A.D. 1308
unipg
UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Dipartimento di Ingegneria
Corso di Laurea in Ingegneria Informatica e Robotica

Reactive Object Handover for Human-Robot Interaction Applications

Candidata
Maria Pennicchi

Relatore
Prof. Gabriele Costante

Correlatrice
Dott.ssa Maria Pozzi

Correlatrice
Dott.ssa Chiara Castellani

Anno Accademico 2024/2025

A chi mi ha sempre voluto bene,
e non mi ha mai dimenticato,
anche quando la distanza sembrava suggerire il contrario.
Alla certezza
e alla tenacia
che mi stanno insegnando a ricordare.

Contents

Chapter 1

Introduction

The increasing emphasis on human–robot collaboration marks a major transition in robotics. No longer confined to isolated work cells, robots are progressively expected to share space with humans, coordinating their actions in real time to achieve joint goals. This trend is supported by the broader paradigm of Industry 4.0, which envisions flexible environments where humans and robots operate side by side as complementary agents. A central operation in this context is object handover - the transfer of an item from a human to a robot (H2R) or vice versa (R2H). Such exchanges play a pivotal role in collaborative tasks because they allow both partners to manipulate the same object at different stages of a process. At the same time, handovers are inherently delicate: they involve close proximity, continuous adaptation, and a high degree of mutual awareness. In industrial applications, carefully structured setups can partially mitigate these difficulties. However, in unstructured domains such as domestic or healthcare environments, variability in human motion, object positioning, and timing makes the task significantly harder.

The potential benefits of robust handover mechanisms are considerable. In everyday life, robots capable of smoothly exchanging objects with humans could assist elderly individuals or people with limited mobility, for example by retrieving and delivering household items. This type of assistance not only supports the successful completion of daily tasks but also enhances the independence of individuals who might otherwise struggle to perform them on their own. By enabling seamless object exchanges in everyday contexts, handover capabilities allow robots to be integrated as supportive and practical partners within domestic environments, thereby easing daily routines. Far from being a purely technical milestone, natural and efficient handover thus represents a gateway for robots to become practical and trustworthy assistants in human-centered environments.

Designing a human–robot handover that approaches the fluidity and efficiency of human–human exchanges remains a demanding challenge. Existing solutions have made important progress, yet many still fall short of the flexibility required for natural collaboration. Human interactions are inherently dynamic: partners frequently adjust their position, trajectory, or grip as the exchange unfolds. To replicate this adaptability,

robotic systems must integrate continuous perception, motion coordination, and safety mechanisms, ensuring that the interaction remains smooth and responsive. Without such responsiveness, the robot risks disrupting the collaborative flow, thereby reducing both efficiency and the user's sense of trust.

To tackle these limitations and enable more natural and adaptive handovers, this thesis transforms a previously validated pre-existing ROS-based handover pipeline into a reactive system. The proposed design continuously monitors the human hand and dynamically adapts the robot's trajectory whenever a displacement is detected. Moreover, a dedicated module was developed to simulate occlusions caused by the robot's own arm within the camera feed, allowing realistic testing despite all experiments being conducted in simulation. The effectiveness of this approach was evaluated under various experimental conditions, including fluid tracking motions and sequences of static poses with rapid changes, both with and without simulated occlusions. The reactive system was compared against a baseline strategy in which the robot returns to its initial pose before replanning. The results demonstrate that the reactive pipeline consistently achieves higher success rates and greater adaptability under varying temporal and perceptual constraints. These findings confirm the importance of reactivity as a design principle in collaborative robotic systems and highlight the potential of perception-driven control in realistic handover scenarios.

This thesis is structured as follows. Chapter 2 presents the related works, providing an overview of existing research on object handover in collaborative robotics. Both Human-to-Robot and Robot-to-Human handovers are discussed, with particular attention to the perception, grasping, and motion planning modules that enable these interactions. Previous studies on reactive strategies are also reviewed, and the pre-existing ROS-based pipeline that served as the starting point for this work is described in detail. Chapter 3 introduces the proposed algorithm to implement the reactive handover. The system is first presented in its non-occluded form, highlighting the mechanisms that enable continuous perception and dynamic replanning, and is then extended to account for occlusions. This chapter also describes the experimental setup and the calibration procedure carried out to determine the optimal value of the reactive threshold parameter. In Chapter 4, a baseline algorithm is outlined in both its non-occluded and occluded versions. The chapter then reports on the three experiments designed to compare the baseline against the proposed approach to validate its effectiveness. These experiments are analyzed in terms of performance, robustness, and adaptability, allowing a clear assessment of the advantages provided by the reactive system. Lastly, Chapter 5 concludes the thesis by summarizing the main contributions and findings. The limitations of the proposed work are discussed, and several directions for future research are proposed based on these observations.

Chapter 2

Related Works

In recent years, there has been a clear shift toward more direct human–robot collaboration. The current momentum of Industry 4.0 supports the idea of fully shared spaces, where robots interact not only with their environment, but also with other agents such as human workers and robots [1, 2]. This vision is made possible by significant advancements in robotic hardware.

While traditional industrial environments, with their structured and predictable setups, have long supported the effective use of robots in dedicated work cells, the same level of integration has not yet been achieved in less structured settings, for example in factories without defined work cells, as well as domestic or healthcare environments. To operate effectively in such dynamic contexts, robots must possess a deeper understanding of the tasks at hand, advanced perception capabilities to recognize and track environmental changes, as well as smart and adaptable motion planning systems that can respond to those changes [3].

Figure 2.1: Human-Robot handover with a Franka Emika Panda [4].

2.1 Object Handover

Within the field of collaborative robotics, one of the key operations is the object handover, which consists in passing an object from the human's hand to the robot (Human-to-Robot Handover, H2R), or the opposite (Robot-to-Human Handover, R2H), as shown in Fig. 2.2. This exchange is fundamental, as it enables both agents to handle the object

and carry out specific parts of a task. However, it also presents safety challenges due to the close physical proximity and interaction required during the handover process.

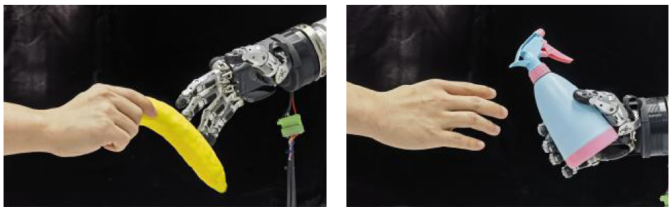


Figure 2.2: Human-Robot object handover [5].

Object handover is a key capability for enabling effective robotic assistance in households or domestic environments. In particular, robots equipped with object handover functionality can provide valuable support to elderly individuals or those with limited mobility by retrieving and delivering frequently used items. [6] In collaborative household tasks, such as cooking or crafting, robots can assist by supplying the necessary tools or materials and subsequently collecting them when no longer needed. This form of assistance not only supports task completion but also promotes greater autonomy for individuals who may face difficulties in performing these actions independently. By facilitating these everyday interactions, object handover enables the integration of robots as helpful and practical assistants in domestic settings, making daily life more manageable. [7] This functionality moves robots beyond simple automation towards becoming true collaborators in the home.

The implementation of a human-robot handover that is as efficient and fluent as the exchanges among humans is an open challenge for the robotics community. Achieving this level of natural interaction requires addressing multiple aspects, including perception, motion coordination, and safety, as well as selecting an appropriate gripper for the task and maintaining continuous situational awareness throughout the interaction. In what follows, we review the main results achieved so far in both Robot-to-Human and Human-to-Robot handover, highlighting the key strategies, experimental findings, and technological advancements that have shaped the current state of the art.

2.1.1 Robot-to-Human Handover

Robot-to-Human handover has been widely studied in recent years, with numerous studies, which seek to identify the features that enhance user comfort during handover interactions and contribute to making the robot appear more reliable and trustworthy.

An instance is given by Grigore et al., in 2013, who explored how incorporating insights from human-human interaction into human-robot collaboration can enhance the effectiveness of robot-to-human handovers. The authors propose a two-layered framework: the first layer addresses the physical dimension of the handover by estimating its current state - categorized as “robot picking up”, “robot holding the object”, “human

grabbing the object", and "robot not holding the object". The second layer models human behaviors typically observed during handovers, such as gaze direction and body posture, and integrates them into the robot's planning process. Experimental results demonstrate that this method improves handover success rates and increases the user's perception of the robot as a trustworthy partner [8].

Subsequently, research shifted toward systems that are more capable of adapting to human behavior. In 2016, Medina et al. used a motion-capture data glove with force-sensitive resistors to monitor human intentions during handovers. They then used the collected data to develop a controller inspired by human dynamics to enable smoother handovers and estimated the most likely handover location based on human movement [9]. Techniques such as learning from human demonstrations and reinforcement learning began to be explored to further improve fluidity and efficiency.

Furthermore, the importance of social aspects and human perception in interaction continues to be explored, together with a data-driven approach, following the current trend in this field. For instance, Khanna et al. analyzed variations in grip force during human handovers with the goal of transferring this skill to collaborative robotic systems. The study involved human handovers in which pairs of participants exchanged a sensor-equipped baton to measure interaction and grip forces. The collected data were used to train a Long Short-Term Memory (LSTM) neural network to learn the relationship between the perceived interaction force-torque and the resulting change in the human donor's grip force. The LSTM architecture was designed to predict the future trend of the donor's grip force based on a time series of interaction force data. This approach enables robotic givers to modulate their grip force in a manner similar to humans during robot-to-human handovers, making the interaction feel natural, intuitive, and preferable for the human user. This research stands out from previous studies that primarily focused on vertical load sharing for grip force modulation [10].

In conclusion, Robot-to-Human handovers typically result in fewer failures compared to Human-to-Robot handovers, mainly due to factors related to control and the predictability of actions. In an R2H scenario, the robot acts as the active agent that releases the object to the human. This allows for more precise control over the object's trajectory and release force. Failures in R2H scenarios - such as object drops or planning timeouts - can often be avoided or mitigated through human intervention, as humans naturally adapt to imprecise object deliveries. In contrast, during H2R handovers, the robot must perceive the human's intention and adjust its grasp and trajectory according to the object being offered, which may have unpredictable orientations or be partially occluded. Learning generalizable H2R handover skills is therefore considered more complex than autonomous robotic control, due to the need to interpret and respond to human behavior and autonomously recover from errors.

2.1.2 Human-to-Robot Handover

The Human-to-Robot handover is a fundamental capability for achieving seamless and effective human-robot collaboration. Although the process may appear intuitive, implementing a system that is efficient, safe, and perceived as natural by the user presents a significant challenge. This complexity arises from the need to integrate various robotic competencies across two main phases: the pre-handover phase and the physical handover phase [11].

During the pre-handover phase, the robot must prepare to receive the object. This involves several coordinated sub-tasks. First, it must detect the human's intention to initiate the handover. This intention can be communicated through various channels, including gestures, static body posture, or more subtle signals captured through wearable sensors that monitor muscle activity. Once the human's intention has been identified, the robot must detect both the human hand and the object being offered and estimate their position and orientation. A key challenge in this phase of perception lies in the robot's ability to handle unknown objects and accommodate the variability in how humans present them. Based on the object's estimated pose, the robot must also plan its grasp. This requires selecting a suitable grasp configuration that ensures stability, object and human safety, as well as human comfort. Grasp planning algorithms must be robust and capable of generalizing across different object types. Finally, the robot must plan its motion toward the object using smooth, safe, and predictable trajectories. These movements play a critical role in how safe and comfortable the interaction feels to the user. During this phase, the robot must also detect changes in the object or hand's pose and consequently dynamically update its grasp and motion planning, effectively enabling a reactive handover.

The physical handover phase begins when the robot makes contact with the object and ends when the object is securely held and the human has fully released it. In this phase, managing physical interaction becomes crucial. When using a parallel-jaw gripper the robot must avoid contact with the human, while using an anthropomorphic hand the contact between human and robot can be used to facilitate the handover, sliding the robot hand onto the human's one. In general, the robot must regulate its grip force carefully to avoid harming the object or the human's hand, often relying on force and torque sensors to do so. It is also essential for the robot to detect the exact moment when the human releases the object so it can finalize its grasp and safely move away. Furthermore, the system should be able to identify and respond to unexpected events, such as object slippage or unanticipated human movements, but dealing with failure is still an open challenge in this field.

Overall, the development of effective H2R handover systems requires careful attention to perception, grasp, motion planning and control. Equally important is the goal of making the interaction as natural, intuitive and safe as possible for the human.

The most relevant aspects will now be discussed in greater detail, drawing on insights

from previous studies and related literature.

End-Effectors

When discussing robotic handovers, one of the fundamental aspects to consider is the type of gripper employed by the robot.

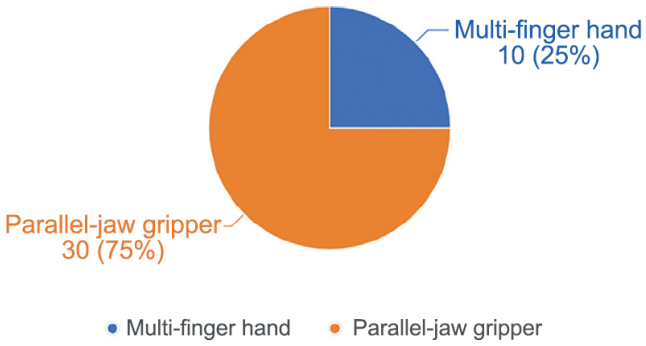


Figure 2.3: Robot end-effector distribution in Human-to–Robot object handover. Data taken from the review [5].

Parallel Jaw Gripper In most of the literature, the end-effector of choice is the parallel-jaw gripper, as shown in Fig. 2.3. This type of gripper, shown in Fig. 2.4, is widely used in Human-to-Robot handover research due to its simple mechanical design and low degrees of freedom, which allow for reliable control in various manipulation tasks [5]. However, its mechanical simplicity also limits its ability to perform more complex operations and to establish stable grasps on irregularly shaped objects - particularly during handovers, where the graspable area may be offset or non-central.

Figure 2.4: Parallel-jaw gripper from Franka Robotics [12].

Anthropomorphic hand gripper In contrast, anthropomorphic robotic hands, which are inspired by the structure and function of the human hand, offer significantly higher degrees of freedom. This enables them to execute more natural and versatile grasps, particularly on everyday objects designed for human use. A notable example is the HIT-DLR II Dexterous Hand 2.5, employed by Duan et al. [13], in conjunction with the

Anthropomorphic Hand Grasp Network (AHG-Net). This neural network predicts suitable grasp configurations for anthropomorphic hands by leveraging five different grasp taxonomies, thereby enabling robots to receive objects in a natural and intuitive way. Such approaches more closely mimic human interaction, allow for more stable grasps across a variety of objects, remove constraints on human handover poses, and facilitate the robot's subsequent manipulation of the object.

Figure 2.5: HIT-DLR II Dexterous Hand performing an handover [13].

Soft Hand Gripper Soft robotic hands, in general, provide inherent adaptability to the shape of the object and ensure safer physical interaction with humans due to their passive compliance. Their mechanical properties make them adaptable, versatile, and robust. In fact, part of the "intelligence" required for grasping can be embedded directly into the hardware design, reducing the burden on planning and control algorithms. As a result, conventional grasp planning strategies - often based on precise finger placements to ensure force closure - must be rethought or extended.

The soft robotic hand used in this thesis is the Pisa/IIT SoftHand, developed by Catalano et al. [14]. This device features 19 joints across four fingers and an opposable thumb (Fig. 2.5), but operates with a single actuator, based on the principle of adaptive synergies - a concept inspired by hand synergies studied in neuroscience. The mechanism is designed so that a single motor command produces coordinated, human-like finger movements that resemble common grasping patterns.

Figure 2.6: Skeleton of the Pisa/IIT SoftHand [14].

The Pisa/IIT SoftHand (Fig. 2.7) has also been used by Pozzi et al. [15], who introduced the concept of a "closure signature" (CS) - a model that describes the hand's

closure behavior and associates it with a preferred grasping direction. Unlike kinematic-based models, the CS captures the functional behavior of the hand, making it especially suitable for soft hands. Thanks to the hand's compliant nature, it is possible to implement simplified grasp planning strategies based on the CS, while still handling uncertainties effectively and safely.

In some implementations, such as that described by Bianchi et al. [16], the SoftHand is equipped with inertial measurement units (IMUs) placed on the fingertips and the back of the hand. These sensors can detect accelerations caused by contact with external objects and can be used to implement tactile-based grasping primitives, suitable for human-robot handover tasks. Upon detecting contact, predefined sequences of arm and hand movements are triggered to complete the grasp.

In conclusion, the compliant and flexible nature of soft hands makes them especially well-suited for physical interaction with humans. Their ability to adapt to contact with the human hand during handover - rather than avoiding it - enhances both the robustness of the grasp and the user experience, ultimately leading to more natural and reliable interactions, as highlighted in [17].

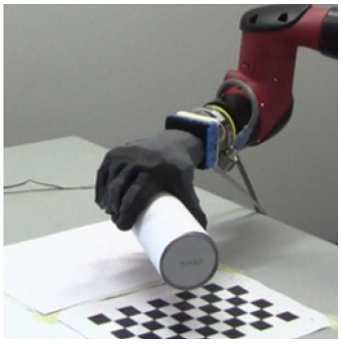


Figure 2.7: Pisa/IIT SoftHand grasping an object [15].

Robotic Arm As for robotic arms, it is commonly observed that while many systems adopt simple grippers, researchers tend to prefer redundant-degree-of-freedom arms, such as the Franka Emika Panda, shown in Fig. 2.8. This choice is likely influenced by the fact that much of the research focuses on motion planning. Due to its user-friendliness and the availability of extensive open-source resources, the Franka Emika Panda is one of the most widely used robotic arms in the research community. It has been employed in handover studies such as those by [7] and [18], and it will also be used in the experimental work of this thesis.

Perception Module

A key component in Human-to-Robot handover is the robot's perception module, which is responsible for providing the robot with the necessary information to understand its surrounding environment. In particular, this module enables the accurate identification

Figure 2.8: Franka Emika Panda with a parallel-jaw gripper as end-effector [19].

of the position of the human hands and the objects to be grasped. This information is essential for planning safe trajectories, generating stable grasps, and ensuring a smooth and collaborative interaction. In the literature, various sensing techniques and processing algorithms are employed within the perception module to achieve these objectives. In the following sections, we will present the most commonly adopted approaches.

Vision-Based Perception Traditionally, vision has been one of the most widely used sensory modalities for perception in robotic handover tasks [11]. Vision-based systems employ one or more cameras, as shown in Fig.2.9, to capture images of the scene and use image processing and deep learning algorithms to extract relevant information about the human hand pose and the objects to be grasped.

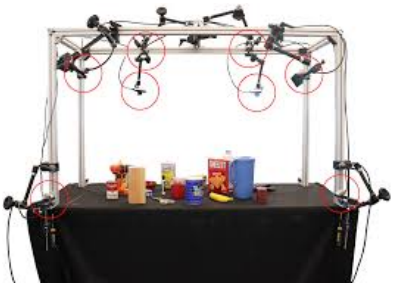


Figure 2.9: Setup with multiple cameras to collect data on human grasping objects [20].

Most approaches rely on RGB-D cameras (Fig.2.10), which acquire both RGB color images and depth information via infrared sensing. The depth data is represented as point clouds, which encode the 3D structure of the environment, as shown in Fig.2.11. This 3D representation proves particularly useful for estimating the distance to objects and reconstructing their shape more accurately than standard RGB images. Various algorithms are applied to RGB-D data to identify and localize the elements involved in the Human-to-Robot handover.

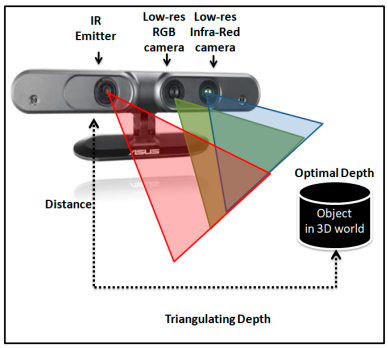


Figure 2.10: An example of RGB-D camera [21].

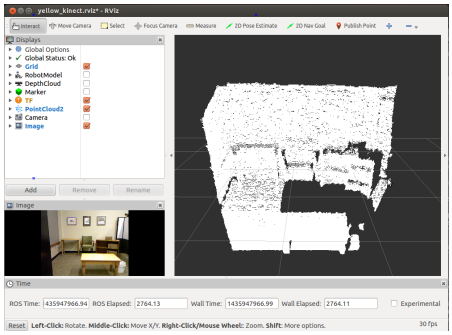


Figure 2.11: A point cloud on Rviz representing a room full of furniture [22].

Instance Segmentation To detect the object to be handed over, segmenting both the object and the human hand is essential in order to isolate the relevant components of the scene. Semantic segmentation involves partitioning an image into regions by assigning a class label to each pixel. However, in cases where it is necessary to differentiate between multiple instances of the same object class - as is typical in handover tasks - the process is referred to as instance segmentation, as shown in Fig.2.12. Semantic and instance segmentation algorithms are commonly used to identify which pixels in the image correspond to the human hand and to the object being transferred.

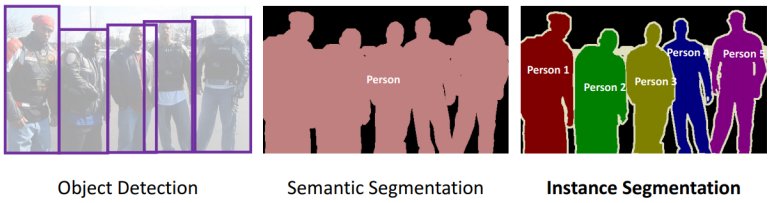


Figure 2.12: Difference between 'object detection', 'semantic segmentation' and 'instance segmentation' [23].

One example is the Feature Pyramid Network (FPN) used by Duan et al., which takes an RGB image as input and applies a fully convolutional network based on a ResNet-50 backbone pretrained on ImageNet [24, 25]. This is followed by a sequence of convolutional and upsampling layers to recover the original image resolution. The output

is a binary segmentation mask that classifies each pixel as either hand or background, effectively isolating the hand region.

Another commonly used approach is to leverage the YOLOv8 architecture for instance segmentation. YOLO (You Only Look Once) is a family of real-time object detection models that, starting from YOLOv5-seg and more comprehensively with YOLOv8, has integrated support for segmentation tasks. In its latest version, YOLOv8 [26] natively supports the generation of precise segmentation masks, enabling it to delineate object boundaries clearly within an image, as shown in Fig.2.13. This allows for the detection and segmentation of a wide range of object types beyond just the hand.

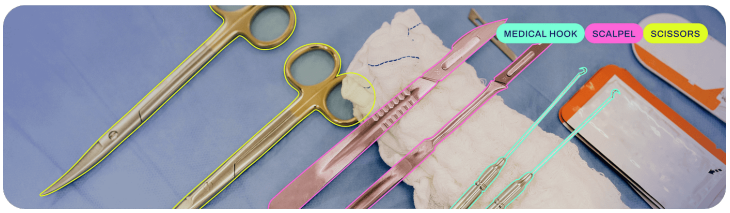


Figure 2.13: Istance segmentation of some medical tools using YOLOv8 [27].

In scenarios where the object is partially occluded by the human hand, directly segmenting the object may prove difficult. In such cases, segmenting the hand and then inferring the presence of the object using depth information has proven to be more effective [7]. Accurate segmentation is critical to prevent the robot from attempting to grasp the user's hand or generating invalid grasp configurations on the object.

Instance Segmentation 3D Another viable strategy for detecting the pose of both the hand and the object is applying instance segmentation directly to the point cloud. A notable deep learning architecture that significantly advanced 3D data processing is PointNet [28]. The key idea behind PointNet is to directly operate on raw point cloud data in a way that is invariant to the ordering of the points. The network's capacity to extract both local and global features has demonstrated its effectiveness in handling 3D input. PointNet has achieved state-of-the-art performance in both classification and segmentation of 3D data.

PointNet++ [29] improves upon this architecture by recursively applying PointNet to small neighborhoods within the point cloud. This enhancement addresses the limitation of the original PointNet in capturing local structures dictated by spatial metrics, which otherwise led to a loss of geometric relationships among neighboring points. PointNet++ successfully recovers these relationships and improves segmentation quality in complex 3D environments.

Hand Keypoint Another viable approach involves detecting the keypoints of the human hand, which can subsequently be integrated with depth information to localize the object being held. Deep learning techniques are commonly employed for keypoint

detection, enabling the robotic system to estimate the pose of the human hand and, potentially, anticipate its future movements.

For instance, the MediaPipe Hands Full model relies on convolutional neural networks (CNNs) and uses RGB images as input. This architecture is designed to predict, in real time, both the 2D positions of hand keypoints on the RGB image and their 3D coordinates relative to the geometric center of the hand - without relying on depth data, as shown in Fig.2.14. Although some methods in the literature exploit depth information to estimate the 3D locations of hand joints, such approaches often suffer from sensitivity to occlusions. In contrast, MediaPipe, by focusing solely on RGB image analysis for hand pose estimation, has demonstrated high accuracy and robustness in close-range human-robot interaction tasks [30, 17]. Other frameworks such as OpenPose, MMPose, and AlphaPose have also been employed for detecting keypoints of the hand and full body. However, they often show reduced accuracy in finger localization and generally perform worse than MediaPipe in the context of fine-grained hand pose estimation [31].

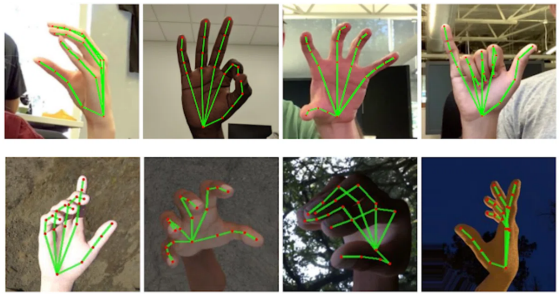


Figure 2.14: Hand keypoints detection on various subjects and various poses using MediaPipe [30].

The transformation of detected 2D keypoints into 3D coordinates is typically achieved by integrating depth data from RGB-D cameras. This approach is also employed in our proposed system. Our perception pipeline, shown in Fig.2.15 is based on a single RGB-D camera. The system first uses the MediaPipe Full Hands model to detect 2D hand keypoints and estimate their 3D coordinates relative to the hand's center, using only the RGB data [17, 30]. Then, the 3D global position of the hand is computed by projecting the midpoint between the third and fourth knuckle from the 2D image into 3D space using the depth information provided by the RGB-D camera. The transformation is performed using the following equations:

$$x_h =\sim \frac{(u_h - c_x) \cdot Z_h}{f_x}, \quad y_h = \frac{(v_h - c_y) \cdot Z_h}{f_y}, \quad z_h = Z_h \tag{2.1}$$

Next, the system estimates the orientation of the hand by fitting a plane to a subset of points belonging to the palm. This is feasible due to the nature of the handover, which assumes an open-palm configuration. The normal vector to the estimated plane is then used as the hand orientation. To localize the object, a cubic region is defined

around the estimated hand position. The cube's vertices are reprojected onto the RGB image to identify a corresponding 2D subregion. YOLOv8 is then applied within this subregion to perform instance segmentation, isolating only the object being held and avoiding false detections of background elements. Finally, the segmented point cloud is projected onto the estimated hand plane, and the Open3D library is used to apply Principal Component Analysis (PCA) on the convex hull of the projected points. This results in an oriented bounding box for the target object [26, 32]. Notably, one face of this bounding box always lies on the hand plane, ensuring alignment with the hand's pose.

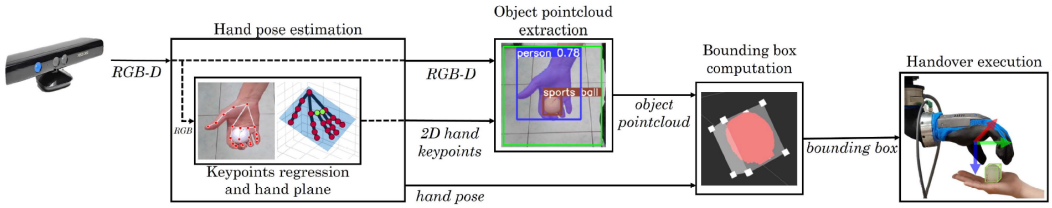


Figure 2.15: Our pipeline of perception used to estimate hand and object position and orientation [17].

Object Variety A final, critical aspect of perception in Human-to-Robot handover tasks is the ability to recognize and handle a broad variety of objects. In real-world environments, such as domestic settings, robots must interact with objects of highly diverse shapes, sizes, appearances, and deformability-including objects they have never encountered before. This variability requires perception systems to be highly generalizable. Evaluating these systems on a wide array of objects allows for more comprehensive and realistic assessments of their performance.

Yang et al. address this challenge by proposing a vision-based system capable of handling the handover of previously unseen objects from humans to robots [7]. Their approach combines closed-loop motion planning with real-time, temporally consistent grasp generation. To overcome the limitations of object detectors trained on fixed datasets, they use a segmentation module to dynamically identify both the human hand and the object. They also extend 6-DOF GraspNet with temporal refinement to ensure the generated grasps are both collision-free and consistent over time, based on the segmented point clouds of the hand and object. This work demonstrates how a careful integration of advanced visual perception and reactive planning can enable robots to effectively interact with a wide range of unknown objects, shown in Fig.2.16-representing a crucial step toward truly general-purpose robotic perception systems.

Perception of Successful Object Grasp The perception module plays a crucial role not only prior to the grasp execution but also afterward, to verify whether the grasp has been successfully completed. Visual feedback alone may not always suffice

Figure 2.16: Unknown object used in a Human-to-Robot handover in Yang et al. work [7].

for this purpose, particularly due to frequent occlusions that may occur during the grasping phase. Fortunately, in addition to vision, tactile sensors are emerging as a valuable sensory modality in robotic handover scenarios. These sensors can provide critical information regarding physical contact between the human hand, the object, and the robotic end-effector - complementing visual data, especially in situations where occlusions hinder visual perception.

For example, in systems equipped with parallel-jaw grippers, tactile sensors mounted on the end-effector can provide key insights into the success of a grasp, as shown by Costanzo et al. [33]. Their sensors measure contact forces and the 6D contact wrench. By detecting a stable contact force within a defined threshold, the system can infer that the object has been securely grasped. Furthermore, by monitoring tangential and torsional forces, it becomes possible to detect early signs of slippage. If these forces exceed a specific threshold or indicate object motion within the grasp, the system can recognize grasp failure and initiate corrective actions.

Similarly, robotic hands can be equipped with tactile sensors on the fingers and palm to detect contact with the object and estimate grasp forces. These sensors are essential for executing stable grasps and for modulating the applied force according to the object's physical properties. For instance, the Pisa/IIT SoftHand primarily in Bianchi et al. work relies on Inertial Measurement Units (IMUs) as grasp sensors [16]. These IMUs are positioned on the back of the fingertips of the underactuated robotic hand, as shown in Fig.2.17, and are used to detect accelerations caused by contact with external objects during interaction.

Failure Prevention The perception module also plays a critical role in predicting potential grasp failures and enabling timely corrective strategies. Averta et al. propose an approach aimed at predicting grasp failure in a soft robotic hand and executing compensatory actions to prevent object loss [34]. In their method, inertial sensors (IMUs) embedded in a glove worn by the soft robotic hand measure the hand's acceleration and angular velocity during the grasping phase. The IMU data is fed into a Recurrent Neural Network (RNN), specifically a Gated Recurrent Unit (GRU) architecture, which is trained to recognize motion patterns that indicate object slippage and an impending grasp failure. If the network predicts a high likelihood of failure, a second robotic

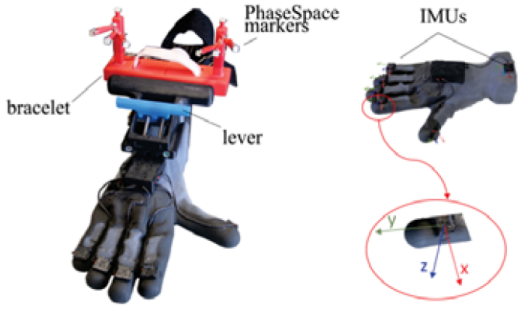


Figure 2.17: Glove with IMUs on the back of the fingertips used to cover the Pisa/IIT SoftHand in Bianchi et al. article [16].

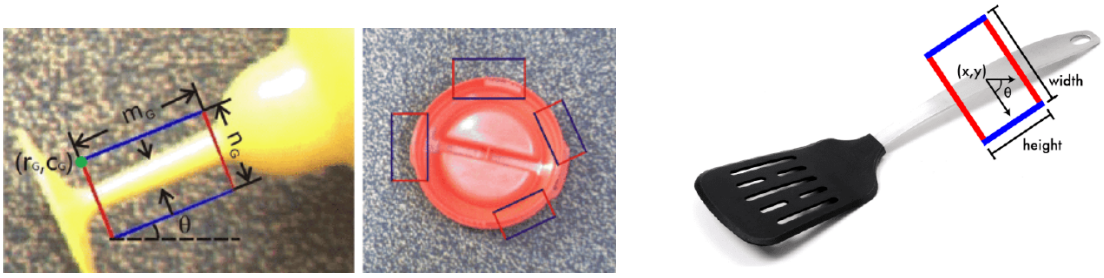
hand is activated to perform a regrasp action, preventing the object from falling. This early detection capability allows the robotic system to respond proactively, significantly enhancing the reliability and success rate of handover tasks.

Grasp Detection Module

Once the poses of the human hand and the object have been estimated, the next step involves determining the most appropriate grasp to execute, using a grasp detection module. Some systems adopt a simplified approach by consistently applying a fixed grasp strategy, however, to enable more natural and robust handovers, the use of grasp detection techniques becomes necessary. These methods aim to predict a set of feasible grasps for a given object in a specific scene, taking into account both the object's characteristics and the manner in which it is held by the human hand.

The input to these methods typically consists of RGB-D images or partial point clouds. Based on this input, various grasp pose representations are employed. When using a parallel-jaw gripper as the end-effector, a commonly adopted representation is the 7-DoF object grasp model [35, 36]. This can be visualized as a rectangle projected onto the image plane, describing the pose that the manipulator arm must assume in order to grasp the object, as shown in Fig.2.18a. It is defined by a 5-tuple (r_g, c_g, n_g, m_g, θ_g), where r_g and c_g denote the coordinates of the top-left corner of the rectangle, n_g and m_g represent the rectangle's dimensions, and θ_g indicates the angle between the rectangle's bottom edge and the x-axis of the image plane. The remaining two parameters describe the depth z_g of the rectangle's center from the camera and the gripper width w for parallel-jaw grasping. These seven parameters provide a complete representation of the grasp pose. From these, both the translation vector t and the orientation θ with respect to the camera reference frame can be derived, which are required by the robotic arm to execute the grasp. Alternatively, the grasp pose can be simplified to a 5-DoF representation, as shown in Fig.2.18b, by constraining the gripper to approach the object orthogonally to the image plane, even though this reduces the variety of possible grasps [37]. The required parameters are (x, y, $width$, $height$, θ),

where x and y are the coordinates of the rectangle's center, *width* and *height* correspond to the gripper's dimensions (and thus its aperture), and θ represents the orientation with respect to the camera frame. To overcome the limitations imposed by this fixed approach vector, grasp poses can also be directly predicted in the partial point cloud and represented in the SE(3) space, as proposed by Pas et al. [38].



(a) 7 DoF (b) 5 DoF

Figure 2.18: Representation of grasp pose using 7 DoF in Fig.(a) and 5 DoF in Fig.(b) [36, 37].

Grasp detection approaches can be broadly categorized into two main groups: analytic (also referred to as geometric) and data-driven (or empirical) methods [39]. Analytic techniques determine grasp configurations by analyzing the kinematic and dynamic properties of the hand-object interaction, as well as the physical attributes of the object, such as its geometry and motion dynamics. These methods typically rely on mathematical models and physical laws to ensure grasp stability. In contrast, data-driven methods employ deep learning algorithms to infer appropriate gripper poses from large datasets. These approaches are particularly advantageous in scenarios where precise object models are either unavailable or challenging to acquire.

Analytic Methods Analytic methods typically rely on mathematical models and principles from mechanics to determine effective grasp strategies [39]. These approaches -commonly employed in conjunction with parallel-jaw grippers-aim to identify a set of potential contact points on the object and to find hand configurations that align the gripper's contact surfaces with these desired points. Often, this involves solving an optimization problem guided by criteria that quantify grasp quality, such as dexterity, force balance, stability, and dynamic behavior. A key assumption underlying many analytic methods is the availability of complete geometric information about the object. This requirement can significantly limit their applicability in real-world scenarios, where objects may be unknown or only partially observed.

One fundamental concept in this category is force closure, which evaluates whether the contact forces exerted by the gripper are sufficient to resist any external disturbance, thereby ensuring that the object cannot move or slip from the grasp. Achieving force closure typically involves analyzing grasp closure properties and the associated grasp

wrench. For example, by analyzing a simple 2D object such as a rectangle grasped by frictionless fingers, one can systematically explore combinations of contact points along the rectangle's edges to identify configurations that satisfy the force closure condition [40]. For each valid grasp, various quality metrics - based on contact geometry and force analysis - can be computed, enabling the selection of the most favorable grasp configuration.

In our system, a top-down grasp from an open palm is always selected [17]. The robot moves its end-effector above the human hand holding the object, then lowers it until contact is made, sliding along the hand to close around the object, as shown in Fig.2.19. This approach allows the method to generalize across different objects, eliminating the need for precise knowledge of the object's characteristics, while avoiding the use of data-driven methods and thus reducing the dependence on large datasets. Although this strategy simplifies the planning process, it simultaneously imposes constraints that may limit the naturalness and flexibility of the handover interaction, as it restricts the range of object orientations the human can use when passing the object.

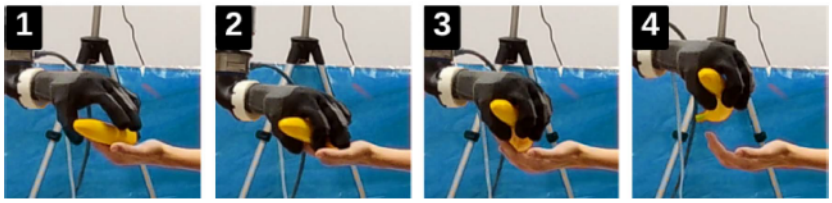


Figure 2.19: Grasp using a soft hand with a top-down motion on an open palm, used in our method [17].

Data-Driven Methods Data-driven approaches are particularly well suited for object-agnostic handover tasks, where the system cannot rely on pre-existing models of the objects involved. Learning-based methods, in particular, employ deep learning architectures trained on large-scale datasets to predict grasp configurations, allowing for generalization to previously unseen objects.

Learning-based techniques can be broadly divided into two categories: pipeline methods, which decouple the grasp generation phase from the motion planning phase, and end-to-end methods, which directly map visual inputs to grasp actions [35]. Pipeline methods typically output either a score indicating the robustness or success probability of grasp candidates, or a structured output that directly identifies a valid grasp pose. In contrast, end-to-end approaches learn control policies directly from image data.

A representative example of a pipeline-based method that leverages point cloud representations is Grasp Pose Detection (GPD) [38]. The process begins with a denoising stage to preprocess the raw point cloud. A large number of grasp candidates are then sampled from a specified region of interest. These candidate grasps are geometrically filtered to ensure that the open gripper will not cause collisions and that the closing

region contains sufficient points from the cloud. A convolutional neural network, trained on labeled grasp pose datasets - often using force closure metrics - is employed to evaluate the grasp candidates. Each candidate is assigned a confidence score that reflects the likelihood of a successful and stable grasp. The highest-scoring, feasible candidates are then selected, discarding those that are incompatible with the robot's kinematics, gripper aperture constraints, or that may cause collisions.

A more advanced method frequently applied in human-to-robot handover scenarios is 6-DoF GraspNet, proposed by Mousavian et al. This method employs a two-stage pipeline: a generative phase for sampling potential grasp poses, followed by a discriminative phase that both evaluates and refines the sampled poses, as shown in Fig.2.20. The first component, the Grasp Sampler, is implemented using a Variational Autoencoder (VAE), which generates a diverse set of 6-DoF grasp candidates. The second component, the Grasp Evaluator, takes both the sampled poses and the point cloud as input. It assigns a quality score to each candidate based on the predicted stability of the grasp and refines grasp poses to improve their accuracy. Both the generator and evaluator modules utilize PointNet++ and consist of three set abstraction layers followed by fully connected layers [41, 29].

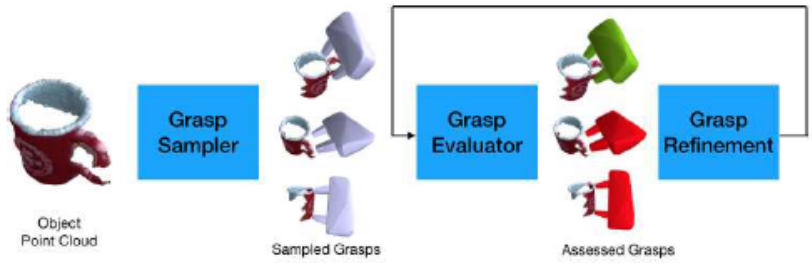


Figure 2.20: Schematic representation of the Grasp Sampler and the Grasp Evaluator interaction to train GraspNet [41].

It is worth noting that many learning-based grasp models are trained on static object datasets, often with objects placed on flat surfaces. This presents a limitation when transferring these models to more dynamic environments, such as human-to-robot handover tasks.

To explicitly address handover scenarios, Yang et al. propose a model that formulates grasp prediction as a human grasp classification problem [42]. Unlike generative methods that predict grasp poses, this approach aims to recognize the type of human grasp being used during the object transfer. The authors define a discrete set of common grasp categories - such as "on-open-palm", "pinch-bottom", "pinch-top", "pinch-side", "lifting", as well as two empty-hand states: "waiting" and "others". To perform this classification, a deep neural network takes as input a point cloud centered around the detected human hand. The network, inspired by PointNet++ [29], includes four set abstraction layers for local feature extraction and a three-layer multilayer perceptron

with batch normalization, ReLU activation, and dropout for global feature learning and final grasp classification. The model is trained on a dataset specifically collected for this work to represent various human handover grasps. Based on the predicted human grasp category, the system adapts the orientation of the robot's gripper accordingly, allowing it to retrieve the object in a more natural and fluid manner. This adaptation reduces the likelihood of physical interference with the human hand, thereby enhancing the safety and intuitiveness of the interaction.

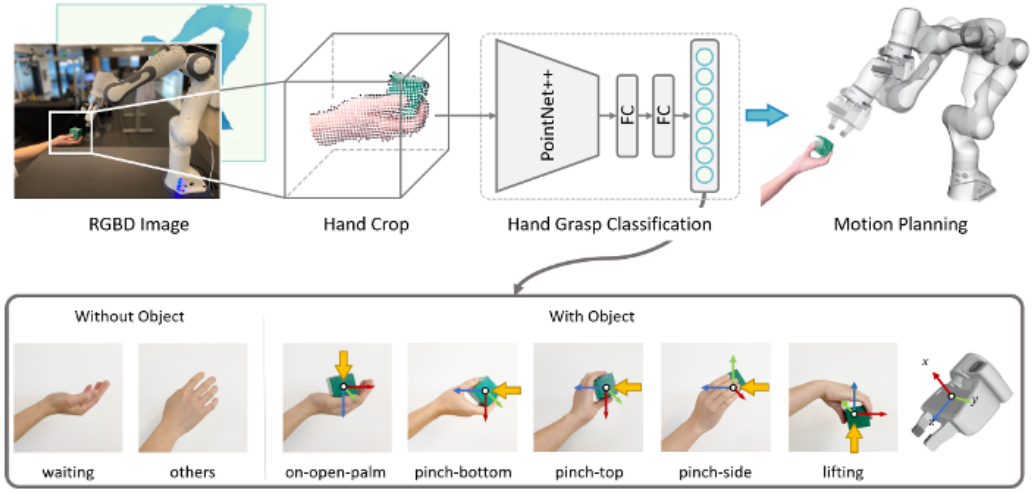


Figure 2.21: Framework to obtain a classification of human grasps between 7 categories from a RGB-D image [42].

A method specifically developed for generating grasp configurations in Human-to-Robot handover scenarios using an anthropomorphic robotic hand is the one proposed by Duan et al. [24]. This approach is based on a deep neural network named Anthropomorphic Hand Grasp-Net (AHG-Net), which takes as input a single-view point cloud of the object captured by an RGB-D camera. The network's objective is to predict dense, stable, and reliable grasp candidates associated with different grasp taxonomies that categorize hand configurations based on finger usage. The architecture of the network is built upon a Transformer framework, employing Fast Point Transformer (FPT) and PointNet++ as its backbone. Additionally, it is enhanced by a novel point cloud representation called Rep-Surf. The network includes a classification component that determines the graspability of each point with respect to various grasp taxonomies, using a cross-entropy loss function [43, 29, 44]. Beyond classification, the network also performs regression on several key parameters necessary for defining a grasp with an anthropomorphic hand, including approach distance, orientation, and joint angles of the fingers. The total loss function used for training combines terms for grasp classification, hand pose estimation, and joint angle prediction. The network is trained exclusively on grasp data collected within the HandoverSim simulation environment [45] and later tested in real-world settings using a sim-to-real approach. This strategy enables effective

model training without requiring real-world data collection. After dense grasp candidates are generated by the network, a grasp selection mechanism identifies the most appropriate configuration for execution, taking into account factors such as predicted grasp confidence, reachability, and tolerance. It is worth noting that the quality of the predicted grasp can be influenced by the way the object is presented by the human operator, including occlusions and visibility constraints.

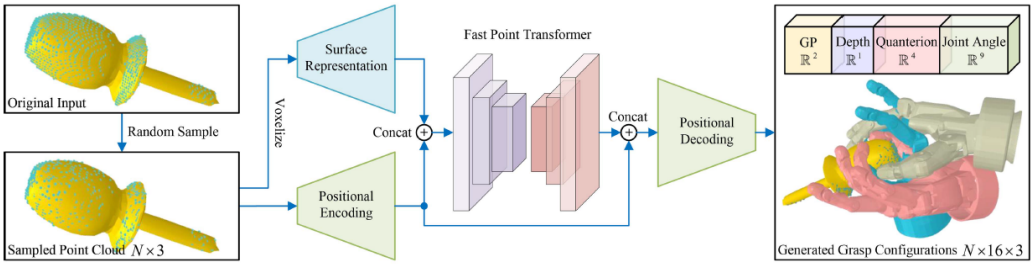


Figure 2.22: Pipeline to obtain a grasp detection strategy using a anthropomorphic hand as end-effector [24].

Motion Planning

Compared to the relatively underdeveloped state of grasp strategy research, the field of motion planning is currently experiencing significant growth. Specifically, advancements in motion planning aim to make robots safer, smoother, and more proactive in object handover tasks. Studies in this area are conducted either using real robots or within simulated environments. We categorize the research conducted on real robots into three main types: learning-based, control-based, and analytic approaches.

Learning-Based Motion Planning At present, learning-based methods can be further divided into supervised learning (SL) and reinforcement learning (RL) approaches.

Some studies employ temporal modeling to train custom datasets that generate robot arm trajectories using RGB-D images as input. For instance, Zhang et al. propose a neural network based on the Transformer architecture to track the robot's approach pose in real time during handover tasks [46]. By leveraging the temporal history of previous frames, the network produces smooth and stable grasping trajectories, avoiding erratic movements that can result from frame-by-frame tracking. This dynamic trajectory generation in real time is presented as a key innovation for achieving flexible handovers, enabling the robot to grasp moving objects successfully, even when the object follows complex and continuous motion paths. This method enhances the robot's ability to adapt to dynamic handover environments and generalize across various object types.

In contrast, Christen et al. focus on learning vision-based control policies specifically for Human-to-Robot handover. They apply the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, an actor–critic type reinforcement learning method [47, 48].

The policy network receives input from a wrist-mounted camera that captures segmented point clouds of the human hand and the object. This input is processed into an embedding, which serves as the state input to the policy network. The network then predicts actions that define relative 6-DoF transformations of the robot's end-effector. The TD3 policy is trained in a three-stage framework, illustrated in Fig.2.23. During the pretraining phase, conducted with a stationary human hand, both imitation learning and reinforcement learning are used, alternating between planned trajectories and exploration data. In the second phase, training is performed using randomly sampled transitions from a replay buffer, which are processed by PointNet++, as well as the actor and critic networks. Finally, in the fine-tuning phase, both the human and the robot move simultaneously. Here, the expert motion planner is replaced by an expert policy that shares weights with the pretrained model. This expert policy is kept frozen and functions as a regularizer for the RL agent. Meanwhile, the agent's actor and critic networks are initialized from the pretrained models and are further updated during fine-tuning.

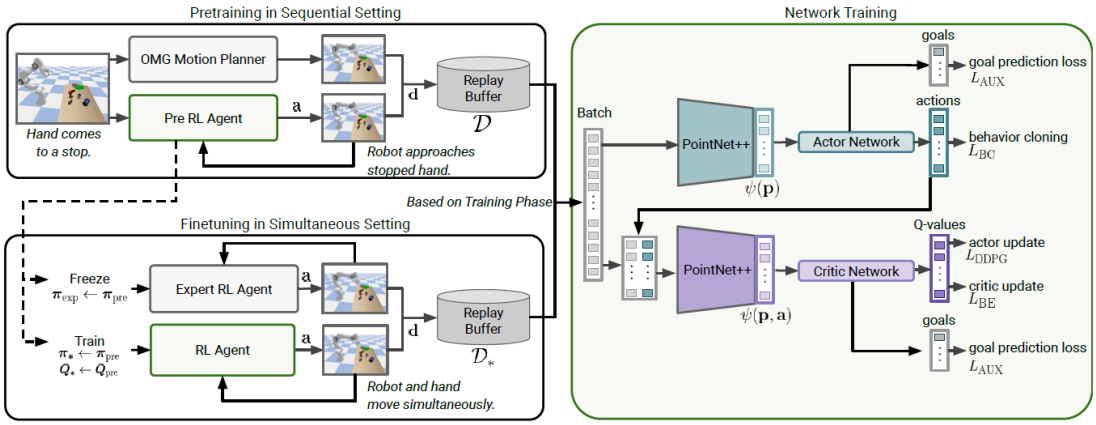


Figure 2.23: Training stages as described in [47]: the pretraining phase is shown in the top-left box, the training phase in the green box on the right, and the fine-tuning phase in the bottom-left box.

Control-Based Motion Planning Other studies opt for control-based methods to address the object handover task between humans and robots.

For instance, in the context of using PID controllers, Neranon and Sutiphotinun combine PID control with impedance control to achieve a Robot-to-Human handover [49]. In this approach, the PID controller ensures that the robot reaches the desired position with stability, while impedance control manages the physical interaction with the environment or human partner by regulating the robot's applied force in response to human hand forces. This is especially critical during the transfer phase of the object, allowing for safe and natural physical collaboration with the human hand.

The work of Yang et al. goes a step further by employing Model Predictive Control (MPC) to implement a responsive Human-to-Robot handover [50]. MPC is used to gener-

ate smooth, human-friendly motion by planning optimal trajectories over a time horizon. It considers the current state of the robot and integrates complex constraints into the cost function, such as collision avoidance and task-specific goals like jerk minimization. This contributes to a more comfortable and predictable human–robot interaction and improves the success rate of handovers.

Analysis-Based Motion Planning Among analysis-based motion planning methods, Dynamic Movement Primitives (DMPs) [51] are widely adopted to enhance robotic performance in human–robot object handover scenarios. DMPs allow for the generation of flexible movement trajectories from a small number of human demonstrations and can adapt to dynamic changes in the target's position without requiring complete trajectory replanning. Compared to real-time optimization methods, DMP-based approaches reduce online computational overhead and enable more efficient, smooth, and reactive robot motions, ultimately improving interaction quality and user comfort.

For example, Wu et al. apply DMPs and their extensions to control robotic arm motion [52]. Specifically, they introduce uncertainty-aware learning via Gaussian Processes for the nonlinear forcing term, which reduces the need for manual parameter tuning. Their framework also includes a weighting function to regulate the transition between the shape (feedforward) and goal attraction (feedback) terms, orientation-based spatial scaling, and online parameter adaptation based on human feedback. Due to their flexibility and adaptability, DMPs are often the preferred solution in scenarios where learning-based methods are difficult to implement.

2.2 Reactive Handover

The Human-to-Robot handover process can be categorized into stationary handovers and reactive handovers, depending on the mode of interaction. A stationary handover refers to a simplified scenario in which both the human partner and the object remain static. This approach eliminates the need to adapt to complex environments or detect human hand movements, thereby allowing for straightforward grasp execution via open-loop control. In contrast, a reactive handover is designed to initially perceive human intent and adaptively respond to environmental changes, making it more aligned with realistic expectations and practical conditions.

For instance, Yang et al. developed a grasp classification framework specifically for reactive Human-to-Robot handover [42]. Their system utilizes a task execution approach based on Robust Logical-Dynamical Systems, which allows for continuous identification of the current logical state and reactive re-planning in response to uncertainty and state transitions, such as human hand displacement or changes in grasp.

Anthropomorphic Hand Unlike parallel grippers, there are relatively few studies on reactive handovers involving anthropomorphic hands, and this area remains underex-

plored due to several inherent challenges. First, anthropomorphic hands tend to occlude a significant portion of the object's point cloud, leading to increased visual obstruction and hindering hand tracking during motion. Second, these hands have a larger collision volume, increasing the risk of contact with the human hand or surrounding environment. Finally, being more dexterous, anthropomorphic hands have a higher number of controllable joints, which complicates motion planning and control [53, 54]. Despite these challenges, anthropomorphic hands offer the potential for more natural and intuitive human–robot interaction, making this direction a valuable area for further investigation.

One example of a reactive handover system is the work by Yang et al., where they adopt a modified version of GraspNet as the grasp planner, chosen for its model-free nature and high efficiency [41, 7]. To enable reactivity, they implement a **temporally consistent grasping strategy**, since the original GraspNet is designed for static objects. In real handover scenarios, however, the human hand typically moves slightly, making static grasp generation inadequate. Simply regenerating new grasps at each time step does not guarantee temporal consistency and can lead to unstable robot behavior. To address this issue, the authors propose a temporal grasp refinement approach. Instead of computing grasps from scratch at every frame, they refine the previous set using Metropolis-Hastings sampling, applying small perturbations to each grasp. These perturbed grasps are evaluated and accepted with a probability proportional to their predicted quality, which helps the system escape local minima while improving grasp stability. However, when the human hand moves too quickly, the object's point cloud may change significantly, and most perturbed grasps may become invalid. In such cases, if the number of valid grasps drops below a threshold, the system resamples entirely to ensure the motion planner always has sufficient candidates. This method maintains smooth and reactive robot motions, even under dynamic interaction conditions.

Duan et al. implement a reactive Human-to-Robot handover using an anthropomorphic hand within a closed-loop framework [24]. The system is structured around four main sub-modules: input, track, grasp, and respond, as illustrated in Fig. 2.24.

The input module collects RGB and depth images from an RGB-D camera (specifically, an Azure Kinect) and simultaneously utilizes the Azure Kinect Body Tracking SDK to extract human body key points, particularly those corresponding to the palm joint. In addition, this module incorporates a human segmentation process using YOLACT to extract a mask of the human body from the RGB image and map it to the depth image, thereby obtaining the corresponding point cloud. This step helps isolate the relevant interaction and extract the point cloud of the object to be handed over. Initially, the **tracking module** is activated to handle the pre-handover phase, allowing the robot to follow the motion of the human hand before attempting the grasp. The objective is to approach the object as closely as possible without occluding the space between the hand and the object. Since the system relies on a single fixed camera, any occlusion during the handover could increase the risk of failure and reduce the safety of the human user. The

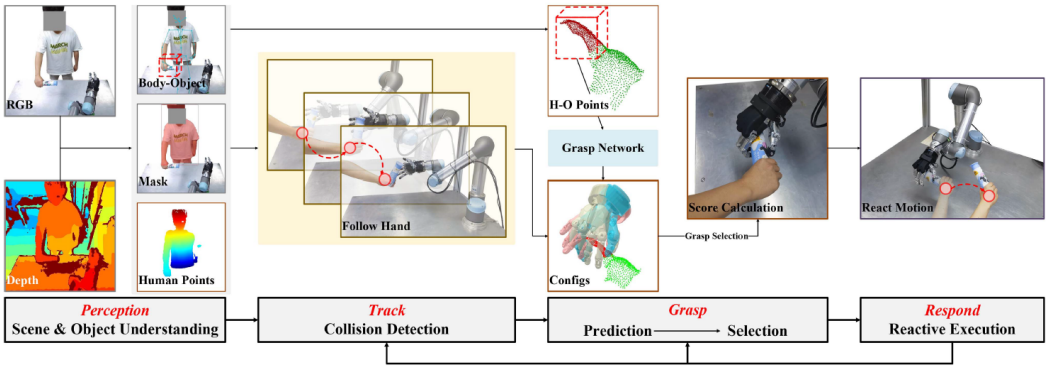


Figure 2.24: Sub-modules used to implement a reactive handover as described in [24]: the input module (here called perception module) is shown on the far left, followed by the tracking module, the grasping module, and finally the response module on the far right.

target position for the Tool Center Point (TCP) of the robot arm is computed relative to the position of the human palm, with a specific translational offset. Notably, the target rotation is ignored during tracking - both because it is determined in the grasp module and to enhance robustness and safety by reducing the chance of occlusion or collision with the human hand. Tracking is triggered when a significant movement of the human wrist is detected. Specifically, the robot begins to follow if the $L1$ norm of the difference between the current target position (U_t) and the previous one (U_{t-1}) exceeds 0.05 m, allowing for tolerance to small jitters or unintentional hand movements. If the movement is below the threshold, it is ignored and the robot remains stationary. Once activated, the robot's motion is calculated by solving the inverse kinematics to determine joint angles needed to reach the target position. For safety, joint velocity is constrained to ensure smooth and secure movements. Moreover, before executing each tracking step, a predictive collision check is performed by interpolating the robot trajectory and verifying collisions with the hand and object point clouds. This check is designed to be highly efficient (within 2 ms), as real-time performance is required. Since the Robot Operating System (ROS) does not meet this real-time constraint due to communication and computation latency, the authors employ the inverse kinematics solver and motion control commands from the Real-Time Data Exchange (RTDE) interface, which comes with the UR robot used in their setup [55, 56, 57]. The transition to the grasp module occurs only when the system detects that the human hand is in a state of minimal movement for a predefined number of consecutive frames ($N_F = 10$ in the experiments). A smaller N_F would make the grasp prediction more dynamic but could increase the occlusion of the human hand and the object by the anthropomorphic hand during the process. The grasp module performs real-time separation of the point clouds of the hand and the object, and feeds the object points into a grasp neural network. A Transformer-based network is employed to predict dense grasp candidates from a single-view point cloud of the object, as described in Section Grasp Detection Module. From the onset of the

actual handover - whether the robot arm is approaching the object or the hand is initiating closure - a **response module** is activated in parallel. This module continuously monitors changes in the position of the human hand. If significant movement is detected before the handover completes, the response module halts the ongoing grasp process and returns the system to the tracking phase. The system is designed to allow the human user to withdraw their hand at any moment, and safely manage this behavior via the tracking module.

The final example of reactive grasping, although not directly related to handover, represents a particularly noteworthy contribution. Bianchi et al. employ a Pisa/IIT SoftHand and focus specifically on enhancing **reactivity during the grasping phase** to prevent failures. This is achieved through the introduction of a novel approach that dynamically adjusts the grasp configuration of soft robotic hands based on tactile feedback [14, 16]. Contact detection is performed using minimalist IMU sensors placed on the fingertips and the back of the robotic hand. These sensors continuously acquire acceleration signals, which are rapidly processed to detect the exact moment of contact. Upon contact detection, the system triggers a predefined grasp "primitive" from its database. These primitives are sensorimotor actions, akin to reflexes, consisting of a coordinated combination of hand closure and wrist movements, as illustrated in Fig.2.25. Additionally, the intrinsic adaptability of the soft hand supports grasp generalization across various object shapes and conditions. Safety considerations are also addressed, such as filtering out contacts from above that could potentially be dangerous.

Figure 2.25: Grasping sequence of a tennis ball using the reactive tactile-based method proposed by Bianchi et al. [16].

2.3 Our Contribution

This thesis builds upon a pre-existing handover framework using a collaborative robot and a soft anthropomorphic hand [17]. The system has been redesigned and extended to achieve reactive and safe human-to-robot handovers. We now describe the system setup and outline the key contributions of the previous work.

Robotic Setup Castellani et al. used a Franka Emika Panda 7-DOF collaborative robotic arm equipped with a Pisa/IIT SoftHand mounted at its end-effector [12, 14]. An ATI Gamma Force/Torque sensor was integrated at the robot wrist to detect physical

interaction during handover and to trigger grasping actions in real time. The robot was mounted on a static table, providing a fixed and reproducible workspace for all experiments. A Kinect v1 camera was used to provide RGB-D images. The setup is shown in Fig.2.26.

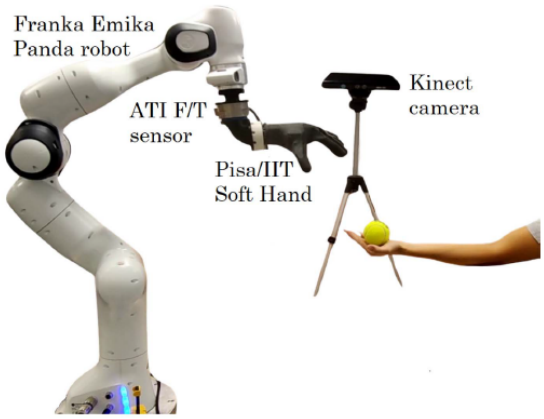


Figure 2.26: Setup of our previous article [17].

Percepion pipeline They developed a vision-based open-palm Human-to-Robot handover system where a soft robotic hand exploits contact with the human hand for improved grasp success and robustness. This results was achieved building a perception pipeline based on a single RGB-D camera, already described in 2.1.2. In this section, we provide a concise overview of the method, illustrated in Fig.2.15. The system begins by leveraging the MediaPipe Full Hands model [30], which extracts 2D hand landmarks and infers their relative 3D coordinates centered on the hand, using solely RGB imagery. Subsequently, the absolute 3D position of the hand is determined by projecting the midpoint between the third and fourth knuckles - identified in the image space - into 3D coordinates. This projection utilizes the depth map from the RGB-D sensor and is computed according to the following relations:

$$x_h =\sim \frac{(u_h - c_x) \cdot Z_h}{f_x}, \quad y_h = \frac{(v_h - c_y) \cdot Z_h}{f_y}, \quad z_h = Z_h \tag{2.2}$$

To infer the hand's orientation, the method fits a plane to a selected set of palm-related points, leveraging the assumption that the hand remains open during handover. The normal vector of the resulting plane serves as an estimate of the palm's orientation. For object localization, a cubic volume is centered at the previously estimated hand position. The corners of this cube are back-projected into the image to delineate a 2D region of interest. Within this region, instance segmentation is performed using the YOLOv8 model [26], allowing the system to isolate the held object while minimizing interference from background elements. The final step involves projecting the segmented 3D point cloud onto the estimated palm plane. The Open3D library [32] is then employed

to perform Principal Component Analysis (PCA) on the convex hull of these projected points. This analysis yields an oriented bounding box for the object, where one face of the box is consistently aligned with the plane of the hand, thereby preserving the relative orientation between the object and the hand. It is important to note that this method can detect an arbitrary set of objects, as shown in Fig.2.27, thereby enabling applicability in real-world scenarios.

Figure 2.27: Objects used in the user study reported in our previous work [17].

Grasp Strategy The system consistently employs a top-down grasp approach initiated from an open-palm configuration, illustrated in Fig.2.19. The robot positions its end-effector above the human hand holding the object and then descends until contact is established. It subsequently glides along the surface of the hand to envelop the object, leveraging the compliance of the soft robotic hand.

Motion Planning and Control In the system, motion planning and control are managed using the MoveIt framework, a widely adopted open-source software for robotic motion planning. MoveIt is integrated seamlessly with the Robot Operating System (ROS), providing tools for trajectory generation, inverse kinematics and collision detection [58, 55]. Within the architecture, MoveIt handles the computation of joint configurations required to reach the target end-effector pose, ensuring that planned trajectories are both kinematically feasible and free from collisions with the environment and, most importantly, the human operator. The planning pipeline is triggered via ROS interfaces, allowing real-time communication between the perception module, which generates target poses, and the robot controller, which executes the trajectories. In particular, they use the MoveGroupCommander interface, a high-level Python API provided by MoveIt, which simplifies the integration of planning and execution commands within the ROS ecosystem. This interface allows for programmatic control of the robot arm, enabling the specification of goals, the planning of collision-free paths, and the execution of those trajectories with minimal overhead.

Soft Hand Advantages The use of a soft robotic hand in this work provides several critical advantages for human-to-robot handover. Its inherent passive compliance enhances safety, allowing for deliberate and gentle physical contact with the human hand. Rather than being avoided, this contact is actively exploitoed as an environmental constraint to improve grasp robustness and success, particularly when handling small or challenging objects or under localization uncertainty. Furthermore, user studies indicate that the soft hand approach is perceived as more natural and reliable, fostering greater trust compared to contact-avoidant strategies. The hand's softness also contributes to a reduced sense of discomfort or perceived risk during interaction.

Prior to the contributions of this thesis, the robot executed a predefined trajectory toward a static grasp point on the object, without accounting for dynamic hand motion. No mechanisms for grasp adaptation or interruption handling were implemented, resulting in limited flexibility and rendering the system unsuitable for real-world scenarios.

Chapter 3

Reactive Handover

This chapter presents a reactive human-to-robot handover pipeline, built upon a pre-existing ROS-based implementation. Using an RGB-D camera, the system detects a human hand presented with an open palm, relatively still and holding an object. Once a stable detection is established, the simulated robot in RViz plans and executes a motion to reach the object, grasp it, and return to a predefined home pose. Unlike the previous static approach, perception remains active during motion: if the user moves the hand while the robot is executing, the system detects the displacement, stops the current trajectory, and replans toward the updated target, thus enabling reactive behavior. This significantly improves the reliability and naturalness of the handover interaction, making the approach suitable for real-world deployment.

In the following sections, the chapter details the implementation of the proposed reactive handover system, starting with the technical development environment and software infrastructure adopted to ensure modularity and reproducibility. It then presents the core algorithmic modifications that enable continuous perception and dynamic replanning during motion execution, followed by the description of key mechanisms introduced to model the occlusion caused by the robot's movements in the image, as well as minor adjustments to the algorithm aimed at improving the overall handover success rate. The experimental setup is described next, illustrating the physical arrangement and data collection procedures used to validate the approach. The experimental protocol is then introduced, clarifying which algorithm was implemented first and thereby helping the reader understand the order in which the experiments were carried out with each method. Finally, the chapter analyzes the tuning of a key system parameter - the reactive threshold - through controlled tests aimed at identifying the most effective trade-off between responsiveness and stability.

3.1 Development Environment

The development and testing of the experimental work were carried out on a workstation running Ubuntu 22.04.5 LTS, equipped with an NVIDIA TITAN Xp GPU and CUDA

version 12.4. All components of the system were containerized using Docker version 28.0.0 to ensure a modular, portable, and reproducible development environment.

Two separate Docker images were created to support the different functional blocks of the system. The first image was responsible for managing the Intel RealSense D415 RGB-D camera [59]. It was based on the official `ros:noetic-ros-base-focal` image and included the installation of necessary dependencies such as Python 3, ROS Noetic, and the RealSense ROS repository [60, 61]. This container handled camera data acquisition and published RGB-D frames and intrinsic calibration parameters as ROS topics.

The second Docker image handled the robotic simulation and perception components. It was based on `nvidia/cuda:11.8.0-devel-ubuntu20.04` [62] and GPU acceleration was enabled through CUDA and cuDNN libraries. The image also included ROS Noetic and MoveIt for robot planning and control, along with the `franka_ros` package for integration with simulated robot hardware. The robot simulation was visualized via RViz, while handover logic and perception modules were implemented in Python. The perception pipeline relied on libraries such as OpenCV, PyTorch, and a set of additional Python packages listed in a dedicated `requirements.txt` file. These included tools for numerical computing, real-time hand tracking, visualization, and pose estimation. PyCharm 2024.3.3 was installed to facilitate Python development within the container.

Although the two containers were launched independently, they were designed to work together: the camera container published ROS topics containing sensor data, which were then subscribed to by nodes running in the robot container to perform perception and motion planning tasks.

3.2 Proposed Algorithm

The architecture adopted in this work is based on a pre-existing ROS-based pipeline for human-to-robot handover, as described in [17]. The original implementation assumed a static handover setup, where the human holds the object with an open palm, keeping the hand still in space. Once the object was detected, the robot would compute a single motion plan and execute it without taking into account further movements of the human hand. This approach, although functional in constrained scenarios, showed limitations in real-world conditions where small user movements or corrections during the handover are common. The robot's inability to adapt to these movements often resulted in failed or suboptimal grasps, as well as the need to reset the system when the conditions changed during execution.

In this work, I analyzed the structure of the existing state machine and the flow of ROS topics published by each node. Through this analysis, I identified the specific states in which perception stopped being active, and where motion execution became disconnected from the real-time sensory input.

The pipeline operates as a sequence of interconnected ROS nodes, shown in Fig. 3.1,

each subscribing to specific topics published by preceding nodes and publishing new topics for subsequent nodes to consume. The overall behavior of the system is governed by a finite state machine (FSM), where the activation and publishing of various topics depend on the current state of the pipeline. Each state represents a specific operational phase of the robot, and transitions occur in response to events published on the `/event` topic. The current state is continuously published on the `/state` topic, allowing other nodes to react accordingly. The pseudocode of the FSM is shown in Algorithm 1. This design ensures that perception, planning, and execution are coordinated according to the system's operational phase.

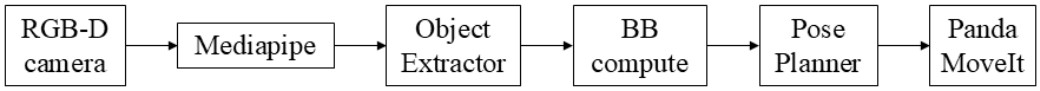


Figure 3.1: ROS nodes involved in the system and their interconnections. Communication through ROS topics is implied but not explicitly shown, to simplify the visualization.

The pipeline begins with the RealSense camera publishing raw frames on the topic `/camera/color/image_raw`. The Mediapipe node subscribes to this topic and publishes 2D hand keypoints on `/2d_kps` and the hand plane on `/hand_plane`. The Object Extractor node listens to `/2d_kps` and `/hand_plane`, processing them to estimate the 3D hand pose, which it publishes on `/hand_pose`. However, in the original implementation, this publication was limited to certain states of the FSM (IDLE, NO_OBJECT, and TRACKING), effectively disabling pose updates during motion execution. I enabled this publication also in the MOVING state, to continue tracking the object even while the robot starts moving towards the hand.

The Bounding Box Compute node subscribes to both `/2d_kps` and `/hand_pose` and calculates the 3D object bounding box, publishing it on `/3d_object_bb`. The Pose Planner node then subscribes to `/3d_object_bb` and publishes the target end-effector poses on `/pose_target` and `/pose_target_typed`. Since this originally occurred only while in the TRACKING state, I added a condition that checks if the state is MOVING and whether the bounding box of the object has moved more than a certain `reactive_thr` (to be calibrated later). If this happens, the last pose is published again and an event `"hand_moved"` is triggered, as shown in Algorithm 2, causing the FSM state to change from MOVING back to TRACKING.

Algorithm 1 Finite State Machine for Handover Process

```
1: Input: Event from /event topic
2: Initialize: State ← STOPPED
3: while system is running do
4:     if State = STOPPED and Event = start then
5:         State ← IDLE
6:     else if State = IDLE and Event = hand_seen then
7:         State ← NO_OBJECT
8:     else if State = NO_OBJECT then
9:         if Event = hand_lost then
10:            State ← IDLE
11:        else if Event = object_seen then
12:            State ← TRACKING
13:        end if
14:    else if State = TRACKING then
15:        if Event = object_lost then
16:            State ← NO_OBJECT
17:        else if Event = hand_lost then
18:            State ← IDLE
19:        else if Event = motion_started then
20:            State ← MOVING
21:        end if
22:    else if State = MOVING then
23:        if Event = target_reached then
24:            State ← MOVING_DOWN
25:        else if Event = hand_moved then
26:            State ← TRACKING
27:        end if
28:    else if State = MOVING_DOWN then
29:        if Event = start then
30:            State ← IDLE
31:        end if
32:    end if
33:    Publish State to /state
34: end while
```

Algorithm 2 Target Pose Update During MOVING State

Trigger: Called upon reception of `/3d_object_bb`

if State is `MOVING` **then**
  pose ← position extracted from bounding box
  if $\|pose - last\_pose\| > reactive\_thr$ **then**
    Publish event `"hand_moved"` on topic `/event`
    Calculate `last_target_pose` from bounding box
    Publish `/pose_target` and `/pose_target_typed`
    `last_pose` ← *pose*
  end if
end if

Finally, the Panda MoveIt Simulation node subscribes to `/pose_target_typed` to execute motion plans and control the simulated robot accordingly. In the previous system, it took the first received pose and moved there without considering any subsequent updates. In our proposed system, every time a new `/pose_target_typed` message is published, the function `handle_pose_target` is called, shown in Algorithm 3. This function checks if the system is in the TRACKING state and, if the robot is not already moving, it starts the motion towards the received target pose. If the robot is already moving and the state returns to TRACKING, the robot is stopped. Moreover, every time `/3d_object_bb` is published, the function `check_target_reached` is called, described in Algorithm 12. This function verifies whether the robot is in the MOVING state and if its end-effector is close enough to the target pose. If so, it publishes the event `"target_reached"` which changes the FSM state to MOVING_DOWN, initiates gripper closure, and then sends the robot back to its initial pose.

Algorithm 3 Robot Motion Planning, Execution, and Interruption Handling

Trigger: Called upon reception of `/pose_target_typed`

if State is `TRACKING` **and** `motion_in_progress` is `true` **then**
  Stop current motion
  `motion_in_progress` ← `false`
else if State is `TRACKING` **and** `motion_in_progress` is `false` **then**
  `target_pose` ← received pose
  `plan_result` ← Plan trajectory to `target_pose`
  Publish event `"motion_started"`
  `motion_in_progress` ← `true`
  Execute plan
end if

Algorithm 4 Target Reached Verification and Grasping Sequence

Trigger: Called upon reception of `/3d_object_bb`

if State is `MOVING` **then**
  if `poses_are_close`(current pose, target_pose) **then**
    Stop current motion
    Publish event `"target_reached"`
    Close the gripper
    Move to initial pose
    `motion_in_progress` ← `false`
  end if
end if

3.3 Proposed Algorithm Considering Occlusion

The reactive handover proposed in Section 3.2 performed well overall, but it presented an undeniable drawback. Since it was tested exclusively in simulation, the video stream used for perception never contained the robotic arm itself. In real handover scenarios, however, the arm inevitably occludes parts of the scene, making perception more challenging and the experiments more realistic. To address this limitation, we implemented a simulated occlusion mechanism that inserts the robotic arm into the camera feed as if it were physically present.

To achieve this, a function called `transform_joint` was first created within the Panda MoveIt Simulation node. This function retrieves the coordinates of the robot joints relative to its base frame, using MoveIt's auxiliary utilities, and then transforms them into the reference frame of the camera. The function is invoked every time the RealSense camera publishes a raw frame on the `/camera/color/image_raw` topic, and it subsequently publishes the transformed joint coordinates in the camera frame on the `/joint_poses` topic.

A new Occlusion node was then developed, responsible for projecting the robot onto the image and publishing the occluded version. This node subscribes to `/joint_poses` and stores the joint positions whenever they are published. It also subscribes to `/camera/color/image_raw`, and each time a new frame is received, it calls the `project` function-whose pseudocode is reported in Algorithm 5-to generate and publish the image containing the robot projection on the topic `/projected_rgb_topic`. The node also maintains an array, `franka_radii`, which stores the estimated radii of the links composing the Franka Emika Panda robot used in our work, enabling a realistic rendering of the arm within the image. Finally, for robustness, the Occlusion node implements an additional function, `republish_last`, which republishes the last available occluded frame every time `/camera/color/image_raw` is updated, ensuring continuity in case of errors during the projection of the most recent frame.

Algorithm 5 Projection of Robot Joints onto Image

Trigger: Called upon reception of `/camera/color/image_raw`

if camera not ready **or** `joint_poses` empty **then**
  return
end if
for each consecutive pair (p_i, p_{i+1}) in `joint_poses` **do**
  Extract 3D points $p1\_3d, p2\_3d$
  Extract radii r_1, r_2
  Compute axis direction of the robot i-link $axis \leftarrow \frac{p2\_3d - p1\_3d}{\|p2\_3d - p1\_3d\|}$
  Compute orthogonal vector to axis and z-camera $ortho \leftarrow \frac{axis \times (0,0,1)}{\|axis \times (0,0,1)\|}$
  Compute projection boundary points:

$$\{p1a, p1b, p2a, p2b\} \leftarrow \{p1\_3d \pm r_1 \cdot ortho,\ p2\_3d \pm r_2 \cdot ortho\}$$

  for each point p in $\{p1a, p1b, p2a, p2b\}$ **do**
    if p invalid **or** $p_z \leq 0$ **then**
      projection $\leftarrow$ None
    else
      $u \leftarrow \frac{p_x f_x}{p_z} + c_x$
      $v \leftarrow \frac{p_y f_y}{p_z} + c_y$
      projection $\leftarrow (u, v)$
    end if
  end for
  if all projections valid **then**
    Draw polygon defined by $\{(u, v)\}$ on `image`
  else
    Log warning
  end if
end for
Publish on `/projected_rgb_topic` and update `last_projected_image`

Finally, in all nodes that originally received the camera image as input, the topic `/camera/color/image_raw` was replaced with `/projected_rgb_topic`, so the ROS nodes connections becomes the one represented in Fig. 3.2.

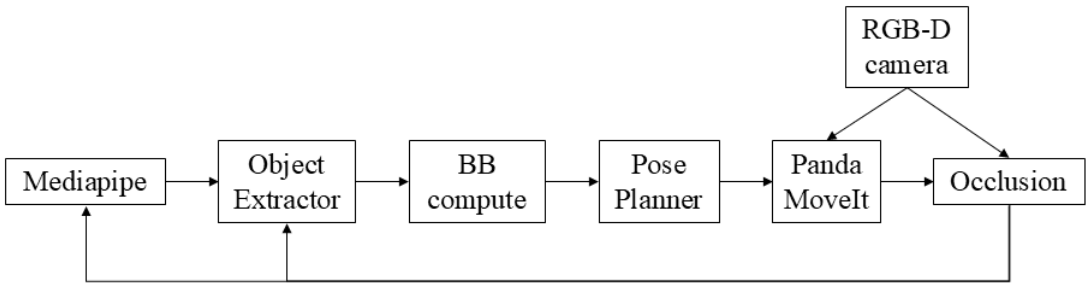


Figure 3.2: ROS nodes involved in the system considering occlusion and their interconnections. Communication through ROS topics is implied but not explicitly shown, to simplify the visualization.

Algorithmic Modifications to Improve Performance

After implementing the occlusion model and testing the previous algorithm under these conditions, I observed a significant decline in performance caused primarily by two factors. First, as the robot approached the hand to grasp the object, perception was often disrupted: the system could no longer recognize the hand, which in turn prevented the completion of the handover. Second, in certain trajectories, the robotic arm itself occluded a large portion of the camera's field of view, remaining in that position until the object reappeared in a different, unobstructed area of the frame.

Acknowledging these practical issues of the initial algorithm, I introduced two key modifications to address them. Specifically, I added a new REACHING state to the finite state machine (FSM). The system transitions into this state once the robot comes within a specified proximity to the target pose (specifically 7 cm). From the REACHING state, the only possible transition is to MOVING_DOWN. In this mode, the algorithm relies solely on the last received target pose and disregards further perception updates, thereby mitigating the risk of losing track of the hand or object as the robot approaches. The pseudocode originally described in Algorithm 1 is thus updated as shown in Algorithm 6. This adjustment is based on the reasonable assumption that once the robot is sufficiently close, the human hand will remain stable at the target location, making the simplification both practical and reliable.

The second issue was addressed by introducing a 1-second timer in the `Timer` node, which is triggered each time the system returns to the IDLE or NO_OBJECT states. If, at the end of the timer, the robot was still obstructing the camera's view, the event `moving_away_to_home` was published. The detailed behavior of this timer is described in the pseudocode in Algorithm 7.

Simultaneously, in the Panda MoveIt Node, I implemented the `moving_away` function, which is invoked whenever an event is published. Upon receiving the `moving_away_to_home` event, this function initiates the movement of the robot back to its initial pose, as illustrated in Algorithm 8.

If, during this motion, the robot moves sufficiently to clear the camera's line of sight and re-detects the object, the motion is interrupted using the previously described `handle_pose_target` function from Algorithm 3, which has been slightly modified to handle the specific case of moving toward the home pose. The final version of this function is presented in Algorithm 9.

3.4 Experimental Setup

The experimental setup consists of a standard desk workspace equipped with an RGB-D camera - specifically an Intel RealSense D415 - mounted on a tripod. The workspace is carefully prepared with masking tape to delineate the camera's field of view and specific target locations. Figure 3.3 shows an overview of the complete setup, highlighting the

Algorithm 6 Finite State Machine for Handover Process — Considering Occlusion

```
1: Input: Event from /event topic
2: Initialize: State ← STOPPED
3: while system is running do
4:     if State = STOPPED and Event = start then
5:         State ← IDLE
6:     else if State = IDLE and Event = hand_seen then
7:         State ← NO_OBJECT
8:     else if State = NO_OBJECT then
9:         if Event = hand_lost then
10:            State ← IDLE
11:        else if Event = object_seen then
12:            State ← TRACKING
13:        end if
14:    else if State = TRACKING then
15:        if Event = object_lost then
16:            State ← NO_OBJECT
17:        else if Event = hand_lost then
18:            State ← IDLE
19:        else if Event = motion_started then
20:            State ← MOVING
21:        end if
22:    else if State = MOVING then
23:        if Event = near_target then
24:            State ← REACHING
25:        else if Event = hand_moved then
26:            State ← TRACKING
27:        else if Event = target_reached then
28:            State ← MOVING_DOWN
29:        end if
30:    else if State = REACHING then
31:        if Event = target_reached then
32:            State ← MOVING_DOWN
33:        end if
34:    else if State = MOVING_DOWN then
35:        if Event = start then
36:            State ← IDLE
37:        end if
38:    end if
39:    Publish State to /state
40: end while
```

Algorithm 7 Move Away Timer — Considering Occlusion

Trigger: Called upon reception of `/event`

if State = `TRACKING` **and** (Event = `object_lost` **or** Event = `hand_lost`) **then**
    `timer` ← current time
end if
if `timer` ≠ None **and** elapsed time > 1 second **then**
    if State ∈ {`NO_OBJECT`, `IDLE`} **then**
        Publish event `"moving_away_to_home"`
        `timer` ← None
    end if
end if

Algorithm 8 Moving Away Towards Home — Considering Occlusion

Trigger: Called upon reception of `/event`

if Event = `moving_away_to_home` **then**
    Stop current motion
    Set joint target ← *initial_pose*
    plan ← Plan motion to target
    `motion_to_home` ← True
    Execute *plan* asynchronously
end if

Algorithm 9 Robot Motion Planning, Execution, and Interruption Handling — Considering Occlusion

Trigger: Called upon reception of `/pose_target_typed`

if State = `TRACKING` **and** `motion_in_progress` = true **then**
    Stop current motion
    `motion_in_progress` ← false
else if State = `TRACKING` **and** `motion_to_home` = true **then**
    Stop current motion
    `motion_to_home` ← false
    Recall `handle_pose_target(msg)`
else if State = `TRACKING` **and** `motion_in_progress` = false **then**
    `target_pose` ← received pose
    Publish event `"motion_started"`
    `plan_result` ← Plan trajectory to `target_pose`
    `motion_in_progress` ← true
    Execute `plan_result` asynchronously
end if

camera placement and the workspace layout.

Figure 3.3: General view of the experimental setup used in the study.

Three distinct objects are used during the experiments: a whiteboard marker (cylinder with a base diameter of 1.9 cm × length 13 cm, 14.9 g), a Jenga block (parallelepiped of 1.5 cm × 2.5 cm × 7.4 cm, 12.1 g), and a ping pong ball (sphere with a diameter of 4.0 cm, 2.7 g), as shown in Fig. 3.4.

(a) marker

(b) jenga

(c) ball

Figure 3.4: The objects used in the experiments in this thesis are: whiteboard marker (a), Jenga block (b), and ping pong ball (c).

The camera is oriented to capture a top-down view of the desk, where hand movements holding the objects take place (Fig. 3.5). The boundaries of the camera's field of view are outlined with masking tape directly on the desk surface, ensuring that all relevant actions remain within the visible area. Within this region, four cross-shaped markers made of tape are placed at fixed positions, labeled A, B, C, and D in counterclockwise order starting from the top-left. These serve as target locations for the user's hand to reach during the experiments. The center of each cross denotes the stopping point for the hand, while the longer arm of the cross indicates the required hand

orientation at that location.

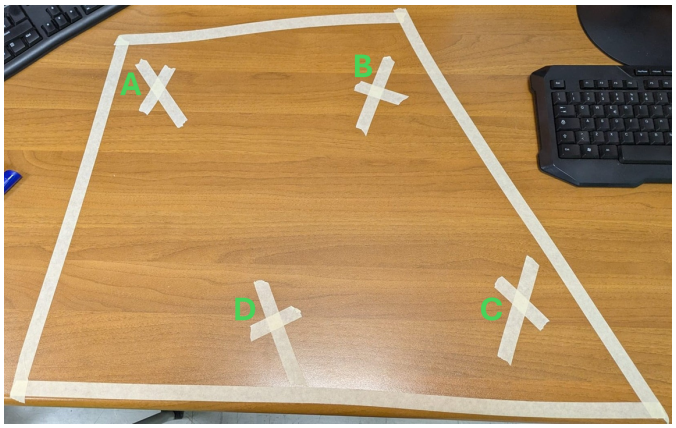


Figure 3.5: Top-down view of the workspace as seen from the RGB-D camera. The boundaries of the camera's field of view are marked with masking tape. Four cross-shaped tape markers (A–D), used as target positions for hand placement, are arranged in counterclockwise order starting from the top-left. The center of each cross indicates the stopping point of the hand, while the longer arm denotes the orientation of the hand.

This configuration was designed to ensure repeatability and precise spatial references across all experimental trials.

To ensure the reproducibility and scientific rigor of the experiments conducted while varying parameters, selected ROS topics were recorded using the rosbag tool during each trial. Specifically, the topics `/camera/color/image_raw`, `/camera/aligned_depth_to_color/image_raw`, `/camera/aligned_depth_to_color/camera_info`, `/camera/depth/image_rect_raw`, `/tf`, and `/tf_static` were captured using the `rosbag record` utility. These streams contain all necessary sensor and transformation data to reproduce the visual and spatial context of each trial. The recorded sessions were replayed using `rosbag play` to test the system's response under controlled and repeatable conditions. This approach allows consistent evaluation of different algorithm configurations without introducing variability caused by human movement or live environmental changes. By decoupling data acquisition from system execution, it becomes possible to isolate the effect of specific parameters or code modifications. This significantly enhances the scientific validity of the experiments, enabling precise comparisons, reproducible results, and facilitates future benchmarking or replication efforts.

Despite the variability introduced by using different objects - requiring separate recordings for each trial - efforts were made to minimize experimental errors and ensure consistency. The use of masking tape markers provided stable spatial references for camera placement and hand positioning, while the recorded rosbag datasets enabled controlled and repeatable playback of sensor data. Together, these measures help limit uncontrolled variability and maintain the scientific rigor of the evaluation across diverse test conditions.

3.5 Experimental Protocol

The development of the reactive handover system followed a sequential process that directly influenced the design and execution of the experiments. Initially, only the proposed algorithm described in Section 3.2 was implemented. This version of the system did not account for camera occlusions caused by the robot's own movements. Consequently, the calibration of the `reactive_thr` parameter, presented in Section 3.6, was performed exclusively with this non-occlusion setup. Similarly, the first set of comparative experiments, later reported in the subsequent chapter (Section 4.2), was also conducted with this initial implementation.

Subsequently, the improved algorithm described in Section 3.3 was developed, explicitly incorporating the modeling of occlusions introduced by the robotic arm. Once this enhanced version was available, the entire set of comparative experiments was repeated using the occlusion-aware algorithm, in order to evaluate its performance under more realistic perceptual conditions. It is important to note, however, that the reactive threshold calibration was not repeated with this second implementation, since it had already been carried out with the original system and proved effective also in the new experiments.

In the next chapter, for each experiment, the results obtained with both algorithms - the first without occlusion and the second with occlusion - are reported in sequence and comparatively analyzed, in order to provide a clearer understanding of the system's strengths, limitations, and overall performance.

3.6 Tuning of Reactive Threshold

The reactive threshold parameter, `reactive_thr`, plays a crucial role in the proposed reactive handover system. It defines the minimum displacement of the detected object's bounding box that triggers the reactive behavior: if the object's bounding box moves more than this threshold, the system interrupts the robot's current motion, switches from the MOVING state back to TRACKING (as described in Section 3.2), and then replans the trajectory accordingly. Selecting an appropriate value for `reactive_thr` is therefore essential to balance responsiveness and system stability.

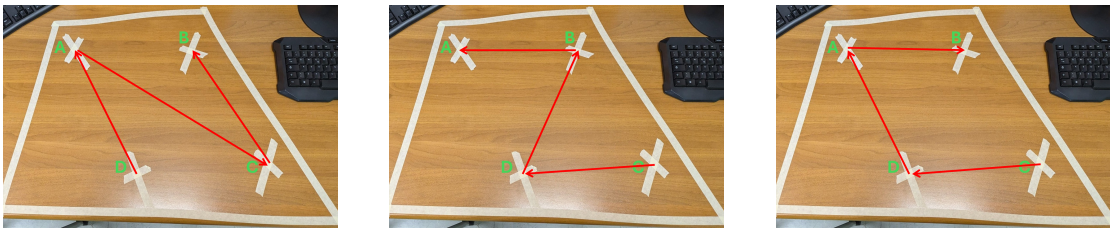
To determine the optimal value for `reactive_thr`, an experiment was conducted testing four candidate thresholds: 1.5 cm, 3 cm, 5 cm, and 7 cm. For this purpose, nine rosbag recordings were collected, each representing a trial in which the human hand holding an object moves fluidly along predefined trajectories. These trajectories pass through the four target locations marked with crosses (A, B, C, D) described in Section 3.4.

Specifically, three distinct trajectories were considered, each repeated with three different objects: the whiteboard marker, the Jenga block, and the ping pong ball, all

introduced in Section 3.4. The trajectories, shown in Fig. 3.6, were designed as follows:

- An "N"-shaped trajectory, passing through the crosses in the order D → A → C → B;
- A "Z"-shaped trajectory, passing through the crosses in the order C → D → B → A;
- A "C"-shaped trajectory, passing through the crosses in the order C → D → A → B.

During each recording, the hand movement was performed smoothly over approximately 40 seconds, capturing continuous transitions across the specified waypoints.



(a) "N"-shaped trajectory (b) "Z"-shaped trajectory (c) "C"-shaped trajectory

Figure 3.6: Top-down view of the workspace with overlaid arrows illustrating the three hand movement trajectories used during the experiment: (a) "N"-shaped, (b) "Z"-shaped, and (c) "C"-shaped.

Subsequently, each rosbag was replayed with the system running without considering occlusions, using the system described in Section 3.2, and with the `reactive_thr` parameter set to one of the four tested values. For each run, key performance metrics were recorded:

- The average computation time for estimating the object's bounding box and the target pose;
- The average planning time for the robot's motion;
- The total time elapsed from the moment the robot detects the hand holding the object until the robot completes the entire trajectory, grasps the object, and returns to the home position.

Throughout all runs, the robot continuously followed the hand, replanning its trajectory every time the hand moved beyond the threshold. Importantly, the system never skipped a waypoint along the trajectory, ensuring full coverage of the intended path.

The results of each run are collected in the tables in this section. Every row represents a different rosbag, while each column represents a different run using a certain value of `reactive_thr`.The average value for each threshold is reported at the bottom of each column. This average is used to choose the optimal value of the `reactive_thr`.

Looking at the Table 3.1 we can observe that the lowest average computation time for estimating the object's bounding box and the target pose can be obtained imposing `reactive_thr` to be 3 cm. We observe that this value is comparable to that obtained with a 7 cm threshold.

Table 3.1: Average pose computation time for each object and trajectory with different `reactive_thr` values (in seconds)

Trajectory	Object	1.5 cm	3 cm	5 cm	7 cm
N	Marker	0.04126	0.00198	0.00225	0.00203
	Jenga	0.04752	0.01905	0.03168	0.01970
	Ball	0.06279	0.05070	0.09550	0.03193
Z	Marker	0.02317	0.01930	0.01900	0.01480
	Jenga	0.01935	0.00180	0.00243	0.01795
	Ball	0.08081	0.06907	0.02914	0.04374
C	Marker	0.04618	0.02028	0.08062	0.08486
	Jenga	0.00260	0.00203	0.00315	0.00233
	Ball	0.01079	0.01910	0.03828	0.00253
Overall Average		**0.03665**	**0.02516**	**0.03747**	**0.02723**

From Table 3.2 we observe that average robot motion planning time remained stable and unaffected by the choice of threshold, as expected given the decoupling between perception and planning stages,

Table 3.2: Average planning time for robot movements for each object and trajectory for different `reactive_thr` values (in seconds)

Trajectory	Object	1.5 cm	3 cm	5 cm	7 cm
N	Marker	0.18958	0.19255	0.15960	0.15735
	Jenga	0.17639	0.16028	0.16533	0.17193
	Ball	0.18067	0.17838	0.17130	0.16090
Z	Marker	0.14424	0.17593	0.17638	0.16655
	Jenga	0.14595	0.12783	0.16943	0.17875
	Ball	0.13965	0.17335	0.16724	0.17080
C	Marker	0.14877	0.17055	0.15652	0.13938
	Jenga	0.19480	0.19040	0.18050	0.14320
	Ball	0.16898	0.14753	0.17180	0.18730
Overall Average		**0.16243**	**0.16553**	**0.16981**	**0.16485**

Table 3.3 shows that the lowest average task completion time was comparable and favorable for `reactive_thr`'s value of 1.5 cm, 3 cm, and 5 cm.

Considering these observations, a threshold of 3 cm was selected as the best trade-off, consistently yielding low perception computation times and competitive overall task durations. This value strikes a balance between sensitivity to object movements and avoiding excessive replanning, thus ensuring the system's reactivity and efficiency.

Table 3.3: Total time for each object and trajectory with different `reactive_thr` values (in seconds)

Trajectory	Object	1.5 cm	3 cm	5 cm	7 cm
N	Marker	32.1984	31.9588	31.9442	31.9414
	Jenga	27.7949	28.0514	27.7720	27.4692
	Ball	28.3641	28.6440	28.6863	28.2629
Z	Marker	36.4098	36.6161	36.0395	36.4538
	Jenga	36.9107	36.6291	35.7173	37.0034
	Ball	40.8051	41.0978	41.6814	45.3839
C	Marker	31.9638	30.0170	31.0333	44.2258
	Jenga	30.1363	30.5357	30.5277	30.4236
	Ball	32.5385	34.7297	34.4291	34.3311
Overall Average		**33.1154**	**33.2901**	**33.2358**	**35.4442**

Chapter 4

Validation of the results

This chapter presents a series of experimental evaluations aimed at assessing the performance of the proposed algorithm, considering occlusion and non, in various handover scenarios. The experiments are designed to simulate realistic human-robot interaction conditions, in which a robotic arm must reach for and follow the movement of a human-held object, exhibiting different levels of temporal and spatial fluidity. The proposed method is compared against a baseline strategy that always returns to a predefined home position before re-planning each movement.

The chapter begins with a description of the baseline algorithm implementation, followed by an improvement of this algorithm in which we consider also the occlusion caused by the robotic arm in the video. The second part of the chapter contains three experiments comparing the baseline with our proposed approach. For each experiment we propose a comparison between the proposed and baseline algorithm without considering the occlusion and then a comparison for the same experiment for the two algorithms that consider the occlusion. The first experiment involves continuous, fluid trajectories and is designed to test the system's ability to follow dynamic targets in real time. The second and third experiments instead focus on semi-static interactions, where the object remains stationary at specific handover points for limited durations - 15 seconds in the second test and 10 seconds in the third (except for the first point, which always lasts 15 seconds). These scenarios introduce increasing levels of time pressure, emphasizing the adaptability and responsiveness of each algorithm.

Various evaluation metrics are considered, including average pose computation time, planning time, overall task duration, and - most importantly - the number of target poses successfully observed or reached, depending on the experiment. Together, these analyses validate our proposed approach and demonstrate its effectiveness in human-robot handover contexts.

4.1 Baseline Algorithm

To properly assess the improvements introduced by the proposed algorithm, a baseline version of the system was implemented for comparison. This baseline algorithm shares the same overall architecture and pipeline structure described in Section 3.2, including the same ROS-based modular design and finite state machine. However, it differs significantly in how the robot reacts to target pose updates during motion execution.

In the proposed algorithm, reactive behavior allows the robot to adapt its motion without interruption, continuously updating its trajectory based on the latest observations. In contrast, the baseline algorithm employs a more conservative and interrupt-driven strategy: if the target hand pose changes significantly while the robot is in motion, the robot aborts its current trajectory, returns to its initial pose, replans a new trajectory from scratch, and only then resumes its movement toward the updated hand position. This strategy ensures that motion is always initiated from a known configuration but sacrifices responsiveness and fluidity in favor of simplicity and robustness.

The baseline approach was designed to simulate the behavior of a different reactive robotic system. By comparing this algorithm with the proposed solution, it becomes possible to evaluate the trade-offs between reactivity and planning overhead, and to demonstrate the benefits of uninterrupted execution.

The first component of the baseline system that differs from the proposed version is the way pose updates are handled. Every time a new message is received on the `/pose_target_typed` topic, the robot stores the previous target pose and updates the current one, as shown in Algorithm 10. This allows for a subsequent comparison between poses to detect significant displacements.

Algorithm 10 Target Pose Update — Baseline Algorithm

Trigger: Called upon reception of `/pose_target_typed`
1: **if** State is `TRACKING` **or** `MOVING` **then**
2: `last_target_pose` ← `target_pose`
3: `target_pose` ← pose contained in received message
4: `last_msg` ← received message
5: **end if**

The central behavioral difference in the baseline system lies in its motion execution logic, described in Algorithm 11. If the robot is already in the `TRACKING` state and a motion is in progress, it immediately stops the current execution upon receiving a new pose, returns to the initial pose, and then recursively re-calls the function to start a new plan. This enforces a complete reinitialization of planning and execution every time the pose changes, reducing adaptability but ensuring consistency in the robot's starting condition. When the robot is in the `MOVING` state, the algorithm checks whether the current and previous target poses differ by more than a threshold (`reactive_thr`). If so, it publishes the event `"hand_moved"`, causing the state machine to switch back to

TRACKING, which in turn triggers the above interruption and replanning process. If no motion is currently in progress and the robot is in the TRACKING state, a plan is computed and executed normally.

Algorithm 11 Robot Motion Planning, Execution, and Interruption Handling — Baseline Algorithm

Trigger: Called upon reception of /pose_target_typed
if State is TRACKING **and** motion_in_progress is true **then**
    Stop current motion
    motion_in_progress ← false
    Move to initial pose
    last_target_pose ← target_pose
    Recursively call handle_pose_target(last_msg)
else if State is MOVING **and** last_target_pose and target_pose are defined **then**
    if Distance between target_pose and last_target_pose > baseline_thr **then**
        Publish event "hand_moved"
    end if
else if State is TRACKING **and** motion_in_progress is false **then**
    target_pose ← pose contained in received message
    plan_result ← Plan trajectory to target_pose
    Publish event "motion_started"
    motion_in_progress ← true
    Execute plan
end if

Finally, the grasping sequence in the baseline algorithm closely mirrors that of the proposed approach. As shown in Algorithm 12, when the robot is in the MOVING state and the end effector is sufficiently close to the target pose, it stops motion, closes the gripper, publishes the "target_reached" event, and returns to the initial pose. This concludes the handover attempt, resetting the robot for a new cycle.

Algorithm 12 Target Reached Verification and Grasping Sequence — Baseline Algorithm

Trigger: Called upon reception of /3d_object_bb
if State is MOVING **and** target_pose is defined **then**
    if poses_are_close(current pose, target_pose) **then**
        Stop current motion
        Publish event "target_reached"
        Close the gripper
        Move to initial pose
        motion_in_progress ← false
    end if
end if

In conclusion, the baseline algorithm provides a suitable reference implementation against which the proposed method can be quantitatively evaluated. Its behavior - based

on interrupting and replanning motion from a fixed home position - offers a clear and consistent basis for comparison with the proposed algorithm, described in Section 3.2.

4.1.1 Baseline Algorithm Considering Occlusion

After implementing the proposed algorithm with the simulated video occlusion caused by the robot's presence, as described in Section 3.3, it became necessary to develop an occlusion-aware version of the baseline algorithm presented in Section 4.1, in order to enable a fair comparison between the two approaches.

Specifically, the publication of joint poses and the Occlusion node are identical to those described in Section 3.3. Likewise, the improvements integrated into the baseline algorithm mirror those of the proposed method, ensuring that the two algorithms remain directly comparable. For instance, the system transitions to the REACHING state under the same conditions - namely, when the end-effector is within 7 cm of the target pose - so as to disregard spurious signals falsely indicating the loss of hand perception.

Regarding the timing mechanism that triggers movement, it follows the same procedure outlined in Algorithm 7. However, the motion logic differs slightly from that of the proposed algorithm. While the latter is designed to reactively move as soon as the object is perceived to change position, the baseline algorithm instead requires the robot to always return to the home position before initiating movement toward a new target. Consequently, in the occlusion-aware baseline, whenever the timer triggers a retreat to the initial pose to clear the visual field and reacquire the object, the robot must fully reach its home configuration before resuming its trajectory toward the hand. For this reason, the algorithm responsible for initiating the return-to-home motion has been slightly modified to ensure consistent operation. This updated logic is presented in Algorithm 13.

Algorithm 13 Moving Away Towards Home — Baseline Algorithm Considering Occlusion

Trigger: Called upon reception of `/event`
1: **if** Event = `moving_away_to_home` **then**
2: Stop current motion
3: Set joint target ← *initial_pose*
4: *plan* ← Plan motion to target
5: `motion_in_progress` ← True
6: Execute *plan* asynchronously
7: **end if**

4.2 Experiments

4.2.1 Experiment 1: Following Fluid Movements

In this first experiment, we aimed to perform a comparative evaluation between the proposed algorithm, with the `reactive_thr` set to 3 cm as previously determined in Section 3.6, and the baseline algorithm using fluid movements of the human hand and the ability of the two approaches to follow smooth hand movements, as well as their timing performances.

The experiment was conducted using the same dataset of rosbag recordings employed in the prior threshold tuning experiment, described in Section 3.6. These recordings consist of three distinct hand movement trajectories performed in a fluid manner by a human operator, with each trajectory executed using three different objects, described in Section 3.4. The three trajectories, shown in Figure 3.6, are a "N"-shaped trajectory, a "Z"-shaped trajectory, and a "C"-shaped trajectory. Each trajectory features smooth and continuous hand motions, lasting approximately 40 seconds per execution, simulating a natural handover process that involves transitioning between multiple predefined waypoints marked in the workspace. The usage of the same rosbag recordings across both algorithms ensures a fair and consistent comparison between their performance.

The evaluation metrics selected for this first experiment include:

- The average computation time required to estimate the bounding box of the object and its corresponding target pose, which reflects the perception efficiency of the system;
- The average planning time for generating the robot's motion trajectories, representing the responsiveness of the motion planner;
- The total task completion time, measured from the initial detection of the hand holding the object until the robot finishes the trajectory, successfully grasps the object, and returns to the home position;
- The number of not observed poses. A pose is defined as not observed if the robot fails to initiate movement towards that pose before the human hand moves on to the next waypoint. This metric captures the system's ability to fully track and follow the hand's motion, which is crucial for ensuring smooth and complete handover operations.

The inclusion of the not observed poses metric is particularly important in highlighting the practical differences between the two algorithms. Since the baseline algorithm mandates returning to home before replanning, it could potentially result in delays that would cause the robot to miss certain positions in the trajectory. In contrast, the proposed algorithm's reactive replanning aims to minimize such skips, maintaining a continuous and timely response to the hand's movements.

Experiment 1: Results without occlusion

The first time this experiment was conducted, the proposed algorithm was compared with the baseline without considering the occluded video produced by the robot's projection, thus relying on the implementations described in Sections 3.2 and 4.1, respectively. This section reports the results of this initial comparison.

The following tables report detailed results for each of the metrics across all combinations of trajectories and objects. Table 4.1 summarizes the average pose computation times, showing that the proposed algorithm consistently achieves lower computation times compared to the baseline.

Table 4.1: Average pose computation time for each object and trajectory (in seconds).

Trajectory	Object	Proposed	Baseline
N	Marker	0.00198	0.03272
	Jenga	0.01905	0.18537
	Ball	0.05070	0.02506
Z	Marker	0.01930	0.01493
	Jenga	0.00180	0.03145
	Ball	0.06907	0.05347
C	Marker	0.02028	0.05570
	Jenga	0.00203	0.02583
	Ball	0.01910	0.02658
Overall Average		**0.02259**	**0.05012**

Table 4.2 lists the planning times, with the proposed approach also exhibiting faster planning performance.

Table 4.2: Average planning time for robot movements for each object and trajectory (in seconds).

Trajectory	Object	Proposed	Baseline
N	Marker	0.19255	0.20400
	Jenga	0.16028	0.18560
	Ball	0.17838	0.16970
Z	Marker	0.17593	0.21107
	Jenga	0.12783	0.19505
	Ball	0.17335	0.18082
C	Marker	0.17055	0.17165
	Jenga	0.19040	0.21363
	Ball	0.14753	0.18860
Overall Average		**0.16853**	**0.19112**

Interestingly, as shown in Table 4.3, the total task completion times between the two algorithms are similar overall, with slight variations depending on the trajectory and

object.

Table 4.3: Total time for each object and trajectory (in seconds).

Trajectory	Object	Proposed	Baseline
N	Marker	31.9588	34.1793
	Jenga	28.0514	28.3918
	Ball	28.6440	30.8818
Z	Marker	36.6161	38.8924
	Jenga	36.6291	43.7208
	Ball	41.0978	47.0801
C	Marker	30.0170	29.1525
	Jenga	30.5357	33.4939
	Ball	34.7297	35.7871
Overall Average		**33.14218**	**32.96840**

However, the real difference becomes evident in the not observed poses statistics. Table 4.4 highlights the specific waypoints missed by the baseline algorithm in several runs, whereas the proposed algorithm successfully observes all poses, as detailed in Table 4.5.

Table 4.4: Positions not seen by the Baseline algorithm. Positions 1 to 4 correspond to temporally ordered observation points.

Position	N-marker	N-jenga	N-ball	Z-marker	Z-jenga	Z-ball	C-marker	C-jenga	C-ball
1				X	X				X
2	X	X	X			X	X	X	
3		X		X			X		X
4									

The baseline algorithm missed a total of 13 poses out of 36, corresponding to an observation rate of approximately 63.9%, while the proposed algorithm maintained a perfect observation rate of 100%.

These results underline the advantage of the proposed reactive algorithm in maintaining continuous tracking and responsiveness during dynamic handover tasks, avoiding the interruptions and inefficiencies introduced by mandatory returns to the home position inherent in the baseline method.

Experiment 1: Results considering occlusion

The second execution of this experiment compared the two algorithms while explicitly accounting for the occlusion introduced by the projection of the robotic arm, relying on the implementations described in Sections 3.3 and 4.1.1, respectively. It is important to note that in this experiment - and only in this one - the robot was required to follow the human hand through all four target poses and grasp the object only at the end of the trajectory. However, since the same pre-recorded rosbag was used as in the non-

Table 4.5: Number of not observed positions for each object and trajectory without occlusions.

Trajectory	Object	Proposed	Baseline
N	Marker	0	1
	Jenga	0	2
	Ball	0	1
Z	Marker	0	2
	Jenga	0	1
	Ball	0	1
C	Marker	0	2
	Jenga	0	1
	Ball	0	2
Total not observed positions (out of 36)		**0**	**13**
Percentage of successfully observed positions		**100%**	**63.89%**

occlusion experiments, the conditions did not fully reflect the improvements introduced in the occlusion-aware implementation. In particular, when using the occlusion-aware algorithm, which transitions to the REACHING state once the robot is within 7 cm of the target pose, the trajectory could not be completed in most cases, as the robot would grasp the object prematurely. To address this limitation, the threshold was reduced to 5 cm for both the baseline and the proposed algorithm, specifically for this experiment. This adjustment decreased reactivity but allowed the robot to complete the trajectory in nearly all trials. While lowering the threshold improves safety by entering the grasping state only when nearer, it also carries the risk of losing sight of the hand when approaching closely, even if the hand is still present. For this reason, the 7 cm threshold remains generally preferable. Nonetheless, in this specific case, reducing the threshold ensured that the same rosbags could be reused, thereby enabling a more rigorous and directly comparable evaluation.

This section presents the results of this second comparison and additionally discusses differences observed when evaluating the same algorithm with and without occlusion.

Table 4.6 reports the average pose computation times. The baseline algorithm achieves lower values compared to the proposed approach; in fact, in the occlusion scenario, the proposed algorithm exhibits a tenfold increase relative to the non-occlusion case. Despite this rise, the computation time remains negligible, averaging around one-tenth of a second.

Table 4.7 presents the planning times, where the baseline again shows slightly faster performance. Nevertheless, if the single outlier trajectory (Z-ball) is excluded, the planning times of the two algorithms are largely comparable, and consistent with those obtained in the non-occlusion condition, since the planning time it's not influenced by the worst perception conditions.

As shown in Table 4.8, the overall task completion times are generally similar for both

Table 4.6: Average pose computation time for each object and trajectory (in seconds).

Trajectory	Object	Proposed occluded	Baseline occluded
N	Marker	0.15786	0.03480
	Jenga	0.02872	0.04013
	Ball	0.14783	0.04325
Z	Marker	0.05934	0.04530
	Jenga	0.05995	0.03844
	Ball	0.23612	0.04845
C	Marker	0.19795	0.05820
	Jenga	0.07094	0.02812
	Ball	0.18200	0.06608
Overall Average		**0.12675**	**0.04475**

Table 4.7: Average planning time for robot movements for each object and trajectory (in seconds).

Trajectory	Object	Proposed occluded	Baseline occluded
N	Marker	0.15520	0.12933
	Jenga	0.14558	0.13047
	Ball	0.16680	0.10000
Z	Marker	0.11668	0.15582
	Jenga	0.12765	0.15946
	Ball	0.97743	0.10005
C	Marker	0.11040	0.12803
	Jenga	0.16317	0.09872
	Ball	0.13994	0.12210
Overall Average		**0.23365**	**0.12489**

algorithms, with small variations depending on the trajectory and object. In particular, for both methods, the average total time is approximately two seconds longer than in the non-occlusion case. Moreover, the baseline yields slightly shorter completion times; however, this advantage is partly due to the fact that in two cases (N-marker and Z-ball) the baseline algorithm grasped the object at the third pose instead of the fourth, thereby not completing the full cycle. By contrast, this occurred with the proposed algorithm only in the Z-ball case, leading to a minor time advantage for the baseline.

The most relevant difference emerges in the analysis of not observed poses. Table 4.9 shows the waypoints missed by the two algorithms across multiple runs, with green circles representing the proposed algorithm and red crosses representing the baseline.

As summarized in Table 4.10, the proposed algorithm did not observe only 4 out of 36 target poses, whereas the baseline failed to initiate movement toward 14, more than three times as many. Consequently, the baseline achieved an observation rate of approximately 61.11%, while the proposed algorithm reached 88.89%. Although both

Table 4.8: Total time for each object and trajectory (in seconds).

Trajectory	Object	Proposed occluded	Baseline occluded
N	Marker	34.8586	27.0645
	Jenga	29.4728	28.3241
	Ball	31.0696	32.5819
Z	Marker	43.9972	41.4890
	Jenga	39.8546	54.0389
	Ball	31.0567	33.0165
C	Marker	32.9671	28.5380
	Jenga	37.3119	32.2142
	Ball	42.8312	30.1214
Overall Average		**35.93552**	**34.15428**

Table 4.9: Positions not seen by the Baseline (red cross) and the Proposed (green circle) algorithms considering considering occlusion. Positions 1 to 4 correspond to temporally ordered observation points.

Position	N-marker	N-jenga	N-ball	Z-marker	Z-jenga	Z-ball	C-marker	C-jenga	C-ball
1	○	X					X	X	
2	○X	X	X	X	X	X			
3			○				X	X	X
4	X					○X			

rates are lower than those observed in the non-occlusion setting - as expected - the performance of the proposed algorithm remains considerably stronger.

Overall, these results highlight the advantages of the proposed reactive approach, particularly in reducing the number of missed poses, at the cost of only a few additional tenths of a second in computation. Even though perception worsens under occlusion, leading to slightly longer execution times due to physical obstruction, this second, more realistic experiment confirms the effectiveness of the proposed method over the more discontinuous baseline, favoring continuous tracking and improved responsiveness during dynamic handover tasks.

4.2.2 Experiment 2: Reaching Static Poses - 15 s per Pose

The second comparative experiment aims to further assess the responsiveness and efficiency of the proposed and baseline algorithms in dynamic human-to-robot handover scenarios. Unlike the previous test, which focused on smooth and continuous movements of the human hand, this experiment investigates how effectively each algorithm can react to temporally discrete pose transitions, simulating a user who pauses at predetermined handover points before moving again.

To this end, the system's execution logic was slightly modified for this and the following experiment: the event triggering the robot to grasp the object and return to the home position was disabled. This adjustment allows the robot to approach the human

Table 4.10: Total not observed positions for each object and trajectory considering occlusion.

Trajectory	Object	Proposed occ.	Baseline occ.
N	Marker	2	2
	Jenga	0	2
	Ball	1	1
Z	Marker	0	1
	Jenga	0	1
	Ball	1	2
C	Marker	0	2
	Jenga	0	2
	Ball	0	1
Total not observed positions (out of 36)		**4**	**14**
Percentage of successfully observed positions		**88.89%**	**61.11%**

hand holding the object without initiating the pickup, and therefore without terminating the execution. Instead, the robot remains in the reactive loop, prepared to replan toward any new position as the hand moves across the scene. Like the other experiments, this one was first conducted using the algorithms that processed the video stream directly from the camera, and subsequently with the algorithms that accounted for occlusion. In the latter case, an additional modification was introduced to preserve the robot's reactivity: specifically, from the REACHING state - present only in this implementation - rather than transitioning to the MOVING_DOWN state upon reaching the target object, the system was designed to return to the TRACKING state. This modification ensures that the robot continuously operates within the reactive loop, maintaining its ability to promptly replan and adapt to new target positions as the hand moves through the scene.

This experiment makes use of a different set of rosbag recordings, where the trajectories and objects are the same as those used in the previous experiments - specifically, the N, Z, and C trajectories (depicted in Figure 3.6), each consisting of four temporally ordered target positions marked on the table surface. However, the motion pattern is different: in this case, the human hand holding the object remains stationary for 15 seconds at each of the four cross-shaped target positions. This duration simulates a user intentionally pausing to allow the robot to react and approach the hand, offering a controlled test of the system's capacity to process, plan, and reach a static target before it changes.

The core metric evaluated in this experiment is the number of hand poses successfully reached by the robot while the human hand is still present at the target position. A pose is considered missed if the robot does not reach that position before the 15-second window expires and the hand moves to the next one. This evaluation directly tests the responsiveness and processing efficiency of the system under time-constrained conditions.

Experiment 2: Results without occlusion

The results highlight a stark contrast between the two approaches that doesn't consider occlusion. The proposed algorithm successfully reaches all 36 target positions across the 9 test runs, demonstrating consistent reactivity and efficient integration of continuous perception with motion planning. On the other hand, the baseline algorithm reaches only 21 out of 36 positions, failing to respond in time in the remaining 15 cases. As shown in Table 4.11, this corresponds to a success rate of 58.33%, compared to the 100% achieved by the proposed system.

Table 4.11: Total missed positions for each object and trajectory.

Trajectory	Object	Proposed	Baseline
N	Marker	0	1
	Jenga	0	2
	Ball	0	2
Z	Marker	0	1
	Jenga	0	3
	Ball	0	0
C	Marker	0	2
	Jenga	0	2
	Ball	0	2
Total missed positions (out of 36)		**0**	**15**
Percentage of successfully reached positions		**100%**	**58.33%**

Further insight is provided by Table 4.12, which details the specific positions missed by the baseline algorithm for each combination of trajectory and object. A clear pattern emerges: position 1 is always reached successfully by the baseline system. This can be attributed to the fact that the robot begins execution from the home pose, which eliminates the delay otherwise introduced by having to return home before replanning. Conversely, the baseline system frequently fails to reach position 2 in time. This is likely because, after reaching position 1, the robot has already moved significantly away from the home pose and must travel a greater distance to return before initiating a new plan. This inefficiency stems from its rigid strategy: every time the hand moves, the robot first returns to the home position before computing a new trajectory, resulting in considerable time overhead that prevents timely responses to hand movement.

Table 4.12: Positions not reached by Baseline algorithm. Positions 1 to 4 correspond to temporally ordered observation points.

Position	N-marker	N-jenga	N-ball	Z-marker	Z-jenga	Z-ball	C-marker	C-jenga	C-ball
1									
2	X	X	X	X	X		X		X
3		X	X		X		X	X	
4					X			X	X

These findings underscore the practical advantages of the proposed reactive control

strategy. By eliminating unnecessary returns to home and enabling continuous perception and immediate replanning, the system achieves both greater temporal efficiency and better coverage of all relevant target poses - critical features for achieving seamless and responsive human-robot collaboration.

Experiment 2: Results considering occlusion

The results of the second version of this experiment - conducted using the algorithms described in Sections 3.3 and 4.1.1 - are presented in this section.

As shown in Table 4.13, the proposed algorithm failed to reach only 4 out of 36 poses, compared to 14 missed by the baseline - nearly four times as many. This corresponds to a success rate of 88.89% for the proposed approach versus 61.11% for the baseline, thereby confirming the robustness of our method even in more realistic settings.

Table 4.13: Total missed positions for each object and trajectory.

Trajectory	Object	Proposed occ.	Baseline occ.
N	Marker	0	1
	Jenga	1	2
	Ball	1	2
Z	Marker	0	1
	Jenga	0	2
	Ball	1	2
C	Marker	0	0
	Jenga	1	2
	Ball	0	2
Total missed positions (out of 36)		**4**	**14**
Percentage of successfully reached positions		**88.89%**	**61.11%**

Table 4.14 details the specific poses missed by the proposed (green circles) and baseline (red crosses) algorithms across different object–trajectory combinations. The pose most frequently missed by the baseline is the second one. This can be explained by the fact that execution begins from the home position, eliminating any delay that would otherwise result from returning home before replanning. Consequently, the robot often fails to reach pose 2 in time, missing it in 6 out of 9 trials. Similarly, in the Z trajectory the fourth pose is always skipped, while in the C trajectory the third pose is missed 2 out of 3 times, and in the N trajectory the second pose is missed 2 out of 3 times (even when the first was not reached in time). Notably, all these poses correspond to the same spatial point, located in the lower-right area of the video. At this location, the robot's movement introduces significant occlusion, covering roughly half of the camera view, making the perception of the hand-held object more challenging. Consequently, reaching this pose reliably becomes more difficult, as the system may mistakenly infer that the hand has moved. While the proposed algorithm failed at this location only

twice, the baseline failed 7 times - over three times as often. This difference is explained by the reactive nature of the proposed algorithm: even if perception is momentarily lost, the robot remains in the vicinity and only relocates after more than one second of continuous loss, thereby requiring smaller corrective movements. In contrast, the baseline algorithm returns to the home position every time perception is lost, wasting valuable time before restarting toward the target.

Table 4.14: Positions not reached by the Baseline (red cross) and the Proposed (green circle) algorithms considering occlusion. Positions 1 to 4 correspond to temporally ordered observation points.

Position	N-marker	N-jenga	N-ball	Z-marker	Z-jenga	Z-ball	C-marker	C-jenga	C-ball
1		X	X						
2		X	○X		X	X		X	X
3						○		○X	X
4	X	○		X	X	X			

Overall, the proposed approach demonstrates superior performance even under realistic occlusion conditions, while the baseline achieved a flawless run in only one trial. This highlights the advantages of immediate responsiveness, despite the fact that the proposed strategy can sometimes be more visually obstructive than the baseline. An interesting pattern, however, emerges when comparing performance under occlusion to the non-occluded case. As expected, the proposed algorithm's performance decreases when real occlusion from the robotic arm is considered. Surprisingly, the baseline not only maintains nearly the same performance but even improves slightly, reducing the number of missed poses from 15 to 14. This improvement can be attributed to the modifications introduced alongside occlusion handling, particularly the addition of the REACHING state. Specifically, in the baseline, even small or erroneous hand movements - sometimes occurring when the hand was very close - would previously trigger a return to home, followed by a new movement toward the target. In the improved version, however, when the hand is within 7 cm, the algorithm enters REACHING and continues directly toward the target. Moreover, because the baseline continually returns to home whenever a movement is detected, the camera often captures an unobstructed view of the hand, resulting in perception conditions very similar to the non-occluded case. In conclusion, despite these adjustments, the baseline continues to perform significantly worse than the proposed algorithm, which instead prioritizes replanning from the closest feasible point rather than returning to home - unless doing so is strictly necessary to disocclude the camera view.

4.2.3 Experiment 3: Reaching Static Poses - 10 s per Pose

The third and final comparative experiment builds upon the setup of the previous test, introducing a more demanding temporal constraint for the robotic system. As before, the robot is required to reach the human hand while it holds a static object at predefined

positions along three trajectories - N, Z, and C - using the same set of three objects. However, unlike in the second experiment, the duration for which the hand remains at each observation point is reduced. Specifically, the hand pauses for 15 seconds at the first position to allow the robot to accurately localize the object, but then remains for only 10 seconds at each of the following three positions before transitioning to the next. This setup increases the urgency for the robot to compute the pose, plan the motion, and execute it swiftly enough to reach the hand before it moves again.

The same code adjustments described in the previous paragraph were also applied in this experiment, disabling the grasping action and allowing the reaching loop to continue across all four poses of the trajectory.

The metrics used for evaluation remain consistent with the previous experiments. In particular, we measure how many handover positions the robot successfully reaches within the allotted time at each point. The main goal of this experiment is to assess the responsiveness and efficiency of the two algorithms under tighter temporal constraints.

Experiment 3: Results without occlusion

As shown in Tables 4.15 and 4.16, this experimental condition presents a significantly greater challenge, especially for the baseline algorithm. While the proposed system continues to perform reliably - missing only a single position across all runs - the baseline algorithm experiences a substantial drop in performance, failing to reach 22 out of 36 positions. This drastic decline highlights its inefficiency when faced with tighter time windows, primarily due to the repeated need to return to the home pose before replanning, which introduces a time overhead incompatible with the 10-second constraint.

Table 4.15: Total missed positions for each object and trajectory

Trajectory	**Object**	**Proposed**	**Baseline**
N	Marker	0	1
	Jenga	0	3
	Ball	0	2
Z	Marker	0	3
	Jenga	0	2
	Ball	0	3
C	Marker	0	3
	Jenga	1	3
	Ball	0	2
Total missed positions (out of 36)		**1**	**22**
Percentage of successfully reached positions		**97.22%**	**38.89%**

Interestingly, the trend observed in the previous experiment continues: position 1 is successfully reached in all cases, as the robot begins execution from the home pose and can therefore move directly towards the first handover point without any delay. In

contrast, position 2 is always missed and positions 3 and 4 are frequently missed by the baseline system.

Table 4.16: Positions not seen by the Baseline (red cross) and the Proposed (green circle) algorithms. Positions 1 to 4 correspond to temporally ordered observation points.

Position	N-marker	N-jenga	N-ball	Z-marker	Z-jenga	Z-ball	C-marker	C-jenga	C-ball
1									
2	X	X	X	X	X	X	X	X	X
3		X	X	X		X	X	X	X
4		X		X	X	X	X	○X	

This emphasizes once again the importance of minimizing unnecessary replanning overhead and maintaining responsiveness in scenarios where interaction opportunities are both spatially and temporally constrained.

Experiment 3: Results considering occlusion

When accounting for occlusion, performance decreases once again compared to Experiment 2, where the hand remained in each pose for a longer duration.

Table 4.17 shows that the proposed algorithm consistently outperforms the baseline even under these more realistic conditions. Specifically, the proposed algorithm fails to reach only 8 out of 36 poses, compared to 21 missed by the baseline. This results in success rates of 77.78% for the proposed method and a mere 41.67% for the baseline, which proves to be ineffective when dealing with such short movement durations, primarily due to its reliance on returning to the home pose. The same trend observed in the previous experiment (Section 4.2.2) is maintained here. In fact, the performance of the proposed algorithm decreases under occlusion (from just 1 missed pose to 8), while the baseline remains almost unchanged, with a slight improvement (from 22 to 21 unreached poses).

Table 4.17: Total missed positions for each object and trajectory

Trajectory	Object	Proposed occ.	Baseline occ.
N	Marker	0	2
	Jenga	1	2
	Ball	0	3
Z	Marker	1	3
	Jenga	2	3
	Ball	2	3
C	Marker	1	1
	Jenga	1	3
	Ball	0	1
Total missed positions (out of 36)		**8**	**21**
Percentage of successfully reached positions		**77.78%**	**41.67%**

As illustrated in Table 4.18, even in the occluded case, position 1 is successfully reached across all trajectories, regardless of the initial pose. This confirms that the

robot begins execution from the home pose and can thus move directly toward the first handover point without delay. By contrast, the remaining three positions are each missed 7 times by the baseline, indicating that the issue is primarily temporal: within the 10-second interval, the system often fails to return to the home pose and reach the intended position on time.

Table 4.18: Positions not seen by the Baseline (red cross) and the Proposed (green circle) algorithms considering occlusion. Positions 1 to 4 correspond to temporally ordered observation points.

Position	N-marker	N-jenga	N-ball	Z-marker	Z-jenga	Z-ball	C-marker	C-jenga	C-ball
1									
2	X	X	X	○X	X	X		X	
3		○	X	X	○X	○X	X	○X	X
4	X	X	X	X	○X	○X	○	X	

These results further confirm that, even under more challenging and realistic conditions with occlusion, the proposed algorithm achieves substantially higher reliability and responsiveness than the baseline, demonstrating its effectiveness for dynamic and time-constrained handover tasks.

4.3 Comparative Analysis and Discussion

The experimental results presented in the previous sections provide a clear and consistent comparison between the proposed reactive handover algorithm and the baseline implementation, both with and without considering the occlusion introduced by the robotic arm. Across all test scenarios - ranging from smooth, continuous movements to more temporally constrained, static pose sequences - the proposed system consistently outperforms the baseline in terms of responsiveness, robustness, and coverage of target positions.

In the first experiment, involving continuous hand motion along predetermined trajectories, the proposed algorithm successfully followed the user's hand throughout the entire trajectory in the non-occluded case, and for nearly 90% of the poses under occlusion, always with high accuracy and exhibiting smooth, timely adaptations to movement. By contrast, the baseline approach struggled to keep pace with the moving hand, often missing portions of the trajectory and successfully observing only around 60% of the poses. This limitation stemmed from its rigid return-to-home strategy, which caused delays and missed opportunities. These findings highlight the critical value of continuous perception and on-the-fly replanning in dynamic interaction contexts.

The second experiment further reinforced this observation by introducing temporally discrete poses, with the hand remaining still for 15 seconds at each location. In the non-occluded condition, the proposed system reliably reached all target positions in every run, whereas the baseline algorithm failed to reach a substantial number of poses - particularly those later in the sequence. When occlusion was considered, the performance of the

proposed algorithm degraded, as expected, but still remained approximately 3.5 times better than that of the baseline under the same conditions. The baseline's shortcomings once again stemmed directly from the inefficiency of its rigid motion policy, which forced a return to the home pose before every new movement, leading to unnecessary delays.

The third and most demanding experiment tested the algorithms under stricter time constraints, reducing the available response window from 15 to 10 seconds (except for the first pose). Even under this increased pressure, the proposed algorithm maintained a high success rate, missing only one position out of 36 in the non-occluded case, and eight positions when occlusion was introduced, corresponding to a success rate of 78%. Meanwhile, the baseline approach's performance deteriorated dramatically, failing to reach over 60% of the targets in both conditions. This experiment underscored the scalability of the proposed method, demonstrating that it remains robust and effective even when the temporal window for decision-making is significantly reduced, whereas the baseline approach proved inadequate.

Interestingly, when comparing occluded and non-occluded cases in the second and third experiments, the baseline algorithm did not exhibit the same performance drop observed with the proposed approach. On the contrary, its success rate even showed a slight improvement (from 15 to 14 missed poses in the second experiment and from 22 to 21 in the third). This counterintuitive effect can be explained by the algorithmic modifications introduced alongside occlusion handling, particularly the addition of the REACHING state when the robot is within 7 cm of the object. In this configuration, small or erroneous hand movements no longer force a complete return to home but instead allow the system to continue directly toward the target. Moreover, the baseline's tendency to reset to the home pose upon detecting motion often results in the camera capturing an unobstructed view of the hand, thereby reproducing visual conditions very similar to the non-occluded case. Nevertheless, despite this marginal improvement, the baseline continues to perform significantly worse than the proposed algorithm, which prioritizes local replanning without unnecessary returns to home.

Taken together, these experiments demonstrate that the proposed reactive system enables a more fluid and natural interaction pattern, closely aligned with human expectations in collaborative tasks. By maintaining active perception during motion execution and reacting immediately to environmental changes, the system avoids unnecessary idle time and responds more reliably to user intent - even under challenging, realistic scenarios involving occlusion. These improvements not only enhance task efficiency but also contribute to a more intuitive and human-friendly robot behavior, which are crucial qualities for real-world deployment in shared human-robot workspaces.

Chapter 5

Conclusion and Future Works

The research presented in this thesis addressed the challenge of improving human-to-robot handovers by extending a pre-existing ROS-based pipeline. The original implementation assumed a static interaction model: once a human hand holding an object was detected, the robot planned a single motion, executed it, and returned to a predefined home pose, without considering subsequent movements. While functional in controlled scenarios, this approach lacked the adaptability required for real-world handovers, where small corrections or displacements by the human are frequent. To overcome these limitations, the present work introduced a modified pipeline in which perception remains active during execution. The system continuously monitors the human hand through an RGB-D camera, and whenever the detected pose changes, the current trajectory is stopped and replanned toward the updated target. This design transforms the interaction into a reactive process, enabling the robot to adapt online to the partner's movements and thereby improving the fluidity and robustness of the handover.

Furthermore, to account for more realistic conditions, an occlusion model was developed: the robot's arm was projected into the camera feed through a dedicated node, ensuring that the perception module operated on a view that included self-occlusions. This enhancement revealed new challenges, as perception often failed when the robot's arm obstructed the view of the hand. To mitigate these issues, some key modifications were introduced in the algorithm, such as the addition of a REACHING state that stabilizes the interaction when the robot is close to the target, and a timer-based retreat mechanism to avoid prolonged self-occlusions.

In order to tune the reactivity of the proposed system, a series of calibration experiments were first performed to evaluate four candidate values of the reactive threshold parameter. This parameter determines the minimum displacement of the detected hand that triggers a replanning event. The results indicated that very low thresholds led to unstable behavior, with frequent replanning even in the presence of small detection noise, whereas excessively high thresholds reduced the robot's responsiveness. A value of 3 cm was ultimately selected as the most effective trade-off, ensuring both stability and reactivity.

Once the proposed reactive algorithm was implemented and calibrated, a baseline system was also developed to provide a reference for comparison. In the baseline implementation, the robot does not update its trajectory from the position were it is. Instead, whenever the human hand is detected to have moved, the robot is forced to return to its home pose before planning and executing a new trajectory toward the updated target. This design reflects a more rigid, restart-based approach to handover, and allows a direct validation of the advantages introduced by the proposed reactive algorithm.

The two systems were then evaluated through three experiments, conducted both with and without occlusion, to test their performance in increasingly demanding conditions. Experiment 1 involved continuous, fluid hand motion along predefined trajectories. The proposed algorithm successfully followed the hand in all poses without occlusion, and nearly 90% of the poses under occlusion. The baseline observed only around 60% of the poses due to its return-to-home policy, highlighting the importance of continuous perception and on-the-fly replanning. Experiment 2 used semi-static interactions, with the hand remaining at each pose for 15 seconds. The proposed algorithm reached all poses in the non-occluded case, and its performance under occlusion remained 3.5 times better than the baseline. Experiment 3 tested stricter timing, reducing the window to 10 seconds per pose (except for the first pose). The proposed algorithm maintained a high success rate, missing only one pose without occlusion and eight under occlusion (78% success rate), whereas the baseline failed to reach over 60% of targets. This confirms the proposed method's robustness under tighter temporal constraints. Overall, the proposed reactive system consistently outperforms the baseline, demonstrating the benefits of immediate responsiveness, active perception, and local replanning, even under occlusion or time pressure. In contrast, the baseline's rigid strategy limits its effectiveness in dynamic handover scenarios.

Despite these encouraging outcomes, several limitations of the current system must be acknowledged. First, all experiments were carried out exclusively with a simulated robot. While simulation provides a controlled and repeatable setting, it cannot fully capture the complexity of real-world deployments, particularly in terms of real occlusion and timing of the execution. A second limitation concerns the dataset used for evaluation: the rosbag recordings were collected using only my hand, which restricts the variability of human movement patterns and may bias the system toward a specific style of interaction. Similarly, the set of objects employed in the experiments was limited in both number and diversity, preventing a comprehensive assessment of the algorithm's adaptability to different shapes, sizes, or textures. From a methodological perspective, the perception pipeline also relied on relatively simple hand-tracking strategies. No advanced tracking techniques, such as state-estimation filters or learning-based predictors, were integrated. As a result, the system remained vulnerable to short-term losses of perception, particularly under occlusion. For example, a Kalman filter could be employed to estimate the latent state of the hand and to predict its short-term trajectory

even when visual input is partially missing. By combining sensor observations with predictive models, such methods could smooth noisy detections, reduce the likelihood of spurious reinitializations, and ultimately improve robustness in dynamic and cluttered environments.

Looking ahead, several promising directions for future research can be identified. A natural first step would be to validate the proposed algorithm on a physical robotic platform, thereby exposing it to the full spectrum of real-world uncertainties. Such experiments should involve a wider variety of participants to capture different hand trajectories, speeds, and styles of motion, thereby ensuring the generalizability of the approach. Furthermore, extending the evaluation to a broader set of everyday objects would allow for a more complete analysis of the system's adaptability to varied handover contexts. On the algorithmic side, integrating predictive models-such as Kalman filter-could significantly enhance robustness against occlusion and temporary perception failures. Another avenue for exploration involves multimodal perception, where combining visual cues with other sensing modalities, such as tactile feedback, could provide more reliable estimates of the human partner's state.

In conclusion, this thesis has delivered a fully functional reactive handover system that extends a static ROS-based framework into a dynamic and adaptive pipeline. The developed architecture successfully integrates continuous perception, on-the-fly replanning, and occlusion handling, demonstrating significant improvements over a rigid baseline in all tested scenarios. Despite being evaluated only in simulation, the system consistently achieved high success rates even under time pressure and occlusion, proving both its effectiveness and robustness. These results confirm the value of the proposed design as a strong foundation for future developments, which can further expand its applicability to real-world collaborative robotics.

Bibliography

[1] E. H. Østergaard, "The role of cobots in industry 4.0." `https://info.universal-robots.com/hubfs/Enablers/Whitepapers/Theroleofcobotsinindustry.pdf`, 2017. [Accessed: April 14, 2025].

[2] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, no. 6446, p. eaat8414, 2019.

[3] A. K. Pandey and R. Alami, "Towards human-level semantics understanding of human-centered object manipulation tasks for hri: Reasoning about effect, ability, effort and perspective taking," *International Journal of Social Robotics*, vol. 6, no. 4, pp. 593–620, 2014.

[4] M. Käppler, I. Mamaev, H. Alagi, T. Stein, and B. Deml, "Optimizing human-robot handovers: the impact of adaptive transport methods," *Frontiers in Robotics and AI*, vol. 10, p. 1155143, 2023.

[5] H. Duan, Y. Yang, D. Li, and P. Wang, "Human–robot object handover: Recent progress and future direction," *Biomimetic Intelligence and Robotics*, vol. 4, no. 1, p. 100145, 2024.

[6] V. Ortenzi, F. Cini, T. Pardi, N. Marturi, R. Stolkin, P. Corke, and M. Controzzi, "The grasp strategy of a robot passer influences performance and quality of the robot-human object handover," *Frontiers in Robotics and AI*, vol. Volume 7 - 2020, 2020.

[7] W. Yang, C. Paxton, A. Mousavian, Y.-W. Chao, M. Cakmak, and D. Fox, "Reactive human-to-robot handovers of arbitrary objects," 11 2020.

[8] E. Grigore, K. Eder, A. Pipe, C. Melhuish, and U. Leonards, "Joint action understanding improves robot-to-human object handover," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. tbc, pp. 4622–4629, Nov. 2013. IEEE/RSJ International Conference on Intelligent Robots and Systems ; Conference date: 03-11-2013 Through 08-11-2013.

[9] J. Medina Hernández, F. Duvallet, M. Karnam, and A. Billard, "A human-inspired controller for fluid human-robot handovers," 11 2016.

[10] P. Khanna, M. Björkman, and C. Smith, "Data-driven grip force variation in robot-human handovers," *arXiv preprint arXiv:2303.16009*, 2023. Accessed: April 14, 2025.

[11] V. Ortenzi, A. Cosgun, T. Pardi, W. P. Chan, E. A. Croft, and D. Kulic, "Object handovers: a review for robotics," *CoRR*, vol. abs/2007.12952, 2020.

[12] Franka Emika, "Accessories – franka emika." `https://franka.de/accessories`, 2025. Accessed: 2025-04-16.

[13] H. Duan, P. Wang, Y. Li, D. Li, and W. Wei, "Learning human-to-robot dexterous handovers for anthropomorphic hand," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 15, no. 3, pp. 1224–1238, 2022.

[14] M. G. Catalano, G. Grioli, E. Farnioli, A. Serio, C. Piazza, and A. Bicchi, "Adaptive synergies for the design and control of the pisa/iit softhand," *The International Journal of Robotics Research*, vol. 33, no. 5, pp. 768–782, 2014.

[15] M. Pozzi, S. Marullo, G. Salvietti, J. Bimbo, M. Malvezzi, and D. Prattichizzo, "Hand closure model for planning top grasps with soft robotic hands," *The International Journal of Robotics Research*, vol. 39, no. 14, pp. 1706–1723, 2020.

[16] M. Bianchi, G. Averta, E. Battaglia, C. Rosales, M. Bonilla, A. Tondo, M. Poggiani, G. Santaera, S. Ciotti, M. G. Catalano, *et al.*, "Touch-based grasp primitives for soft hands: Applications to human-to-robot handover tasks and beyond," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7794–7801, IEEE, 2018.

[17] C. Castellani, E. Turco, V. Bo, M. Malvezzi, D. Prattichizzo, G. Costante, and M. Pozzi, "Soft human-robot handover using a vision-based pipeline," *IEEE Robotics and Automation Letters*, 2024.

[18] W. Yang, B. Sundaralingam, C. Paxton, I. Akinola, Y.-W. Chao, M. Cakmak, and D. Fox, "Model predictive control for fluid human-to-robot handovers," 2022.

[19] TurboSquid, "3d model: Franka emika panda robot." `https://www.turbosquid.com/it/3d-models/3d-model-franka-emika-panda-robot-1344347`. Accessed: 2025-08-31.

[20] Y.-W. Chao, W. Yang, Y. Xiang, P. Molchanov, A. Handa, J. Tremblay, Y. S. Narang, K. Van Wyk, U. Iqbal, S. Birchfield, *et al.*, "Dexycb: A benchmark for capturing hand grasping of objects," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9044–9053, 2021.

[21] K. A. Tychola, I. Tsimperidis, and G. A. Papakostas, "On 3d reconstruction using rgb-d cameras," *Digital*, vol. 2, no. 3, pp. 401–421, 2022.

[22] K. Tychola, I. Tsimperidis, and G. Papakostas, "On 3d reconstruction using rgb-d cameras," *Digital*, vol. 2, pp. 401–423, 08 2022.

[23] P. L. Liu, "Single stage instance segmentation — a review," 2020. Accessed: 2025-08-31.

[24] H. Duan, P. Wang, Y. Yang, D. Li, W. Wei, Y. Luo, and G. Deng, "Reactive human-to-robot dexterous handovers for anthropomorphic hand," *IEEE Transactions on Robotics*, 2024.

[25] ImageNet, "Imagenet," 2024. Accessed: 2025-04-20.

[26] Ultralytics, "Yolov8 models," 2023. Accessed: 2025-04-17.

[27] Rafael, "Training a yolo object segmentation model for your needs," 2024. Accessed: 2025-08-31.

[28] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

[29] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[30] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.

[31] J. Docekal, J. Rozlivek, J. Matas, and M. Hoffmann, "Human keypoint detection for close proximity human-robot interaction," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pp. 450–457, IEEE, 2022.

[32] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[33] M. Costanzo, G. De Maria, and C. Natale, "Handover control for human-robot and robot-robot collaboration," *Frontiers in Robotics and AI*, vol. 8, p. 672995, 2021.

[34] G. Averta, "Learning to prevent grasp failure with soft hands: From on-line prediction to dual-arm grasp recovery," in *Human-Aware Robotics: Modeling Human Motor Skills for the Design, Planning and Control of a New Generation of Robotic Devices*, pp. 221–235, Springer, 2022.

[35] M. Dong and J. Zhang, "A review of robotic grasp detection technology," *Robotica*, vol. 41, no. 12, p. 3846–3885, 2023.

[36] Y. Jiang, S. Moseson, and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3304–3311, 2011.

[37] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[38] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," 2017.

[39] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 57, 2018.

[40] M. Roa and R. Suárez, "Grasp quality measures: review and performance," *Autonomous Robots*, vol. 38, no. 1, pp. 65–88, 2015. Published online: July 31, 2014.

[41] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2901–2910, 2019.

[42] W. Yang, C. Paxton, M. Cakmak, and D. Fox, "Human grasp classification for reactive human-to-robot handovers," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11123–11130, 2020.

[43] C. Park, Y. Jeong, M. Cho, and J. Park, "Fast point transformer," 2022.

[44] H. Ran, J. Liu, and C. Wang, "Surface representation for point clouds," 2022.

[45] Y.-W. Chao, C. Paxton, Y. Xiang, W. Yang, B. Sundaralingam, T. Chen, A. Murali, M. Cakmak, and D. Fox, "Handoversim: A simulation framework and benchmark for human-to-robot object handovers," 2022.

[46] G. Zhang, H.-S. Fang, H. Fang, and C. Lu, "Flexible handover with real-time robust dynamic grasp trajectory generation," 2023.

[47] S. Christen, W. Yang, C. Pérez-D'Arpino, O. Hilliges, D. Fox, and Y.-W. Chao, "Learning human-to-robot handovers from point clouds," 2023.

[48] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[49] P. Neranon and T. Sutiphotinun, "A human-inspired control strategy for improving seamless robot-to-human handovers," *Applied Sciences*, vol. 11, no. 10, p. 4437, 2021. Published: 13 May 2021.

[50] W. Yang, B. Sundaralingam, C. Paxton, I. Akinola, Y.-W. Chao, M. Cakmak, and D. Fox, "Model predictive control for fluid human-to-robot handovers," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 6956–6962, 2022.

[51] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," *The International Journal of Robotics Research*, vol. 42, no. 13, pp. 1133–1184, 2023.

[52] M. Wu, B. Taetz, Y. He, G. Bleser, and S. Liu, "An adaptive learning and control framework based on dynamic movement primitives with application to human–robot handovers," *Robotics and Autonomous Systems*, vol. 148, p. 103935, 2022.

[53] Y. Lee, W. Do, H. Yoon, J. Heo, W. Lee, and D. Lee, "Visual-inertial hand motion tracking with robustness against occlusion, interference, and contact," *Science Robotics*, vol. 6, Sept. 2021.

[54] F. Ficuciello, A. Migliozzi, G. Laudante, P. Falco, and B. Siciliano, "Vision-based grasp learning of an anthropomorphic hand-arm system in a synergy-based control framework," *Science Robotics*, vol. 4, Jan. 2019.

[55] Open Source Robotics Foundation, "Robot operating system (ros)," 2025. Accessed: 2025-05-03.

[56] Universal Robots, "Real-time data exchange (rtde) guide," 2024. Accessed: 2025-05-03.

[57] Universal Robots A/S, "Universal robots - sito ufficiale," 2025. Accesso: 3 maggio 2025.

[58] PickNik Robotics, "Moveit studio by picknik robotics," 2024. Accessed: 2025-04-28.

[59] Intel Corporation, "Intel realsense depth camera d415," 2024. Accessed: 2025-07-28.

[60] O. Robotics, "Ros noetic docker images." `https://hub.docker.com/_/ros`, 2020. Accessed: 2025-07-28.

[61] Intel, "Realsense ros package." `https://github.com/IntelRealSense/realsense-ros`. Accessed: 2025-07-28.

[62] N. Corporation, "Cuda docker images." `https://hub.docker.com/r/nvidia/cuda`. Accessed: 2025-07-28.

Ringraziamenti

Prima di tutto vorrei ringraziare il mio relatore, il professor Gabriele Costante, per la sua guida esperta e la disponibilità dimostrata nei miei confronti in questo momento cruciale della mia carriera universitaria. Successivamente, ci tengo a ringraziare la professoressa Maria Pozzi per la sua estrema competenza e attenzione in ogni momento. I suoi consigli illuminanti sono stati determinanti per il successo di questo lavoro. Vorrei poi ringraziare la dottoranda Chiara Castellani per il supporto e la presenza di questi mesi. Infine, desidero ringraziare il professor Francesco Crocetti per la disponibilità continua e per il sostegno offertomi nei momenti di maggiore necessità.

Un pensiero speciale va alla mia famiglia, che non ha mai smesso di incoraggiarmi e di credere in me, spingendomi a dare sempre il meglio. Un grazie di cuore ai miei genitori, che da sempre mi hanno trasmesso l'amore per lo studio e la curiosità verso il mondo. Grazie per avermi sempre permesso di fare esperienze che mi formassero, non solo dal lato professionale, ma anche dal lato umano. Un grazie pieno di affetto va anche a Elio e Celeste: con i loro giochi e le risate condivise negli anni mi hanno insegnato che la vita non è fatta solo di studio, ma anche di leggerezza. Un ringraziamento speciale va infine ai miei nonni, che fin da piccola mi hanno accompagnata con mille giochi, soprattutto di carte, arte fondamentale per poter affrontare questi anni ad ingegneria! So di poter contare sempre su tutti voi. Vi voglio bene.

Un grazie dal profondo del cuore va poi a Dario, che in questi anni mi è stato sempre vicino, anche se non sempre fisicamente. Non sono stati sempre momenti semplici, ma su di te ho sempre potuto contare e pure quando la distanza diventava difficile da gestire tu eri lì a ricordarmi dei momenti più spensierati e di quanto mi vuoi bene. Grazie perché non ti sei mai arreso e, soprattutto per le cose importanti, ci sei stato in ogni momento. Grazie per la tua certezza, e grazie per sopportarmi quando faccio la testona. Grazie per le notti passate alla Mezcaleria, le facce buffe al telefono e le frangette alla Justin Bieber, grazie per le serate passate abbracciati a guardare Boris sul proiettore, per il braccialetto del Napoli, ma soprattutto grazie per i tanti nostri discorsi, piccoli o grandi che siano. Infine, grazie per voler crescere assieme.

Vorrei poi ringraziare i miei compagni di quest'esperienza. Primo tra tutti Mattia, che, anche da un'altra Università, non ha mai mancato di aiutarci tutti, anche su esami che lui stesso non aveva mai visto. In te non ho trovato solo un collega, ma un Amico con la a maiuscola, con il quale fare discorsi profondi, ma anche scannarci sui giochi da tavolo.

Poi ci sono Stefan, Matteo e Sebastiano, che oltre ad avermi insegnato più cose sulle auto di chiunque altro, sono stati dei simpaticissimi compagni di studio (e cactus) in questi due anni, sui quali ho potuto sempre contare. Ringrazio poi Davide, Filippo e Giovanni, con i quali le lezioni si sono fatte più leggere e gli esami più affrontabili, anche se in fondo in fondo forse siamo tutti un po' dei san pietrini. Grazie anche a William, Marilù, Bilaal, Samuele, Alessandro M., Emanuele, Alessandro R., Francesco M., Francesco A., e Federica perché, anche se in questa magistrale ci siamo visti sporadicamente, quelle volte era come se non fosse passato neanche un giorno. Grazie, infine, a Billal & Co. perché in quest'ultimo periodo mi avete fatto morire dal ridere con i vostri discorsi divertenti, ma anche profondamente interessanti. E Billy, arriverà il giorno in cui le capriole saranno il tuo pezzo forte!

Un ringraziamento davvero di cuore va poi a Nathalia, in lei ho trovato una persona sensibile e premurosa, sempre attenta agli altri in qualunque momento. Un grazie sincero va anche a Chiara, Eleonora ed Elena, compagne preziose di questi anni, resi ancora più belli dai nostri aperitivi, dai film trash e dalle passeggiate insieme.

Un immenso grazie va poi a Sofia, che con le sue risate contagiose, i giochi e gli scherzi ha reso più gioiose le mie giornate e col tempo passato insieme mi ha insegnato cose che non si possono spiegare a parole.