in estimating both the positions and dimensions of objects, although challenges arise with irregularly shaped objects. PCL-based computations tend to be more accurate, with smaller errors compared to the Localization Server, particularly for the x and y axis measurements. However, discrepancies in the z-axis measurements are notable, particularly for the bottle, due to its irregular shape.

In terms of object dimensions, most measurements were within acceptable error limits, particularly for simpler shapes such as the box and the Fanta can, where errors were typically less than 1%. The height of the bottle presented the largest deviation because the segmentation only captured part of the bottle, resulting in an estimated height of 0.102 m instead of the actual 0.23 m, with an error of 55.57%. As before, the height measurement of the other two objects deviates of 3cm for the same reason explained in the previous section.

The segmentation of boxes sometimes presented problems, particularly in complex environments. This performance discrepancy primarily arises from the lack of a dedicated box segmentation function in the Point Cloud Library (PCL), which hinders the detection of three orthogonal planes. Approximately 20% of the time, the sensitivity of the cylinder segmentation algorithm can lead to the mis-classification of cubes as cylinders. Even when the algorithm correctly identifies that an object is not a cylinder, it can still struggle to effectively segment the three planes, which occurs in a further 20%. This limitation may result in only 2 out of 3 objects being successfully sorted, as the cube is not segmented properly. These challenges highlight the need for more robust segmentation methods to ensure accurate detection and sorting of all object types.

In the 10% of cases, bottles are incorrectly positioned, which, due to their smaller size, can lead to errors in the pick and place planning. However, the system is robust and capable of sending a new request to ensure successful task completion, allowing it to recover from such challenges and maintain reliable sorting.

As the previous set of tests, for each experiment conducted, the total application time was measured. The average result was 86.03 seconds.

5.3.3 Sorting Policy based on Object's Type

The third sorting policy tested is based on the type of object. Specifically, the system distinguishes between bottles and cans to determine the appropriate sorting location. If the object is identified as a bottle, it is placed in one location, while a can is directed to another. In this method, the key for sorting is determined by

extracting the object type from its name. The system identifies the last underscore (_) in the object's name, and everything before that is considered the base type (e.g., "bottle" or "can"). This extracted name is then used as the sorting key, allowing the system to apply specific rules based on the object type.

A new Gazebo environment was built with three objects positioned in front of the robot, training the feature-matching algorithm as before. The chosen objects include two cans, specifically a beer and a Fanta, and a bottle of Coca-Cola.

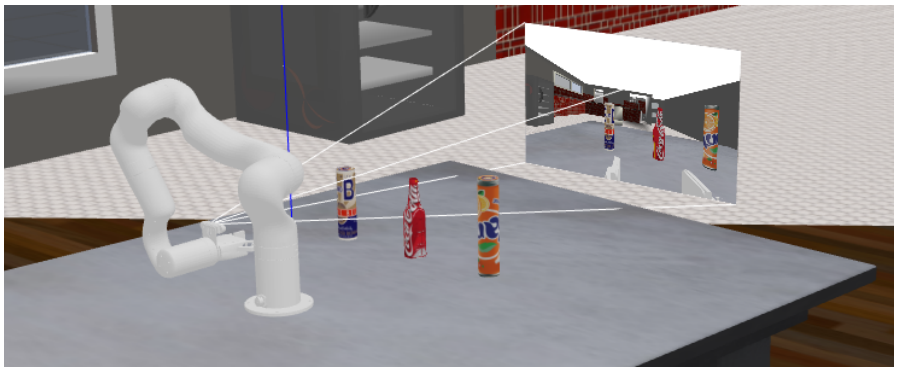

Figure 5.9: Gazebo environment for testing the type-based sorting policy.

The table below presents all the known information for each object, which will be compared with the values derived from the servers.

Object	X position(m)	Y position (m)	Z position(m)	radius	height
bottle_Cola	0.65	0.1	0.115	0.034	0.23
Can_beer	0.65	0.4	0.115	0.03	0.23
Can_fanta	0.65	-0.2	0.15	0.027	0.3

Table 5.13: Real position and geometric dimensions of the object from Gazebo using the type-based policy.

The sorting application is initiated following the process outlined earlier. After the objects are identified and their positions determined, the MoveIt Planning Scene is set up for execution.

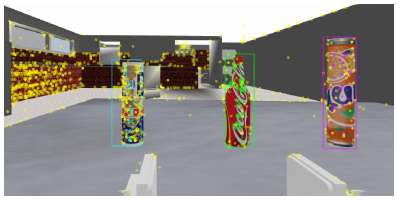

Figure 5.10: Object Recognition using the type-based policy.

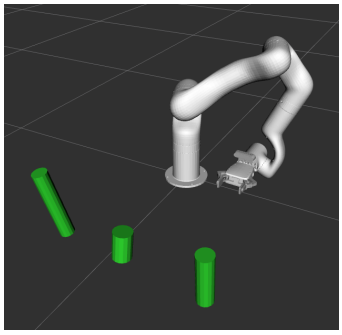

Figure 5.11: Planning Scene using the type-based policy.

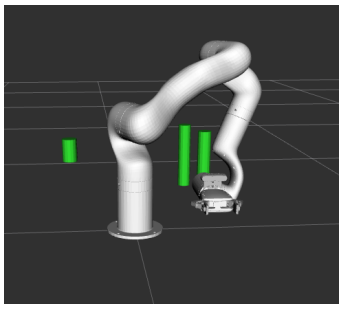

Figure 5.12: Objects after sorting using the type-based policy.

The results of the set of experiments are shown in the tables below.

Object	**Type of measure**	**Exact Value (m)**	**Localization Server computation (m)**	**Localization Server (Error %)**	**PCL Computation (m)**	**PCL (Error %)**
Bottle: cola	Position (x) Position (y) Position (z)	0.65 0.1 0.115	0.622 0.097 0.1146	4.28 3.16 0.32	0.633 0.102 0.083	2.55 2.08 28.18
Can: beer	Position (x) Position (y) Position (z)	0.65 0.4 0.115	0.626 0.3975 0.124	3.67 0.62 7.63	0.633 0.3972 0.130	2.58 0.69 13.51
Can: fanta	Position (x) Position (y) Position (z)	0.65 -0.2 0.15	0.625 -0.186 0.151	3.89 6.76 0.53	0.641 -0.195 0.166	1.37 2.65 10.55

Table 5.14: Position measurements and errors for objects using the type-based policy.

Object	Dimension Type	Exact Value	PCL (Measured)	Error %
bottle_cola	Radius	0.034	0.0339	0.21
	height	0.23	0.103	55.42
Can_beer	Radius	0.03	0.0299	0.33
	height	0.23	0.199	13.54
Can_fanta	Radius	0.027	0.0269	0.17
	height	0.3	0.268	10.54

Table 5.15: Dimension measurements and errors for objects using the type-based policy.

The results show that the system performs generally well in estimating both the positions and dimensions of objects, with most errors being within acceptable margins. The x- and y-position estimates for all objects display relatively low errors, typically below 5%. However, there are some discrepancies in the z-axis measurements, particularly for the Coca Cola bottle, which shows a significant error of 28.18%. This is likely due to partial segmentation of the object, a known issue with non uniform shaped objects.

In terms of dimensions, the system accurately measured the radii, with an error of less than 1%. However, the height estimate for the Coca Cola bottle deviates considerably (55.42%), again likely due to the object's irregular shape, while the other height's measurements show the same trend of the previous experiments. Despite these challenges, the system is robust enough to handle the objects effectively during the sorting process, even when some measurements are less precise.

In 10 experiments, the system encountered three notable failures. In one case, Find Object 2D only recognized one out of the three objects, and in another, it matched only two. These mismatches led to the failure of the initial identification process. In such instances, the system automatically sent a new request to both the localization and PCL algorithms, as the experimental setup required all three objects to be correctly identified for the task to proceed. In addition, an error occurred during the planning phase when a bottle was positioned too skewed, probably due to its irregular dimensions. This misalignment caused the pick and place operation to fail, demonstrating a sensitivity to the positioning of the bottle during the sorting process.
For each experiment conducted, the total application time was measured. The average result for this set of tests was 80.02 seconds.

5.3.4 Sorting Policy based on Object's Height

The final sorting policy is based on the object's height. In this approach, the key for sorting is determined by the height of the object. The system iterates through a configuration file that contains predefined height ranges. For each object, the height value is compared against these ranges to find a match. If the object's height falls within a particular range, the system assigns the corresponding range as the sorting key. This key is then used to retrieve the specific sorting instructions from the configuration file.

In this case, the Gazebo world is composed by three different cans with different heights, so they will be placed in three different positions.

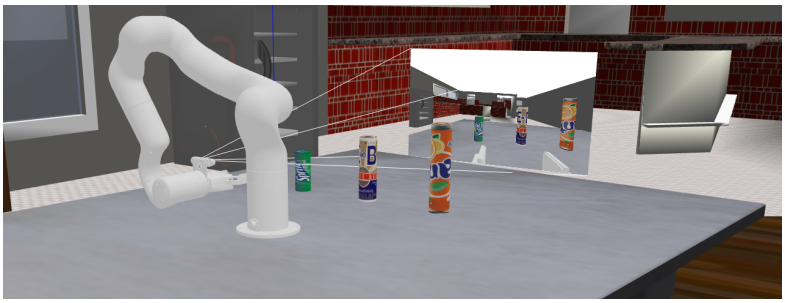

Figure 5.13: Gazebo environment for testing the height-based sorting policy.

The table below presents all the known information for each object, which will be compared with the values derived from the servers.

Object	X position(m)	Y position (m)	Z position(m)	radius	height
beer	0.65	0.1	0.115	0.032	0.23
sprite	0.65	0.4	0.075	0.03	0.15
fanta	0.65	-0.2	0.15	0.035	0.3

Table 5.16: Real position and geometric dimensions of the object from Gazebo using the height-based policy.

As in previous cases, the sorting application is started by detecting the objects and determining their positions. Following this, the MoveIt Planning Scene is prepared for execution.

Figure 5.14: Object Recognition using the height-based policy.

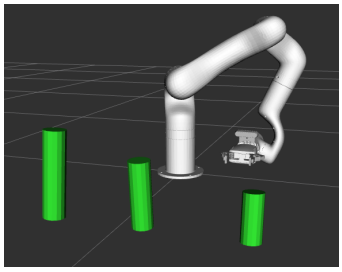

Figure 5.15: Planning Scene using the height-based policy.

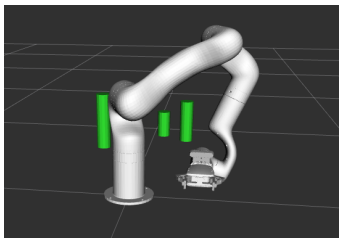

Figure 5.16: Objects after sorting using the height-based policy.

The results of the set of experiments are shown in the tables below.

Object	**Type of measure**	**Exact Value (m)**	**Localization Server computation (m)**	**Localization Server (Error %)**	**PCL Computation (m)**	**PCL (Error %)**
Beer	Position (x)	0.65	0.627	3.46	0.633	2.53
	Position (y)	0.1	0.092	7.63	0.102	1.99
	Position (z)	0.115	0.127	10.33	0.134	16.76
Sprite	Position (x)	0.65	0.621	4.42	0.633	2.58
	Position (y)	0.4	0.399	0.24	0.399	0.31
	Position (z)	0.075	0.066	11.49	0.0755	0.72
Fanta	Position (x)	0.65	0.616	5.2	0.640	1.49
	Position (y)	-0.2	-0.195	2.58	-0.194	2.91
	Position (z)	0.15	0.151	0.83	0.162	8.28

Table 5.17: Position measurements and errors for objects using the height-based policy.

Object	**Dimension Type**	**Exact Value**	**PCL (Measured)**	**Error %**
Beer	Radius	0.032	0.0319	0.33
	height	0.23	0.198	13.68
Sprite	Radius	0.03	0.0299	0.33
	height	0.15	0.119	20.93
Fanta	Radius	0.035	0.0349	0.33
	height	0.3	0.2685	10.49

Table 5.18: Dimension measurements and errors for objects using the height-based policy.

The results show that the system performs well in estimating both the position and dimensions of the objects, with relatively low errors across most measurements. For position estimates, the PCL computation tends to be slightly more accurate than the Localization Server, particularly in the x- and y-axes. Errors remain under 5% for the x and y coordinates of all objects, except for the beer can in the y-direction, which exhibits a 7.63% error in the Localization Server's output. The z-axis measurements show slightly larger errors, especially for the beer can, where the PCL estimate has a 16.76% error and the Localization Server shows a 10.33% error. This discrepancy is likely due to segmentation challenges in capturing the exact vertical position of the object. Regarding dimensions, the PCL measurements for radius are very accurate, with errors below 1% for all objects. Regarding the height, again there is present the same systematic error of the previous cases.

Similar to the previous experiments, in 10 experiments, the system encountered three failures. In two instances, Find Object 2D matched only two out of the three objects, leading to a partial identification failure. However, this is not a problem, as it was explained in the previous sections. Additionally, another failure occurred during the planning phase of the smallest object. Despite being a regular cylinder, its small size caused issues during the pick-and-place operation, leading to a failure. As for the other policy, the total application time was measured, with average result equal to 72.88 seconds.

5.4 Hardware Testing and Scalability Assessment

To further evaluate the scalability and real-world applicability of the system, the Ufactory Lite6 manipulator, a smaller model from the same Ufactory family as the one used in the simulation, was selected for integration. Despite its compact size, the Lite6 was readily available within the company and provided an opportunity to test the adaptability of the system on real hardware. It is characterized by six-degree of freedom and 600g of payload, with a maximum speed of 0.5m/s. It is able to reach 440mm along horizontal axis. Compared to the UF850 model used in the simulation, which is designed for more demanding tasks, the Lite6 offers reduced performance. However, it still allowed for a valuable assessment of the system's flexibility on physical hardware.

Figure 5.17: ULite6 manipulator

As part of UFactory, all the tools for motion planning and control are available in the xArm repository, which is also used for the simulation case. This repository contains the launch files that allow the robot to be controlled by Moveit in Rviz with the real hardware.

5.4.1 Set up and Integration

In order to integrate with the real hardware, several preliminary steps were required. First, the robot had to be connected to the PC via a static IP address to establish communication. Once connected, the system could be started by specifying the robot's IP address and activating the gripper and RealSense camera. When the application was started, it was found that MoveIt was working correctly, but the RealSense camera was not enabled due to missing plugins for the real hardware. To

solve this, the ROS Wrapper for Intel RealSense Cameras was downloaded from the official website. This wrapper integrates the functionality of the RealSense camera into the ROS2 system, allowing depth data, RGB images and point clouds to be published. Once the wrapper was installed, the RealSense camera was successfully enabled and integrated into the system, enabling object detection, segmentation and localization in the scene as intended.

Regarding the setup of the scene, due to the smaller opening range of the Lite6 gripper compared to the UF850, the selected objects had to be much smaller than those used in the simulation. Specifically, two chess pieces were selected: the horse and the queen. These objects are about 5 cm high, while the objects used in the simulation were between 15 cm and 30 cm high. As a result, several segmentation parameters had to be adjusted to accommodate the smaller objects. This included modifying the filter limits in the Pass Through filter and adjusting the radius limit in the Extract Cylinder function, as well as refining the distance threshold for object detection. In addition, clustering parameters needed to be modified, specifically the minimum and maximum cluster size and cluster tolerance settings. All of these adjustments were necessary because, without them, the system struggled to segment these smaller objects effectively.

As the simulation experiments, a training phase for the feature-matching algorithm was necessary. In this case, more than in the simulation, proper lighting was crucial to ensure accurate object recognition.
The environment created was simple with a white background and a white plane to avoid recognition and segmentation problems. This because the aim of this set of experiments was to verify the versatility of the developed sorting application, demonstrating its scalability to different systems, rather than emphasizing the limitations of the recognition algorithm, which were already known.

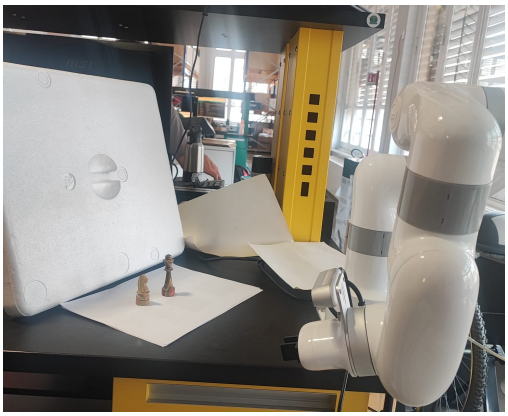

Figure 5.18: Set up of the scene in the real case.

For the Pick and Place server node, it was necessary to modify the task planning group and specify the gripper frame to ensure the correct execution of the Pick and Place operations. Also in this case, the sorting parameters were specified in a configuration file, taking into account the physical limit of the new robot used when it was defined the placement position. In particular, the policy based on the name of the object was updated by adding two new sections for these two new objects, "horse" and "queen", thus defining the appropriate sorting parameters for effective operation.
Again, the server nodes were included in the main launch file. When the application was launched, similar to the simulation case, the objects were recognized and localized, and the MoveIt planning scene was constructed. Finally, the pick-and-place task was executed.

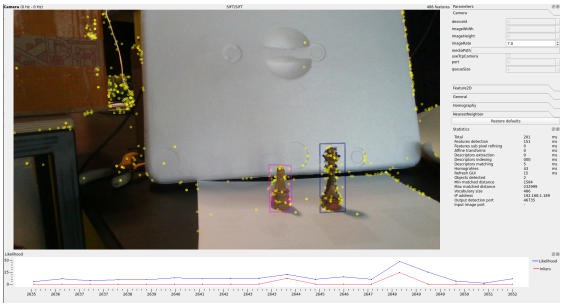

Figure 5.19: Object Recognition in the real case.

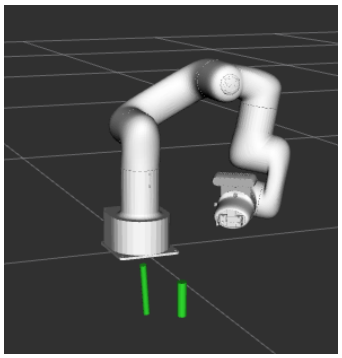

Figure 5.20: MoveIt Planning Scene for the real case.

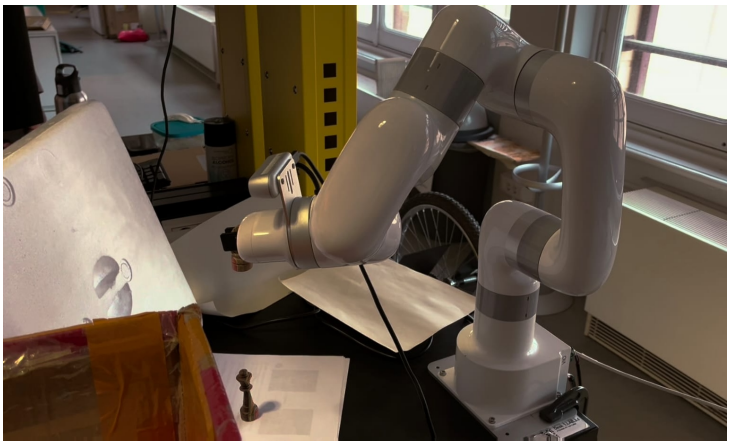

Figure 5.21: ULite6 during the Lift stage of the object.

As can be seen in 5.20, although the two objects do not had regular shapes, the Compute Dimension server based on PCL approximated them as cylinders and this information was used by the Pick and Place server to build the planning scene and execute the task.

The system demonstrated easy integration with the new hardware, requiring only minor modifications to adapt to the Lite6 manipulator. This seamless transition highlights the versatility and scalability of the developed sorting application, affirming its effectiveness in real-world scenarios.

Chapter 6

Conclusions and future researches

The primary objective of this thesis was to develop a flexible and efficient sorting system based on a robotic manipulator, utilizing computer vision technologies and motion planning algorithms. The system demonstrated its ability to recognize, segment, and manipulate objects both in simulated environments and on a real hardware setup. The use of flexible algorithms allowed the system to adapt to a different hardware configuration with only minor modifications, proving its scalability and versatility. Additionally, the implementation of Docker containers enhanced the system's scalability and made it production-ready by providing a consistent and portable environment for deployment, facilitating easy integration and management across various platforms.

The system exhibits both robustness and efficiency, sorting approximately 2–3 objects per minute. In fact, the planning time per object is around 2.8 seconds, with an execution time of 17.766 seconds, making the Pick and Place task for three objects approximately 61.775 seconds. Given that the average total application time is 80.224 seconds, the additional time includes recognition, segmentation, and communication between various ROS nodes. Considering the hardware used, the system demonstrates efficient operational speed.

The feature-matching algorithm performs effectively in controlled, well-lit environments, where conditions are optimized for accurate object recognition and localization. However, when the lighting is reduced, the algorithm struggles to recognize objects reliably, highlighting its sensitivity to changes in illumination and the potential need for more robust techniques to handle varying light conditions.

In a controlled environment, the localization performance of both the feature matching algorithm and the point cloud based techniques is generally reliable. The algorithms are able to achieve high accuracy in object localization, with errors typically staying within a 7% range, which is acceptable for many industrial applications. However, certain factors, such as variations in object geometry and surface reflections, occasionally introduce larger errors. These inaccuracies often result in only a portion of the object being segmented, leading to a perceived height shorter than the true value. Despite these segmentation challenges, the overall sorting task remains unaffected, as the system continues to perform effectively in its pick-and-place operations.

Cylinder segmentation is highly robust, with errors in radius determination below 1%, while box segmentation shows lower performance, particularly in complex environments. This performance discrepancy is attributed to the lack of a dedicated segmentation function for boxes in the Point Cloud Library (PCL), which sometimes results in difficulties in detecting three orthogonal planes. Additionally, the sensitivity of the cylinder segmentation algorithm occasionally leads to misclassification of cubes as cylinders.

Handling very low objects occasionally presents challenges, but the system's robustness is evident as it automatically resubmits requests if recognition, localization or pick and place planning fails, ensuring continuous operation.
Considering the total number of experiments conducted (40 in simulation, 10 per policy), the task is successfully executed on the first attempt in 90% of cases. For the remaining 10%, success is achieved after subsequent requests, demonstrating the system's adaptability and reliability in executing sorting tasks.

As mentioned above, the current performance of the system in complex or visually cluttered environments is limited by existing techniques. Incorporating deep learning or neural networks can help to overcome these limitations by improving both the geometric and semantic capabilities of the system. In particular, it can offers several advantages:

- **Improved Object Recognition and Localization**: Deep learning can enhance the system's ability to identify and classify objects in challenging environments. By using convolutional neural networks (CNNs) and other advanced architectures, the system can learn to recognize objects with different textures, lighting conditions, and partial occlusions. As a consequence, the improved recognition would also enhance object localization, ensuring more accurate positioning and handling, even in complex or cluttered environments, thus improving the overall system's precision.

- **Enhanced Segmentation**: while the Point Cloud Library (PCL) offers specific segmentation functions for objects like spheres that could be integrated, deep learning techniques would provide more flexibility, enabling the segmentation of complex geometries. Furthermore, cube segmentation in PCL has proven less robust than cylinder segmentation due to the lack of a dedicated function, an issue that deep learning could potentially resolve by offering more accurate and adaptable segmentation method.

Regarding the Pick and Place task, future research can focus on evaluating alternative motion planning algorithms, improving path planning in complex environments and allowing for smoother and more effective movements while maintaining the essential speed required for sorting operations. In addition, future work could explore how to integrate mesh-type objects into the MoveIt environment, by seamlessly receiving data from computer vision algorithms, as is currently done with primitive shapes. This integration would allow the system to handle more complex geometries and increase its flexibility in sorting tasks.

Appendix A

RANSAC Algorithm

The "RANdom SAmple Consensus" (RANSAC) is an iterative method that is used to estimate parameters of a mathematical model from a set of data containing outliers. The RANSAC algorithm assumes that all of the data we are looking at is comprised of both inliers and outliers. Inliers can be explained by a model with a particular set of parameter values, while outliers do not fit that model in any circumstance. Another necessary assumption is that a procedure which can optimally estimate the parameters of the chosen model from the data is available [38]. As pointed out by Fischler and Bolles (the authors that proposed this algorithm), unlike conventional sampling techniques that use as much of the data as possible to obtain an initial solution and then proceed to prune outliers, RANSAC uses the smallest set possible and proceeds to enlarge this set with consistent data points [39].

Algorithm 1 RANSAC

1: Select randomly the minimum number of points required to determine the model parameters.
2: Solve for the parameters of the model.
3: Determine how many points from the set of all points fit with a predefined tolerance.
4: If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold, re-estimate the model parameters using all the identified inliers and terminate.
5: Otherwise, repeat steps 1 through 4 (maximum of N times).

The number of iterations, N , is chosen high enough to ensure that the probability p (usually set to 0.99) that at least one of the sets of random samples does not contain an outlier.

An advantage of RANSAC is its ability to robustly estimate the model parameters, while a disadvantage is that there is no upper bound on the time it takes to compute these parameters. When the number of iterations computed is limited the solution obtained may not be optimal, and it may not even be one that fits the data in a good way. In this way RANSAC offers a trade-off; by computing a greater number of iterations the probability of a reasonable model being produced is increased.

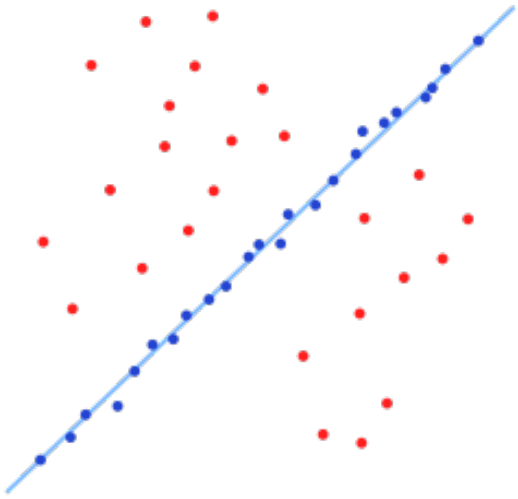

Figure A.1: Application of the RANSAC algorithm on a 2-dimensional set of data. The outliers are in red, while inliers in blue. The blue line is the result of the work done by RANSAC.

Acknowledgements

Questo elaborato sancisce la fine del mio percorso di laurea magistrale e per questo mi sembra doveroso fare dei ringraziamenti.
Innanzitutto, ringrazio il mio relatore, il professor Marcello Chiaberge, per la sua disponibilità e per avermi permesso di entrare in contatto con l'azienda Reply, presso cui ho svolto il progetto che ha portato alla realizzazione di questa tesi.
Ringrazio anche il mio relatore aziendale, il dottor Claudio Chieppa, per la presenza e la gentilezza sempre dimostrata. Grazie soprattutto per l'opportunità che mi hai dato permettendomi di svolgere il mio lavoro di tesi in un ambiente lavorativo così interessante e dinamico, che mi ha arricchito sia professionalmente che umanamente.
Un grazie particolare va a Simone e Matteo Carlone, per avermi guidato durante tutto il mio percorso di tesi e per avermi trasmesso la passione per la robotica.
A Sam per i consigli e le risate, e a tutti i colleghi di Concept Engineering per avermi accolta e messa a mio agio.

Un grazie va alla mia famiglia, per avermi sempre sostenuta.
Ai miei genitori, per essersi sempre fidati di me e per avermi supportato in ogni mia scelta.
A mia sorella, per esserci sempre stata.
A mio fratello, per aver sempre fatto il tifo per me.
Ai miei nipotini, per la gioia che inconsapevolmente portano nella mia vita.

A Matteo, il mio ragazzo, per aver sopportato tutte le mie paranoie, per i consigli e la tranquillità che mi hai sempre trasmesso.

Ai miei compagni universitari con cui ho condiviso questo percorso, in modo particolare a Giulia e Alessia, ormai amiche dalla triennale. Spettegolare insieme a voi, tra una pausa caffè e l'altra, ha reso tutto più leggero.

Ai miei amici di sempre e a tutti coloro che hanno creduto in me, grazie.

Bibliography

[1] *Sorting Robots in Express Logistics Industry – A Complete Guide*. URL: https://www.unboxrobotics.com/sorting-robots-in-express-logistics-industry-a-complete-guide/ (cit. on p. 1).

[2] Maria F. Mogro. «Sorting Line Assisted by A Robotic Manipulator and Artificial Vision with Active Safety». In: *Journal of Robotics and Control (JRC)* (2024) (cit. on pp. 2, 6, 7).

[3] Nikita V. Belov and Andrey G. Vovik. «Adaptive Control System for the Process of Sorting Objects Using a Robotic Arm». In: *International Russian Smart Industry COnferences* (2024) (cit. on pp. 2, 7).

[4] Georg Maier. «A Survey of the State of the Art in Sensor-Based Sorting Technology and Research». In: *IEEE Access* (Jan. 2024) (cit. on p. 6).

[5] Guiran Liu and Binrong Zhu. «Design and Implementation of Intelligent Robot Control System Integrating Computer Vision and Mechanical Engineering». In: *International Journal of Computer Science and Information Technology* (2024) (cit. on p. 7).

[6] Meijing Song. «Research on Intelligent Logistics Sorting Robot Control Based on Machine Vision». In: *ADS: a multidisciplinary journal* (June 2024) (cit. on p. 7).

[7] Usama Shakoor et others Hafiz Muhammad Tayyab Abbas. «Automated Sorting and Grading of Agricultural Products based on Image Processing». In: *IEEE* (Feb. 2020) (cit. on p. 7).

[8] Ming Zhong et others Shuo Zhou. «Framework of rod-like crops sorting based on multi-object oriented detection and analysis». In: *ELSEVIER* (Jan. 2024) (cit. on p. 8).

[9] Haitao Wu et others Botao Zhong. «Mapping computer vision research in construction: Developments, knowledge gaps and implications for research». In: *Automation in Construction* (2019) (cit. on p. 8).

[10] C.S.George Lee King-Sun F 'King-Sun Fu. *ROBOTICS: CONTROL, SENSING, VISION AND INTELLIGENCE*. 0071004211: MCGRAW-HILL EDUCATION (ISE EDI, 1987 (cit. on p. 8).

[11] Siva Prasad Ebrahim Karami and Mohamed Shehata. «Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images». In: *arXiv.org* (Oct. 2017) (cit. on p. 9).

[12] Aly Amin Abdelmgeid M. Hassaballah and Hammam A. Alshazly. «Image Features Detection, Description and Matching». In: *Springer* (Feb. 2016) (cit. on pp. 9, 10).

[13] Kamal Jain Surendra Kumar Sharma and Anoop Kumar Shukla. «A Comparative Analysis of Feature Detectors and Descriptors for Image Stitching». In: *MDPI* (May 2023) (cit. on pp. 10, 11).

[14] Chunping Wang et others Ying Yu. «Techniques and Challenges of Image Segmentation: A Review». In: *MDPI* (Feb. 2023) (cit. on pp. 12–14).

[15] *The Mechanics Behind Pick and Place Robots*. URL: `https://conveyorsystemsltd.co.uk/the-mechanics-behind-pick-and-place-robots/` (cit. on p. 14).

[16] *What is a pick and place robot?* Apr. 2023. URL: `https://6river.com/what-is-a-pick-and-place-robot/` (cit. on p. 15).

[17] *Pick and Place Robots: An In-Depth Guide to Their Functionality and Applications*. Mar. 2024. URL: `https://www.wevolver.com/article/pick-and-place-robots-an-in-depth-guide-to-their-functionality-and-applications` (cit. on p. 15).

[18] Ioan A. Sucan et others Mark Moll. «Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization». In: *IEEE Robotics and Automation Magazine* (Aug. 2015) (cit. on p. 16).

[19] Kevin Lynch et others Howie Choset. *Principles of Robot Motion*. The MIT Press, Cambridge, Massachusetts, 2005 (cit. on pp. 16, 17).

[20] M. Elbanhawi. «Sampling-Based Robot Motion Planning: A Review». In: *IEEE Access* (Jan. 2014) (cit. on pp. 16, 17).

[21] N. Ratliff and et al. «CHOMP: Gradient Optimization Techniques for Efficient Motion Planning». In: *IEEE International Conference on Robotics and Automation* (May 2009) (cit. on pp. 16, 18).

[22] Ioan A. Sucan et others Ryan Luna. «Anytime Solution Optimization for Sampling-Based Motion Planning». In: *IEEE International Conference on Robotics and Automation* (May 2013) (cit. on p. 17).

[23] Howie Choset. *Robotic Motion Planning: Sample-Based Motion Planning*. URL: https://www.cs.cmu.edu/~motionplanning/lecture/Chap7-Prob-Planning_howie.pdf (cit. on p. 17).

[24] B. Siciliano and et al. *Robotics: Modelling, Planning and Control*. Springer, 2009 (cit. on p. 17).

[25] Open Robotics. *ROS2 Documentation*. 2024. URL: https://docs.ros.org/en/humble/index.html (cit. on p. 19).

[26] Docker community. *docker docs*. 2024. URL: https://docs.docker.com/guides/docker-overview/ (cit. on p. 21).

[27] PickNik Robotics. *MoveIt 2 Documentation*. 2024. URL: https://moveit.picknik.ai/main/index.html (cit. on pp. 22, 24–26, 32).

[28] B. Sumukha and C.S. Asha. «Gesture Controlled 6 DoF Manipulator with Custom Gripper for Pick and Place Operation using ROS2 framework». In: *IEEE* (May 2024) (cit. on p. 23).

[29] *Planners Available in MoveIt*. URL: https://moveit.ros.org/documentation/planners/ (cit. on p. 25).

[30] Sung-Ho Lee et others Hyeong Ryeol Kam. «RViz: a toolkit for real domain data visualization». In: *Springer* (Apr. 2015) (cit. on p. 27).

[31] Nathan Koenig and Andrew Howard. «Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator». In: *IEEE International Conference on Intelligent Robots and Systems* (Oct. 2004) (cit. on p. 27).

[32] *UFactory 850 user manual*. URL: https://www.ufactory.cc/wp-content/uploads/2023/07/UFactory-850-User-Manual-V2.1.0.pdf (cit. on p. 28).

[33] *Intel RealSense*. URL: https://www.intelrealsense.com/depth-camera-d435i/ (cit. on p. 29).

[34] Husarion. *Husarion docs*. 2024. URL: https://husarion.com/tutorials/ros-tutorials/5-visual-object-recognition/ (cit. on p. 39).

[35] *OpenCV Documentation*. URL: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html (cit. on p. 40).

[36] Zoltan-Csaba Marton Aitor Aldoma et al. «Point Cloud Library Three-Dimensional Object Recognition and 6 DoF Pose Estimation». In: *IEEE Robotics & Automation Magazine* (Sept. 2012) (cit. on p. 42).

[37] *PCL Walkthrough*. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/walkthrough.html (cit. on p. 42).

[38] *How to use Random Sample Consensus model*. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/random_sample_consensus.html (cit. on p. 75).

[39] Konstantinos G. Derpanis. «Overview of the RANSAC Algorithm». In: *Image Rockester NY* (May 2010) (cit. on p. 75).