POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Smart mobile manipulation for human-robot assistive applications

Supervisors

Prof. Marina INDRI

Prof. Pangcheng David CEN CHENG

Candidate

Cesare Luigi BLENGINI

April 2024

Summary

Within the field of Robotics, human-robot interaction and cooperation are fascinating topics with important real-world applications. The overall objective is to program robots capable not only of safely sharing the environment with a human but also of actively interacting with the person to jointly accomplish tasks. The main challenges are meeting safety requirements and making the interaction feel as natural as possible, so that the robot can easily integrate with the human workflow. Recent advancements in machine learning have made it easier and more convenient to develop such solutions. They provide powerful computer vision algorithms which require only camera sensors to operate. The present thesis tries to address these issues by experimenting with a mobile manipulator, the LoCoBot wx250s. The robot is equipped with a mobile base and a 6DOF arm. It can monitor its surroundings with a LIDAR and a RGB-D camera. Its task is to navigate in an environment, such as a laboratory or warehouse, to pick up a designated object and to safely hand it to a human. The robot is controlled through the ROS framework, which is the standard for robotic applications. The navigation and robot's arm control systems are managed through standard ROS packages. Custom solutions leveraging a mix of classical and machine learning algorithms were developed to take care of object recognition, grasp point prediction and object handover. The robot must be able to detect the target object, grasp it in a way that is functional to its subsequent handover and then perform the handover itself. It must be ensured that the human is presented with the safe part of the object. The solution for the grasp point prediction is based on the concept of part affordance, where each part of the object is labeled with its apparent purpose. In this application, the focus is on detecting which parts of the object are dangerous or safe to grasp for the human. Since it relies on visual data, affordance detection is tightly linked with object detection, so both independent and joint solutions were tested. The proposed solutions leverage popular algorithms such as YOLO, DeepLab and others. The handover solution combines the MediaPipe framework for hand tracking and the depth camera stream to map the human hand in the 3d space. The robot is then able to extend its arm towards the human hand and release the object once the handover is completed. The present thesis discusses in detail the implementation

of the algorithms, the challenges encountered, the possible alternative solutions and the future developments, and it showcases the results on the real robot.

Table of Contents

Chapter 1

Introduction and problem statement

Robots are widely employed across various sectors, particularly in industry and logistics. They are traditionally utilized for repetitive tasks requiring precision and/or physical strength. Typically, they perform a series of predefined actions, such as in assembly lines. Even when they possess a certain degree of autonomy, they operate in highly controlled environments leading to the development of extremely specific solutions.

Recently, there has been growing interest in service robots. This type of robot is designed to work closely with humans, providing assistance and collaborating on shared tasks. However, developing these robots poses significant challenges. Firstly, they must operate in dynamic and unstructured environments that are difficult to model beforehand. Robots need to interact with and manipulate a wide range of objects and navigate new and dynamic surroundings. Furthermore, since they work in close proximity to humans, ensuring safety is fundamental, and it is important to make interactions feel natural and pleasant. Given all the requirements at play, recent advancements in machine learning algorithms can be immensely helpful. These algorithms have seen significant development, particularly in the field of computer vision, offering a wide array of tools to enhance robotic capabilities. From sophisticated neural networks for object detection, classification, and modeling to advanced algorithms for understanding human poses and gestures, these technological advancements play a pivotal role in improving human-robot interaction.

To develop service robots, it is crucial not to only focus on software and algorithms but also to ensure the hardware platform is well-suited. Mobile manipulators are the ideal choice for this purpose. They possess a mobile base, upon which one or more robotic arms are mounted. Combining a moving base with a manipulator

grants these robots both the mobility of mobile robots and the dexterity of manipulators. They are highly versatile and efficient, making them ideal for completing pick-and-place and object-handling tasks in large workspaces.

1.1 Thesis goal

The aim of this thesis is to explore various techniques and strategies for object manipulation to facilitate collaboration and interaction between humans and robots. Specifically, it seeks to experiment with new solutions enabled by advancements in machine learning. To achieve this goal, the design of a robotic assistant based on a mobile manipulator is proposed, tasked with assisting humans in their work by identifying and delivering requested objects. The robot must autonomously navigate to an object repository, identify the requested item, pick it up, navigate back to the human, and deliver it. Ensuring safety and providing a pleasant and natural experience for humans is crucial throughout the process. In particular, the robot must recognize hazardous parts of the object to consider them during transport and delivery. To address this problem, it is necessary to develop:

- An object recognition system that not only identifies the object but also provides additional information about its hazardous parts.
- A grasping system that takes into consideration the dangerous part of the object and facilitates the subsequent handover.
- A handover system which allows the robot to autonomously and safely deliver the object to the human.

1.2 Thesis structure

This thesis is organized as follows. In Chapter 2, the state-of-the art and recent advancements in subjects related to the thesis goal will be analyzed. Chapter 3 presents an overview of the main software platform used to build the solutions. In particular, it focuses on the Robot Operating System (ROS). Chapter 4 describes the hardware of the mobile manipulator. Chapters 5 and 6 describe in great detail possible models and solutions for handover and grasping. Chapter 7 describes the machine learning model used, with emphasis on the building of the dataset. Chapter 8 describes how handover, grasping and navigation solutions are implemented on the real robot through dedicated software modules. Chapter 9 describes all the experiments run on the robot to test the developed solutions. Finally, the conclusion and future works can be found in Chapter 10.

Chapter 2

State of the art

The context in which this thesis is developed is quite complex, as it encompasses various research domains. It transitions from classical robotics problems, such as managing robotic arm movement and navigation, to interdisciplinary issues, like interaction and modeling of the surrounding environment. Part of the challenge lies in integrating traditional robotics algorithms with novel tools. In this regard, particular emphasis is placed on machine learning algorithms, which in recent years have made significant strides forward, enabling much more natural interaction with humans and the environment. In this chapter, an analysis of the state-of-the-art related to the research problem will be presented.

2.1 Mobile base and manipulator planning

Every robotics application depends on the possibility of controlling the robot's movement. The motion planning problem can be divided into path planning and trajectory planning. Path planning involves determining a feasible path or route from the robot's current position to a designated goal location within its environment. It focuses on high-level decision-making, considering some factors, such as kinematic constraints, obstacles, terrain, and efficiency, to find an optimal or near-optimal path. On the other hand, trajectory planning deals with the detailed execution of movements along the planned path, considering the robot's dynamic constraints. Motion planning for manipulators depends on the inverse kinematic problem which aims to find the joint angles or positions needed to reach a desired end-effector pose. The computed solution can be employed to create a path and then a trajectory within the joint space. The Open Motion Planning Library (OMPL) [1] offers many state-of-the-art sampling-based motion planning algorithms and is the default in the popular robotic framework ROS.

Mobile base motion planning consists of global and local planners. A global

planner generates a high-level path from the robot's current position to its destination, considering the overall environment and obstacles. On the other hand, the local planner focuses on the immediate surroundings of the robot, adjusting the path generated by the global planner to account for real-time changes and obstacles encountered during navigation. In [2], the authors made a comprehensive review of state-of-the-art mobile robot planning algorithms.

For mobile manipulators, there are two ways to handle the planning of the mobile base and manipulator motion. The simpler method involves using separate planners for each system. This divides the robot's movement into two tasks, but it does not ensure the best overall solution for the entire robot. Alternatively, a holistic planner can be used, treating the mobile manipulator as one integrated system and planning for both the manipulator and base simultaneously. Although this creates a more complex system with higher degrees of freedom, it allows for simultaneous movement of the base and manipulator, resulting in smoother and more natural motion.

In this thesis, the decision has been made to use two independent planners. While it may not be the most efficient solution, it simplifies development and experimentation since it allows for the development of separate modules for base and manipulator control.

2.2 Affordance estimation

To let the robot perform complex actions with objects, like robot tool use and robot-to-human or human-to-robot handover, manipulable objects must be modeled in a much finer way, conveying information on the apparent use of each object part. This is the concept behind part affordance, which refers to the perceived and actual properties of an object that determine how it can be used. Affordance can be estimated by means of classical algorithms by relying on object geometrical or material properties. However, machine learning offers a wide range of different solutions. In [3], the authors propose an algorithm for simultaneous object detection and affordance estimation. The algorithm is based on convolutional neural networks (CNNs) and outputs a bounding box of the object plus an affordance map, where an affordance label is assigned to each pixel inside the bounding box. So the network is both an object detection network and an image segmentation network. The algorithm is trained on the IIT-Affordance dataset presented in [4]. A similar approach is proposed in [5] but translated to 3D objects. The authors designed 3D AffordanceNet, a dataset with point-wise affordance annotations over the object's point clouds. The dataset, containing 23 object classes and 18 affordance labels, is then tested on popular 3D point cloud classification algorithms such as PointNet++ [6] and DGCNN [7]. Tests were also performed on partial view point

clouds. In addition to these more generic algorithms, specific techniques have also been proposed to tackle more targeted problems. For instance, [8] introduces Ganhands, a neural network for estimating how a human would grasp objects. This can prove highly beneficial for human-robot interaction tasks, as it enables the robot to understand the manipulable part for the human and it facilitates a handover accordingly. The authors propose a generative model, which takes a single RGB image as input, regresses the 3D shape and pose of the objects in the scene, estimates the grasp types, and predicts the pose of a 3D hand model that minimizes a graspability loss.

2.3 Object detection and pose estimation

For any task involving autonomous object manipulation, the initial crucial step is undoubtedly the ability to recognize and identify the object of interest within the working environment. In this realm, machine learning has proven to be exceptionally effective, representing the state of the art of performance. Detection over 2D RGB images will be taken into consideration. Detecting an object involves predicting either its bounding box (a rectangular frame outlining the object's spatial extent in the image) or, for more detailed detection, a segmentation mask, which precisely outlines the boundary of each object within the image. Two main approaches exist: (i) two-stage and (ii) single-stage detectors. In two-stage detectors, a region proposal network (RPN) generates potential object bounding boxes, which are then refined and classified by a subsequent network. Examples of this approach are Faster R-CNN [9], for bounding box prediction, and Mask R-CNN [10] for instance segmentation. Single-stage detectors directly predict bounding boxes and class probabilities for all objects in a single step, making them faster and more suitable for real-time applications. The main algorithm using this approach is YOLO [11], which in its latest version, YOLOv8, offers both instance segmentation and bounding box models.

Pose estimation goes a step further from object detection. It refers to the process of determining the position and orientation (pose) of objects or other entities relative to a reference frame. This is a fundamental problem in robotics, since the grasping pose of the object must be known. The robot uses sensors such as LIDARs, cameras and depth cameras to gather information about its environment, then an algorithm is applied to perform the estimate. Classical algorithms involve manual feature extraction and are mainly based on geometric features. Machine learning has been proven very useful in this field, especially to work with limited input data, such as only RGB streams. In [12], the authors present a network that uses CNNs to directly regress object pose from a single RGB image.

2.4 Grasp pose estimation

Alongside traditional grasping techniques involving separate object detection, pose estimation, optional affordance detection and a hand-crafted grasp pose, recent advancements in machine learning have facilitated the development of end-to-end solutions that handle all or some of these tasks simultaneously. Grasping can be performed with multiple end-effectors, the most popular ones are parallel grippers and suction cups, but a lot of work is being done on anthropomorphic hands as well. Since the mobile manipulator used for developing this thesis is equipped with a parallel gripper, only this kind of solutions will be taken into consideration. Numerous grasping algorithms exist, but due to the robot's setup with only a single RGB-D camera, only algorithms that utilize either a single-shot color or depth image will be considered. Estimating a grasp pose usually means predicting a gripper pose, characterized by a point and a rotation relative to a fixed frame and the gripper target width. In [13], the authors propose a network called GPD, to directly predict the grasping of unknown objects in a dense object clutter without explicitly relying on pose estimation. The algorithm samples grasp candidates from the point cloud and then classifies them using a CNN. This work is extended in [14], where the GPD algorithm is improved and integrated with an affordance estimation algorithm, based on mask R-CNN, to filter out, before classification, all the grasping points incompatible with a target task. A more complex approach for task-constrained grasping is explored in [15]. In that paper, the authors propose an architecture comprised of two modules: a grasp affordance module and a visual affordance module. The grasp affordance module predicts coarse grasp candidates from depth data based on the target task, and then the prediction is refined with the information from the visual affordance module, which builds a 3D heat map of object affordance combining geometrical and color information. The visual affordance module is trained on the already mentioned 3D AffordanceNet. The grasp affordance module is trained on an expanded version of the ACRONYM dataset [16], a synthetic dataset containing 3D object models labeled with grasps obtained in simulation, where each grasp was also labeled with a task-related label. The performance of that machine learning algorithm depends heavily on the training dataset. Datasets for general object manipulation are very difficult to build and annotate. The authors of [17] proposed REGRAD, a synthetic dataset containing cluttered scenes annotated with object bounding boxes, 3D and 2D grasp poses, segmentation masks and, most interesting a relational hierarchy of objects in the scene, which describes how objects overlap together. At the time of writing, one of the best-performing task-agnostic object grasping algorithms in cluttered scenes is AnyGrasp [18], an improvement over [19]. Some of its performance advantages depend on its training on the Graspnet-1billion dataset [19], which contains more than 90000 real-word RGB images annotated with 1 billion grasps.

2.5 Handover

Handover is an integral part of human-robot assistive applications, since it enables the robot to interact with a human by sharing objects with that person. As we are dealing with a mobile manipulator, handover presents challenges related to both the movement of the mobile base and the planning of the robot arm's motion. In [20], the authors devised a framework for grasping an object and planning a handover on a stationary robot. It assumes that the 3D model of the object is available. The robot's grasp position for the object is determined by predicting how a human would grasp the object, using the previously mentioned Ganhands neural network. The handover point is then computed by tracking the human hand using the MediaPipe framework [21]. In [22], the authors explore various options for mobile handovers, with a specific emphasis on the receiver's perception. They particularly investigate on-the-go handovers, where the exchange occurs without necessitating the robot to come to a complete stop.

Chapter 3

Software platform

3.1 ROS

The Robot Operating System [23] is a framework that makes possible to easily build robotic applications. Despite its name, it is not an operating system, but a collection of tools and libraries, offering drivers, state-of-the-art algorithms, development tools and more. ROS is available on multiple platforms. However, since it works very closely with the computer's operating system and libraries, every ROS version is tested and certified to work on a specific Ubuntu LTS release.

At the time of writing ROS is transitioning from version 1 to version 2. ROS 2 introduces a lot of breaking changes and the code written in the two versions is not interoperable and needs porting. Because of this, some ROS 1 packages are still not available in ROS 2, and this is slowing the transition.

3.1.1 Architecture

The main idea behind ROS is to divide a complex task into smaller and simpler tasks, each carried out by a dedicated process, which exchanges information with the other processes. A crucial characteristic of ROS is the possibility of making these subprocesses run on multiple machines, connected to the same network. This makes it possible to offload heavy computation from the robot to a workstation next to the robot's environment.

The main components of the software architecture are: nodes, topics, messages, services and the parameter server. These elements can be grouped and distributed in packages.

Nodes

Nodes are the backbone of ROS-based robotic systems, embodying modularity and encapsulation. Each node represents a distinct unit of computation that performs a specific task, such as processing sensor data, controlling actuators, or executing algorithms. Nodes communicate with each other through ROS topics, services, and actions, facilitating a distributed architecture, where functionalities are decoupled and can run concurrently. This modularity enables developers to create complex systems by composing and connecting smaller, reusable components, fostering code reusability and maintainability. Each node is uniquely identified by its name. Nodes can be written in both Python and C++ languages.

In each ROS system, there is a node running as the *ROS master*. It is the first node to be started. It provides naming and registration services to other nodes and it tracks services and topics. It also manages the parameters server. Overall the job of the *ROS Master* is to enable the nodes to locate each other. After the location process, the nodes communicate with each other peer-to-peer.

Topics

Topics provide a publish-subscribe communication mechanism that enables nodes to exchange messages. Topics serve as communication channels, through which nodes can publish messages to be consumed by other nodes that have subscribed to the same topic. In this way, nodes do not need to interact directly but can use topics as a bridge, ignoring the underlying implementation. This approach reduces complexity and allows for flexible and scalable architectures, where nodes can be added, removed, or replaced without disrupting the overall system functionality. Furthermore, topics support one-to-many communication, enabling multiple nodes to subscribe to the same topic and receive relevant data concurrently, enhancing system flexibility and efficiency. Topics are created the first time a node publishes or subscribes to it. Each of them is characterized by its name and the message type it can serve.

Messages

Messages serve as the means by which nodes exchange data asynchronously. Messages are structured units of information defined by specific message types, encapsulating various types of data, such as sensor readings, control commands, or state updates. Message types are defined using ROS message description language and are typically stored in ROS packages. This standardized format ensures interoperability between different components of the system, allowing nodes to communicate seamlessly despite potential differences in programming languages or hardware platforms. Additionally, the lightweight and efficient nature of ROS

messages facilitates real-time processing and distributed computation, enabling robotic systems to handle diverse data sources and perform complex tasks.

Services

Services offer a request-response communication pattern that allows nodes to call specific functionalities provided by other nodes. Services define a clear interface for invoking operations or accessing resources offered by nodes, enabling synchronous communication between clients and servers. This interaction model is particularly useful for tasks that require direct interaction or coordination between nodes, such as requesting sensor calibration, querying robot status, or triggering specific actions. Services provide a reliable and deterministic means of communication, ensuring that requests are processed and responses are received promptly.

Parameter server

The ROS Parameter Server is a centralized key-value storage system that enables nodes within a ROS-based system to store and retrieve parameters dynamically at runtime. It provides a convenient mechanism for sharing configuration parameters, settings, and other data among different nodes.

3.1.2 Robot description

To be able to effectively use and control the robot's hardware, ROS provides abstractions for describing the physical characteristics and kinematics of robots. This is done through URDF files and the transformation tree.

URDF

ROS Unified Robot Description Format (URDF) is a markup language, widely used in the ROS ecosystem for describing the physical characteristics and kinematics of robotic systems. URDF files define the geometry, visual appearance, inertial properties, joint structure, and sensor configurations of robots in a structured XML (eXtensible Markup Language) format. These descriptions are essential for the visualization, simulation, motion planning, and control of robots within ROS-based environments. URDF files provide a standardized way to represent robotic models, making it easier for developers to create, share, and integrate robotic systems in ROS.

Transformation tree

The transformation tree in ROS refers to the hierarchical relationship between coordinate frames, known as the TF tree. This tree represents the spatial transformations between different coordinate frames in a robotic system, allowing for accurate localization, navigation, and sensor data fusion. Each node in the TF tree corresponds to a specific coordinate frame, such as the base link of a robot or the frame of a sensor. The transformation tree is managed by the *TF package* [24]. This package keeps track of multiple coordinate frames in time and enables to make conversions between them.

A standard transformation tree is usually defined as something like:

$$map \rightarrow odom \rightarrow base\_link \rightarrow sensor\_link$$

. In this transformation tree:

- *map* is the static fixed frame
- *odom* is the frame where the robot localizes itself. In the case of mobile robots this changes over time, otherwise is fixed.
- *base_link* is a coordinate frame fixed to the base of the robot.
- *sensor_link* is a frame fixed to one of the robot sensors. For each sensor, there will be a specific frame of this type.

3.1.3 Simultaneous mapping and localization

Simultaneous mapping and localization (SLAM) is the process of a robot constructing a map of its environment while simultaneously determining its position within that map. Thanks to SLAM, a map of the environment can be built without prior knowledge by just moving a robot into it.

RTAB-Map

RTAB-Map [25] (Real-Time Appearance-Based Mapping), solves the SLAM problem by merging odometry data from motion sensors with measurements from LIDARs and RGB-D cameras. This is achieved by performing loop closure, which is the process of detecting and correcting errors in the estimated trajectory of a robot by recognizing previously visited locations. The robot extracts meaningful features from its sensors and the loop closure detector uses a *bag-of-words* approach to determine how likely these features come from a previous location or a new location. RTAB-Map uses a memory management technique to limit the number of locations considered as candidates during loop closure detection. This greatly

improves performance and makes it possible to run the algorithm in real time. RTAB-Map uses a graph to represent robot poses at different times. Once a loop closure is detected, graph optimization is performed to refine the map by taking into consideration spatial constraints obtained by sensor data.

SLAM toolbox

SLAM toolbox [26] is a simpler and more lightweight alternative to RTAB-Map. It uses odometry data and lidar scans to build a representation of the environment. Localization is then performed by matching observed features from the scans to features of the map. Although it does not offer the same advanced features as RTAB-Map, the performance are still satisfactory and it is preferred for resource-constrained applications.

3.1.4 Navigation and localization

Navigation in ROS is managed by the *navigation stack* [27], which is a collection of different software that enables autonomous navigation. Once a map of the environment is provided, the stack employs path-planning algorithms to determine the robot trajectory needed to reach the desired pose while avoiding obstacles. The position of the robot in the map is constantly updated through localization algorithms, and it is possible to implement dynamic obstacle avoidance.

Cost-map

The aim of a cost-map is to represent the reachability of a location in the environment, by taking into consideration obstacles, map boundaries and robot physical constraints. At its core, a cost-map divides the robot's operating space into a grid of cells, with each cell assigned a label indicating whether it is free, occupied or unknown. This occupancy grid is then enriched by a process called inflation, in which cost values assigned to cells in the cost-map are adjusted to create a buffer zone around obstacles. The purpose of cost-map inflation is to ensure that the robot maintains a safe distance from obstacles while navigating and to make the planning algorithm aware of narrow passages. Usually, inflation is performed by specifying a radius and expanding the boundaries of obstacles by that radius. The cost-map can also be dynamically updated from sensors, such as cameras or LIDARs to represent dynamic obstacles.

Path planning

Path planning is managed by two components: a local and a global planner. Global path planning involves generating a high-level path from the robot's current position

to a specified goal while considering the overall layout of the environment. Usually, only static obstacles are taken into consideration. ROS offers many algorithm options, such as A* or Dijkstra's. Local path planning manages small adjustments in the robot path to keep the globally planned path while avoiding obstacles. ROS packages related to navigation implement popular algorithms such as Dynamic Window Approach (DWA) and Timed Elastic Band (TEB).

Localization with AMCL

While SLAM Toolbox and RTAB-Map both offer their own localization algorithm (primarily based on feature matching), another popular localization algorithm is the Adaptive Monte Carlo Localization (AMCL) [28]. The AMCL approach uses a particle filter to track the pose of a robot against a known map. The probability distribution of the robot pose is represented by particles scattered over the map. The particles move around following the odometry data, then their position is refined based on information about the robot's surroundings obtained by the lidar.

3.1.5 Perception

It is crucial that the robot is aware of its surroundings, in particular, to manage obstacle avoidance during navigation and arm movement. The most basic form of perception is done through the LIDAR, which gives a planar representation of obstacles. A more advanced representation of obstacles in 3D space can be achieved with depth cameras and Octomap [29].

Octomap

Octomap takes a point-cloud or depth image as input and constructs a voxel-based representation of an obstacle in 3D space. A 3D occupancy grid is built, where every voxel can be labeled as either free or occupied space. This information is saved in an octree data structure. An octree is a hierarchical data structure commonly used in computer graphics, robotics, and computational geometry to partition three-dimensional space. At the top level, the octree represents the entire 3D space as a single cube. As the structure is recursively subdivided, each cube is split into eight smaller cubes, and this subdivision continues until a certain condition is met, such as reaching a predefined depth or having a minimum size for each cube. Thanks to the efficiency of this data structure, Octomap can be used to track obstacles in real-time. Efficiency can be further improved by Octomap's tuning options that enable the user to specify voxel size, limit the input camera view distance and more.

3.1.6 Motion planning

Motion planning involves generating feasible trajectories for the robot's manipulator to move its end-effector from an initial configuration to a desired goal configuration while satisfying constraints. Motion planning in ROS is handled by the MoveIt! framework.

MoveIt!

MoveIt! [30] is an open-source framework, which offers a convenient and high-level interface to control a manipulator and also to manage grasps. The framework must be correctly set up to work with the desired manipulator by providing a physical description (from a URDF file) and the actuator's physical constraints.

The user can specify a desired end-effector pose and the framework will try to compute a feasible path. Different planning algorithms are available. By default MoveIt! relies on the OMPL framework for planning, offering CHOMP and STOMP as alternatives. Constraints of various types can be imposed, such as joint states, links position and orientation and visibility of a sensor. The framework offers three kinds of interfaces:

- C++ interface, to write C++ code.
- Python interface, to write Python scripts.
- RViz GUI, a graphical interface used for testing.

Both the C++ and Python interfaces provide a *MoveGroupCommander* object, equipped with methods for querying desired poses, imposing constraints, and planning and executing motion. Additionally, a *ScenePlanningInterface* object is accessible, offering functions for incorporating obstacles into the planning scene in the form of 3D shapes and meshes.

MoveIt! further extends its functionality through a plugin system. For instance, an Octomap plugin is available to dynamically manage obstacle mapping in real time.

3.2 Machine learning

As previously discussed, machine learning is becoming increasingly used in robotics applications. It offers state-of-the-art solutions for object detection and lately, a lot of work has been also done on grasping and affordance detection algorithms. As it will be discussed in Chapter 7, machine learning models were trained and adapted to accomplish some of the robot's tasks. The model implementations were done using popular machine learning frameworks such as PyTorch [31] and TensorFlow [32].

3.3 MediaPipe

For the handover task, it is crucial to know the position of the human receiving the object. This can be done in a convenient way using the MediaPipe framework [33]. MediaPipe is a collection of machine learning algorithms for computer vision developed by Google. Each algorithm is written and trained in TensorFlow with the goal of being efficient and have good performances on mobile devices. Mediapipe provides ready-made solutions with models already trained for tasks like:

- Object detection
- Image segmentation
- Pose estimation
- Hand tracking
- Facial recognition

Python, java, javascript and ios libraries and examples are available for each solution. These allow to quickly and easily write code for desktop, web and mobile applications. MediaPipe also provides the C++ framework in which the solutions have been developed, thus allowing the development of customized solutions.

Hand landmark solution

This solution takes as input an RGB image and returns the position of 21 key points of the hand (corresponding to the joints), as in Figure 3.1, and their position in 3D space relative to the geometric center of the hand. This permits to track the position of the hand in 2D space and to reconstruct a 3D model of the hand. The solution allows the tracking of one or two hands and can classify a hand as the right or left hand.

The machine learning model is divided into two components that work in cascade:

- Hand detector: it is a CNN that behaves like a single shot detector to estimate the region of the image where the hand is located.
- Hand tracker: it is a CNN that performs a regression on the portion of the image returned by the hand detector to estimate the key points.

This solution is further developed in the gesture recognition solution to not only predict keypoints, but to also classify hand gestures. The hand landmarks are inputted into another neural network dedicated to classification. Eight gesture labels are defined: *Unknown*, *Closed_Fist*, *Open_Palm*, *Pointing_Up*, *Thumb_Down*, *Thumb_Up*, *Victory* and *ILoveYou*.

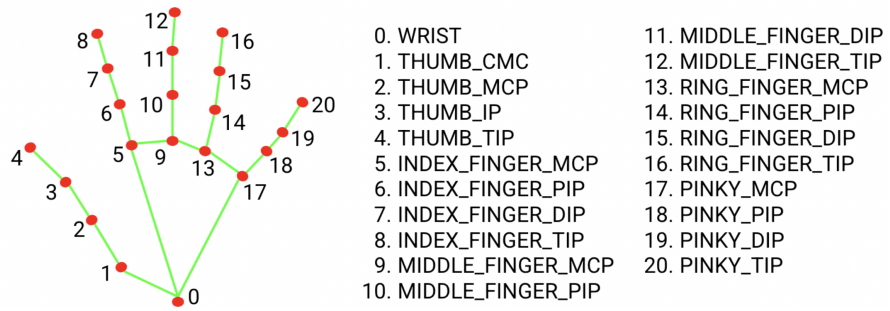


Figure 3.1: Hand landmarks predicted by MediaPipe hand landmark solution [34].

3.4 2D and 3D image manipulation

To perform handover and grasping tasks, the robot must understand its environment by analyzing RGB-D camera streams. This involves using tools to process 2D images and manage 3D representations of the surroundings. Additionally, preparing data for training neural networks also necessitates the use of image manipulation libraries.

OpenCV and PIL

OpenCV (Open Source Computer Vision Library) [35] is a popular and feature rich multi-platform framework for image processing and computer vision. At its core, OpenCV provides a vast collection of optimized algorithms and functions designed to tackle a diverse range of tasks, including image and video acquisition, manipulation, enhancement, and analysis. From fundamental operations like image resizing, color space conversions, and filtering to more advanced functionalities, such as feature detection, object recognition, and motion estimation, OpenCV offers a comprehensive toolkit for both research and practical applications.

PIL (Python Imaging Library) [36] is a versatile library primarily focused on image processing and manipulation tasks in Python. It provides simple yet powerful tools for tasks such as opening, saving, and converting images in various file formats, as well as performing basic operations like resizing, cropping, and rotating images. Additionally, PIL offers a range of filters and enhancements for adjusting image properties such as brightness, contrast, and color balance. While not as extensive as OpenCV, PIL excels in its simplicity and ease of use.

Open3D

Open3D [37] is a powerful Python open-source library designed to facilitate 3D data processing tasks, including point cloud manipulation, geometry processing, and scene reconstruction. One of its key capabilities is its ability to efficiently handle large-scale 3D data, enabling tasks such as point cloud registration, alignment, and downsampling with ease. Moreover, Open3D offers a rich set of functions for geometric operations, including surface mesh generation, and voxelization, which are essential for tasks like 3D modeling and simulation. Another notable feature of Open3D is its support for segmentation and clustering algorithms.

Chapter 4

Hardware platform

The robot in use is a LoCoBot WX250 6DOF [38] from Trossen Robotics. It is a small mobile maniplator equipped with a mobile base, an arm with 6 degrees of freedom and various environmental sensors. It is suited for applications where the robot needs to navigate in an environment and to pick and place small objects.

Manipulation hardware

The robot is equipped with a WidowX 250 manipulator with 6 degrees of freedom (DOF), capable of reaching up to 650 mm with a payload capacity of 250 g. It is powered by Robotis' DYNAMIXEL X Series servos, achieving a movement accuracy of 5 mm to 8 mm and a repeatability of 1 mm. It offers advanced features including high resolution (4096 positions), customizable PID parameters, temperature and positional feedback, among other functionalities.

The gripper is simply made of two parallel fingers actuated by an electric servo, which is enough for most applications. No force sensor is present.

Mobile base

The robot's mobile base is the Kobuki base; it is a low-cost mobile base designed for education and research on state of art robotics. Its onboard battery guarantees 90 minutes of autonomous navigation. It is driven by two powered wheels located on diametrically opposite sides of the base, while stability is granted by 2 caster wheels, one on the front side and the other on the back one. This means that the base is equipped with a differential drive. Its highly accurate odometry and calibrated gyroscope enable precise navigation.

On-board computer

All the computations and hardware control are managed by an Intel NUC. A NUC is a small form factor computer, with somewhat constrained computational capabilities, especially regarding GPU performance, but with great connectivity and low power consumption. It runs Ubuntu 20.04 with ROS Noetic and comes with ROS packages from Trossen Robotics to control all of its peripherals.

LIDAR

The robot is equipped with the RPLIDAR A2, a 2D LIDAR designed for indoor use, offering a complete 360-degree scanning capability. With a rapid rotation speed, each unit can capture up to 8000 laser range samples per second. It can perform comprehensive 2D scans operating within a 12-meter range. The resulting 2D point cloud data facilitates tasks such as mapping, localization, and object/environment modeling.

Camera

The robot mounts an Intel RealSense D435 camera. The camera captures two video streams, one with RGB data and another with depth information. The ideal operating range of the camera is between 0.3 m and 3 m. The depth information is obtained by projecting infrared light on the scene and analyzing how the light is reflected. By combining these two streams, the robot can reconstruct a scene in 3D space. This is particularly useful in navigation, object detection and object grasping. The camera is not in a fixed position and can be tilted and panned thanks to two motors.

Chapter 5

Handover

The handover is a procedure where the robot hands to the human the object it picked up. Since the human and the robot must interact together, the procedure is very delicate and must be carefully planned. The two main challenges are safety and figuring out how to make the interaction feel natural.

5.1 Scene Modeling

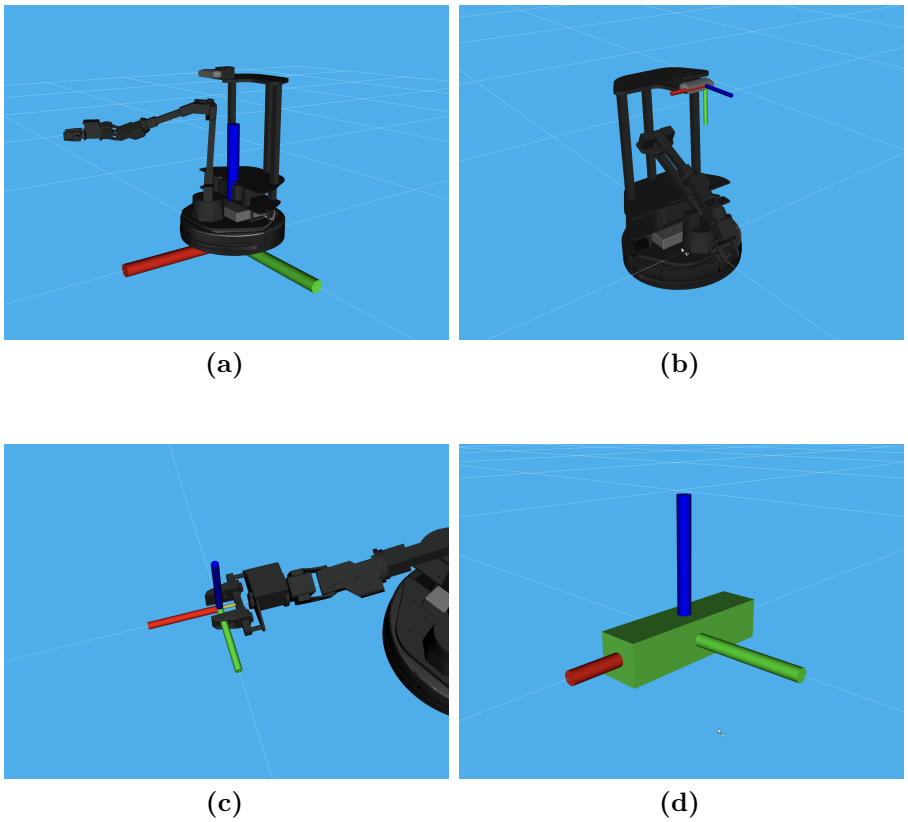
In a basic handover scenario, the robot and human face each other. It may be feasible for the robot to track the human's movements and position itself directly in front of the person. This could be achieved using a human pose detection model, like MediaPipe Pose Landmark Detection. Further exploration of this approach could be considered for future research. The current study focuses on a static handover, where the robot is already in the handover area and will reach the handover point without moving its base. It is essential to create models of both the object being grasped and the human hand, in order to calculate the handover point and pose accurately. These models ensure that the object is both comfortable and safe for the human to grasp. Additionally, the planned arm movement needs to ensure that the object does not move in any dangerous manner.

5.1.1 Reference frames

To better understand the handover scene, four coordinate frames are defined:

- The **world frame**, centered at the robot base with axes oriented as in Figure 5.1a. The XY-plane represents the plane the robot is moving on.
- The **camera frame**, centered in the middle of the camera sensor with axes oriented as in Figure 5.1b. It describes the position and orientation of the robot camera.

- The **gripper frame**, centered on the gripper with axes oriented as in Figure 5.1c. It describes the position and orientation of the gripper
- The **object frame**, centered on the object with axes oriented as in Figure 5.1d. It describes the position and orientation of the object.



(a) **(b)**

(c) **(d)**

Figure 5.1: World frame (a), camera frame (b), gripper frame (c) and object frame (d). Red represents the x-axis, green the y-axis and blue the z-axis.

5.1.2 Object representation

The object the robot is designed to interact with has sharp edges, blades, and other hazardous parts. In objects modeling, it is crucial to consider these elements to ensure safety. To keep things simple, objects are modeled as boxes, where the z-dimension (height), the x-dimension (width) and the y-dimension (depth) are fixed and chosen as the maximum width, height and depth of the manipulable

objects. On the x-axis, the end of the box pointing towards the positive direction represents the object handle, while the other is the dangerous part. More complex options for object modeling will be discussed in Section 6.1.2.

5.1.3 Hand representation

The position and the geometry of the hand are obtained by the MediaPipe hand tracking solution already described in Section 3.3. The robot takes pictures of the human standing in front of it with the RGB-D camera. The RGB stream is fed to MediaPipe, configured to recognize a single hand. If a hand is detected, the keypoints are overlayed onto the depth images and projected in the 3D space by using the depth camera intrinsic parameters provided by ROS. The camera is modeled as a pinhole camera and is characterized by an intrinsic matrix of the form:

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where:

- **fx** and **fy** are the focal lengths on the x and y axis expressed in pixels.
- **cx** and **cy** indicate the offset in pixels between the pixel coordinates and the camera coordinate system. The camera coordinate system usually has its origin in the center of the image while the pixel coordinate system of the image has its origin in the top-left corner.

The hand keypoints coordinates in the camera coordinate system are the following. The Z value is the distance measured by the depth camera. The X and Y values are obtained as:

$$X = (x - cx)\frac{Z}{fx} \tag{5.1}$$

$$Y = (y - cy)\frac{Z}{fy} \tag{5.2}$$

where [**X,Y,Z**] are the coordinates in the camera system and [**x,y,z**] are the coordinates in the image system. The keypoints coordinates can then be converted from the camera frame into the world frame by ROS.

This method has a few limitations. The primary one is that it does not consider hand self-occlusion, which means that keypoints hidden by the hand may appear closer to the camera than their actual positions. Another limitation is related to the precision of the depth camera. Occasionally, slender objects like fingers and fingertips may not be accurately detected by the depth camera, causing keypoints to be mistakenly placed in the background.

To mitigate this, the hand is modeled as a sphere as in Figure 5.2. The center of the sphere is computed as the center of the 6 palm keypoints. The sphere radius is modeled around an approximation of a finger length. This lets the robot know the position and the encumbrance of the hand without any information on where the hand is facing. This model was chosen because it is simple and takes into account all possible hand positions, making it safer.

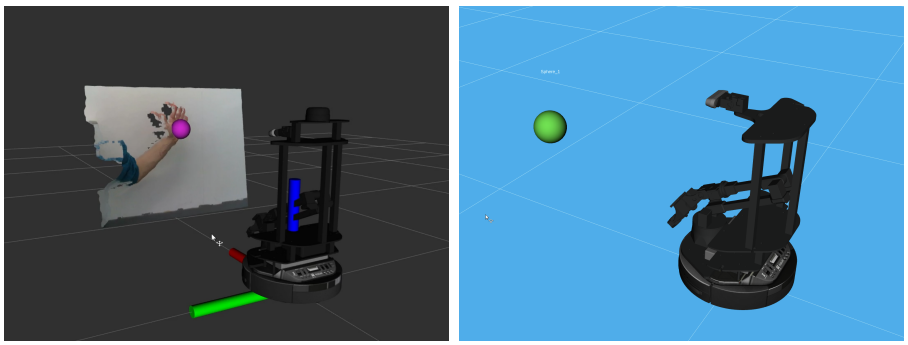


Figure 5.2: The hand modeled as a sphere over the point cloud.

A more complete model could be built by exploiting MediaPipe 3D hand reconstruction capabilities. MediaPipe Hands also provides the keypoints in 3D-world coordinates, using a coordinate system that has its origin at the geometric center of the hand. By using the pinhole camera model, it is possible to get the transformation between this coordinate system and the camera coordinate system and build a mesh of the hand.

5.1.4 Handover pose

The robot performs the handover by moving its 6 DOF arm while holding the object inside the gripper. Te arm movement is controlled by ROS through MoveIt!, which computes the path to bring the gripper to a desired pose. So the final pose reached by the object during the handover is not controlled directly, but through the gripper pose. The gripper pose is described as a point (handover point) and a quaternion (handover orientation) representing the offset and the rotation between the gripper frame and the world frame. It is assumed that the human is in front of the robot, the human hand is modeled as a sphere as described in Section 5.1.3, and the object is held perpendicular to the gripper. The target final pose for the object is perpendicular to the XY-plane (the ground) with the handle pointing upwards, so that the dangerous part of the object is not pointing towards the human.

The handover point is located on the line connecting the base link of the robot arm to the center of the sphere representing the human hand (Figure 5.3). This

point is located at a distance from the hand center equal to the radius of the sphere plus a safety margin. The gripper needs to reach this point while remaining parallel to the XY-plane (ground). To increase the reach of the robotic arm, the gripper is rotated around the world Z-axis to align with the line connecting the arm base link to the hand. The target gripper pose is shown in Figure 5.3

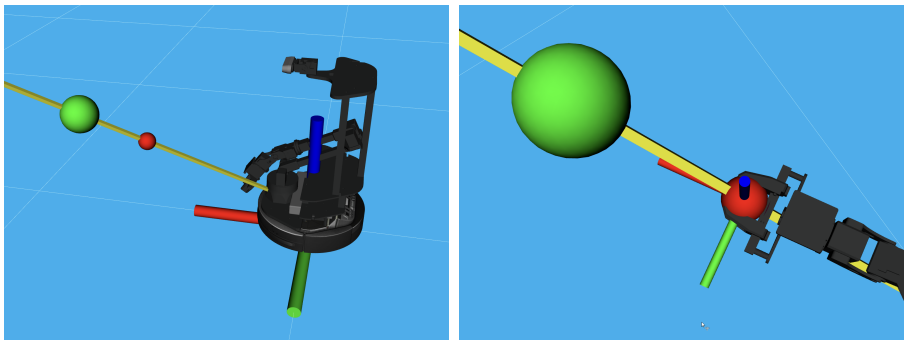


Figure 5.3: The handover point (red sphere) and the final gripper pose.

5.1.5 Handover path

It is essential for the robot to move predictably and safely to ensure that no hazardous parts of the object are exposed during its movement. This also helps the human to anticipate where the object will be coming from. The handover starts with an object held close to the robot's body and already oriented vertically. The planned path must ensure that the robot keeps the object vertical and the movement occurs in a constrained portion of space. In this case, the gripper should be moved only along the line connecting its starting point to the handover point, as in Figure 5.4, and be only free to rotate around the world Z-axis.

Through the OMPL planner, MoveIt! offers the possibility to impose constraints over the planned path, as already mentioned in Section 3.1.6. To achieve the desired motion two kinds of constraints are used:

- **Position constraints**. A bounding volume is defined for a specific link (in this case the gripper), ensuring that the position of the link remains within this volume throughout the motion. The volume dimension can be manipulated so that it becomes a plane or a line.
- **Orientation constraints**. A quaternion is assigned to a specific link to maintain its predefined orientation. Tolerances ranging from 0 to 1 are set for each axis, indicating the degree of deviation allowed from the specified orientation. A value of 1 indicates unrestricted rotation along that axis.

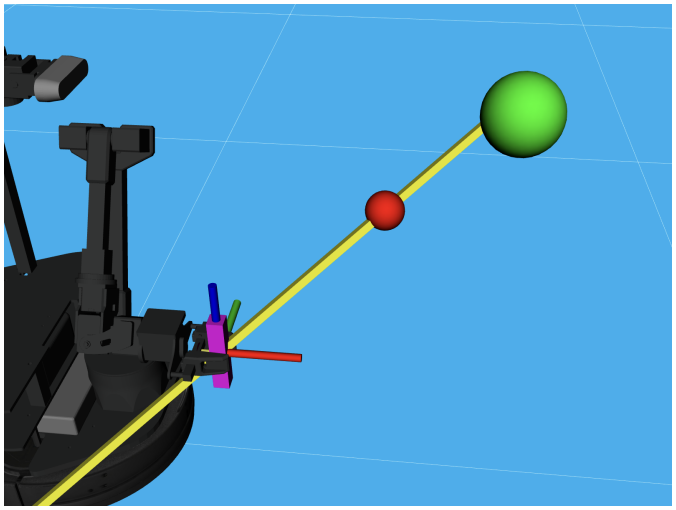


Figure 5.4: The handover path (yellow line). The pink box represents the object.

5.2 Procedure

The handover process begins with the robot positioned at the handover point, its arm retracted in the carrying position, and the object held in the gripper. From there, it proceeds through the following steps:

- When a human hand is inside the robot's field of view, the robot can start receiving commands.
- When the command to start the handover is given, the robot locks the current hand position and extends its arm towards it. If during this step the human hand moves, the new position will not be taken into account.
- Once the arm's final position is reached, the arm locks in place and the robot is ready to receive further commands.
- At this point the human can safely grasp the object and then give the robot the command to open the gripper.

5.2.1 Commands

The robot and the human must synchronize before performing certain steps of the handover procedure. This is achieved by making the robot wait for human commands. The commands can be issued in multiple ways. The simplest one is through a computer program, but it makes the interaction very unnatural, as the

human is not directly interacting with the robot. A more elegant way is to use hand gestures. As stated in Section 3.3, MediaPipe Hands offers a solution for hand gesture recognition. It is possible to map a command to a MediaPipe gesture, triggering actions when a gesture is detected. For instance, the handover sequence could initiate upon recognizing the *Open_palm* gesture.

Chapter 6

Grasping and Detection

Grasping is a fundamental and challenging part of robot object manipulation. There are multiple valid approaches to grasping, each dependent on the task requirements and the target objects. For this application, it is important that the performed grasp is functional to the subsequent handover and guarantees safety. In this scenario, the specific object to grasp is not predetermined. Instead, the robot is trained to identify objects from a diverse range of classes. When presented with a scene, the robot's task is to detect an unfamiliar object, determine its location, and then use the grasping algorithm to pick it up.

6.1 Scene Modeling

It is assumed that the robot has reached the pick-up area and the objects are laying in front of it. The robot must be able to recognize the correct object and calculate a feasible path to reach it and grasp it with the gripper. The grasp pose must be functional to the subsequent handover, taking into consideration the different affordances of the objects. To describe the scene a world and camera frame are defined as in Section 5.1.1.

6.1.1 Object detection and affordance representation

Object detection makes the robot able to detect and classify objects by using a machine learning network like YOLO to predict 2D bounding boxes over the camera RGB images. The bounding boxes are used to isolate each object from the scene clutter. Information on object part affordance is obtained by giving each pixel of an input image its appropriate affordance label. Four different affordance labels were defined for this task:

- **Background**, each pixel which is not part of the object.

- **Danger**, the parts of the objects that are directly used to perform some kind of job, like blades, points etc. These are usually also the most dangerous parts of the object.
- **Handle**, the part of the objects used to hold the tool, likes handles. Ideally, these parts must be kept free for the human to grasp.
- **Grasp**, for each class of objects some parts are defined so that they are suitable for the robot to grasp but do not interfere with human grasping.

The affordance mask obtained in this way can be overlayed on top of the point cloud to know the affordance label of each point and infer affordance to the object 3D representation. This detailed affordance representation can be obtained by training a segmentation network, like DeepLab, over the whole RGB image taken by the robot camera or just a portion of the image cropped around the object bounding box. Further details can be found in Section 7.4. A simpler representation of affordance can be achieved by defining a keypoint for each affordance label, except for the background. These keypoints can be generated from the affordance mask by calculating the center of the pixels associated with each affordance label. A machine learning model, such as the YOLO Pose, can then be employed to identify the object's bounding box and forecast the keypoints. More information on this process can be found in Section 7.3. The affordance information can be brought from the 2D image to the object point cloud following the steps detailed in Section 5.1.3. This makes it possible to define affordance volumes in 3D space. The same can be done for affordance keypoints.

6.1.2 Improved object representation and pose estimation

The depth and color data captured by the RGB-D camera can be utilized to construct a three-dimensional model of an object. However, due to the limitations of capturing the image from a single viewpoint and the occlusion of the object itself, only an approximate model can be created, resulting in a loss of detail. First of all the point cloud of the scene is constructed by projecting the captured image in 3D space (Figure 6.1), as discussed in 5.1.3. The points belonging to the objects are extracted. This can be done on the color image by computing the affordance mask and removing all pixels labeled as background. Alternatively, it can be done on the point cloud itself, by using a RANSAC algorithm to find the plane where the object is laying on, and remove all the points that belong to it. The result is shown in Figure 6.1. The point cloud of the object can be turned into a 3D model by computing its convex hull, which is the minimal convex shape which completely encloses all the point cloud points. Using PCA (principal component analysis) it is possible to compute the three-dimensional bounding box of the convex hull, as

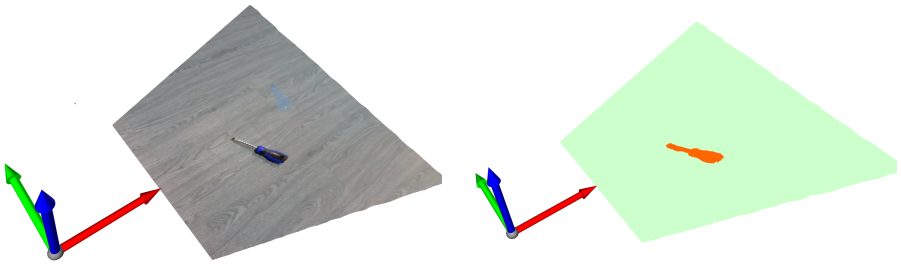


Figure 6.1: The object point cloud and the segmented point cloud.

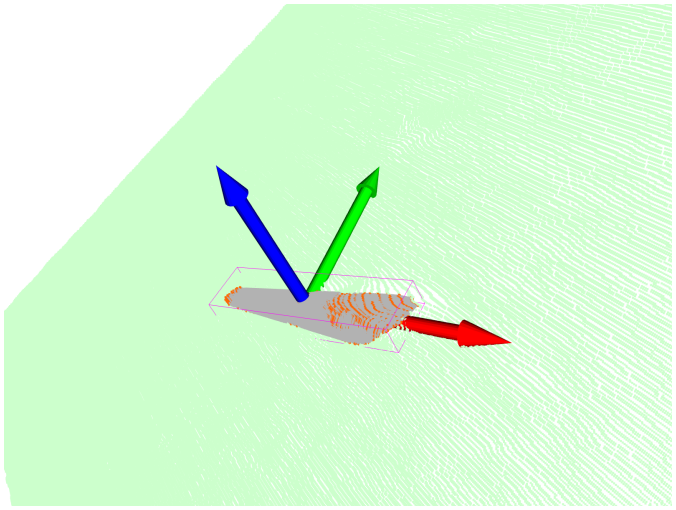


Figure 6.2: The object convex hull enclosed by the oriented bounding box obtained from PCA.

shown in Figure 6.2. This can be used to model the object as a box with width, length and eight equal to the ones of the bounding box. The object is assumed to lay on a plane parallel to the XY-plane (the ground). Its position is defined by a point representing its center and an angle describing its rotation around the Z-axis. These two parameters can be determined from the three-dimensional bounding box. The orientation of the bounding box is represented by a coordinate frame (object frame) with axes aligned to the object's principal components. Specifically, the x-axis aligns with the object's major axis. Therefore, the rotation angle is calculated by measuring the angle between the world system's X-axis and the projection of the object frame's x-axis onto the world XY-plane. The center is

simply obtained as the center of the bounding box. The positive direction of the object frame's x-axis should align with the object handle to indicate the location of the object's safe and hazardous areas. To ensure that affordance information is used, a guide vector is defined as the projection over the XY-plane of the vector connecting the center of the object "handle" volume to the center of the object "danger" volume. If the object frame x-axis and the guide vector lay on opposite quadrants, the object frame is rotated by 180° to align with it. The final object model is shown in Figure 6.3.

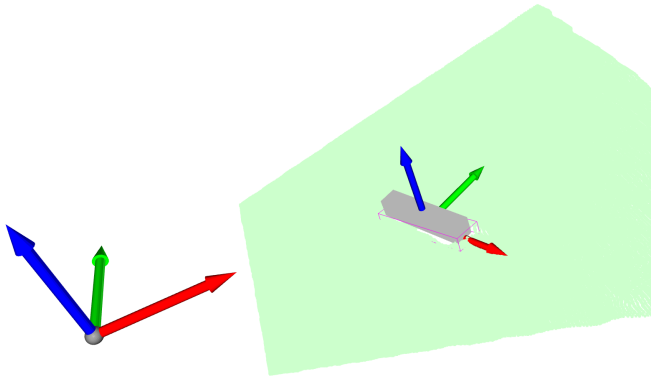


Figure 6.3: The estimated object pose represented with the grey box.

6.1.3 Grasp pose

The robot performs a parallel grasp. To do that, the gripper must be aligned with the object, so the gripper frame must be rotated to match the object frame. Furthermore, the gripper must be perpendicular to the world XY-plane (the ground), so a 90° rotation over the gripper y-axis is performed to align its x-axis with the world Z-axis. The grasp point is computed as the center of the "grasp" volume.

6.1.4 Obstacles modeling

The robot must be aware of its surroundings to be able to plan a collision-free path for the grasp. MoveIt! provides the option to consider obstacles during planning. These obstacles can be incorporated into the planning scene as 3D meshes or represented by voxels, which can be dynamically updated through integration with Octomap. The integration with Octomap is suitable for environments that

change dynamically. However, it can be computationally intensive, and issues may arise when the robot's arm blocks the camera's view. Octomap tracks the robot model's real-time position and uses the data to exclude areas blocked by the robot's components from the obstacle representation. This works well except when the robot's arm gets really close to the camera. Due to the camera characteristics, depth perception degrades when the object is closer than 30 cm. This creates artefacts in the point cloud used by Octomap, which are wrongly detected as obstacles. As the robot is not intended to grasp objects in a dynamic environment, we can model the scene by converting its point cloud into a 3D model, which can then be loaded into MoveIt! Obtaining a mesh from a point cloud is not trivial and surface reconstruction techniques must be used. Some of them are:

- **Alpha shapes**. The alpha shape is a generalization of the convex hull. It is essentially a bounding structure that encapsulates the original points, allowing the identification of concave and convex regions within the point cloud. The parameter alpha determines the level of detail in the reconstruction, influencing the size of geometric elements in the resulting model. Higher alpha values yield simpler shapes, while lower values capture finer details. With alpha set to its maximum level, the alpha shape is equal to the convex hull.

- **Ball pivoting**. Conceptually a 3D ball with a specific radius is dropped onto a point cloud. If the ball touches three points (and does not pass through them), it forms a triangle. Then, the algorithm initiates pivoting the ball from the edges of these triangles, creating additional triangles whenever it encounters three points where the ball does not fall through. Smaller values of the ball's radius produce more detailed surfaces, while larger values result in smoother approximations.

- **Poisson surface reconstruction**. This method solves an optimization problem to obtain a smooth surface. It has usually better performance than the other two methods.

Another method involves defining a voxel grid and marking a voxel as occupied whenever a point from the point cloud falls within it. Subsequently, this voxel representation can be transformed into a 3D mesh. Some scene modeling example are shown in Figure 6.4.

6.2 From grasp to handover

The grasp should conclude in a position conducive to subsequent navigation and handover. This entails holding the object and arm as close to the robot's body as possible to avoid obstacles during navigation. Additionally, aligning the gripper

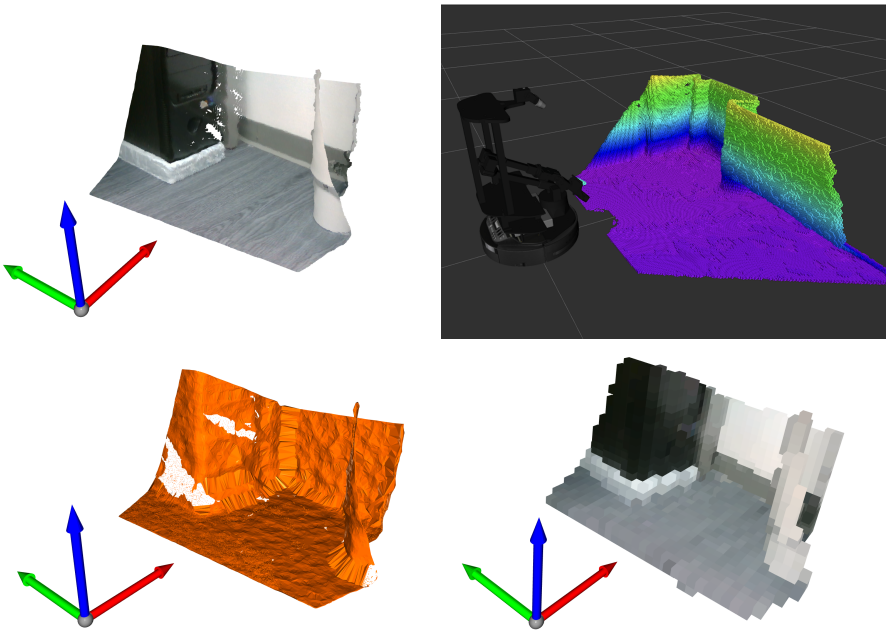


Figure 6.4: From the top-left (clockwise): scene point cloud, Octomap occupancy grid, scene mesh reconstructed from alpha shape and scene voxels representation.

parallel to the ground is advantageous due to the orientation constraints outlined in Section 5.1.5. To achieve these objectives, a standard arm and gripper pose, referred to as the "carrying position", is engineered by manually assigning appropriate values to the manipulator joints. MoveIt! simplifies motion planning to this pose by providing methods to specify desired target joint values. The carrying pose is shown in Figure 6.5.

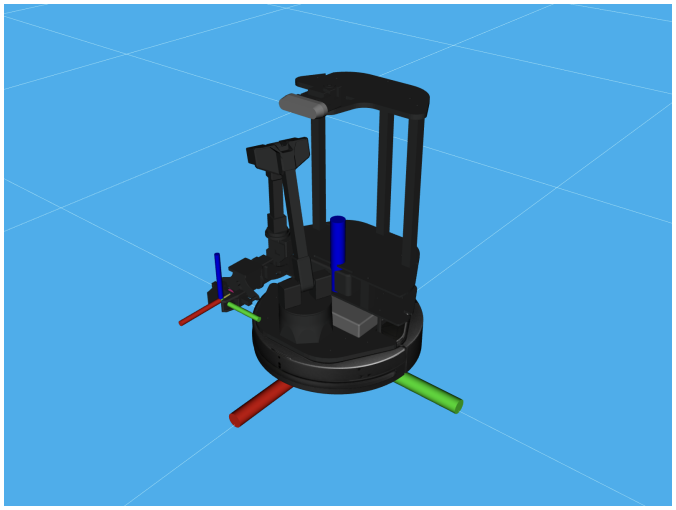


Figure 6.5: Robot carrying position.

Chapter 7

Models training

Machine learning models are used for object and affordance estimation tasks, as already stated in Chapter 6. Three different solutions were trained:

- Simultaneous object and full affordance detection using AffordanceNet.
- Simultaneous object and simplified affordance detection using YOLO Pose.
- Object detection with YOLO and subsequent full affordance detection with DeepLab.

7.1 Dataset

A custom dataset was built to tackle both object and affordance detection, ensuring compatibility across all three solutions without the need for any modifications. The dataset contains 7 classes of heterogeneous objects and 3 affordance labels. The affordance labels are already discussed in Section 6.1.1 and are "danger", "handle" and "grasp". The classes are: knife, hairbrush, razor, screwdriver, paintbrush, lighter and highlighter. These objects were chosen because the parts used to accomplish a task are clearly distinguishable from the handles. Additionally, their shapes and weights were considered to ensure that the robot could easily distinguish them using the depth camera and physically manipulate them. The same objects appear multiple times with pictures taken from multiple angles.

Each entry of the dataset is comprised of the following:

- **Rgb image** of the object.
- **Object mask**, an image labeling the pixel as either part of the object or as part of the background.

- **Affordance mask**, an image labeling object parts with their corresponding affordance labels.
- **Serialized affordance mask**, a ".sm" file containing the affordance mask serialized using the *pickle* Python package. It is used by AffordanceNet.
- **YOLO annotation file**, a text file containing the position of the object bounding box, the object class and, optionally, the keypoints for the YOLO Pose model.
- **AffordanceNet annotation file**, an XML file containing the position of the object bounding box and the object class.

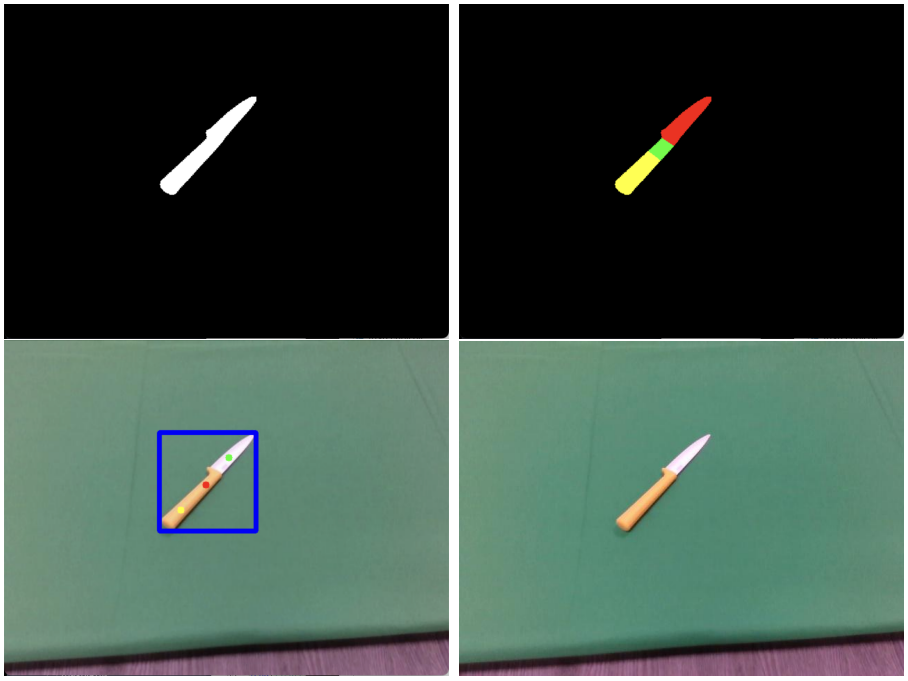


Figure 7.1: An entry of the dataset with the RGB image, segmentation mask, object mask, bounding box and keypoints.

7.1.1 Image acquisition and annotation

The pictures are taken from the robot's point of view with its Intel RealSense D435 camera. A simple ROS program was written to perform the task. The robot is controlled by a DualShock 4 controller, the mobile base can be moved around with the left analog stick and the shoulders button, the camera can be tilted and

panned with the right analog stick and pictures can be taken by pressing the "X" button. The depth and RGB streams are aligned and synchronized and are accessible through Rviz to see what the robot is looking at. The object is placed in front of the robot and multiple pictures are taken from different angles. To allow better generalization, the dataset is complemented with some pictures taken with a phone camera.

Once the images are taken, bounding boxes and object mask must be extracted. This process is automated with the help of the U2Net machine learning model for background removal [39]. The model is pre-trained on a wide variety of objects and shows to work well on the objects included in the dataset. It outputs an RGBA image (Figure 7.2), where the alpha channel is used to make the background transparent and highlight the object. The object mask is constructed by extracting the alpha channel, thresholding it so that each pixel has either value 0 or 255 and exporting it into a gray-scale image. The bounding box is obtained by taking the object mask and extracting the minimum enclosing rectangle with the OpenCV *findContours* method.

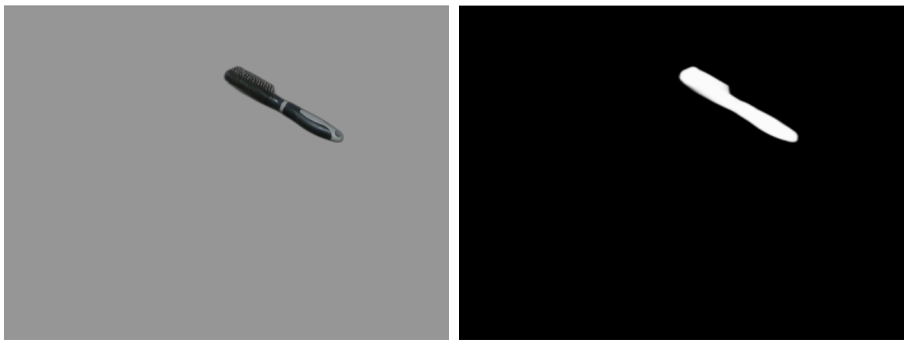


Figure 7.2: Output of U2Net and mask.

The affordance mask is obtained by manual annotation using a custom-made Python script built using OpenCV. The script iterates over each image contained in a specified folder, displays it to the user and lets the person draw two rectangles over the picture, a blue one surrounding the handle of the tool and a red one surrounding the dangerous part of the tool. Then, the object mask is applied so that the portions of the rectangles outside the object are merged into the background. Finally, the segmentation mask is created as a gray-scale image, where each pixel in the original masked image is assigned a new value according to the following procedure: the background black pixels get a value of 0, the red pixel value of 3, the blue pixel value 2, and the remaining pixels, representing the part of the object graspable by the robot, value 1. The full process is detailed in Figure 7.3.

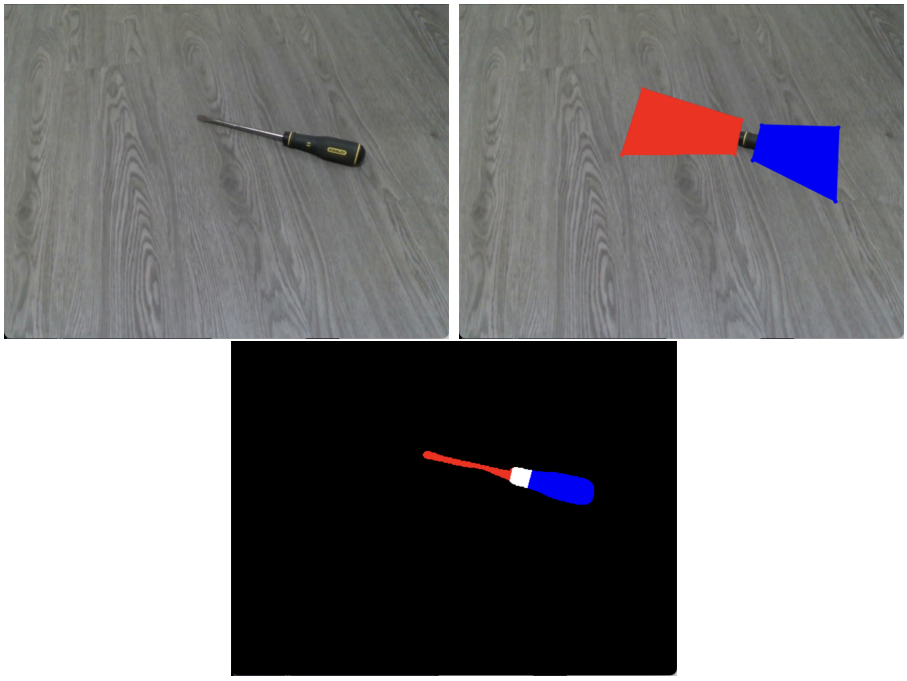


Figure 7.3: The full process of affordance annotation.

Data augmentation

Since the dataset is very small, data augmentation is of great help in avoiding overfitting. Data augmentation involved randomly adjusting the brightness, sharpness, and contrast of the images. Additionally, new images were generated by changing the backgrounds. A script was developed to isolate the object using the object mask, blur the edges of the object, and then place it onto the new background. A sample of augmented images is shown in Figure 7.4.

7.2 AffordanceNet

AffordanceNet performs both object detection and affordance estimation. It takes as input the color image of a cluttered scene and for each object outputs the bounding box, the class label and the affordance mask. Each pixel inside the bounding box is classified with its appropriate affordance label.

The network structure is described in Figure 7.5. A CNN backbone is used to extract features and works jointly with a region proposal network to output a feature map and the regions of interest (bounding boxes) where objects could be located. The *RoIAlign layer* takes the feature of the regions of interest and pools them into a

Figure 7.4: A sample of augmented images.

fixed size. These fixed-size feature maps are then fed to two branches. The first one is an object classification branch made of two fully connected layers, which outputs the object class and regress the object location. The second one is an affordance detection branch made of convolutional layers, which outputs the affordance mask. The loss function is comprised of 3 terms: the output of the classification layer, the output of the regression layer in the object classification branch and the output of the affordance detection branch. The original implementation is built in the Caffe framework, but a TensorFlow implementation is also available [40].

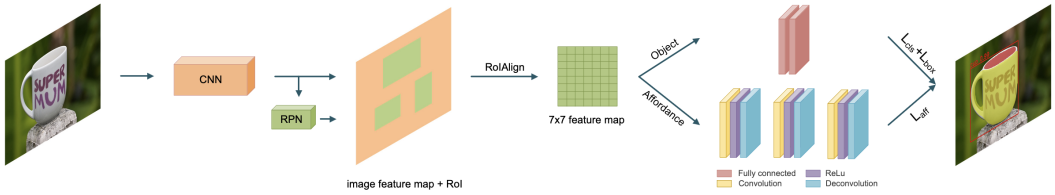


Figure 7.5: AffordanceNet architecture [3].

7.2.1 Training and results

The custom dataset was formatted in the VOC2012 format. The network was slightly modified to work with the correct number of classes and labels. For each image in the dataset, AffordanceNet necessitates an XML file containing information on the bounding box and a serialized affordance map. The XML file is created with the *xml.etree.ElementTree* Python module. Each file contains the object class, the coordinates of the top-left and bottom-right corners of the bounding box, the filename and its dimensions. The serialized affordance mask is simply obtained by converting the affordance mask image as a *numpy* array, and serializing it using the *pickle* package in a binary ".sm" file.

Due to hardware limitations training was not possible.

7.3 YOLO

YOLO stands out as a leading solution for object detection in cluttered scenes. It processes RGB images and provides outputs of bounding boxes and class labels for detected objects. The implementation utilized here is YOLOv8 from Ultralytics [41], which offers models with added functionalities, such as predicting oriented bounding boxes and poses. YOLOv8's architecture mainly comprises convolution layers.

Training sessions were conducted for both the YOLOv8 Base model and the YOLOv8 Pose model. The Pose model not only outputs bounding boxes but also keypoints that represent significant points in the detected objects. In the Pose model, object affordances are encoded as these keypoints. Each keypoint serves as the geometric center for points labeled with specific affordances, as illustrated in Figure 7.1. Given the shapes of the objects in the dataset, this simplified representation of affordances should be enough to provide the robot with sufficient information to perform its task. The keypoints enables the detection of the direction of the hazardous object part and the identification of a safe grasping point.

7.3.1 Training and results

Training and evaluation can be easily performed using the *Ultralytics* Python package. In addition to the augmentations already applied, the package provides a broad range of augmentation options. Particularly noteworthy is the ability to merge multiple images into one, called mosaicing, allowing the model to train for cluttered scenes, even if the original dataset only includes single objects. An example is reported in Figure 7.6.

For each image in the dataset, YOLOv8 necessitates a text file containing the object class label, the position and dimension of the bounding box and keypoints

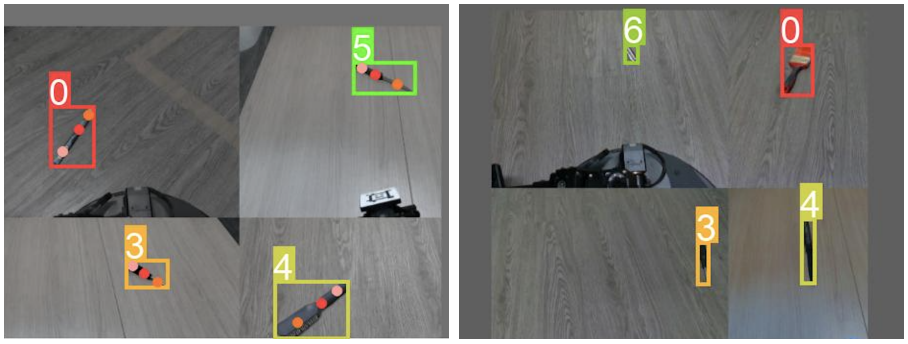


Figure 7.6: An example of mosaicing for YOLOv8 Base and Pose models.

location for the Pose model. The file contains a row for each object in the picture. The metrics used to evaluate the network are:

- Precision and Recall: Precision measures the proportion of true positive predictions among all positive predictions made by the model. Recall measures the proportion of true positive predictions among all actual positive instances in the dataset.
- IoU (Intersection over Union): IoU measures the overlap between the predicted bounding boxes and the ground truth bounding boxes. It is calculated as the ratio of the area of intersection to the area of union between the two bounding boxes. A high IoU score signifies accurate localization of objects.
- F1 Score: F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is often used as a single metric to evaluate model performance.

The training of both models was performed on a train set of 1700 images. The validation set contained 28 images. The hyperparameters used are described in Table 7.1. In both cases, the mosaicing augmentation is turned off in the latest 50 epochs. The evolution of the model losses on the train and validation set is described in Figures 7.7 and 7.8. The Pose model achieves a mean average precision (computed across IoU thresholds from 50 to 95) of 0.98 for pose estimation and 0.95 for bounding box detection. The mean average precision for the Base model is 0.96. Some of the results of both models over the validation set are shown in Figure 7.9

	Pose model	Base model
Batch size	16	16
Epochs	210	250
Image size	640	640
Learning rate	0.01	0.01
Class loss weight	0.5	0.5
Box loss weight	7.5	7.5
Pose loss weight	12.0	0

Table 7.1: YOLOv8 models hyperparameters.

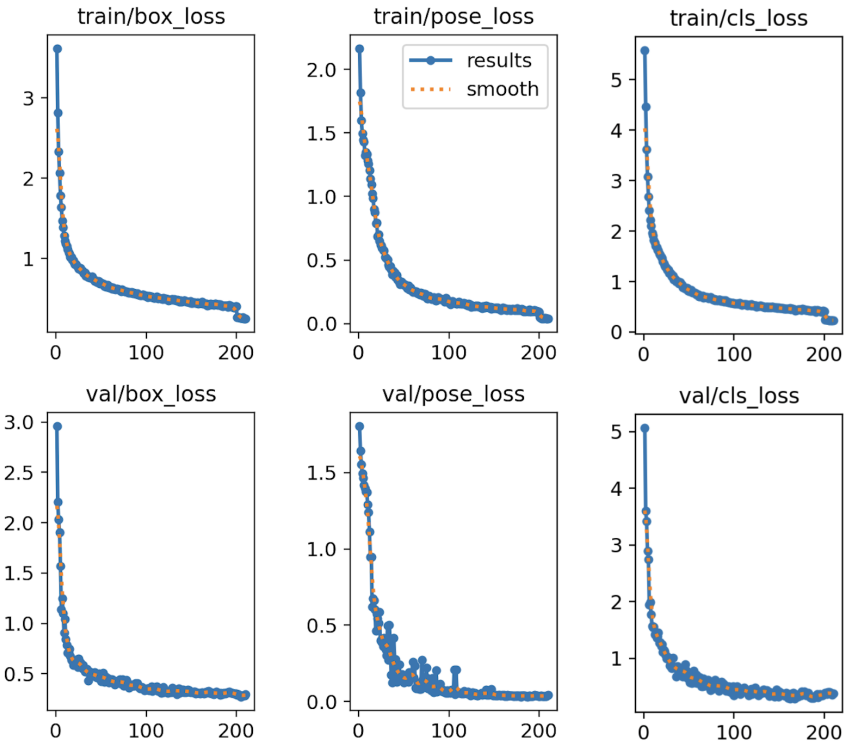


Figure 7.7: Losses in the YOLOv8 Pose model training.

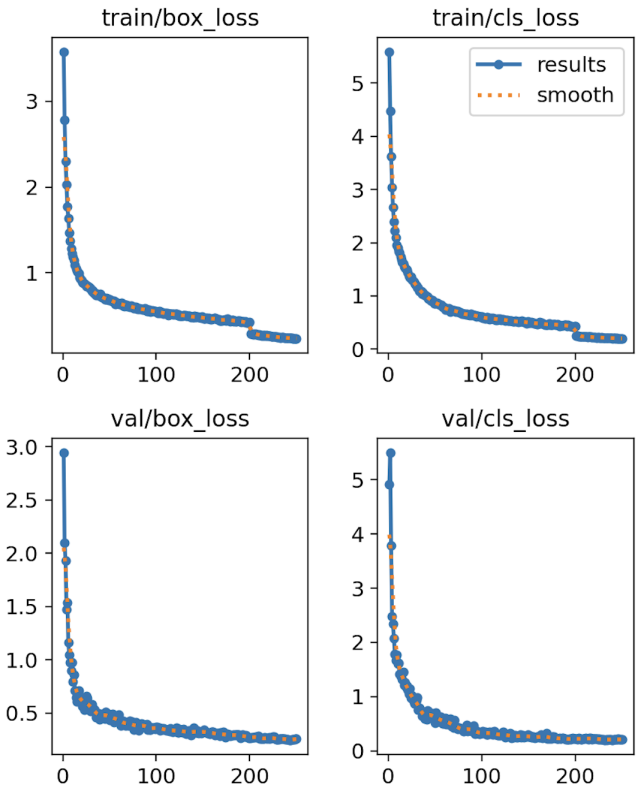


Figure 7.8: Losses in the YOLOv8 Base model training.

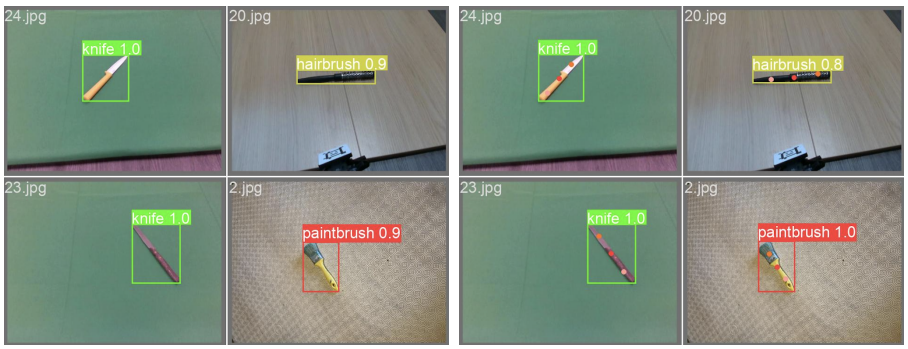


Figure 7.9: YOLOv8 models predictions over the validation images.

7.4 DeepLab

DeepLab is a type of semantic segmentation network designed to analyze images by assigning a class label to each pixel. In its implementation, DeepLabV3Plus utilizes MobileNet as the backbone for feature extraction, which is pre-trained on ImageNet. However, other backbones like ResNet or Xception can also be employed. DeepLab operates as a fully convolutional neural network, leveraging atrous convolution extensively. This technique involves spacing out kernel weights, allowing the network to encompass a wider area of input without inflating the parameter count. As a result, it can capture more context and detail without significantly increasing computational requirements.

7.4.1 Training and results

The training utilizes a DeepLabV3Plus implementation in PyTorch [42]. The model is trained on images from the dataset along with affordance masks cropped around annotated bounding boxes. To diversify the dataset and mitigate overfitting, images and masks in the training set are cropped using randomly varying bounding boxes, achieved by applying random scaling factors to the original bounding boxes. Moreover, the images are scaled to 256x256 pixels. To prevent distortion a black padding is added if needed. An example is shown in Figure 7.10.

Figure 7.10: Cropped images and affordance masks to be used with DeepLabV3Plus.

The main metric used for evaluation is the intersection over the union between the affordance mask and the segmented image. The training was performed on a set of 3200 images (more augmentation was performed) with a validation set of 28

images. The hyperparameters used are described in Table 7.2. The model reaches a main IoU of 0.88. The loss and mean IoU curves are in Figure 7.11

	DeepLabV3PlusMobileNet
Batch size	16
Iterations	15K
Learning rate	0.01
Crop size	256

Table 7.2: Caption

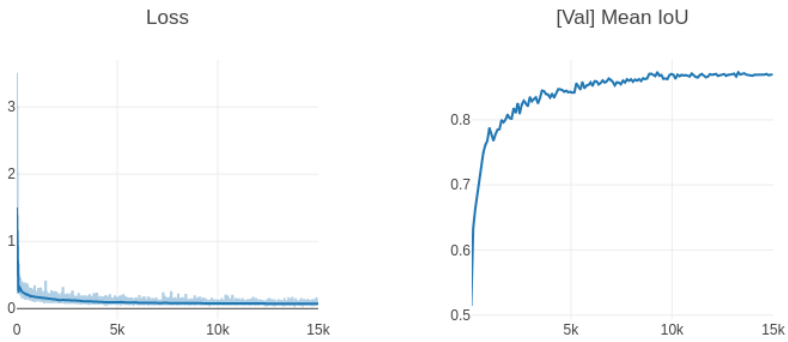


Figure 7.11: Loss and mean IoU during training for DeepLabV3MobileNet.

Chapter 8

Software architecture

The code governing the robot's operations is segmented into various modules. Each module handles specific subtasks, contributing to the overall task. This modular approach enhances scalability and aligns with the design principles of ROS. Not all modules run directly on the robot, enabling the utilization of algorithms beyond its computational limits.

Each module consists of one or more ROS nodes. Communication between them is managed through ROS topics, services and Python interfaces. There are three service modules:

- The **image acquisition module** takes depth and image streams from the camera.
- The **navigation module** builds the map of the robot environments and controls the mobile base making the robot move and avoid obstacles.
- The **manipulation module** controls the robot's arm for manipulation.

There are two main modules:

- The **grasping module** identifies the object and its affordances, calculates the optimal pose for grasping the object, and commands the arm to execute the grasp.
- The **handover module** controls the arm to perform the handover sequence.

The overall structure is sketched in Figure 8.1

The service modules heavily rely on the software packages offered by Trossen Robotics for LoCoBot development. These packages provide fundamental functionalities that are essential for the operation of the grasping and handover modules (main modules).

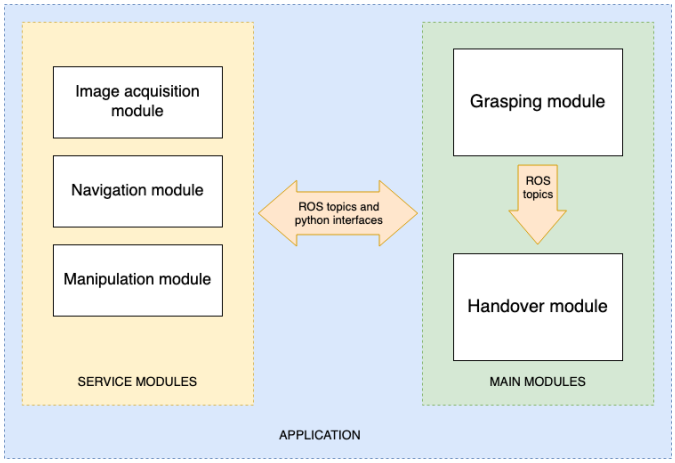


Figure 8.1: The overall software architecture.

8.1 Image acquisition module

The image acquisition module manages the Intel Realsense D435 camera using the Realsense2 ROS package from Intel. It offers configurability, enabling users to set the frame rate and resolution for both the color and depth streams. Additionally, it aligns the field of view of the two streams, ensuring perfect overlap between the depth and color images. It also dynamically publishes intrinsic camera parameters, crucial for constructing 3D projections and point cloud generation. Optionally, it can publish point clouds as well.

Interface

These are the data published by the module:

- **Aligned depth stream** on topic */camera/aligned_depth_to_color/image_raw*.
- **Color stream** on topic */camera/color/image_raw*.
- **Intrinsic parameters** on topic */camera/aligned_depth_to_color/camera_info*.
- **Point cloud** on topic */camera/depth/color/points*.

8.2 Manipulation module

The manipulation module takes care of the robot arm and end-effector movement. Planning and control are done trough MoveIt! framework.

8.2.1 Motion planning

Motion planning is performed by one of the algorithms provided by the OMPL Planner library, which offers a collection of state-of-the-art sampling-based motion planning algorithms. In particular, the RTT algorithm is used. The desired end-effector pose is declared through a *PoseStamped* message containing a quaternion representing the desired orientation and a point representing the desired position. It is also possible to specify the desired state for each joint and plan a path to reach those states. The handover use case requires imposing some path constraints, as discussed in Section 5.1.5. Constraints are declared in a constraints message, which contains four arrays, one for each kind of constraint. Position constraints are represented as a bounding volume delimiting the area where the specified link can be moved. Orientation constraints for the specified link are represented as absolute tolerances (in radians) over the orientation described by a quaternion. To improve the planner performance in the presence of path constraints, the *enforce_joint_model_state_space* option was set to true in the *ompl_planning.yaml* configuration file. This forces the use of the joint space for all plans.

8.2.2 Obstacles and object modeling

The grasped object is depicted as a rectangle with fixed dimensions. After a successful grasp, the object is incorporated into the robot gripper model. This ensures that future planning takes into account the object to prevent collisions with everything except the robot gripper. For simplicity, it is assumed that the robot faces the object without any additional obstacles, so only the floor is modeled. Octomap was also evaluated for dynamic obstacles modeling. Upon spawning the rectangle representing the object, the corresponding area in the map is cleared to enable collisions between the gripper and the object.

Interface

The module functionalities are made available through the MoveIt! Python interface; in particular, the following classe are involveds:

- ***MoveGroupCommander*** allows users to establish the desired end-effector pose or the desired joint states, initiate and halt motion, impose constraints, and define velocity and acceleration scaling factors. Two instances of this class were instantiated, one for controlling the gripper and another for controlling the arm.

- ***PlanningSceneInterface*** allows users to add and remove objects from the planning scene and to attach objects to the robot's joints.

8.3 Navigation module

The navigation module manages the movement of the robot's mobile base within the working environment. It primarily utilizes the *move_base* package [43] for base control and integrates other packages for mapping, localization, obstacle detection, and path planning.

8.3.1 Mapping

For effective navigation, the robot requires a representation of its environment in the form of a map. The map creation process is managed by the SLAM Toolbox. During this process, the robot is manually guided around the room by a human operator using a DualShock4 controller. The SLAM Toolbox integrates LIDAR data from various positions and merges them to construct the map. Real-time updates of the map can be visualized using the RViz interface. The map can be saved either through an RViz plugin or a rosservice call, and it is stored both as a ".pgm" image for use with the *map_server* [44] node and localization methods like AMCL, and as a serialized pose-graph for localization within the SLAM Toolbox itself and potential map refinement.

Localization on the finalized map was also tested using SLAM Toolbox, which matches recent LIDAR scans to the saved pose-graph. A comparison with the standard AMCL algorithm is elaborated in Section 9.1.

8.3.2 Path planning

The *move_base* node handles path planning through its implementation of both local and global planners. Users can customize the planner by modifying the *move_base_params.yaml* configuration file. Other configuration files are available, with options to adjust parameters such as map layer inflation, map update frequency, base recovery behavior, and planning attempts. The map inflation considers the obstruction caused by the object held in the gripper during movement. For the global planner, Dijkstra's algorithm from the *global_planner* [45] package is utilized, while the local planner employs the TEB algorithm from the *teb_local_planner* [46] package. Moreover, the *move_base* node can dynamically manage obstacle avoidance by receiving obstacle position information and updating costmaps accordingly. LIDAR serves as the primary information source for obstacle detection, but it has limitations in detecting objects of certain heights. To address this, depth scans are also utilized to detect small objects in front of the robot camera. This functionality is achieved using three rtabmap nodelets: *rgbd_sync*, *points_xyzrgb*, and *obstacle_detection*. These nodelets filter out the robot model to account for self-occlusion and generate a downsampled point cloud of the robot's

environment, which is then fed to the *move_base* node. The target goal for the mobile base is specified as a *MoveBaseGoal* message which contains a *PoseStamped* message.

Interface

The module functionalities are made available through the *actionlib* Python package. In particular, it offers the following classes:

- ***SimpleActionClient*** interfaces with the *move_base* node and enables to send pose goals.

8.4 Grasping module

The grasping module consists of two nodes: the *grasp_control_node* and the *object_modeling_node*. The *grasp_control_node* is responsible for manoeuvring the robot to the object depot, executing the grasp on the detected object, and transporting it into the carrying position. On the other hand, the *object_modeling_node* handles object detection, affordance estimation, and pose estimation. Together, they incorporate the functionalities outlined in Chapter 6.

8.4.1 Grasp control node

The node waits until it receives a target object from the human, then it proceeds with the following operations:

- The node interfaces with the navigation module to navigate from its starting position to the depot area.

- Upon reaching the depot area, the *object_modeling_node* is activated. If the target object is detected, its affordance is estimated, and its pose is modeled. This information is then sent back to the *grasp_control_node*, which remains idle, awaiting instructions.

- When the object's pose is received, it is converted into a suitable grasp pose and the grasp is executed by interfacing with the manipulation module.

- Once the object is securely held in the carrying position described in Section 6.2, a message is dispatched to the handover module, signaling its readiness.

8.4.2 Object modeling node

The node uses a simplified version of the technique described in Section 6.1.2 to model the object pose. The node uses YOLO v8 Pose to predict the object bounding box together with the affordance keypoints. The keypoints are then projected into 3D space. The object is approximated with a rectangle with fixed dimensions, so to determine the pose it is sufficient to find its center and its rotation. The center is set to the coordinates of the grasp keypoint, while the rotation is described by the vector connecting the grasp keypoint to the danger keypoint.

Interface

The module uses the following topics for external communications:

- */grasp/grasp_status*, to signal a successful grasp to handover module as a *String* message.
- */grasp/grasp_target*, to receive the target object from the user as a *String* message.

In addition, some topics are declared to manage communication internally between the two nodes:

- */grasp/model/object_pose*, to exchange the estimated object pose as a *PoseStamped* message.

8.5 Handover module

The handover module is made of two nodes: the *handover_control* node and the *hand_tracking* node. The first one takes care of moving the robot from the grasp area to the handover area, and performs the handover. The second one takes care of tracking the human hand and detecting hand gestures. They implement the functionalities described in Chapter 5.

8.5.1 Handover control node

This node waits until it receives a message from the grasping module indicating that the grasp was successful, then it proceeds with the following operations:

- The node interfaces with the navigation module to navigate from its current position to the handover area.
- Once the handover area is reached and the motion is completely stopped, the *hand_tracking* node is started and the waiting status to receive the hand position begins.

- When the hand position is received, the manipulation module is tasked to bring the object towards the human exactly as described in Section 5.1.4. Once the final pose is reached, the robot waits 2 seconds, then it starts slowly opening the gripper so that the human can take the object.

8.5.2 Hand tracking node

The node utilizes MediaPipe to track the position and gestures of the human hand and to model its position as described in Section 5.1.3. In order to project the hand keypoints in 3D space, perfect alignment between the color and depth images is essential, along with the availability of camera intrinsic parameters. The image acquisition module provides three separate topics for color and depth images and camera parameters, necessitating synchronization using a *TimeSynchronizer* from the *message_filters* package [47]. Subscribing to the *TimeSynchronizer* triggers a callback each time a depth and a color image message, along with a camera info message sharing the same timestamp, are received. The color images are inputted into the Mediapipe gesture recognition model. Upon detecting the *Open_Palm* gesture, the identified keypoints are used to compute the center of the hand sphere model, represented as a *PointStamped* message. To ensure accurate 3D projection, depth images are clipped at a distance of 2 meters, as the robot cannot extend beyond this range.

8.5.3 Interface

Some topics are declared to manage communication internally between the two nodes:

- */handover/model/hand_pose*, to exchange the hand pose as a *PointStamped* message.

Chapter 9

Experiments

Multiple experiments were designed to assess the functionalities of the architecture described in Chapter 8 within the real-world scenario.

Experiments were carried out in a controlled laboratory setting, featuring a level floor surface, consistent lighting, and static obstacles such as walls and desks. Occasionally, the laboratory environment was augmented by introducing cardboard boxes to simulate additional obstacles.

9.1 Mapping

The mapping capabilities of the navigation module (Section 8.3) were tested. The results of these experiments will be utilized for future experiments involving navigation. A human operator pilots the robot around the room using a DualShock4 controller to evaluate the mapping capabilities of SLAM Toolbox, as described in Section 8.3.1. Only the portion of the laboratory used for the experiments is fully mapped. The resulting map, shown in Figure 9.1, demonstrates satisfactory outcomes, albeit with some noise. SLAM Toolbox localization is tested by placing the robot in random places in the environment. Localization succeeds only if the robot position is next to the map origin or a rough estimate of its initial position is published on the */intialpose* topic and red by SLAM toolbox. Once the initial position is successfully determined the localization goes on without any issue. Figure 9.2 shows an example of successful localization with SLAM Toolbox. To overcome SLAM Toolbox limitation an experiment was conducted on ACML as well. Initially, ACML assumes a uniform distribution of the robot position on the whole map. By moving the robot around, this estimate is continuously refined until the robot's real position is matched as shown in Figure 9.3.

Figure 9.1: The map obtained with SLAM Toolbox.

Figure 9.2: Localization with SLAM Toolbox. The red lines represent the lidar output and match with the map.

9.2 Object detection and affordance estimation

The algorithms for object detection and affordance estimation developed in Chapter 7 were put to the test in a real-world setting. Objects were placed on the laboratory floor, as in the scenario shown in Figure 9.4. The robot was then manoeuvred around to assess how well it could detect objects and estimate their affordances in real time. The algorithms ran on a remote computer leveraging the distributed compute capabilities of ROS, rather than directly on the robot hardware.

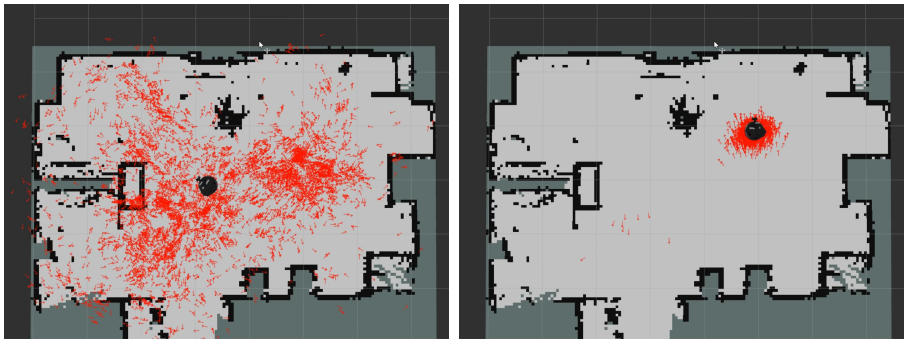


Figure 9.3: Localization with AMCL. The red arrows represent the position distribution, which improves over time.

Figure 9.4: The setup for detection and affordance testing.

YOLOv8 Pose model

The initial experiment utilized the YOLOv8 Pose model. This model predicts bounding boxes for objects along with their class labels and affordance keypoints. In this context, the red keypoint indicates the hazardous area, the yellow keypoint represents the handle, and the green keypoint denotes the part suitable for grasping. Notably, the model demonstrates robust performance even when multiple objects are present within the field of view. The results are depicted in Figure 9.5.

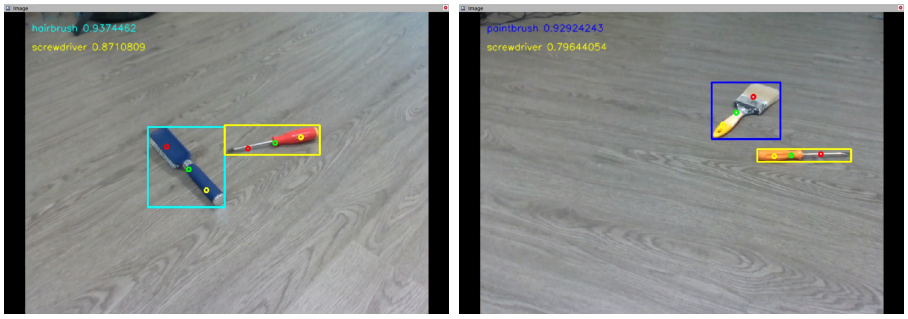


Figure 9.5: The result of YOLOv8 Pose on four different objects.

YOLOv8 plus DeepLabV3Plus

The second experiment employed a combination of the YOLOv8 Base model for object detection and DeepLabV3Plus for affordance segmentation. YOLO analyzes the entire image to generate bounding boxes, while DeepLab processes the cropped image around each predicted bounding box to produce an affordance mask. In the resulting affordance mask, red indicates hazardous areas, yellow corresponds to handle locations, and green represents areas suitable for grasping. Both networks exhibited excellent predictive capabilities. However, utilizing two different networks had a noticeable impact on performance, resulting in slower processing times. Figure 9.6 displays the outputs of the two networks side by side, while Figure 9.7 shows the merged outputs.

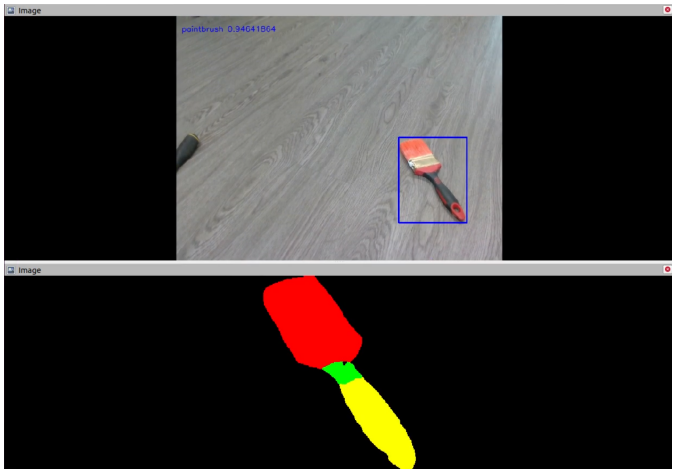


Figure 9.6: Outputs of YOLOv8 Base model and DeepLabV3Plus side by side.

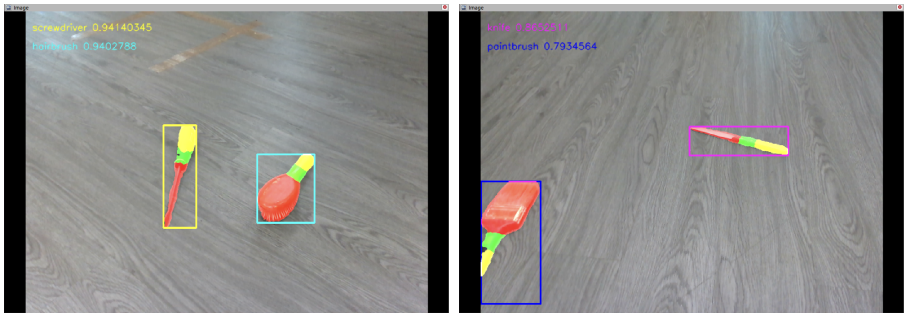


Figure 9.7: Merged outputs of YOLOv8 Base model and DeepLabV3Plus on four different objects.

9.2.1 Considerations on the different solutions

Both solutions have demonstrated excellent performance. The advantages of using YOLOv8 Pose are training a single network and achieving better real time performance. On the other hand, the solution based on YOLOv8 Base model and DeepLabV3Plus offers a more comprehensive representation of affordance and is better suited for analyzing complex objects. However, using two separate networks makes the solution slower during execution and more challenging to train. Nevertheless, employing two separate networks can also have its benefits. Firstly, it allows training the two networks on different datasets. Since object detection is a well-studied problem, using a pre-existing dataset to train YOLOv8 Base can result in a more robust detection. Additionally, this approach is more modular, providing the option to perform affordance estimation only for a subset of manipulable objects. In the upcoming experiments, YOLOv8 Pose will be employed for both detection and affordance estimation due to its speed and simplicity.

9.3 Grasping

The experiments were structured to evaluate the functionalities of the grasping module outlined in Section 8.4 in a real-world setting. Objects were positioned in front of the robot, requiring it to accurately estimate their poses and plan effective grasps, factoring in affordances and handover necessities.

Grasp pose estimation

The technique for estimating object poses and grasp poses outlined in Sections 6.1.3 and 6.1.2 is put to the test. However, rather than executing the entire process of pose estimation and object modeling, a simplified approach is employed, as

implemented in the *object_modeling_node*, Section 8.4.2. This method utilizes the output of YOLOv8 Pose projected into 3D space. The object's pose, and consequently, the grasping pose, are defined by the grasp keypoint (depicted as a green sphere) and the rotation around the world Z-axis of the vector connecting the grasp keypoint (green sphere) to the danger keypoint (red sphere).

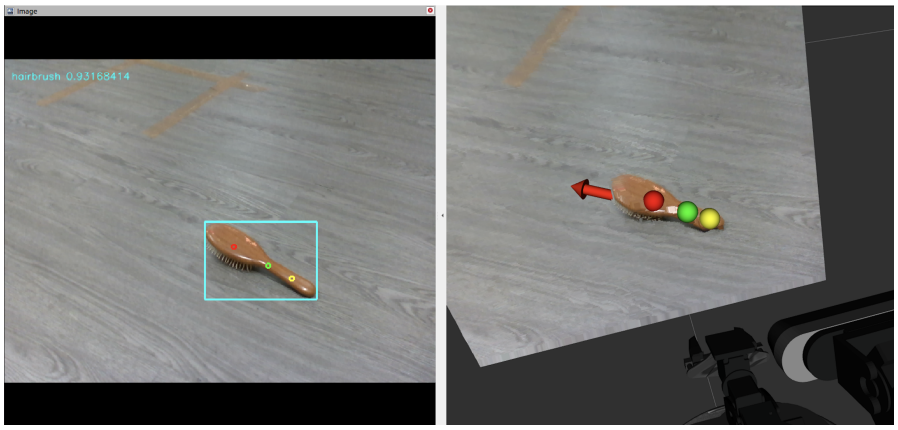


Figure 9.8: YOLOv8 Pose predictions in 3D space (red, green, yellow spheres) and simplified object pose estimation (red arrow) over the scene point cloud.

Stationary grasp

In this experiment, the robot is tasked with several steps: detecting an object, estimating its pose, planning a grasp while considering affordances, and then positioning the object for handover, as implemented in the *grasp_control_node*, Section 8.4.1. As depicted in Figure 9.9, the grasp sequence operates seamlessly. Figure 9.10 illustrates how the grasp pose varies for different target objects with distinct orientations.

9.4 Handover

The experiments aimed to assess the performance of the handover module, Section 8.5, and its integration with the grasping module, Section 8.4, in a real-world environment. The experiments involved the robot selecting an object, navigating to the designated handover point, and transferring the object to a human operator.

Figure 9.9: A real-world grasp.

Figure 9.10: Different gripper orientations for different object poses.

Hand tracking

The functionalities of the *hand_tracking_node*, Section 8.5.2, are tested. The node uses MediaPipe to model the hand in 3D space as a sphere, as described in Section 5.1.3. During testing, a human operator sits in front of the robot, while moving the hand. If the *Open_palm* gesture is detected, the sphere representing the hand moves; otherwise, it remains stationary. The results are shown in Figure 9.11

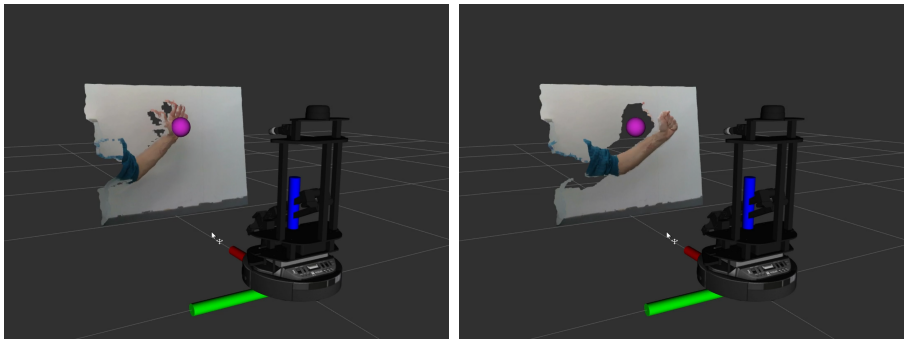


Figure 9.11: Hand tracking and gesture recognition.

Stationary handover

Functionalities of the *handover_control_node*, Section 8.5.1, which involve hand tracking, gesture recognition, and arm motion planning, were put to the test. A human operator sits in front of the robot, while the robot holds an object in the carrying position. Upon detecting an *Open_palm* gesture from the human, the robot initiates the handover process by extending its arm toward the hand, adhering to the constraints and final pose specifications detailed in Sections 5.1.4 and 5.1.5. The handover point is set at 20cm from the center of the human hand. Once the robot reaches its final position, the gripper opens, allowing the human to take the object. The entire process is illustrated in Figure 9.12.

Full handover

All functionalities of the handover module underwent testing. The robot initially begins from a predefined resting position next to the human's desk, holding an object in the carrying position. Its objective is to autonomously navigate through the environment to a predetermined handover location, facing the human. To increase the difficulty level, obstacles are introduced into the scene. Once the robot reaches the designated area and identifies the correct gesture, the handover process proceeds similarly to the stationary scenario. The complete procedure is outlined in Figure 9.13.

Grasp plus handover

The coordination between the handover and grasp modules was evaluated in a static setting. An object is positioned in front of the robot, which must detect it, plan a grasp, and transport it into the carrying position. Subsequently, the robot

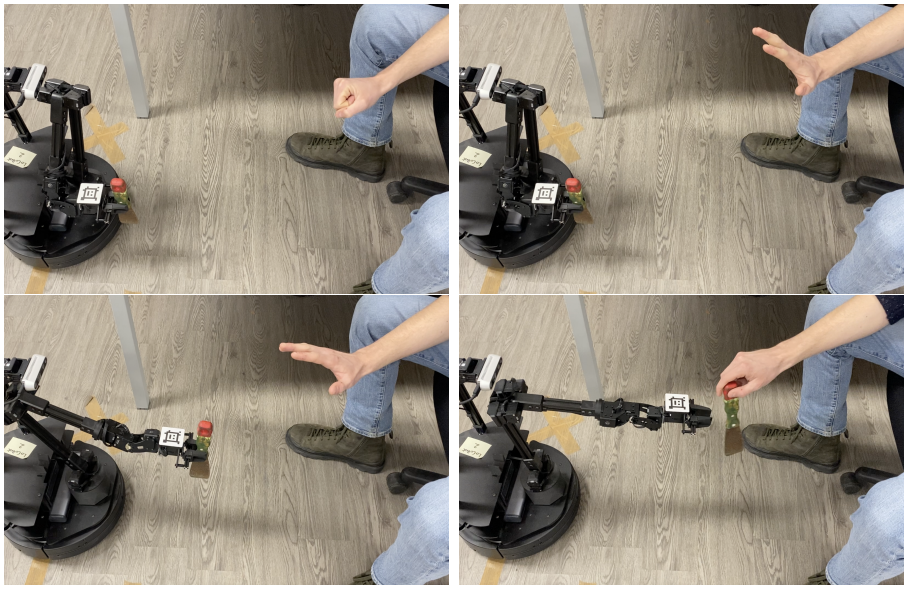


Figure 9.12: A real world handover sequence. The robot waits for the activation gesture, then the object is delivered.

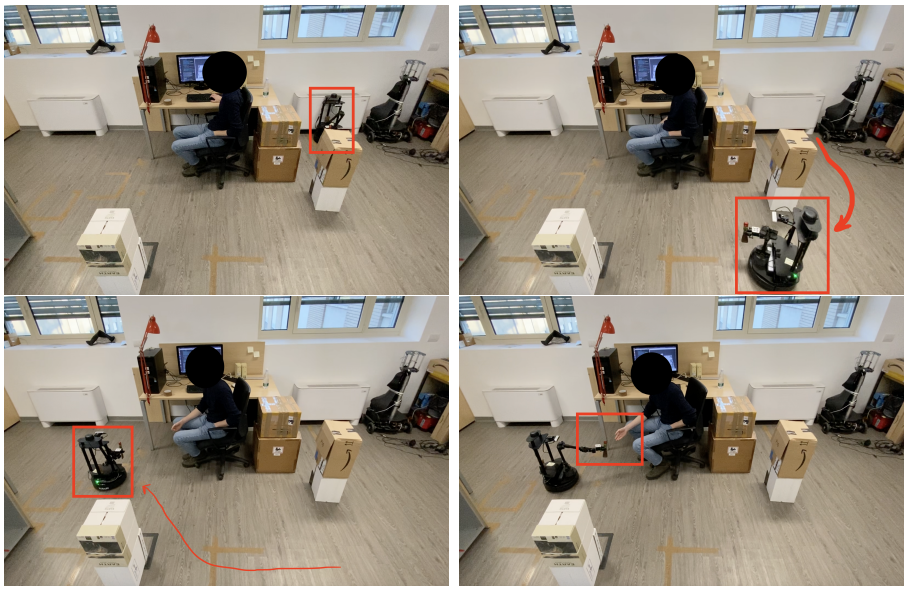


Figure 9.13: The full task of navigating the environment to reach the human and deliver the object.

remains idle until it detects the appropriate gesture from the human, performing the handover process. The complete procedure is shown in Figure 9.14.

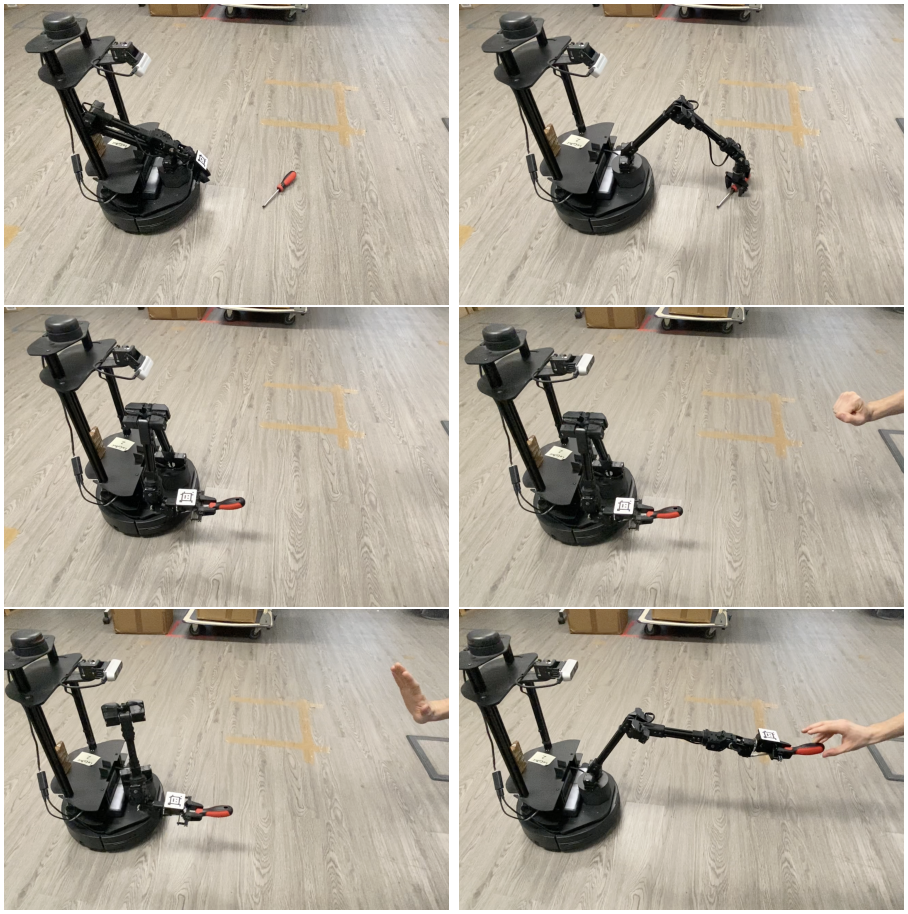


Figure 9.14: A grasp followed by a handover with the robot in a stationary position.

Chapter 10

Conclusions and future works

The aim of the thesis was to explore the topic of object manipulation in the context of the development of robotic applications with humans in collaborative tasks. In particular, the goal was to explore the possibilities offered by recent developments in machine learning algorithms, with the objective of finding safe solutions and achieving human-machine interaction as naturally as possible.

The contribution of this thesis is the implementation of grasping and handover algorithms. These algorithms allow a mobile robot to manipulate novel objects, belonging to a predefined set of object classes, and deliver them to a human in real-world scenarios. Since the end goal was to safely perform the handover, the grasp had to be planned accordingly. That was made possible by machine learning.

Thanks to affordance estimation algorithms, it is possible to obtain crucial information about the object's characteristics and consequently implement a grasp that takes into account its orientation and hazardous parts. In particular, two solutions based on neural networks for affordance estimation were developed, both inspired by the current state of the art. The first is based on the YOLOv8 Pose estimation model, which outputs the object bounding box together with a compact affordance representation based on keypoints. The second solution is based on the joint use of YOLOv8 and DeepLabV3Plus, for object detection and image segmentation respectively, with the aim of obtaining object location alongside a more complete affordance representation based on a segmentation mask.

Subsequently, to train and test the solutions, a custom dataset was developed, containing images and annotations (bounding boxes and affordance masks/keypoints) of instances of manipulable object classes. The dataset contains 7 object classes with 3 affordance labels. Both solutions yielded excellent results and allowed the creation of intelligent grasps that prioritized safety.

Due to the small size of the dataset, it is not possible to provide reliable data on performance. However, tests indicated that the approach is valid and capable of yielding good results.

In the future, expanding the dataset could allow for a more detailed analysis of performance.

Handover planning allowed to consider aspects related to naturalness in interactions between robots and humans. The handover must ensure the safe delivery of the object while also being natural and intuitive for the recipient.

The thesis conceived and implemented, using tools offered by ROS, a sequence of movements that satisfies these requirements and taht is predictable. Once again, machine learning proved extremely useful. The handover algorithm uses the MediaPipe hand-tracking solution to determine the human hand position in real time and compute the best object handover pose relative to the recipient's hand. Additionally, considering the robot's mobile base and its ability to navigate the surrounding environment, handover and grasping were also planned with this robot's capability in mind. The final experiments provide concrete examples, in a real scenario, of the robot's capabilities.

Unfortunately, the hardware of the robot exhibited some limitations that affected overall performance. Firstly, the accuracy of the depth camera makes it difficult to distinguish small variations in depth (between 0.5 cm and 1 cm) at the working distance of 1 meter. This results in an inaccurate representation of objects seen by the robot and makes grasp planning for slender and low objects more difficult. Additionally, the precision of the mobile base and arm is not excellent. The arm-planned movement suffered from errors between 1 cm and 2 cm, while the mobile base was sometimes off from 3 cm to 5 cm.

As further developments are concerned, it would be beneficial, as already mentioned, to improve and expand the dataset used for training affordance estimation and object detection solutions so that the model can generalize better and performance can be measured. It could also be worthwhile to try extending the types of objects manipulable by the robot, transitioning to more complex and intricate objects. In this case, with objects that are more difficult to model manually, it would be interesting to try using generative grasping solutions based on machine learning. More generally, it would be possible to expand the robot's ability to interact with humans, even by making greater use of the mobile base. For example, it would be interesting to expand the possibilities of handover by developing a system for tracking and following people, to autonomously reach the person to assist without any prior knowledge.

Bibliography

[1] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. «The Open Motion Planning Library». In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82. DOI: 10.1109/MRA.2012.2205651 (cit. on p. 3).

[2] Liwei Yang, Ping Li, Song Qian, He Quan, Jinchao Miao, Mengqi Liu, Yanpei Hu, and Erexidin Memetimin. «Path Planning Technique for Mobile Robots: A Review». In: *Machines* 11.10 (2023). ISSN: 2075-1702. DOI: 10.3390/machines11100980. URL: https://www.mdpi.com/2075-1702/11/10/980 (cit. on p. 4).

[3] Thanh-Toan Do, Anh Nguyen, and Ian Reid. *AffordanceNet: An End-to-End Deep Learning Approach for Object Affordance Detection*. 2018. arXiv: 1709.07326 [cs.CV] (cit. on pp. 4, 38).

[4] Anh Nguyen, Dimitrios Kanoulas, Darwin G. Caldwell, and Nikos G. Tsagarakis. «Object-based affordances detection with Convolutional Neural Networks and dense Conditional Random Fields». In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 5908–5915. DOI: 10.1109/IROS.2017.8206484 (cit. on p. 4).

[5] Shengheng Deng, Xun Xu, Chaozheng Wu, Ke Chen, and Kui Jia. *3D AffordanceNet: A Benchmark for Visual Object Affordance Understanding*. 2021. arXiv: 2103.16397 [cs.CV] (cit. on p. 4).

[6] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV] (cit. on p. 4).

[7] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV] (cit. on p. 4).

[8] Enric Corona, Albert Pumarola, Guillem Alenyà, Francesc Moreno-Noguer, and Grégory Rogez. «GanHand: Predicting Human Grasp Affordances in Multi-Object Scenes». In: *2020 IEEE/CVF Conference on Computer Vision*

and Pattern Recognition (CVPR). 2020, pp. 5030–5040. DOI: 10.1109/CVPR42600.2020.00508 (cit. on p. 5).

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (cit. on p. 5).

[10] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV] (cit. on p. 5).

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV] (cit. on p. 5).

[12] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*. 2018. arXiv: 1711.00199 [cs.CV] (cit. on p. 5).

[13] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. *Grasp Pose Detection in Point Clouds*. 2017. arXiv: 1706.09911 [cs.RO] (cit. on p. 6).

[14] Kun Qian, Xingshuo Jing, Yanhui Duan, Bo Zhou, Fang Fang, Jing Xia, and Xudong Ma. «Grasp Pose Detection with Affordance-based Task Constraint Learning in Single-view Point Clouds». In: *Journal of intelligent & robotic systems* 100.1 (2020-10), pp. 145–163. ISSN: 0921-0296 (cit. on p. 6).

[15] Wenkai Chen, Hongzhuo Liang, Zhaopeng Chen, Fuchun Sun, and Jianwei Zhang. «Learning 6-DoF Task-oriented Grasp Detection via Implicit Estimation and Visual Affordance». In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 762–769. DOI: 10.1109/IROS47612.2022.9981900 (cit. on p. 6).

[16] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. «ACRONYM: A Large-Scale Grasp Dataset Based on Simulation». In: *Under Review at ICRA 2021*. 2020 (cit. on p. 6).

[17] Hanbo Zhang, Deyu Yang, Han Wang, Binglei Zhao, Xuguang Lan, Jishiyu Ding, and Nanning Zheng. «REGRAD: A Large-Scale Relational Grasp Dataset for Safe and Object-Specific Robotic Grasping in Clutter». In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2929–2936. DOI: 10.1109/LRA.2022.3142401 (cit. on p. 6).

[18] Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu. «AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains». In: *IEEE Transactions on Robotics (T-RO)* (2023) (cit. on p. 6).

[19] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. «Graspnet-1billion: A large-scale benchmark for general object grasping». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11444–11453 (cit. on p. 6).

[20] Chongxi Meng, Tianwei Zhang, and Tin lun Lam. «Fast and Comfortable Interactive Robot-to-Human Object Handover». In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 3701–3706. DOI: 10.1109/IROS47612.2022.9981484 (cit. on p. 7).

[21] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. *MediaPipe Hands: On-device Real-time Hand Tracking*. 2020. arXiv: 2006.10214 [cs.CV] (cit. on p. 7).

[22] Kerry He, Pradeepsundar Simini, Wesley P. Chan, Dana Kulic, Elizabeth Croft, and Akansel Cosgun. «On-The-Go Robot-to-Human Handovers with a Mobile Manipulator». In: *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, Aug. 2022. DOI: 10.1109/ro-man53752.2022.9900642. URL: http://dx.doi.org/10.1109/RO-MAN53752.2022.9900642 (cit. on p. 7).

[23] Open Robotics. *Robot Operating System*. https://www.ros.org/. [Online; accessed March 2024] (cit. on p. 8).

[24] *tf package*. http://wiki.ros.org/tf. [Online; accessed March 2024] (cit. on p. 11).

[25] *RTAB-Map*. http://introlab.github.io/rtabmap/. [Online; accessed March 2024] (cit. on p. 11).

[26] SteveMacenski. *SLAM Toolbox*. https://github.com/SteveMacenski/slam_toolbox. [Online; accessed March 2024] (cit. on p. 12).

[27] *Navstack*. http://wiki.ros.org/navigation. [Online; accessed March 2024] (cit. on p. 12).

[28] *AMCL*. http://wiki.ros.org/amcl. [Online; accessed March 2024] (cit. on p. 13).

[29] *OctoMap*. https://octomap.github.io/. [Online; accessed March 2024] (cit. on p. 13).

[30] *MoveIt!* https://moveit.ros.org/. [Online; accessed March 2024] (cit. on p. 14).

[31] Meta AI. *PyTorch*. https://pytorch.org/. [Online; accessed March 2024] (cit. on p. 14).

[32] Google. *TensorFlow*. https://www.tensorflow.org/. [Online; accessed March 2024] (cit. on p. 14).

[33] Google. *MediaPipe*. https://developers.google.com/mediapipe. [Online; accessed March 2024] (cit. on p. 15).

[34] Google. *Hand landmarks detection guide*. https://developers.google.com/mediapipe/solutions/vision/hand_landmarker. [Online; accessed March 2024] (cit. on p. 16).

[35] *OpenCV*. https://opencv.org/. [Online; accessed March 2024] (cit. on p. 16).

[36] *Pillow (PIL Fork)*. https://pillow.readthedocs.io/. [Online; accessed March 2024] (cit. on p. 16).

[37] *Open3D*. https://www.open3d.org/. [Online; accessed March 2024] (cit. on p. 17).

[38] Trossen Robotics. *LoCoBot WidowX-250 6-DOF*. https://docs.trossenrobotics.com/interbotix_xslocobots_docs/specifications/locobot_wx250s.html. [Online; accessed March 2024] (cit. on p. 18).

[39] Renato Violin. *Background Removal with Deep Learning)*. https://github.com/renatoviolin/bg-remove-augment. [Online; accessed March 2024] (cit. on p. 36).

[40] Beatriz Pérez. *AffordanceNet TensorFlow*. https://github.com/beapc18/AffordanceNet. [Online; accessed March 2024] (cit. on p. 38).

[41] Ultralytics. *YOLOv8*. https://docs.ultralytics.com/. [Online; accessed March 2024] (cit. on p. 39).

[42] Gongfan Fang. *DeepLabV3Plus-Pytorch*. https://github.com/VainF/DeepLabV3Plus-Pytorch. [Online; accessed March 2024] (cit. on p. 43).

[43] *Move base*. http://wiki.ros.org/move_base. [Online; accessed March 2024] (cit. on p. 48).

[44] *Map server*. http://wiki.ros.org/map_server. [Online; accessed March 2024] (cit. on p. 48).

[45] *Global planner*. http://wiki.ros.org/global_planner. [Online; accessed March 2024] (cit. on p. 48).

[46] *TEB local planner*. http://wiki.ros.org/teb_local_planner. [Online; accessed March 2024] (cit. on p. 48).

[47] *Message filters*. http://wiki.ros.org/message_filters. [Online; accessed March 2024] (cit. on p. 51).