Implementation of a Pick-and-Place Robotic System Using Franka Arm and Vision-Based Object Detection

Technical Report - 2025

Azeez Jimoh
Ahmad Reza Zarei
Juha Plosila
Hashem Haghbayan
Department of Computing
University of Turku
February 11, 2025

Abstract

This report presents the design, implementation, and evaluation of a pick-and-place robotic system using the Franka Emika robotic arm and a Stereolabs ZED2 camera for object detection and localization. The system employs 3D depth sensing through the ZED2 camera, mounted on the robot's end-effector, to provide real-time 3D position data of the target object. These coordinates are transformed into the robot's base frame using a calibrated transformation matrix, enabling precise manipulation. The motion planning is executed using the Robot Operating System (ROS) framework, ensuring smooth and collision-free trajectories. The system dynamically updates the object's position during the pick-and-place sequence, allowing for accurate handling even in cases of displacement. Experimental results demonstrate the system's capability to execute high-precision pick-and-place tasks, highlighting its suitability for advanced applications such as precision assembly, warehouse automation, and medical robotics.

Keywords: Pick-and-Place Automation,3D Object Detection, Real-Time Localization, Robotic Manipulation.

Contents

Chapter 1

Introduction

The advancement of robotics and automation has led to significant breakthroughs in tasks that require precision, adaptability, and efficiency. Among these, pick-and-place operations hold a crucial place in a wide array of applications, ranging from manufacturing and logistics to laboratory automation. The Franka Emika Arm (Figure 1.1), a robotic manipulator designed for versatility and precision, features 7 degrees of freedom (DOF) that enable it to perform complex tasks requiring a high level of dexterity and control.In the experimental setup, the robot arm is situated on a Husky unmanned ground vehicle(UGV). This report focuses on the development and evaluation of a pick-and-place robotic system that integrates the Franka Emika arm with a vision-based detection system. Leveraging the capabilities of the Stereolabs ZED2 camera, the system achieves accurate object localization and manipulation.

1.1 Objectives

The primary goal of this project is to design, implement, and evaluate a pick-and-place robotic system that integrates advanced manipulation and vision-based detection technologies. The following objectives outline the key steps undertaken to achieve this goal:

1. Development of Pick-and-Place Robotic System: Design and implement a robotic system by integrating the Franka Emika robotic arm with a vision-based object detection system using the Stereolabs ZED2 camera. This includes ensuring accurate object localization and seamless manipulation.
2. Transformation Matrix Calibration: Establish and calibrate the transformation matrix between the camera frame and the robot's base frame to enable precise mapping of object positions for manipulation.
3. Motion Planning and Execution: Implement motion planning using the ROS to generate smooth, collision-free trajectories for the pick-and-place task.
4. Dynamic Object Position Updating: Develop a real-time system to track and update the object's position dynamically during the pick-and-place sequence, ensuring reliable operation in cases of object displacement.
5. System Testing and Validation: Test and validate the system's performance in a controlled environment to ensure accurate and repeatable pick-and-place operations.

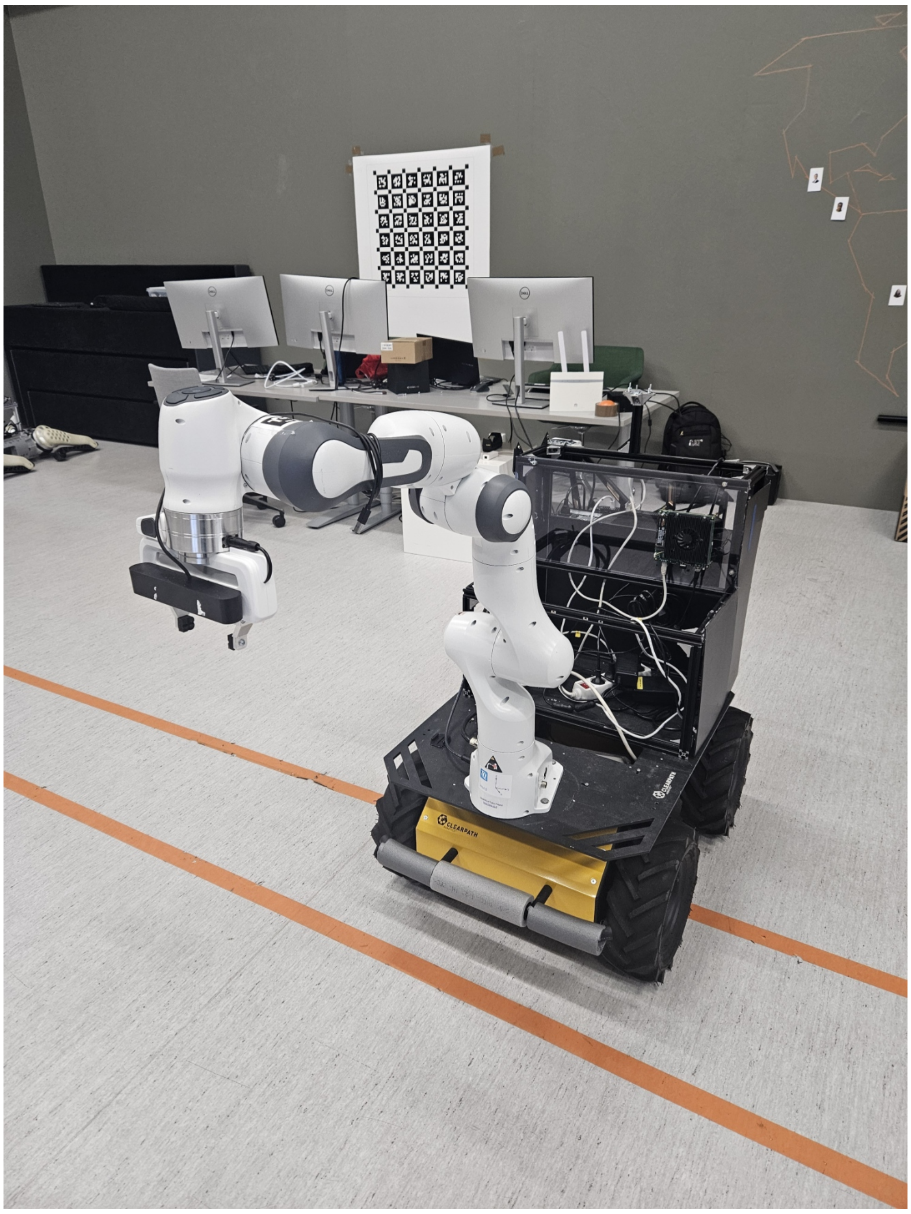


Figure 1.1: The Husky UGV with the Franka Emika Panda Arm.

1.2 Significance

Pick-and-place operations are integral to a variety of industries, offering the potential to enhance efficiency, accuracy, and automation. By utilizing the Franka Emika arm and vision-based detection, this project demonstrates how integrating robotic manipulators with real-time perception systems can overcome challenges such as object misalignment or movement during the process. The system's adaptability and precision make it a valuable solution for applications such as automated assembly lines, warehouse sorting, and laboratory sample handling. Furthermore, the project contributes to the growing field of collaborative robotics, showcasing how robots can perform complex tasks safely and reliably alongside human operators.

1.3 Scope

The scope of this project encompasses the following:

1. Component Integration: Integration of the Franka Emika robotic arm and the Stereolabs ZED2 camera to develop a functional pick-and-place robotic system.

2. Transformation Matrix Calibration: Calibration of the transformation matrix between the camera frame and the robot base frame for accurate object localization.

3. Motion Planning and Execution: Development of collision-free motion trajectories using the ROS to perform precise pick-and-place operations.

4. System Testing and Validation: Testing and validation of the system in a controlled environment to evaluate its accuracy, reliability, and potential for real-world applications.

Chapter 2

System Hardware

2.1 Franka Emika Arm

The Franka Emika Panda, also referred to as the Franka Emika Research 3, is an advanced robotic system that includes a 7-degree-of-freedom manipulator arm equipped with torque sensors at each joint, providing precise force measurements. The arm is designed with exceptional industrial-grade pose repeatability, achieving ± 0.1 mm accuracy. Its innovative design and capabilities make it a versatile tool for both industrial and research applications, offering force sensitivity and unparalleled precision for complex robotic tasks.[2]

The system is enhanced by the Franka Control Interface (FCI), which facilitates a fast and direct low-level bidirectional connection to the arm and gripper via an external workstation connected through Ethernet. As illustrated in Figure 2.1, the FCI enables real-time control with a frequency of 1 kHz through multiple interfaces, allowing gravity and friction-compensated torque commands, joint position or velocity commands, and Cartesian pose or velocity control. It also provides access to detailed robot status data, such as joint positions, velocities, externally applied forces, and collision information, alongside the robot model library for forward kinematics, Jacobian matrices, and dynamic properties. Moreover, the Franka-ROS package integrates the robot with the ROS ecosystem, enabling advanced visualization, kinematic simulations, and gripper control via MoveIt.[2]

To effectively run the FCI and ensure seamless operation with the robot, specific system and network requirements must be met. The workstation PC must operate on Linux with a PREEMPT-RT patched kernel or Windows 10 (experimental) and include a network card supporting 100BASE-TX for high-speed Ethernet communication. To minimize latency, configurations such as disabling CPU frequency scaling are recommended, alongside other optimizations tailored to the system. The workstation must connect directly to the LAN port of the Control, avoiding intermediaries like switches, as these can introduce delay, jitter, or packet loss, significantly impacting performance. To maintain optimal performance, the total time for the round trip between the PC and FCI, the motion generator or control loop execution, and robot data processing must remain under 1 ms. Exceeding this constraint leads to packet drops, and if more than 20 consecutive packets are lost, the robot will halt with a communication error. Direct LAN connections and pre-measuring network performance, including bandwidth, delay, and jitter, are essential to ensure reliable and precise operation.[1]

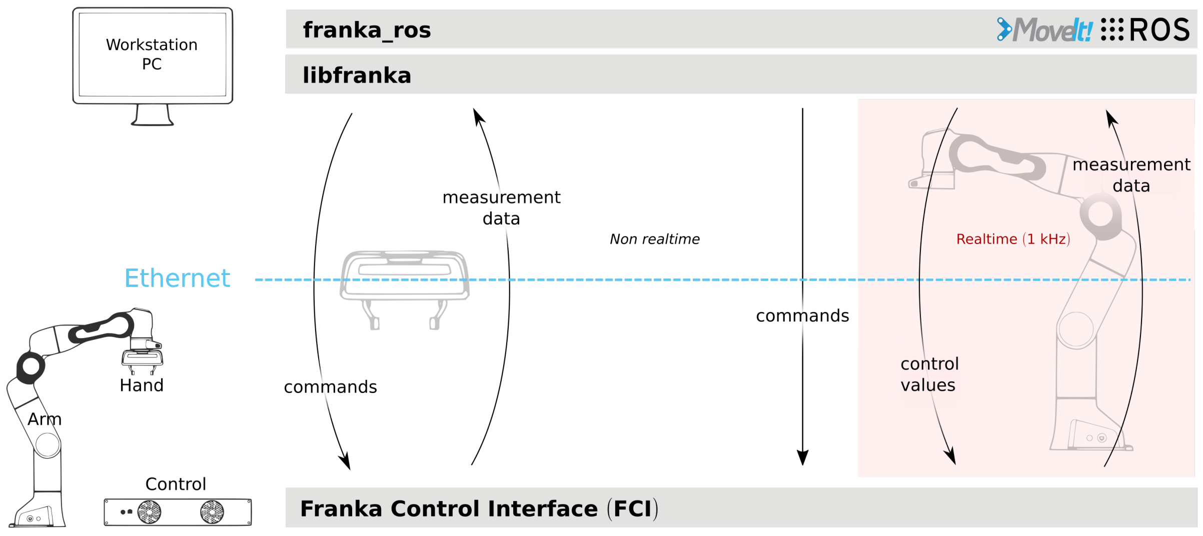


Figure 2.1: Schematic overview of the interaction between franka-ros and FCI for controlling the Franka arm. [2]

2.2 Franka Hand

The Franka Hand, shown in Figure 2.2 , is a highly versatile and fully integrated two-finger parallel gripper developed by Franka Emika, making it an essential component for this project's manipulation tasks. With a continuous grasping force of 70 N, a maximum force of 140 N, and a travel stroke of 80 mm, the Franka Hand can securely grip a wide range of objects with precision and reliability. The Hand's 3 kg payload capacity and lightweight design (0.7 kg) ensure compatibility with various grasping scenarios. In this project, the Franka Hand is crucial for enabling precise and adaptable pick-and-place operations, as its sensitive gripping capabilities allow it to handle diverse object geometries and weights effectively, aligning with the project's goal of achieving reliable and accurate robotic manipulation.

Figure 2.2: Franka Hand.

2.3 Workstation PC

The workstation PC used for control and communication with the Franka arm is an AAEON UPX-TGL01 board from the UP Xtreme Series by Intel As illustrated in figure 2.3, a platform designed to combine flexibility, performance, and scalability for edge computing applications. This host device is powered by an 11th Gen Intel® Core™ i7-1185GRE @ 2.80GHz $\times$ 8 processor, Mesa Intel® Xe Graphics (TGL GT2), 16GB of SO-DIMM memory, and a 250GB Samsung V-NAND SSD. It operates on Ubuntu 20.04.4 LTS, with a kernel compiled specifically for real-time communication with the FCI.[6]

Figure 2.3: AAEON UPX-TGL01 board.

2.4 ZED2 camera

The Stereolabs ZED2 camera, shown in Figure 2.4, is a critical component of this project, enabling precise object detection and tracking. Utilizing its dual-lens setup and triangulation, the ZED2 provides a three-dimensional understanding of the observed scene, including the real-time 3D position of detected objects. This spatial awareness is essential for generating accurate trajectories for the robotic arm to grasp objects effectively. Powered by the ZED SDK, the camera employs optimized algorithms for depth perception, tracking, and spatial AI, ensuring reliable performance in dynamic environments. Additionally, the ZED2 stands out for its versatility and ease of integration, supporting a wide range of platforms from desktop PCs to embedded systems.[5]

Figure 2.4: Zed 2 Camera

2.5 ZED Box Orin

To handle computationally intensive tasks such as object detection and tracking, this project employs the ZED Box Orin™ NX 8GB, shown in Figure 2.5, as the computer vision platform. The ZED Box is specifically designed to process the outputs of the ZED2 camera in real time, enabling applications such as object detection, tracking, and depth estimation. In this project, the ZED Box performs object detection and tracking, processing the camera's output and sending critical information, including the class name and 3D position of detected objects, to the workstation PC. This data is then used by the workstation to generate precise trajectories for the pick-and-place operations. Powered by a 1,024-core NVIDIA Ampere GPU with 32 Tensor Cores and a 6-core Arm® Cortex®-A78AE CPU, the ZED Box delivers 70 TOPS of AI performance, ensuring reliable and efficient processing for real-time applications.

Figure 2.5: Zed Box Orin.

Chapter 3

Implementation

3.1 Hardware System Integration

A host-client communication protocol is implemented to integrate the hardware components of the pick-and-place system (see Figures 3.1 and 3.2). The primary reason for adopting the host-client communication protocol is discussed in Appendix 3. The host device, an AAEON UPX-TGL01 from the Intel UP Xtreme series, is directly connected to the Franka Emika Arm via the Franka Control Interface (FCI). The client system comprises a ZED Box Orin and a ZED2 camera, which are responsible for performing real-time object detection and tracking.

The ZED2 camera captures the scene, and the ZED Box processes the camera's output to detect objects and determine their class and 3D coordinates. This information is transmitted to the host device using a ROS Node. On the host device, the trajectory for the pick-and-place task is generated and subsequently sent to the Franka Arm through the libfranka API for execution. This setup ensures seamless integration and communication between components, enabling precise and efficient pick-and-place operations.

3.2 Software Components

The project utilizes the ROS as the primary framework for communication and coordination among the components. Key libraries include libfranka and franka-ros, provided by Franka Emika, which facilitate low and high-level control of the Franka arm. These libraries enable GUI-based control through ROS Rviz for motion visualization and trajectory planning, as well as API-based control for specifying target coordinates and trajectories directly in code. This flexibility allows precise manipulation and seamless integration of the robotic arm into the pick-and-place system.

The ZED ROS wrapper, developed by Stereolabs, integrates the ZED2 camera with the ROS ecosystem, providing access to features such as object detection, depth estimation, and spatial AI. It retrieves critical object information, including 3D positions and classifications, which are used for trajectory planning and manipulation. Additionally, the hand calibration package is employed to perform eye-in-hand calibration between the ZED2 camera and the Franka Arm, using joint data, camera images, and an ArUco marker to calculate the transformation matrix between the camera frame and the arm's coordinate frame. This calibration ensures accurate alignment of object positions with the robot's base frame, enabling precise pick-and-place operations.

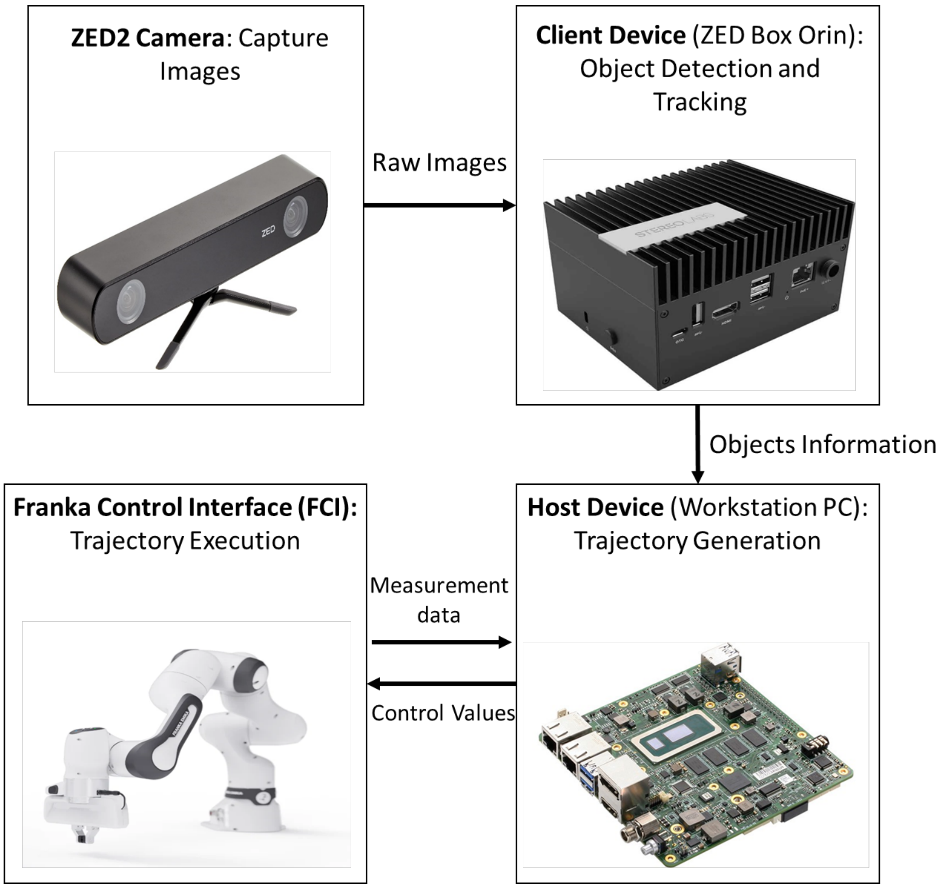

ZED2 Camera: Capture Images
Raw Images
Client Device (ZED Box Orin): Object Detection and Tracking
Objects Information
Franka Control Interface (FCI): Trajectory Execution
Measurement data
Control Values
Host Device (Workstation PC): Trajectory Generation

Figure 3.1: Diagram of hardware integration

3.3 Calibration Process

Calibration is a critical process in robotics to ensure accurate mapping between different coordinate frames, such as the camera, robot, and object frames. It includes intrinsic calibration, extrinsic calibration, and specific setups like eye-in-hand and eye-to-hand calibration, which are essential for precise object detection, localization, and manipulation. Intrinsic calibration determines the internal parameters of a camera that affect how it projects 3D points onto a 2D image plane. These parameters include the focal lengths (f_x, f_y), the principal point (c_x, c_y), and lens distortion coefficients. The camera intrinsic matrix is represented as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where:

- f_x, f_y: Focal lengths of the camera.
- c_x, c_y: Coordinates of the principal point.

Lens distortions are corrected using coefficients for radial distortion (k_1, k_2, k_3) and tangential distortion (p_1, p_2). These parameters are estimated by capturing images of a known pattern (e.g., a checkerboard) and using algorithms such as those provided by OpenCV. Extrinsic calibration involves determining the transformation between the camera frame and an external reference frame, such as the robot base or end effector. This

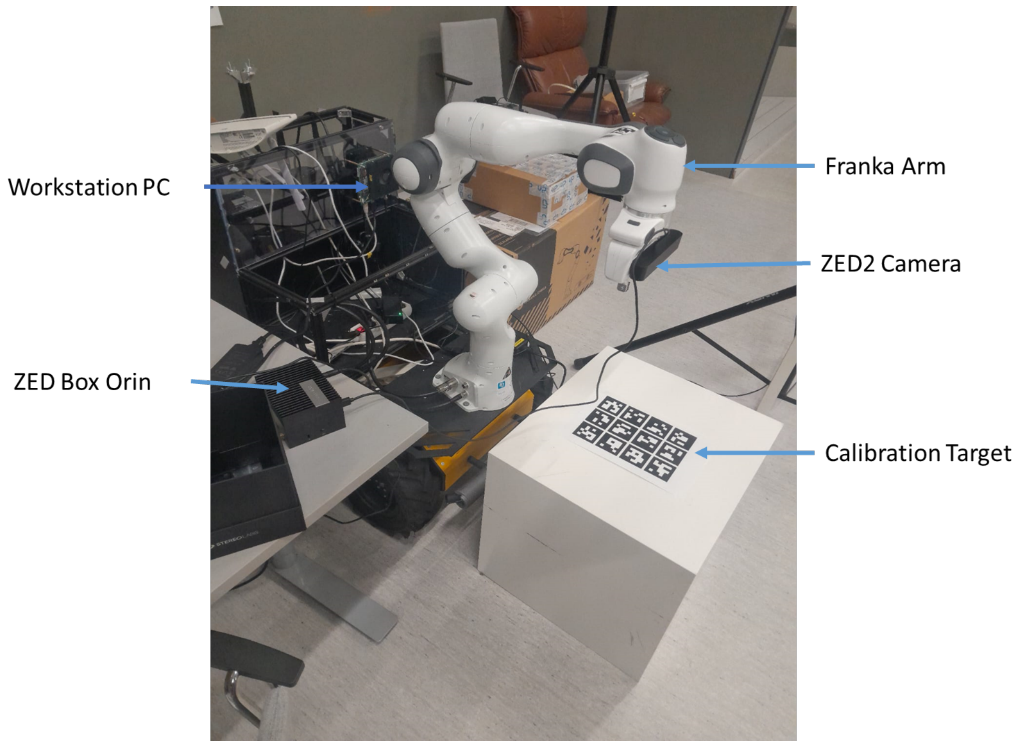


Figure 3.2: Real world hardware integration.

transformation is expressed as:

$$T_{\text{ext}} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

Where:

- R: A 3×3 rotation matrix representing the orientation.
- T: A 3×1 translation vector representing the position.

Extrinsic calibration ensures that the coordinates of objects detected by the camera can be accurately mapped into the robot's workspace, enabling precise manipulation. In an eye-in-hand setup, the camera is mounted on the robot's end effector, moving with the gripper or tool. The goal of eye-in-hand calibration is to compute the transformation $T_{\text{hand\_cam}}$ between the camera frame and the end-effector frame. This is critical for object detection relative to the gripper. The transformation is determined by observing a known target, such as an ArUco marker, from various end-effector poses:

$$T_{\text{hand\_cam}} = T^{-1}_{\text{hand\_base}} T_{\text{cam\_base}}$$

Where:

- $T_{\text{hand\_base}}$: Transformation from the robot base to the end-effector.
- $T_{\text{cam\_base}}$: Transformation from the robot base to the camera.

By moving the robot to various poses and recording the target's position in the camera frame, the transformation $T_{\text{hand\_cam}}$ can be calculated.

In an eye-to-hand setup, the camera is mounted in a fixed position, and the transformation $T_{\text{base\_cam}}$ between the robot base frame and the camera frame is calculated. This calibration enables mapping object positions in the camera's coordinate frame into the robot's base frame. The transformation is expressed as:

$$T_{\text{base\_cam}} = T_{\text{base\_obj}} T_{\text{cam\_obj}}^{-1}$$

Where:

- $T_{\text{base\_obj}}$: Transformation from the robot base to the object.
- $T_{\text{cam\_obj}}$: Transformation from the camera to the object.

By observing a known target from different robot poses, this relationship is computed to align the camera's observations with the robot's workspace.

For calibration, the MoveIt Calibration package is utilized, offering plugins and a graphical interface for performing hand-eye camera calibration. This tool supports both eye-to-hand calibration, where the camera is rigidly mounted to the robot base frame, and eye-in-hand calibration, where the camera is attached to the robot's end effector [3]. Successful calibration requires a camera with accurate intrinsic parameters and a well-defined coordinate frame. The Stereolabs ZED2 camera used in this project comes pre-calibrated with precise intrinsic parameters, eliminating the need for manual intrinsic calibration. Additionally, the ZED ROS wrapper package provided by Stereolabs supplies the coordinate frames required for integration with ROS, further simplifying the calibration process.

The calibration process begins with the MoveIt GUI to create a distinctive target, such as an ArUco marker, featuring a known pattern and size. This allows the pose of the target in the camera's coordinate frame to be estimated [3]. The target is printed and placed under the camera (see Figure 3.2), which is mounted on the end effector of the robotic arm. Frames, including the camera frame, robot base frame, end-effector frame, and target frame, are linked using ROS, with each device publishing its respective topics. During calibration, the robotic arm is moved to various positions, ensuring the target remains visible to the camera at all times. Once the target is detected, samples are taken, and the process is repeated at least five times, with higher accuracy typically achieved after 12 to 15 samples. This iterative approach ensures accurate transformation values, critical for aligning the camera and robotic frames for precise pick-and-place operations.

As detailed in Appendix 1 (final Code for Pick and Place with Franka arm and Zed2 Camera), the function calculate-object-position-robot first determines the transformation of the object position from the camera frame to the hand (gripper) frame. Subsequently, this result is used to calculate the transformation of the object position from the hand frame to the robot base frame. These transformation values are then used to generate a trajectory for the robotic arm, enabling it to perform the pick-and-place task with precision and reliability.

3.4 Object Detection

The ZED Software Development Kit (SDK) utilizes AI and neural networks to identify objects present in both the left and right images captured by the ZED2 camera. It computes the 3D position of each object and their bounding boxes using data from the depth module. Additionally, the SDK provides object tracking capabilities, allowing objects to be tracked over time even when the camera is in motion. The distance of an object from the camera is calculated in metric units (e.g., meters) and is measured from the back of the left lens of the camera to the object in the scene.[4]

The ZED SDK can detect various objects, including people, vehicles, animals, electronics, and fruits/vegetables. In this project, the fruit class was selected to detect specific fruits, such as apples, oranges, and bananas, for pick-and-place tasks. Figure 3.3 shows the results of fruit detection using the ZED SDK along with the 3D positions of the detected objects. The code for detecting these objects and publishing their positions in real time to the host device is provided in Appendix 2. The detected object's position is transformed from the camera frame to the robot base frame, enabling precise pick-and-place operations. If the object's position changes, the updated position is published in real time, allowing the host device to dynamically generate a new trajectory, ensuring precision and adaptability.

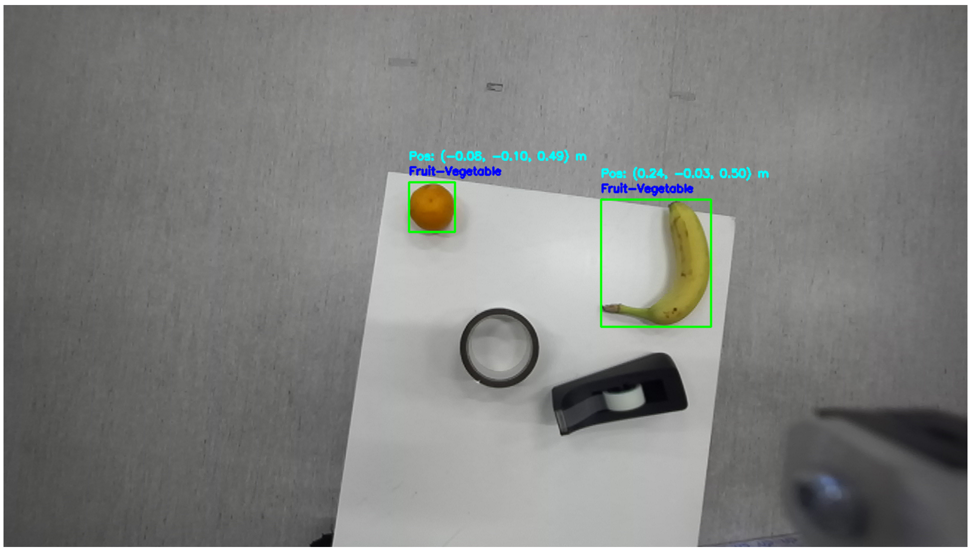


Figure 3.3: Fruit detection and 3D position estimation using the ZED SDK.

3.5 Overview of the Pick-and-Place Process

The pick-and-place system begins with the ZED2 camera capturing the scene, providing real-time stereo vision data. This data is processed using the ZED SDK, which identifies the object's class, bounding box, and 3D position within the camera's coordinate frame. The object detection step ensures that the system can identify specific objects of interest, such as fruits, and accurately determine their location in the environment. The overall workflow of this process is illustrated in Figure 3.4, which provides a detailed view of each step involved in the pick-and-place operation.

Once the object's 3D position is published in the camera frame, it undergoes a coordinate transformation to map its position into the robot base frame. This transformation aligns the detected object's location with the robot's base frame, enabling precise manipulation. Based on the transformed position, the system generates a trajectory for the Franka Arm to execute the pick-and-place operation. The trajectory planning ensures smooth and collision-free movements, facilitating efficient object handling.

During the execution phase, the robotic arm attempts to grasp and place the object at the specified destination. If the object's position changes due to external factors, the system dynamically updates the position using real-time data from the ZED2 camera. These updates allow the trajectory to be recalculated, ensuring the system adapts to changes and maintains precision. After releasing the object at the target position, the arm returns to its designated home position. Both the home position and destination position can be set to any reachable position within the robot's workspace, as detailed in Appendix 1. The implementation of object detection, tracking, and pick-and-place operations is further described in Appendix 1 and Appendix 2. This approach ensures accuracy, reliability, and efficiency, even in dynamic and unstructured environments.

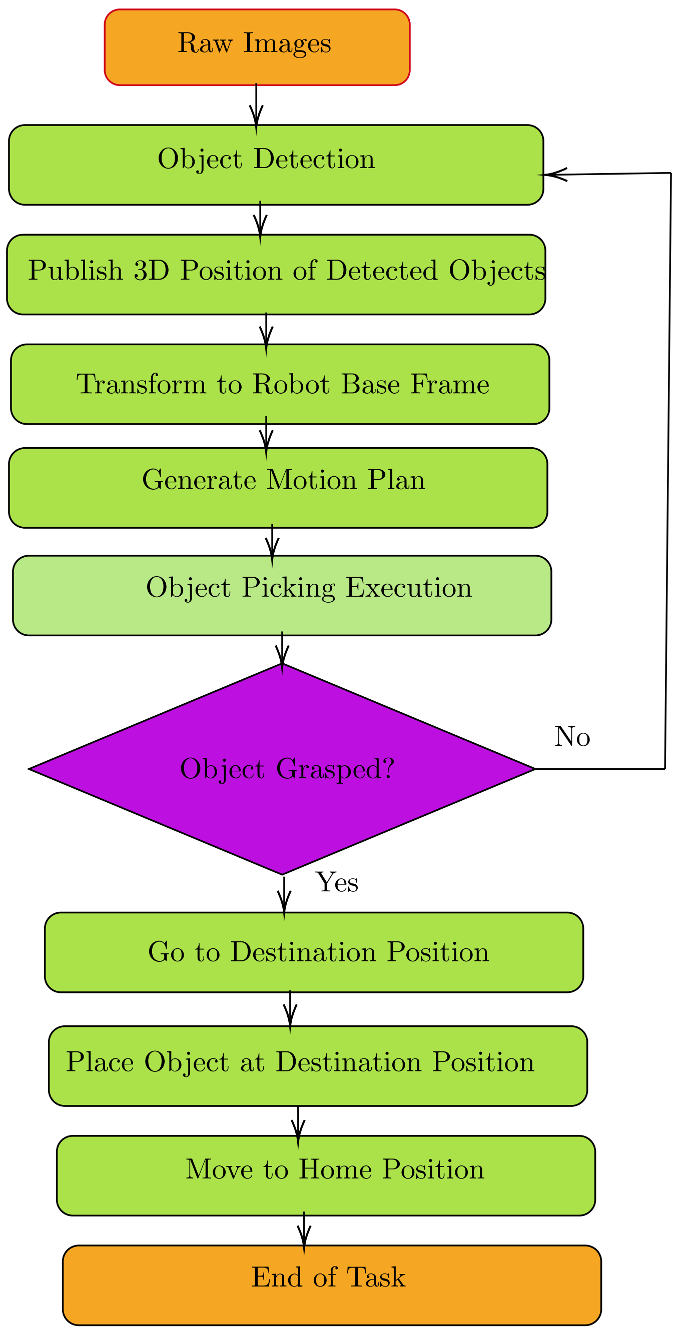

Raw Images
Object Detection
Publish 3D Position of Detected Objects
Transform to Robot Base Frame
Generate Motion Plan
Object Picking Execution
Object Grasped?
No
Yes
Go to Destination Position
Place Object at Destination Position
Move to Home Position
End of Task

Figure 3.4: Flowchart of the pick-and-place process.

Chapter 4

Result

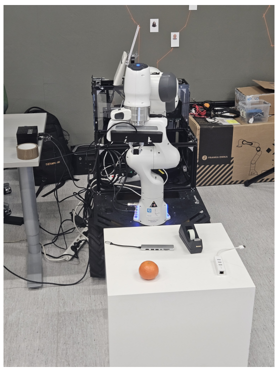
This chapter presents the step-by-step process and results of the pick-and-place operation performed using a robotic arm equipped with an ZED2 camera. The operation demonstrates the integration of object detection, trajectory planning, and motion execution in a dynamic environment. Each step, from initializing the system to detecting, picking, and placing the object, is detailed to provide a comprehensive understanding of the workflow and the robotic system's capabilities.

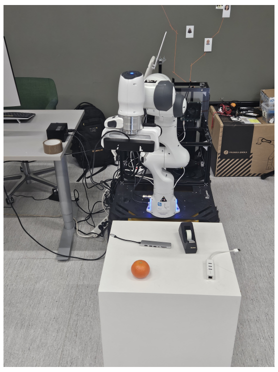
Figure 4.1 illustrates the sequence of the pick-and-place operation for fruit, showcasing the robotic arm's stepwise functionality. The process begins with the starting position (Figure 4.1a), where the robotic system is idle and ready to detect objects within its workspace. At this stage, the ZED2 camera scans the environment to identify any detectable objects, such as fruits. Once an object is detected, the system transitions to the gripper initialization phase (Figure 4.1b). Here, the detected 3D position of the object in the camera's reference frame is transformed into the robot's base frame using a calibrated transformation. Simultaneously, the gripper is opened to prepare for grasping.

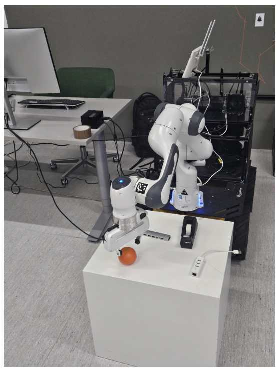
In the picking phase (Figure 4.1c), the robotic arm generates a trajectory based on the transformed position of the object and moves toward it. If the object's position dynamically changes during this phase, the camera detects the updated position, and the trajectory is recalculated in real time, ensuring accurate alignment with the new position. Once the arm reaches the object, the gripper closes to secure the grasp. The system then transitions to the moving phase (Figure 4.1d), where the robot generates a new trajectory to transport the object to the destination. After reaching the target location, the gripper opens to release the object (Figure 4.1e). Finally, the arm returns to its home position (Figure 4.1f), resetting the system for the next pick-and-place operation. This process showcases the robotic system's adaptability and precision in dynamic environments.

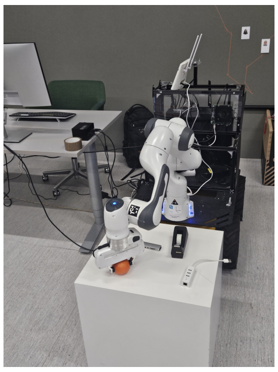
Figures 4.2 and 4.3 illustrate key aspects of the robotic arm's performance during a repetitive pick-and-place operation conducted four times. Figure 4.2 displays the joint movement graphs, highlighting significant variations in certain joints, such as joints 4 and 6, which play critical roles in orienting the end-effector for precise object manipulation. In contrast, in some joints, for example, joint 2, the transitions are smoother and more moderate, reflecting their relatively stable contributions to the motion. The periodic nature of the joint movement graphs confirms the consistency and repeatability of the operation.

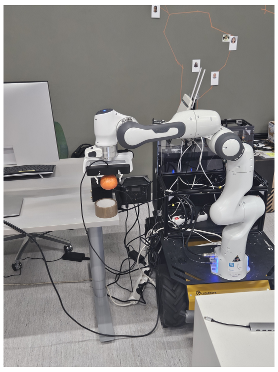
Figure 4.3 depicts the end-effector's trajectory, starting consistently from a fixed initial position, moving to different pick locations, and transitioning to a deterministic end position. The repetitive nature of the trajectory across all operations demonstrates


(a) Starting position


(b) Gripper initialization


(c) Picking the object


(d) Moving to the destination position


(e) Placing the object at the destination position

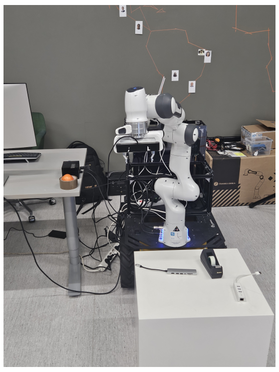

(f) Returning to home position

Figure 4.1: Sequence of the pick-and-place operation for fruit. Subfigures illustrate the key steps of the process, from starting position to object placement and return to home position.

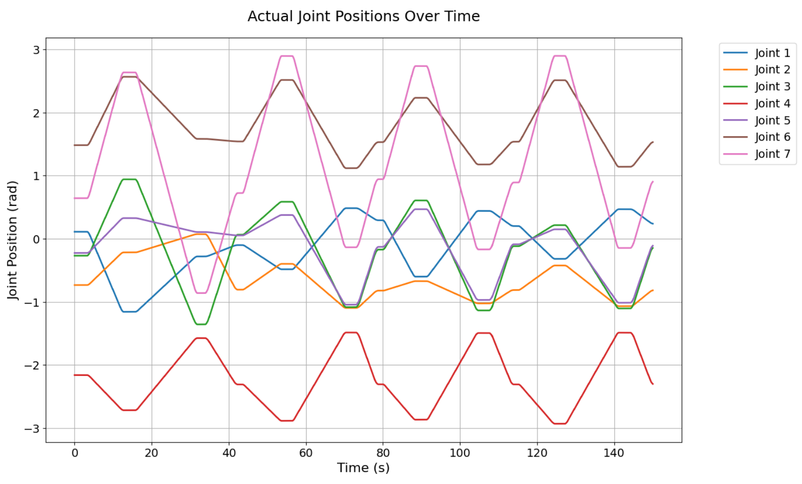


Figure 4.2: Joints position over time.

the system’s reliability and precision in executing dynamic pick-and-place tasks.

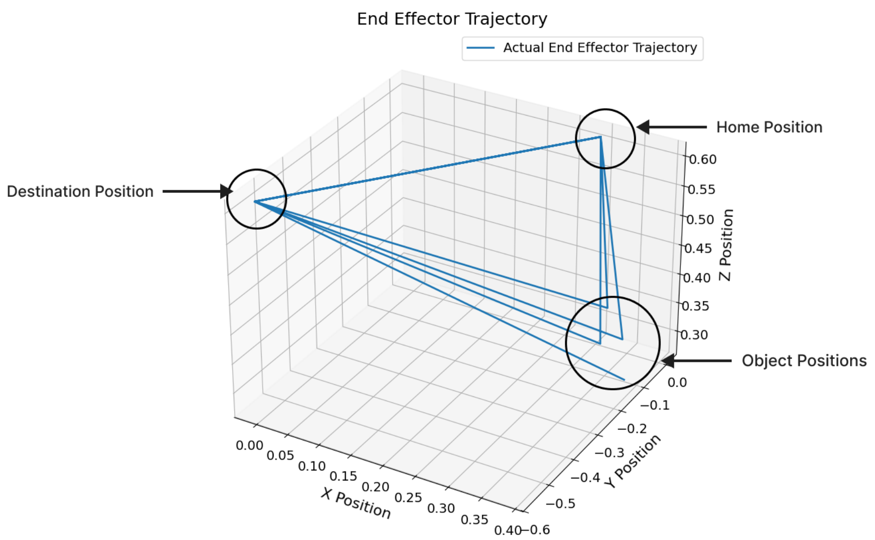


Figure 4.3: Trajectory of end-effector for pick and place.

Chapter 5

Conclusion

This project successfully developed and implemented a pick-and-place robotic system that integrates the Franka Emika Arm with a vision-based detection system using the ZED2 camera. The system leverages real-time object detection, tracking, and coordinate transformation to enable precise manipulation and trajectory planning. By utilizing ROS as the communication framework, the system seamlessly integrates hardware and software components, including the ZED Box Orin for image processing and the AAEON UPX-TGL01 for managing high-frequency communication with the robotic arm. The iterative calibration process and dynamic updates to object positions significantly enhanced the system's accuracy, ensuring reliable pick-and-place execution in dynamic environments.

The outcomes of this project highlight its potential for applications in automated manufacturing, logistics, and laboratory operations. The system's ability to dynamically adapt to changes in object positions and its modular design make it highly versatile for various use cases. Future work could focus on integrating more advanced AI-based algorithms for enhanced detection accuracy, expanding the range of detectable objects, and optimizing the scalability of the system. These improvements would further solidify the system's applicability in real-world robotic automation tasks, providing a robust foundation for continued innovation in autonomous manipulation technologies.

Appendix

.1 Code for Pick and Place with Franka arm and Zed2 Camera

```
##########DEPENDENCIES#############

#!/usr/bin/env python3

import sys
import rospy
import moveit_commander
import tf2_ros
import numpy as np
import csv
from scipy.spatial.transform import Rotation as R
from geometry_msgs.msg import Pose, PoseStamped
from tf2_geometry_msgs import do_transform_pose
from sensor_msgs.msg import JointState
import actionlib
from franka_gripper.msg import GraspAction, GraspGoal, MoveAction, MoveGoal

# Global variables for logging
joint_states = []
last_pose = None
threshold = 0.05  # Movement threshold to trigger the robot motion

#########FUNCTIONS#################

# CSV Writers
def init_csv_writers():
    global joint_writer, ee_writer, event_writer
    joint_file = open("joint_states.csv", "w", newline="")

    joint_writer = csv.writer(joint_file)

    # Write headers
    joint_writer.writerow(["Time", "Joint1", "Joint2", "Joint3", "Joint4", "Joint5", "
        Joint6", "Joint7"])

# Log data
def log_joint_states(msg):
    timestamp = rospy.get_time()
    joint_writer.writerow([timestamp] + list(msg.position))
    joint_states.append((timestamp, list(msg.position)))

# Callback to record joint states
def joint_state_callback(msg):
    log_joint_states(msg)

# Home Position
def home_pos():
    destination_pose = Pose()
    # Set position values
    destination_pose.position.x = 0.2987534986896804
    destination_pose.position.y = 0.0009212556053579501
    destination_pose.position.z = 0.6157771379931122
    # Set orientation values
    destination_pose.orientation.x = -0.924961916293293
    destination_pose.orientation.y = 0.37960735831204023
    destination_pose.orientation.z = -0.016490071471464182
    destination_pose.orientation.w = 0.008472571348736812
    return destination_pose
```

```python
def move_robot_to_home(position):
    group = moveit_commander.MoveGroupCommander("panda_arm")
    group.set_pose_target(position)
    success = group.go(wait=True)
    group.stop()
    group.clear_pose_targets()
    if success:
        rospy.loginfo("Robot successfully moved to the home position.")
    else:
        rospy.logwarn("Robot failed to reach the home position.")

# Destination Pose
def get_destination_pose():
    destination_pose = Pose()
    destination_pose.position.x = -0.007743033245198012
    destination_pose.position.y = -0.5606458367899725
    destination_pose.position.z = 0.614265142713894
    destination_pose.orientation.x = -0.9247894103805966
    destination_pose.orientation.y = 0.38005733197149827
    destination_pose.orientation.z = -0.014763964167281547
    destination_pose.orientation.w = 0.010148705566830401
    return destination_pose

def move_robot_to_destination(position):
    if position is None:
        rospy.logerr("Cannot move the robot: Position is None.")
        return
    group = moveit_commander.MoveGroupCommander("panda_arm")
    group.set_pose_target(position)
    success = group.go(wait=True)
    group.stop()
    group.clear_pose_targets()
    if success:
        rospy.loginfo("Robot successfully moved to the destination.")
    else:
        rospy.logwarn("Robot failed to reach the destination.")

# Gripper control functions
def open_gripper():
    client = actionlib.SimpleActionClient('/franka_gripper/move', MoveAction)
    client.wait_for_server()
    goal = MoveGoal(width=0.08, speed=0.1)
    client.send_goal(goal)
    client.wait_for_result()
    rospy.loginfo("Gripper opened successfully.")

def close_gripper():
    client = actionlib.SimpleActionClient('/franka_gripper/grasp', GraspAction)
    client.wait_for_server()
    goal = GraspGoal(width=0.05, force=20.0, speed=0.1)
    client.send_goal(goal)
    client.wait_for_result()
    rospy.loginfo("Gripper closed successfully.")

def calculate_object_position_robot(x, y, z):
    camera_to_hand_translation = np.array([-0.0362371, -0.0622288, 0.0630396])
    camera_to_hand_quaternion = [0.684326, 0.00118371, 0.729135, 0.00771406]

    object_position_camera = np.array([x, y, z])

    # Transform object position from camera to hand frame
    camera_to_hand_rotation = R.from_quat(camera_to_hand_quaternion).as_matrix()
    camera_to_hand_transformation = np.eye(4)
    camera_to_hand_transformation[:3, :3] = camera_to_hand_rotation
    camera_to_hand_transformation[:3, 3] = camera_to_hand_translation

    object_position_camera_homogeneous = np.append(object_position_camera, 1)
    object_position_hand = np.dot(camera_to_hand_transformation,
        object_position_camera_homogeneous)

    # Transform object position from hand to base frame
    tf_buffer = tf2_ros.Buffer()
    tf_listener = tf2_ros.TransformListener(tf_buffer)

    try:
        rospy.loginfo("Waiting for panda_hand to panda_link0 transform...")
        transform = tf_buffer.lookup_transform("panda_link0", "panda_hand", rospy.Time
            (0), rospy.Duration(5.0))

        hand_to_base_translation = np.array([
            transform.transform.translation.x,
            transform.transform.translation.y,
            transform.transform.translation.z
        ])
        hand_to_base_quaternion = [
            transform.transform.rotation.x,
            transform.transform.rotation.y,
            transform.transform.rotation.z,
            transform.transform.rotation.w
        ]

        hand_to_base_rotation = R.from_quat(hand_to_base_quaternion).as_matrix()
        hand_to_base_transformation = np.eye(4)
        hand_to_base_transformation[:3, :3] = hand_to_base_rotation
        hand_to_base_transformation[:3, 3] = hand_to_base_translation

        object_position_hand_homogeneous = np.append(object_position_hand[:3], 1)
        object_position_base = np.dot(hand_to_base_transformation,
            object_position_hand_homogeneous)

        return object_position_base[:3]
    except Exception as e:
        rospy.logerr(f"Failed to get transform: {e}")
        return None


# Move to target position
def move_robot_to_object(position):
    if position is None:
        rospy.logerr("Cannot move the robot: Position is None.")
        return
    group = moveit_commander.MoveGroupCommander("panda_arm")
    target_pose = Pose()
    target_pose.position.x = position[0] + 0.06
    target_pose.position.y = position[1]
    target_pose.position.z = position[2] + 0.1
    target_pose.orientation.x = 0.0
    target_pose.orientation.y = 1.0
    target_pose.orientation.z = 0.0
    target_pose.orientation.w = 0.0
    group.set_pose_target(target_pose)
    success = group.go(wait=True)
    group.stop()
    group.clear_pose_targets()
    if success:
        rospy.loginfo("Robot successfully moved to the target object.")
    else:
        rospy.logwarn("Robot failed to reach the target object.")

# Post-action sequence: Drop object, return home, and look for another object
def post_action_sequence():
    open_gripper()
    home = home_pos()
    move_robot_to_home(home)

# Callback for object detection
def callback(msg, args):
    move_group, tf_buffer = args
    global last_pose
    detected_pose = msg.pose
    rospy.loginfo("Received detected pose: Position (x: %f, y: %f, z: %f)",
              detected_pose.position.x, detected_pose.position.y, detected_pose.
                  position.z)

    if last_pose is None or np.linalg.norm(
        np.array([last_pose.position.x, last_pose.position.y, last_pose.position.z]) -
        np.array([detected_pose.position.x, detected_pose.position.y, detected_pose.
            position.z])
    ) > threshold:
        last_pose = detected_pose
        object_position_robot = calculate_object_position_robot(
            detected_pose.position.x, detected_pose.position.y, detected_pose.position.z
        )
        if object_position_robot is not None:
            move_robot_to_object(object_position_robot)
            close_gripper()
            destination_pose = get_destination_pose()
            move_robot_to_destination(destination_pose)
            post_action_sequence()
    else:
        rospy.loginfo("No significant movement detected, skipping.")

# Listener setup
def listener():
    moveit_commander.roscpp_initialize(sys.argv)
    move_group = moveit_commander.MoveGroupCommander("panda_arm")
    tf_buffer = tf2_ros.Buffer()
    tf_listener = tf2_ros.TransformListener(tf_buffer)
    rospy.Subscriber("/detected_object_position", PoseStamped, callback, (move_group,
        tf_buffer))
    rospy.spin()

##########MAIN EXECUTABLE###########

if __name__ == "__main__":
    rospy.init_node("pick_and_place_node")
    init_csv_writers()
    rospy.Subscriber("/joint_states", JointState, joint_state_callback)
    open_gripper()
    listener()
```

Listing 1: Python code for pick and place operation

.2 Objects Information Publisher

```
#!/usr/bin/env python

import rospy
from zed_interfaces.msg import ObjectsStamped
from geometry_msgs.msg import PoseStamped

def object_list_callback(msg):
    """
    Callback function to process detected objects and publish their positions.
    """
    global object_pub
    rospy.loginfo("Received object list with {} objects.".format(len(msg.objects)))

    for obj in msg.objects:
        # Skip invalid detections
        if obj.label_id == -1:
            continue

        rospy.loginfo(
            f"Object: {obj.label} [{obj.label_id}], Position: [{obj.position[0]:.2f}, {
                obj.position[1]:.2f}, {obj.position[2]:.2f}]"
        )

        # Prepare PoseStamped message
        object_pose = PoseStamped()
        object_pose.header.stamp = rospy.Time.now()
        object_pose.header.frame_id = "camera_frame"  # Change to appropriate frame ID

        # Set position
        object_pose.pose.position.x = obj.position[0]
        object_pose.pose.position.y = obj.position[1]
        object_pose.pose.position.z = obj.position[2]

        # Default orientation (identity quaternion)
        object_pose.pose.orientation.x = 0.0
        object_pose.pose.orientation.y = 0.0
        object_pose.pose.orientation.z = 0.0
        object_pose.pose.orientation.w = 1.0

        # Publish the object position
        object_pub.publish(object_pose)

def main():
    global object_pub

    # Initialize the ROS node
    rospy.init_node("zed_object_publisher", anonymous=True)

    # Subscribe to ZED2 detected objects topic
    rospy.Subscriber("/zed2/zed_node/obj_det/objects", ObjectsStamped,
        object_list_callback)

    # Create a publisher for object positions
    object_pub = rospy.Publisher("/detected_object_position", PoseStamped, queue_size
        =10)
    rospy.loginfo("ZED object publisher node started. Waiting for object data...")
    rospy.spin()

if __name__ == "__main__":
    main()
```

Listing 2: Python code for object detection and publishing

.3 Reasons for Client-Host Communication

The Franka Emika Arm operates at a 1 kHz signal transmission frequency, which is essential for receiving commands such as joint positions, Cartesian poses, and gravity- and friction-compensated joint-level torque commands. This high-frequency communication enables the arm to provide real-time joint data, including joint positions, velocities, externally applied torques, and forces. Additionally, the system offers detailed information about collisions and contact events, ensuring precise and safe operation during manipulation tasks [2]. To facilitate this communication, the board used for the project initially required kernel compilation to enable real-time scheduling. The ZED Box Orin was the original choice for this role, but it did not support kernel compilation, a limitation confirmed by the board's manufacturer.

This restriction necessitated a switch to a host-client protocol, where the AAEON UPX-TGL01 was configured as the host device. The kernel of the AAEON UPX-TGL01 was successfully compiled to achieve real-time scheduling and enable 1 kHz communication with the arm. However, the AAEON UPX-TGL01 could not be used as the sole board for the entire project, as the ZED2 camera required CUDA (Compute Unified Device Architecture) for proper operation—a feature unavailable on the AAEON UPX-TGL01 due to its Intel-based architecture. As a result, the ZED Box Orin was utilized as the client system for handling image processing tasks, while the AAEON UPX-TGL01 was dedicated to managing communication with the Franka Arm.

Bibliography

[1] Franka Emika. *Minimum System and Network Requirements*. Accessed: January 7, 2025. n.d. URL: https://frankaemika.github.io/docs/requirements.html.

[2] Franka Emika. *Franka Emika Robot Arm - Overview*. Accessed: 2024-09-05. 2024. URL: https://frankaemika.github.io/docs/overview.html.

[3] MoveIt. *MoveIt calibration*. 2025. URL: https://github.com/moveit/moveit_tutorials/blob/master/doc/hand_eye_calibration/hand_eye_calibration_tutorial.rst.

[4] StereoLabs. *Object detection*. Accessed: 2025-01-07. 2025. URL: https://www.stereolabs.com/docs/object-detection.

[5] StereoLabs. *Zed 2 Camera*. Accessed: 2025-01-07. 2025. URL: https://www.stereolabs.com/en-fi/store/products/zed-2.

[6] Up Xtreem. *Up xtreem board*. Accessed: 2025-01-06. 2024. URL: https://www.aaeon.com/en/product/detail/up-series-developer-board-up-xtreme-lite.