THESIS

Umut Ilbay

B.Sc. Mechanical Engineering

Gödöllő

2025

Hungarian University of Agriculture and Life Sciences Szent István Campus

Mechanical Engineering

ROS2 and Python Implementation of Motion, Vision Planning for Franka Emika Panda Robot

Supervisor: **János Tóth**

Author: **Umut Ilbay**

Institute: **Institute of Technology, Engineering Department**

Gödöllő

2025

TABLE OF CONTENT

1. ABBREVIATIONS

ML (Machine Learning)

RL (Reinforcement Learning)

IL (Imitation Learning)

LLM (Large Language Model)

LVLM (Large Vision-Language Models)

ROS (Robot Operating System)

ROS2 (Robot Operating System 2)

DDS (Data Distribution Service)

YOLO (You Only Look Once)

LiDAR (Light Detection and Ranging)

RGB (Red/Green/Blue)

GUI (Graphical User Interface)

XML (Extensible Markup Language)

URDF (Unified Robot Description Format)

DOF (Degrees of Freedom)

HSV (Hue, Saturation, Value)

SDF (Simulation Description Format)

PID (Proportional–Integral–Derivative)

2. INTRODUCTION

In today's rapidly evolving world, where automation and technological advancements influence every aspect of modern industry, robotics plays a key role in operational processes. From manufacturing to healthcare, and especially within logistics and production lines with regards to automation, robots are increasingly responsible for tasks that require precision, efficiency, and reliability. One of the most significant areas within robotics is the manipulation of robotic arms, which enables machines to interact with their environment, perform complex movements, and handle objects with accuracy.

The manipulation of robotic arms involves the precise control of joint movements, end-effector positioning, and trajectory planning, allowing robots to carry out tasks such as assembling products, packaging goods, or conducting rather complex and sensitive procedures. As industries try to optimize productivity, while ensuring workplace safety, and improve the consistency of their operations, the importance of advanced robotic manipulation systems continues to grow.

Despite the progress in this field, challenges still exist regarding flexibility, real-time adaptability, and the safe interaction of robotic arms with dynamic environments. This thesis aims to explore these challenges and try to improve real-time adaptability by adding a camera to the robot to get real-time feedback from the environment as well as the robot itself. With this addition, we will also increase our flexibility regarding more sensitive procedures such as in the medical field.

In the medical field, like the approach demonstrated by Gode et al. (2025), where reinforcement learning and 3D visual feedback were successfully applied to improve the precision of autonomous suturing tasks and got 96.8% needle placement accuracy better than teleoperated robot and manual human surgeons. This study is limited to ROS2 (ROBOT OPERATING SYSTEM 2), and python.We will see how cameras can be implemented in robotic systems to improve in this area.

3. OBJECTIVE

The main objective of this thesis is to design and develop a visual-based robotic manipulation system using ROS2 and Python. The aim is to enable a robotic arm to perceive and interpret its environment through a camera and to utilize this visual information for real-time control and decision-making. By integrating computer vision techniques with robotic control algorithms, the system is intended to detect visual cues such as o color-based features and use them to generate feedback that assists in refining the robot's motion and improving manipulation accuracy. This visual feedback loop will allow the robot to continuously monitor its environment, analyze changes, and made necessary movements, accordingly, thereby enhancing its ability to perform precise and adaptive manipulation tasks in both structured and dynamic settings.

The specific objectives of the thesis are:

1. To design and implement a ROS2-based control system for a robotic arm using Python.
2. To integrate a camera module for detecting visual cues such as object position, orientation, and color-based markers.
3. To develop image processing algorithms for real-time object detection, tracking and feedback using Python libraries specifically OpenCV.
4. To test and evaluate the performance of the robotic arm's visual manipulation capabilities in a simulated or physical environment.

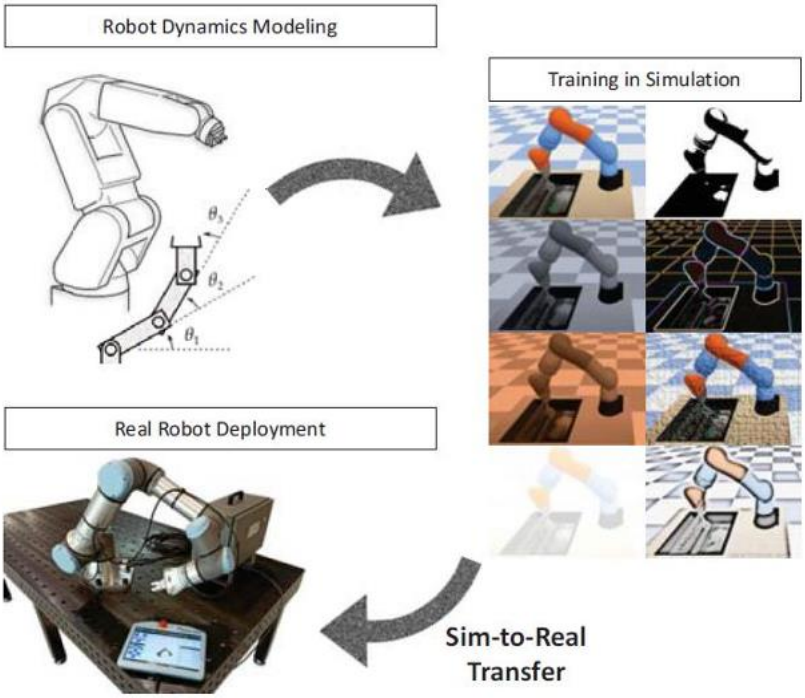
4. LITERATURE REVIEW

In this chapter, an overview of the relevant research and studies related to robotic manipulation, vision-based control, and reinforcement learning is presented. The purpose of this review is to identify existing methodologies, highlight technological advancements, and determine the research gaps that the current work aims to address.

4.1. Introduction to Robotic Manipulation

Robotic manipulation can be defined as the ability of robotic systems to interact with and control objects within their environment by sensors, actuators, and decision-making algorithms. This capability is fundamental for automating complex tasks in dynamic and moving environments, especially those involving object handling, assembly, and human-robot collaboration. Historically, robotic manipulators always used pre-programmed, rigid trajectories, and analytical models of motion and contact. While these approaches were sufficient for pre-determined environments, they failed to use these programs for unpredictable, real-world settings due to their limitation to deal with the environmental uncertainties and dynamic object interactions (Weinberg et al., 2024). From simulation to real life transfer of a robot illustrated in Figure 1.

Figure 1: Conceptual view of a Sim2Real transfer process in reinforcement learning for robotics. Reproduced from Thaker (2024, p. 2).



In response to these limitations, the field has been leaning towards learning-based approaches, particularly for tasks such as in-hand manipulation, where maintaining grasp stability and dexterous control of objects is challenging. Learning-based methods, including

model-based learning, RL, and IL, offer a more adaptive system by helping robots to learn manipulation skills from real-world interactions or simulations. These techniques allow systems to handle task variations and environmental changes without extensive manual programming (Weinberg et al., 2024).

Reinforcement learning has gained importance for its ability to autonomously develop control policies through trial-and-error interactions, requiring minimal human involvement. Imitation learning adds expert demonstrations to train manipulation policies, while model-based methods aim to learn the dynamics and representations of the system to improve manipulation performance. The combined use of these approaches has improved robotic in-hand manipulation by optimizing dexterity, improving task success rates, and making it possible to use robotic manipulation in complex environments (Weinberg et al., 2024).

Recently, the integration of large language models (LLMs) and large vision-language models (LVLMs) into robotic manipulation has further expanded the capabilities of robotic systems in perception, decision-making, and planning. These models allow robots to interpret natural language instructions, recognize, and reason objects in their environment, and plan complex tasks. However, this increased reliance on intelligent agents has introduced new challenges, including security vulnerabilities such as adversarial and backdoor attacks, which can compromise robotic operations in both digital and physical environments (Wang et al., 2024). Recent work like Haresh et al. (2024) shows how compositional language–vision skills can scaffold robotic perception and planning, strengthening the case for integrating LVLMs in manipulation pipelines.

Overall, robotic manipulation has evolved from rigid, deterministic systems to highly adaptive, learning-based systems capable of addressing the variability of real-world tasks. This literature highlights the progress and persistent challenges in enabling robots to achieve human-like dexterity and reliability in manipulation tasks.

4.2. In-Hand Manipulation and Visual Feedback

In-hand manipulation is a fundamental challenge in robotics, including the capacity of a robot's hand or end-effector to adjust an object's position and orientation within the grasp, often while maintaining continuous or non-continuous contact. As mentioned in the comprehensive review by Weinberg et al. (2024), in-hand manipulation is essential for achieving the high level of dexterity observed in human hands, especially for complex tasks such as reorienting, assembling, or handling objects in unsystematic environments. Early attempts in this area focused on analytic modeling of motion and contact; however, such methods proved insufficient in handling real-world complexities and uncertainties. As a result, the field has moved toward learning-based approaches such as RL and IL which offer the adaptability needed for dynamic tasks (Weinberg et al., 2024).

Weinberg et al. (2024) provide a categorization of in-hand manipulation, distinguishing between dexterous and non-dexterous approaches. Dexterous in-hand manipulation involves

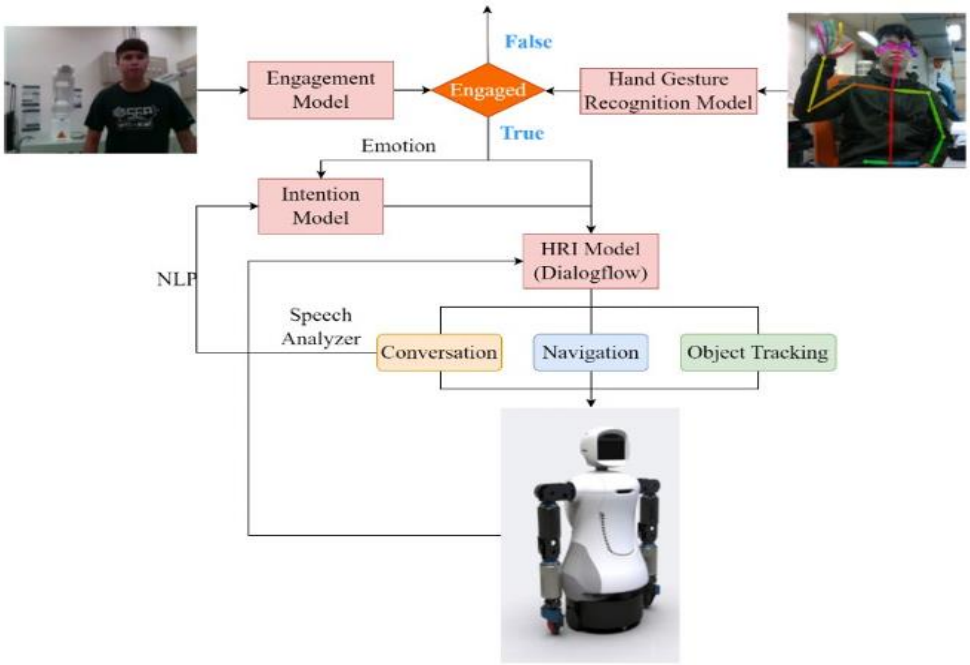
multi-fingered, human-like hands capable of rolling, pivoting, sliding, and in-grasp repositioning objects, generally requiring hands with a high number of degrees of freedom. In contrast, non-dexterous approaches utilize simpler parallel grippers or underactuated hands, often relying on external methods such as pushing objects against surfaces or regrasping through arm motion to achieve similar effects (Weinberg et al., 2024). Different grippers for various tasks shown in Figure 2.

Figure 2: Examples of dexterous and non-dexterous robotic hands. Reproduced from Weinberg et al. (2024, p. 7),

Visual feedback is a very important concept for modern in-hand manipulation systems, providing essential information for real-time correction of manipulation strategies. Robust visual perception enables robots to detect and track objects, maintain grasp stability, and respond adaptively to changes or uncertainties in the environment. According to Orozco-Soto and González Dorantes (2024), integrating a camera-based visual feedback loop with robotic manipulator control can significantly reduce joint position errors, especially for robots using low-cost actuators that does not have any internal feedback sensors. Their study demonstrated that visual feedback implemented as a secondary closed-loop controller successfully adjusted for the effects of gravity, friction, and mechanical backlash, enabling more precise and robust

manipulation (Orozco-Soto and González Dorantes, 2024). A visual feedback model that uses human-robot interaction framework is shown in Figure 3.

Figure 3: Visual feedback control loop for a robot manipulator. Reproduced from Orozco-Soto and González Dorantes (2024, p. 3)



The literature distinguishes between various strategies for vision-based control, or "visual servoing." As detailed by Weng et al. (2022), visual servoing is typically categorized as position-based, image-based), or hybrid reconstructs the 3D pose of the object relative to the camera and uses this information for control but is sensitive to camera calibration errors. , in contrast, relies on features extracted directly from the image plane and is more robust to calibrate errors, though it requires careful design of control mappings. Hybrid visual servoing aims to combine the strengths of both approaches, using both 2D and 3D cues to achieve robust and flexible control (Weng et al., 2022).

Robustness in visual perception is also highlighted by the challenges resulting from blockage, changing lighting, or background interference. For in-hand manipulation, algorithms must reliably extract, and track features needed even under partial visibility or environmental variability. Weng et al. (2022) describe systems where feature extraction, segmentation, and visual tracking are implemented using libraries such as OpenCV, and then integrated with the robot control system for closed-loop operation. This integration of visual feedback is critical not only for routine operation, but also for the system's ability to adapt in real-time to unexpected events or disturbances.

In summary, in-hand manipulation is fundamentally dependent on robust visual feedback, which supports dexterous object handling, enables adaptation to uncertainties, and enhances overall system resilience. The integration of cameras, real-time image processing, and closed-loop control remains central to achieving high levels of performance in robotic

manipulation, as demonstrated in the most recent research (Weinberg et al., 2024; Orozco-Soto and González Dorantes, 2024; Weng et al., 2022; Wang et al., 2024).

4.3. Python and ROS2 for Robotic Manipulation

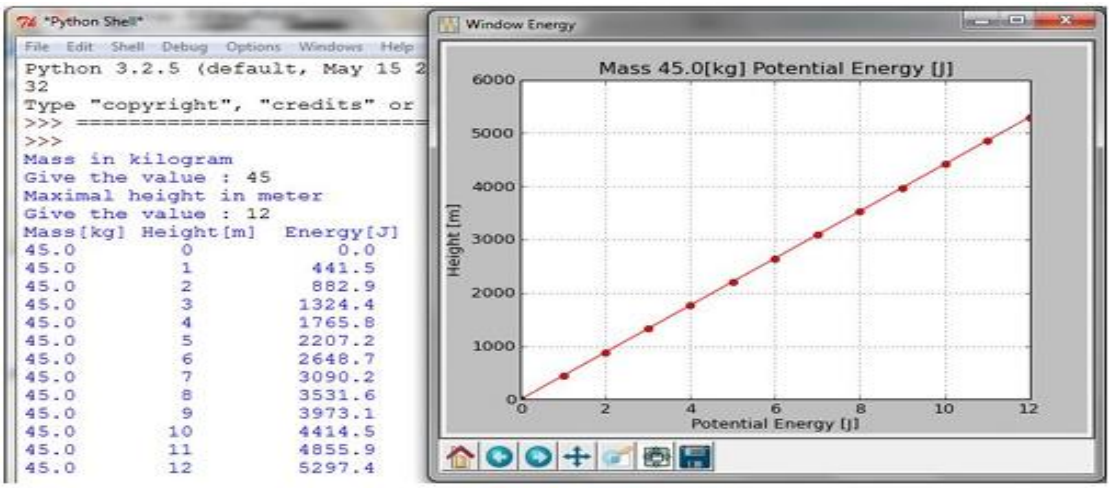
Python and the Robot Operating System (ROS), as lately more ROS2, have become important in the current development of robotic manipulation systems, providing an accessible, powerful framework for both educational and research usage.

The usage of Python as a primary development language in robotics is motivated by its simplicity, readability, and the large number of libraries available for all kinds of tasks such as computer vision, control, and ML. ROS2,improved version of ROS, add new necessary upgrades for communication, real-time performance, and modularity qualities that are crucial for robust, scalable, and responsive manipulation systems.

4.3.1. Python in Robotics

Python's influence and superiority in robotics education and development is well known. Fraanje et al. (2016) describe the broad integration of Python in mechatronics and robotics education, highlighting its role in supporting tasks varying from basic algorithm development to complex simulation, visualization, and control. The language's object-oriented features, effortlessness of integration with other programming languages (such as C and C++), and platform independence have made it a standard choice for teaching and rapid prototyping. The authors highlight that Python, in conjunction with specialized libraries such as OpenCV for computer vision, python-control for control systems, and scikit-learn for machine learning, enables students and engineers to implement and test complex algorithms without steep learning curves or licensing restrictions (Fraanje et al., 2016). Figure 4 demonstrates how Python can be integrated into robotics education to visualize fundamental physical concepts such as potential energy computation and plotting.

Figure 4: Python integration across robotics education stages. Reproduced from Fraanje et al. (2016, p. 2)



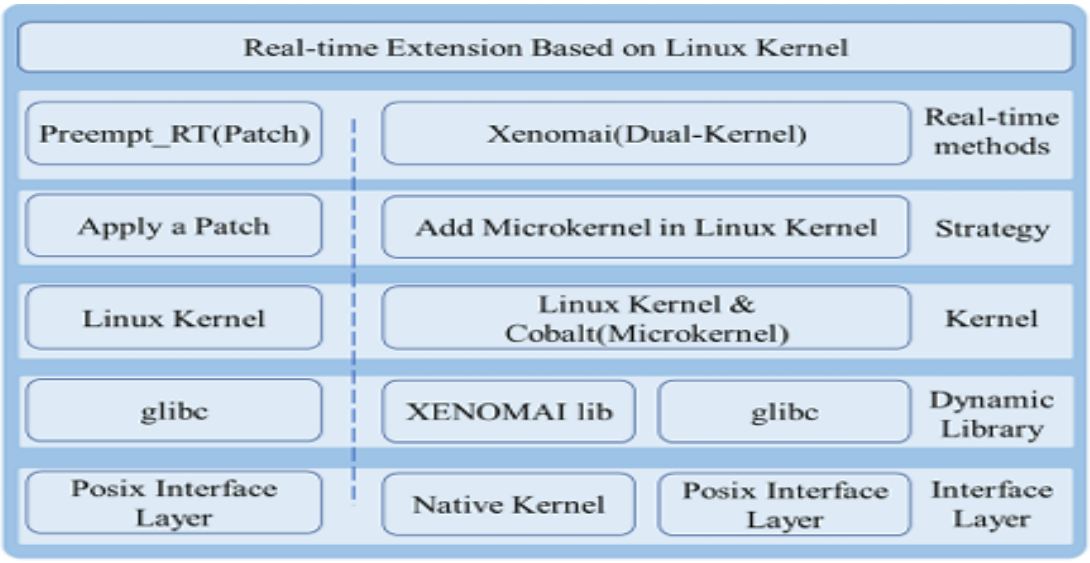
Similarly, Kumar and Sreenivasulu (2019) demonstrated the use of Python for solving inverse-kinematics problems, confirming its practicality for implementing analytical and numerical control algorithms in manipulator design. Building on Python's adaptability, Moghadam et al. (2013) introduced domain-specific languages for modular robotic systems, demonstrating Python's suitability for creating flexible and reconfigurable control architectures.Python's modularity is particularly advantageous for robotics, allowing engineers to build versatile code components for perception, planning, and actuation. Nedelcu and Latinovic (2025) emphasized Python's expanding ecosystem for robotics interfaces, showing how frameworks such as wxPython and Streamlit enable intuitive visualization and monitoring of robotic processes. This modularity supports the repeated process of experimentation and refinement that characterizes robotics research. As noted by Blank et al. (2003), Python enables exploration "beyond LEGOs," empowering users to design and program custom robotic platforms with greater flexibility than traditional commercial kits. Their work illustrates the use of Python in developing multi-agent robotic systems, stressing that Python's clean syntax and extensive library support reduce development time and facilitate debugging and collaborative research (Blank et al., 2003).

4.3.2. ROS2: Real-Time Performance and Modularity

ROS has become the standard for developing distributed, modular robotic systems. ROS2 was designed to overcome some of the limitations of ROS, especially regarding real-time operation, multi-platform compatibility, and secure, scalable communication (Ye et al., 2023). One of the most critical requirements for manipulation systems, especially those integrating visual feedback, is the ability to process sensor data and update control commands in real time. Real-time performance makes sure that the robotic arm can respond without delay to dynamic changes in the environment.

Ye et al. (2023) provide a thorough evaluation of ROS2's real-time capabilities, describing how the system architecture leverages Data Distribution Service (DDS) and middleware optimizations to improve response time and reliability. The authors go back and forth about the importance of kernel-level real-time patches, such as Preempt_RT, for minimizing jitter and achieving predictable communication and computation cycles. Their evaluation compares native and optimized ROS2 architectures, demonstrating that performance improvements such as reduced response time and increased data transfer rate are essential for tasks like visual servoing, where delay or unpredictability can lead to instability or task failure (Ye et al., 2023). Figure 5 shows the real-time extension methods based on the Linux kernel for ROS 2, comparing the Preempt_RT patch and the Xenomai dual-kernel approaches.

Figure 5: Real-time extension methods based on the Linux kernel for ROS2. Reproduced from Ye et al. (2023, p. 3)



The structure of ROS2 allows for the separation of perception, planning, and actuation processes into independent nodes, each potentially running on different hardware. This modularity is particularly advantageous when integrating computationally intensive tasks such as image processing, as these can be offloaded to separate nodes without affecting real-time control loops. ROS2's support for multi-node architectures, robust Quality of Service policies, and cross-platform compatibility (including microcontrollers via micro-ROS) extends its applicability to a wide range of robotic arms and environments (Ye et al., 2023).

4.3.3. Integration of Python and ROS2

The cooperation between Python and ROS2 is an exceptionally good base for vision-based learning systems. Python's ROS2 client libraries provide straight forward interfaces for subscribing to camera streams, publishing control commands, and integrating with perception and planning modules. This integration speeds up development and helps with rapid prototyping, as highlighted by Fraanje et al. (2016), who report the success of Python-based curricula in helping students to quickly progress from simulation to real hardware.

Python's compatibility with computing libraries (NumPy, SciPy), machine learning frameworks (scikit-learn, TensorFlow, PyTorch), and visualization tools (matplotlib,) allows developers to develop perception, decision-making, and actuation all within a single language (Fraanje et al., 2016). This is particularly valuable for research projects aiming to develop and test new algorithms for object detection, tracking, and closed-loop control, which can be implemented and assessed in simulation before deployment on physical robots.

Elsner (2023) further illustrated the effectiveness of Python in real-world robot programming by integrating the Franka Emika Panda with ROS, highlighting Python's strength in achieving seamless communication and motion contr Addition to that Python and ROS2

together enable compatibility with other technologies essential for manipulation tasks, such as Ether CAT-based motor controllers and real-time data acquisition systems (Ye et al., 2023). This flexibility supports integration with industrial robotic arms, mobile manipulators, and even multi-robot systems. The open-source nature of both Python and ROS2 helps to have a huge community, providing extensive documentation, code repositories, and huge libraries. And user support. This collective knowledge base speeds up problem-solving and supports consistent results in robotics research. Blank et al. (2003) note that access to open-source tools and libraries enables even small research groups or educational institutions to take part in state-of-the-art robotics development without extra costs or restrictions.

4.4. Visual Cue Detection Object Localization, Pose, and Tracking

Vision is one of the important types of information for dynamic environments such as autonomous systems, enabling robots to perceive, localize, and interact with objects in dynamic and unorganized environments. The main idea behind gathering this environmental information is through camera systems, which allow for adaptable sensing that surpass many limitations of fixed or proprioceptive sensors. The effectiveness of a manipulation system is related to its capability to detect relevant visual cues, accurately estimate object position, and reliably track cues during motion or disturbance.

Recent implementations in robotics research showcase a varying range of visual perception architectures that blend classical image processing with state-of-the-art learning-based techniques. In the work of Weng et al. (2022), a dual-arm mobile robot is equipped with a visual perception system capable of handling human-robot interaction tasks, including object detection, gesture recognition, and visual servoing. This system demonstrates the value of combining multiple visual tasks into a unified perception pipeline. Hand-gesture recognition, for example, is accomplished by extracting key-point features and employing a voting mechanism to robustly interpret human intent. For object tracking and manipulation, the system utilizes YOLO, a deep-learning-based object detector, for initial identification, and then maintains object pose and geometry information using a hybrid, model-based tracking approach. The integration of deep learning and model-based methods provides robustness to partial blockages and variable lighting, which are typical in real-world settings (Weng et al., 2022).

Marker-based approaches, such as attaching colored or geometric markers to objects, remain common in laboratory and industrial applications where environmental conditions can be very controlled and precise unlike the real-world. These markers are easily detected using standard computer vision techniques and can provide highly accurate position and orientation estimates, especially when used in conjunction with calibrated camera setups. For example, Wen et al. (2024) describe a high-precision positioning system for composite robots that employs camera-based visual feedback to refine the end-effector's pose after coarse navigation with LiDAR. The system uses visual markers to precisely locate objects and make fine adjustments, achieving sub-millimeter accuracy critical for industrial automation and precision

assembly tasks (Wen et al., 2024). In Figure 6, a composite robot is shown integrating a LIDAR and camera system for precision feedback and navigation.

Figure 6: Physical setup of composite robots integrating LIDAR navigation and camera-based precision feedback. Reproduced from Wen et al. (2024, p. 2),

However, reliance on markers can be very often impractical or not so wanted in many real-world scenarios, such as logistics, service robotics, or healthcare, where objects may be unmarked or may change frequently. Marker-less detection strategies address this limitation by leveraging machine learning and advanced image processing to recognize and localize objects based on inherent visual features such as shape, color, or texture. Weng et al. (2022) demonstrate the use of YOLO for fast, real-time object detection that does not require special preparation of the workspace. The output of the object detector is then used as input to a tracking module, which maintains a continuous estimate of the object's position and orientation throughout the manipulation sequence.

Underlying these perception systems is a robust pipeline of image processing and feature extraction. Fraanje et al. (2016) highlights the critical role of Python and its libraries, particularly OpenCV, in enabling flexible and accessible image processing for robotics. Tasks such as filtering, segmentation, contour detection, and object tracking are all implemented efficiently using these libraries, making it possible to process real-time video streams and extract actionable geometric information (Fraanje et al., 2016). Feature extraction methods allow the identification of key points, centroids, and contours, which serve as the foundation for pose estimation. Pose estimation itself may include combining two-dimensional feature locations with geometric models or prior knowledge of the object's shape to infer three-dimensional position and orientation.

Visual tracking, the continuous process of updating the estimated position of objects, is especially important for closed-loop control. In scenarios where the robot or the environment is subject to movement, vibration, or blockage, robust tracking ensures that the robot maintains situational awareness and can adjust its actions dynamically. Weng et al. (2022) show that merging deep-learning-based detection with model-based tracking results in a system that can follow objects even when they are briefly hidden or change appearance. This tracking is essential for visual servoing applications, where the robotic controller repeatably adjusts movement in response to the changing visual information.

Integration with other sensory modalities can further enhance the precision and reliability of visual cue detection. Wen et al. (2024) provide an example where LiDAR is used for large-scale navigation and environment mapping, while cameras are dedicated to high-precision pose correction and manipulation. By combining these sensory modalities, the system can overcome the limitations of any single sensor and maintain robust performance in complex, changing, or cluttered environments.

In summary, the ability to detect, localize, and track visual cues is a foundational part of modern robotic manipulation. The literature illustrates that successful manipulation systems depend on the careful combination of classical image processing, deep learning, model-based estimation, and multi-sensor integration. These advances allow robots not only to identify and localize objects with high accuracy but also to adapt their manipulation strategies on the fly, maintaining performance in the face of real-world uncertainties (Weng et al., 2022; Wen et al., 2024; Fraanje et al., 2016).

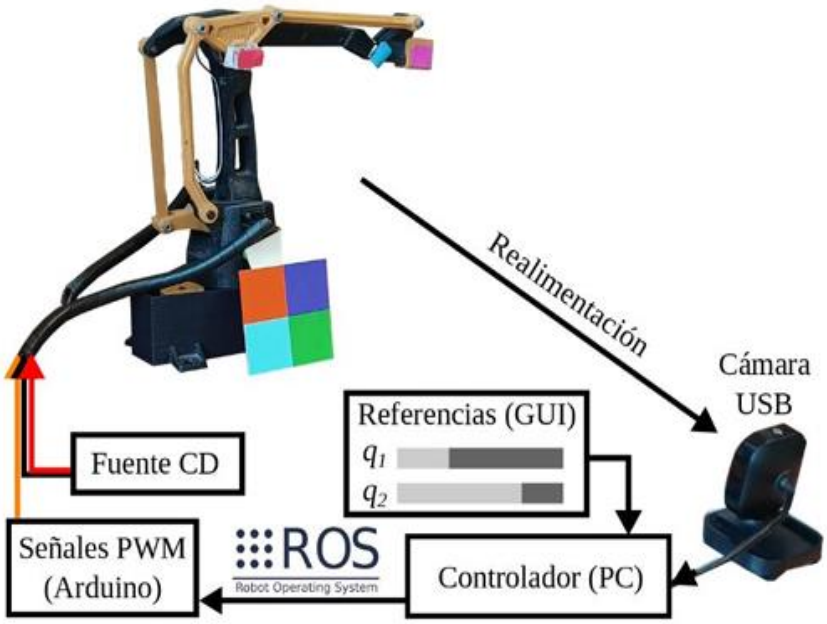
4.5. Real-Time Visual Feedback and Closed-Loop Control

Achieving reliable, precise, and adaptable robotic manipulation depends on the implementation of real-time visual feedback and closed-loop control. These approaches enable robots not only to perform pre-programmed duties but also to dynamically adjust their behavior in response to real-world variations, disturbances, and sensor uncertainties.

In vision-based manipulation systems, closed-loop control refers to the continuous use of sensory input, such as camera images, to guide robotic motion in real time. This feedback loop allows the system to adjust for errors that could occur from mistaken models, actuation inaccuracies, external disturbances, or unmodeled dynamics. A convincing demonstration of this concept is found in the work of Orozco-Soto and González Dorantes (2024), who present a robust visual feedback strategy for the control of a robot manipulator actuated by position-controlled servomotors. Their method leverages an external camera to provide continuous measurements of the robot's joint positions and the end-effector, which are then used as input for a secondary control loop. This visual feedback loop allows the system to correct deviations caused by gravity, friction, gear backlash, or external forces, ensuring the robot's actual motion aligns with the desired trajectory. Significantly, the authors report significant reductions in joint position errors and improvements in the repeatability of manipulation tasks when visual feedback is employed, compared to purely open-loop or proprioceptive control schemes

(Orozco-Soto and González Dorantes, 2024). Figure 7 shows the visual feedback control setup of a robotic manipulator using ROS, Arduino-based PWM signals, and a USB camera for real-time position correction.

Figure 7: Dual-loop control architecture combining actuators and visual feedback loops for robotic manipulators. Reproduced from Orozco-Soto and González Dorantes (2024, p. 3)



The advantages of visual feedback are not just limited to academic or laboratory robots. Wen et al. (2024) describe an industrially oriented system in which composite robots, combining mobile bases and robotic arms, utilize high-precision visual feedback for end-effector positioning. In their system, coarse navigation and mapping are performed using LiDAR, but the final pose adjustment for manipulation relies on high-resolution camera feedback. This visual feedback mechanism detects and localizes markers in real time, enabling the system to achieve very precise absolute positioning accuracies on the order of tenths of a millimeter an essential requirement in industrial manufacturing, precision assembly, or quality control scenarios. The feedback is used to perform real-time compensation for errors introduced by mobile base localization, mechanical tolerances, or environmental changes, demonstrating the particularly important role of real-time vision in robust, high-performance manipulation (Wen et al., 2024).

From a control theory perspective, the integration of visual feedback into closed-loop systems is often achieved with visual servoing. In the system described by Orozco-Soto and González Dorantes (2024), the secondary visual feedback loop is implemented in parallel with the primary actuator control. This dual-layer control structure is robust to changes in camera position, orientation, or workspace configuration. Experiments conducted with the camera

rotated at different angles confirm that the algorithm maintains performance despite significant changes in the visual input, emphasizing the algorithm's resilience and adaptability.

Weng et al. (2022) further illustrated the application of real-time visual feedback in a dual-arm robot tasked with complex human-robot interaction and object manipulation. Their approach uses continuous visual servoing to track and regulate the position and orientation of the manipulated object throughout the interaction, deep learning to identify objects, and hybrid tracking algorithms to follow their movement. This real-time adaptation makes it possible for the robot to interact safely and effectively with human users, maintain object stability during handovers, and adjust to ever-changing unpredictable environmental events.

Real-time performance is not only a function of perception and control algorithms but is also very closely related to the underlying system infrastructure. As described by Ye et al. (2023), the use of ROS2 as a middleware especially when optimized with real-time kernels such as Preempt_RT ensures that visual feedback, sensor data processing, and actuation commands are exchanged with minimal latency and jitter. Such system-level optimizations are critical for maintaining the responsiveness of the feedback loop, preventing instability, and achieving the high bandwidth necessary for fast and accurate manipulation (Ye et al., 2023).

In summary, the use of real-time visual feedback in closed-loop control architectures is a basis for modern robotic manipulation. It makes it possible for robots to correct errors on the go, adapt to unforeseen changes, and achieve levels of accuracy and robustness that are not possible with open-loop or purely model-based methods.(Orozco-Soto andGonzález Dorantes, 2024; Wen et al., 2024; Weng et al., 2022; Ye et al., 2023).

4.6. Learning-Based Manipulation and Reinforcement Learning

The field of robotic manipulation has experienced a transformative shift with the utilization of learning-based approaches such as RL for enabling adaptive, robust, and intelligent control of robotic arms. Traditional model-based methods, which rely on rigid kinematic and dynamic models, often fall short when robots are deployed in unorganized or changing environments, where exact modeling is not possible, which is the case for most situations. Learning-based methods, particularly RL, have developed as powerful alternatives, capable of providing robots with the ability to autonomously improve performance through trial-and-error and adapt to new tasks and environments with minimal human involvement.

As outlined by Harrington (2025), reinforcement learning provides a formal mathematical framework for agents to learn optimal control policies through iterative trial-and-error interactions within dynamic environments. RL frames the control problem as a sequential decision-making process, where the robot (agent) interacts with its environment, observes states, selects actions, and receives feedback in the form of rewards. Through repetition, the robot learns a directive that maximizes cumulative reward, leading to optimal results for complex tasks (Lawrence, 2024; Thaker, 2024). When we are talking about robotic manipulation, RL allows a robot to develop sophisticated skills such as grasping, sorting,

assembling, or in-hand object reorientation, often surpassing what can be achieved with static, hand-crafted controllers.

The literature shows that the integration of RL into robotic manipulation is particularly advantageous for handling uncertainty of the real world, sensor noise, and the high-dimensional state and action spaces typical of dexterous manipulation. For example, Weinberg et al. (2024) provide a comprehensive survey of learning-based approaches for in-hand manipulation, demonstrating that RL and IL are now the dominant paradigms for enabling robots to master manipulation tasks that require fine motor skills and the ability to adapt in the work itself. These approaches are especially effective when combined with precise sensory feedback, including visual input, which allows the learning agent to reason about object position, orientation, contact state, and other relevant cues during task execution (Weinberg et al., 2024).

Despite its promise, RL in robotics presents unique challenges, particularly in the "sim-to-real" area. Collecting substantial amounts of real-world interaction data can be costly, time-consuming, and potentially harmful to hardware. As a result, researchers often train RL policies in simulation, where there is a lot of data and it is risk-free. However, the differences between simulated and real-world environments (the sim-to-real) can cause procedures that work well in simulation to fail when transferred to physical robots (Lawrence, 2024; Thaker, 2024). To address this, advanced techniques such as domain randomization, meta-learning, and transfer learning are used to improve the applicability and robustness of learned policies (Lawrence, 2024; Thaker, 2024).

Thaker (2024) explores sim-to-real transfer, IL, and transfer learning as solutions to these obstacles. Domain randomization, for instance, introduces variability in simulation parameters (such as object textures, lighting, and dynamics) so that the learned policy becomes less sensitive to environmental changes that may occur. Transfer learning and imitation learning enable robots to leverage prior knowledge either from other tasks or from expert demonstrations to accelerate learning and improve performance on new manipulation tasks (Thaker, 2024). This trend is further supported by Lawrence (2024), who highlights the importance of safety mechanisms, efficient data collection, and robust generalization strategies for successful deployment of RL in real-world robotics.

An even better advantage of RL is its suitability for continuous improvement and adaptation. Once a policy has been trained and deployed, the system can continue to refine its behavior based on new experiences or in response to changes in the environment. This enables robots to maintain high performance with more time, even as task requirements or operational contexts evolve and require more. As described by Lawrence (2024), the repetitive nature of RL facilitates real-time adaptation and optimization, making it an ideal approach for manipulation tasks that involve dynamic, uncertain, or complex, harsh conditions.

The effectiveness of RL in robotic manipulation is evidenced by its successful application across a broad range of scenarios from industrial automation and assembly lines to service robotics and collaborative human-robot interaction. A notable medical application was presented by Gode et al. (2025), who employed reinforcement learning with 3D visual feedback

for autonomous suturing, demonstrating RL's potential for high-precision manipulation in surgical settings. Addition to that procedures that have been learned through RL has enabled robots to autonomously discover more effective grasp strategies, adjust to novel object shapes and sizes, and even recover from manipulation failures without explicit reprogramming (Weinberg et al., 2024; Lawrence, 2024).

But it also should be mentioned that the increasing reliance on learning-based control brings new risks, including vulnerabilities to adversarial and backdoor attacks. Wang et al. (2024) demonstrate that vision-language models, which are often part of the perception stack in RL-driven manipulation systems, can be exploited by adversaries to trigger undesired behaviors, raising concerns about the safety and security of autonomous robots in the real world.

In conclusion, learning-based manipulation and reinforcement learning have redefined what is possible in robotic manipulation. They enable adaptive, robust, and intelligent control, allowing robots to perform complex tasks in environments where traditional approaches would not be able to go part with. (Weinberg et al., 2024; Lawrence, 2024; Thaker, 2024; Wang et al., 2024).

4.7. Hardware Constraints and Vision Optimization in Robotic Manipulation

The integration of visual perception in robotic manipulation systems often introduces computational and hardware limitations that restrict processing capability, response time, and accuracy. Recent studies have presented several approaches to overcoming challenges through improved simulation environments, accurate dynamic modelling, and the development of lightweight computer vision algorithms designed for embedded hardware.

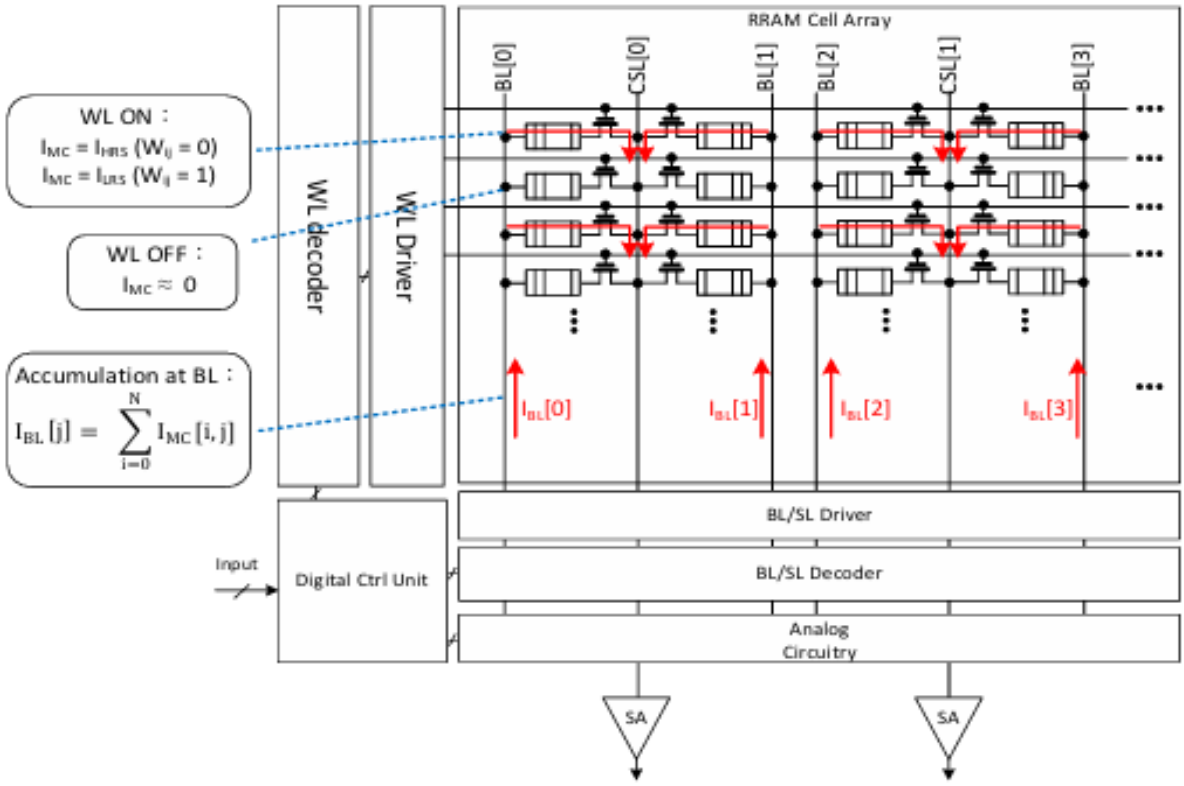
Platt and Ricks (2022) conducted a comparative study between the ROS Gazebo and ROS Unity3D simulation frameworks to analyse their suitability for robotic applications. Their findings showed that while Gazebo provides computational efficiency for small-scale simulations, Unity3D offers superior visual realism and supports more complex environments. The enhanced graphical rendering and detailed lighting models of Unity3D make it particularly suitable for validating camera-based perception systems, where realistic lighting and object textures are vital for accurate performance evaluation before transferring algorithms to real robots.

Accurate modelling of manipulator dynamics is another major factor influencing control precision. Soto, Maina and Byiringiro (2024) examined the nonlinear parameter identification of the Franka Emika Panda robot using the Newton–Euler formulation and several friction models such as Stribeck, Lugre, and Sigmoidal. The study demonstrated that including nonlinear friction behavior significantly improved torque estimation and reduced trajectory deviation, leading to smoother and more predictable motion. This approach is particularly beneficial for visual-servoing systems, in which minor modelling inaccuracies can cause feedback instability and degraded object-tracking accuracy.

Hardware efficiency in visual perception has also been advanced through lightweight neural network architectures. Kee et al. (2024) proposed an optimized pick-and-place detection method using an SSD Mobile-Net V2 FPN Lite model. Their research combined hyperparameter tuning with RGB image enhancement to achieve a seven percent improvement in mean Average Precision and a 97 percent detection rate when deployed on a Tensor Processing Unit. This work highlights how careful network optimization and image pre-processing can offset hardware constraints while maintaining reliable detection accuracy in real time.

Complementing this, Song (2025) presented a low-cost vision solution using HSV-based color segmentation implemented in OpenCV on a Raspberry Pi platform. The study found that the HSV color model is more resilient to varying illumination compared with the RGB model. By employing contour detection and shape recognition, the method achieved stable and efficient object detection suitable for small-scale or educational robotic platforms. This approach shows that effective color segmentation can deliver dependable vision performance without requiring heavy computational resources. Figure 8 shows an in-memory RRAM computing architecture designed for energy-efficient convolution operations and hardware–software co-optimization.

Figure 8: Schematic representation of an in-memory RRAM computing macro used for energy-efficient convolution operations and hardware–software co-optimization (reproduced from Chiang et al., 2022).



At the hardware-design level, Chiang et al. (2022) explored in-memory RRAM computing as a strategy for improving the efficiency of deep-learning-based object detection. Their hardware–software co-optimization technique reduced the effects of device variation and voltage instability while maintaining detection accuracy within a four percent deviation. The approach also lowered power consumption, indicating a potential direction for future robotic systems that integrate energy-efficient computing hardware directly into their perception and control loops.

Overall, these studies demonstrate that it is possible to achieve accurate, responsive, and efficient robotic manipulation even within limited hardware conditions. Continuous advancements in simulation realism, dynamic modelling, and embedded vision architectures contribute to the development of flexible and reliable robotic systems capable of operating in real time.

In conclusion, the integration of realistic simulation platforms, precise mechanical modelling, hardware-aware optimization, and computationally efficient vision networks enables adaptive and high-performance robotic manipulation (Platt and Ricks, 2022;Soto, Maina and Byiringiro 2024; Kee et al., 2024; Song, 2025; Chiang et al., 2022)

5. MATERIALS AND METHODS

In this chapter, the experimental setup, materials, and methodologies used throughout the research are described in detail. It outlines the software components of the robotic system, including the Franka Emika Panda manipulator, the integrated RGB camera, and the ROS2-based control framework implemented in Python. All experiments were conducted in the Gazebo simulation environment, which provided a safe and controlled platform for developing, testing, and validating robotic manipulation and vision algorithms.

5.1. The Environment Setup

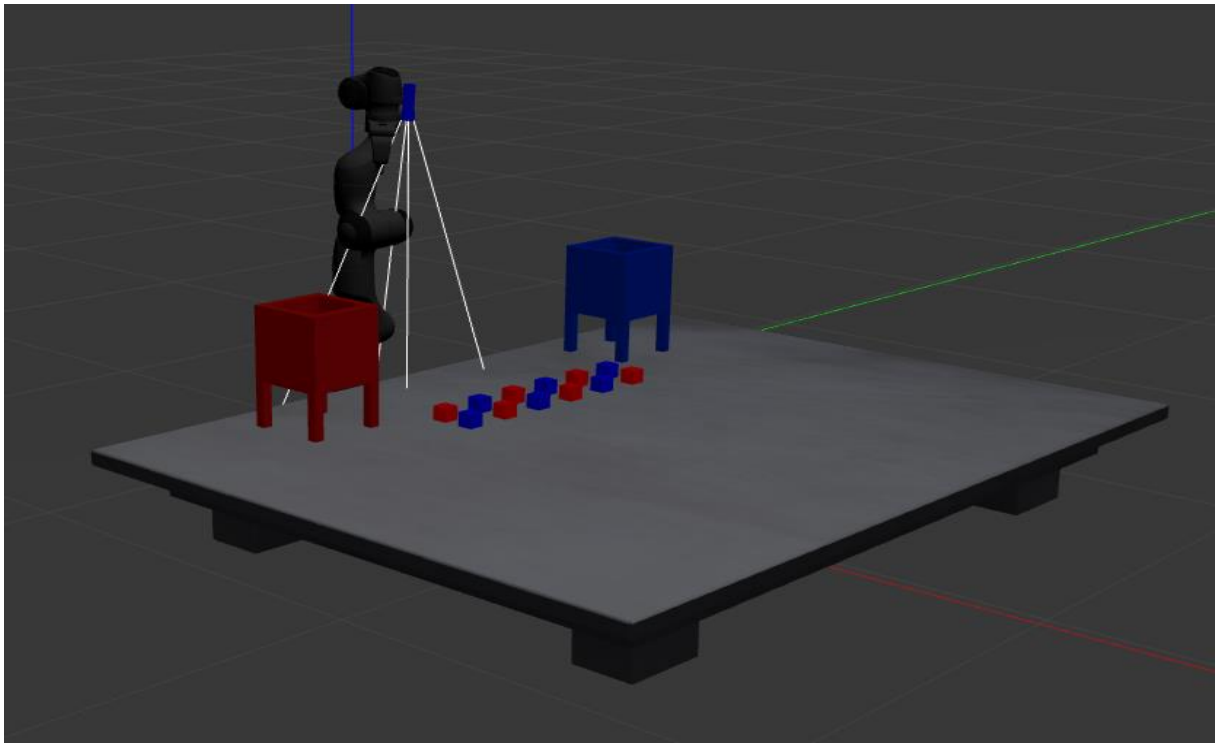
The experimental environment was designed to provide a controlled and repeatable setting for developing, testing, and evaluating the manipulation tasks of the robotic arm system. All experiments were conducted in a simulation workspace that maintained constant variables to ensure fair comparison between both methods.

The robotic system was built around the Franka Emika Panda 7-DOF manipulator, which was fixed at the origin. The Panda arm was equipped with a parallel gripper for grasping and manipulating various test objects. An RGB camera was installed on the panda_link6 to provide real-time visual feedback for object detection and tracking. The camera's field of view covered the area in Front of the gripper to capture visual cues and use this data for visual servoing.

The computational environment consisted of an Ubuntu 22.04 LTS workstation running ROS 2 Humble as the communication framework for the nodes. All perception, control, and planning algorithms were implemented in Python 3, utilizing the OpenCV library for image processing and the *rclpy* client library for ROS 2 node development. The system architecture followed the standard ROS 2 node-based design, where individual nodes handled specific functions such as image acquisition, object recognition, inverse kinematics computation, and trajectory execution.

The robot's environment was modeled in simulation to ensure consistency during virtual testing and for comparing results between object detection and color-based detection. Gazebo and RViz were used for visualization and verification of motion planning before deployment on the physical robot. The workspace included a flat surface with a uniformly colored background to enhance visual segmentation, and objects were placed within a predefined region reachable by the manipulator. Figure 9 shows the general layout of the workspace.

Figure 9: The workspace layout



This environment setup ensures reliable integration between perception and control subsystems, providing a robust foundation for testing the proposed manipulation strategies based on visual feedback.

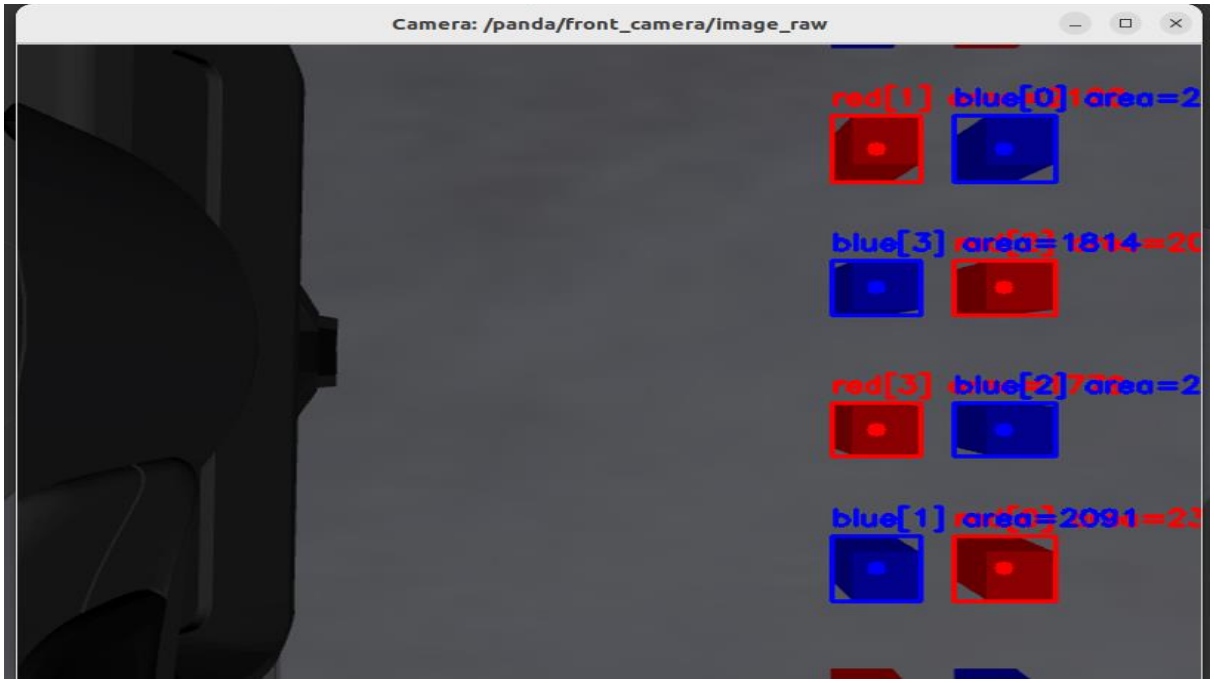
5.2. Robot and Hardware Components

The Franka Emika Panda robotic arm was selected as the primary manipulator because it has great ROS 2 compatibility and a dexterous robot. The arm provides seven degrees of freedom (7-DOF), which allows for flexible and dexterous movements suitable for tasks involving object grasping and other tasks.

At the end of the manipulator, a parallel gripper was attached to handle as two panda fingers and it can be used for grasping with the help of IFRA_LinkAttacher. The gripper's opening width is controlled via ROS 2 messages, allowing smooth integration with the control node. The workspace and object dimensions were selected to ensure that all targets remained within the manipulator's reachable area.

A fixed RGB camera was mounted on joint link 6, aligned with the area in front of the robot gripper. The camera continuously captured visual data used for detecting and localizing target objects. The messages were transformed using OpenCV to analyze the data and create a projection of images back into the GUI. The camera provided real-time image streams at a resolution sufficient for color and feature-based tracking, enabling the implementation of the visual feedback control strategy. Figure 10 shows the color-based detection using the camera.

Figure 10: The RGB camera view



The entire system was operated by a workstation running Ubuntu 22.04 LTS, equipped with an Intel i7 processor, 16 GB RAM, and an NVIDIA GPU to accelerate image processing tasks. The computer communicated with the simulated robot through the ROS 2 Humble middleware, using the rclpy interface for Python nodes.

This hardware configuration was used for object training, color-based visual recognition, and simulation to emulate a workspace.

5.3. Software Setup and ROS 2 Architecture

The software system was developed within a single ROS 2 workspace to maintain modularity and ease of deployment. The workspace, named ros2_ws, contains all custom packages related to robot description, control, vision, and motion planning inside the my_robot_controller directory. Each package was organized following the ROS 2 conventions, which ensured compatibility with existing tools and simplified integration with simulation and visualization environments.

The urdf folder contains the robot's URDF/Xacro model, including link and joint definitions, sensor configurations, and the <ros2_control> hardware interface. Another package named meshes includes the mesh files and material definitions used for visualization in Gazebo.

The launch folder has the launch file named robot_camera_panda.launch.py for launching the robot and all associated nodes. It includes launch files that initialize the robot description, controllers, and camera drivers.

It gets its configuration from YAML files inside the config folder, specifically the robot_controllers.yaml file for the controller manager. The parameters were also stored within this file.

The components directory contains different models and their required SDF and XML files to spawn entities in the world. The worlds folder contains my_gazebo.world, which includes information regarding world physics, entity locations, and names so they can spawn in the simulation environment. The relevant information is derived from the components file.

The my_robot_controller directory contains the executable Python nodes that perform various tasks. These nodes are all executable and can be called from the terminal.

The nodes are listed on the table below.

Table 1: The Node Table

NODE NAME	PURPOUSE
my_yolo_detector.py	Converts dataset annotations (e.g., VOC or COCO format) into YOLO format, generating .txt label files that match YOLO's training requirements. Useful for preparing custom datasets.
voc2coco_redcube.py	Converts Pascal VOC annotations to COCO JSON format. Helps when training detectors that require COCO-style annotations.
my_RGB_detector.py	Creates a basic GUI for color detection using pixels and trajectory control.
my_yolo_detector.py	Runs a YOLO object detection model with the same GUI and structure as my_RGB_detector.py.

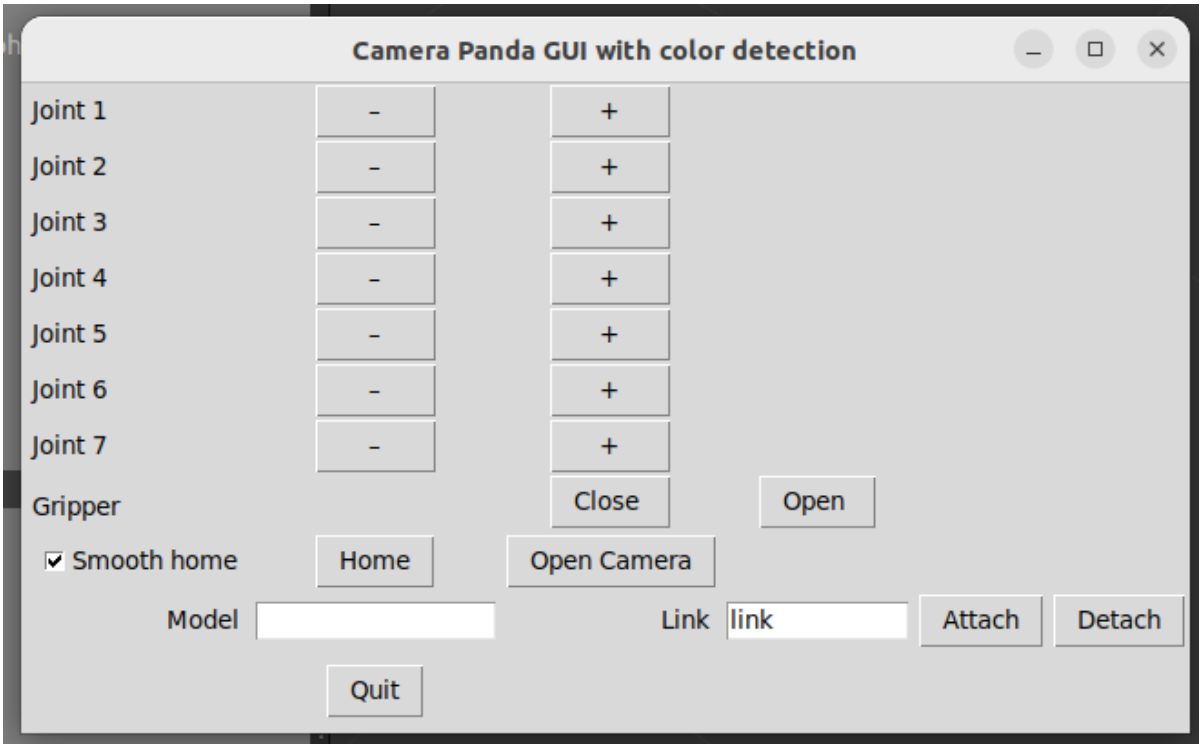
Each package was built using colcon, ROS 2's official build system. Dependencies and environment variables were managed through the setup files (setup.bash), allowing integration between modules. This modular organization ensured that each subsystem description, perception, control, and planning could be developed, tested, and maintained independently while maintaining full system interoperability.

5.3.1. Robot Description and Configuration:

The robot model was defined using the URDF.Xacro file and parameterized through Xacro macros to ensure flexibility, easier modification, and debugging. The description package contained a main Xacro file that generated the complete URDF of the Franka Emika Panda robot with a camera. This model included all essential physical and visual properties such as links, joints, masses, inertial parameters, and visual meshes that were derived from the component's directory. The Xacro structure also allowed separate macro definitions for different components such as the camera, controller interface, and the robot itself. This helped reduce redundancy and improve readability.

The robot's end-effector was modeled with a parallel gripper, defined as panda_link7, which has two fingers that can be controlled through the GUI. A camera link was also added to the panda_link6 frame to represent the visual sensor used for object detection. The camera pose was defined by inserting it in joint 6 so that it moved relative to that joint, matching the field of view used in the simulation. This ensured that the camera's orientation and position remained consistent across experiments. Figure 11 shows the GUI used for controlling the Panda robot.

Figure 11: A basic GUI f or joint and camera control



To enable simulation and control, the robot description incorporated <ros2_control> tags that defined the hardware interface, transmission elements, and actuator types. These specifications allow the the ROS 2 Control framework to manage joint states and trajectories through the controller_manager. The controllers were configured via YAML files located in the my_robot_controller/config directory. The configuration included a joint trajectory controller for the arm's seven joints, a gripper controller for end-effector actuation, and a joint state broadcaster for publishing the current joint positions and velocities.

The robot's TF (transform) tree was automatically generated using the robot_state_publisher, which published transforms for each joint based on the URDF structure. The hierarchy followed the standard convention. Each link and joint in this hierarchy defined a unique coordinate frame, ensuring accurate spatial relationships across the robot model. The robot_state_publisher node continuously broadcasted these transformations in real time based on the joint states received from the controllers. This structure allowed all subsystems' vision, motion planning, and control to operate within a unified reference frame.

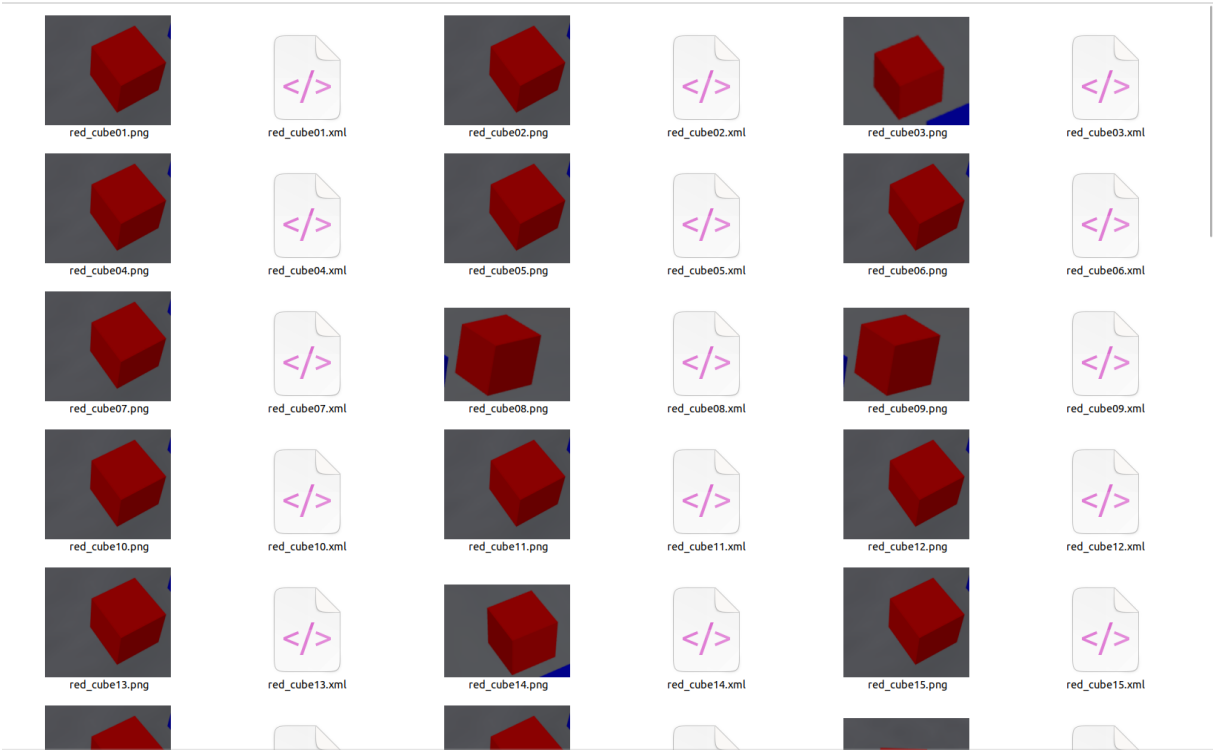
Static transforms were used for fixed frames, such as between the camera link and gripper since their relative positions did not change during operation. These were defined using the static_transform_publisher and verified through visualization tools such as rqt_tf_tree. Maintaining a consistent and verified TF structure was crucial for transforming object coordinates detected in the image frame into the robot's Cartesian coordinate system for manipulation tasks.

This complete configuration ensured that the robot model, controllers, and planning framework shared a consistent reference structure, allowing accurate motion planning, trajectory execution, and visual feedback integration within the ROS 2 simulation environment.

5.3.2. The YOLO training

For this part of the project, a YOLO-based object detection model was trained to identify the red cube used in the simulation environment. The training process began with the collection of approximately fifty images of the cube from different angles and lighting conditions in Gazebo. Each image was manually labeled using a dataset annotation tool, such as LabelImg, which allowed defining bounding boxes around the object of interest. These annotations were saved in the Pascal VOC (.xml) format, describing the coordinates of the bounding boxes for each image. Figure 12 shows all the training data along with their corresponding XML annotation files.

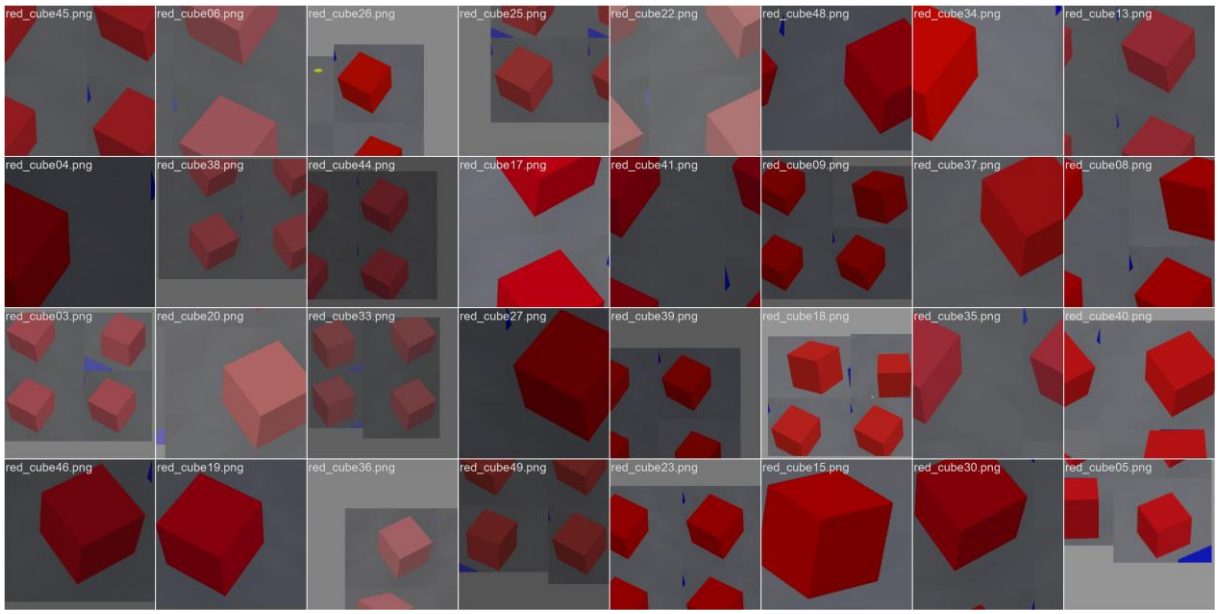
Figure 12: The data set



Since YOLO requires annotations in its own specific format, the dataset was converted into two steps. The first conversion was performed using the **voc2co_redcube.py** script, which transformed all VOC XML files into a **COCO (.json)** dataset structure. This JSON file stored

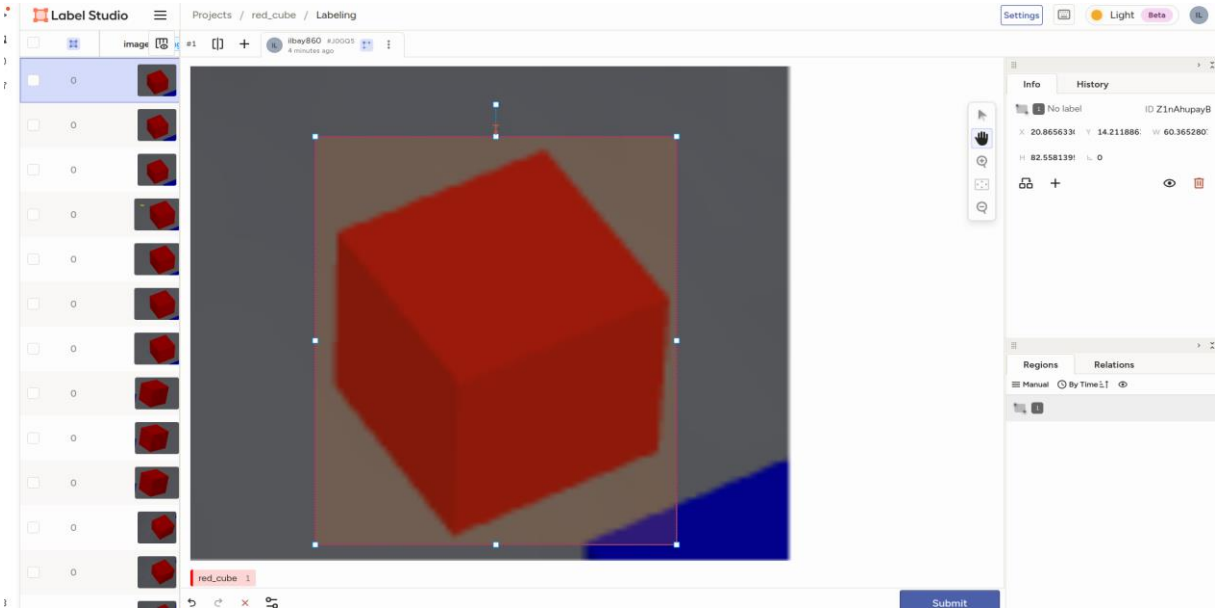
image file names, dimensions, bounding box coordinates, and object categories in a standardized format suitable for computer vision models. The second script, **convert_to_yolo.py**, was then used to convert the COCO dataset into the **YOLO (.txt)** format required for training. In this stage, each image received a corresponding text file containing normalized coordinates of the bounding box and the class identifier, expressed as numerical values between 0 and 1 relative to image width and height. Figure 13 shows the first training batch (0–1) consisting of sample images used for model learning and validation.

Figure 13: Training batch 0-1



The final dataset structure consisted of separate trains and validation directories, each containing image and label subfolders. A configuration file named data.yaml was created to define the dataset paths, number of classes, and class names (in this case, “redcube”). The YOLO model was then trained using this dataset by running the standard YOLO training command with parameters specifying image size, batch size, and number of epochs. During training, the model learned to associate the visual features of the red cube with its labeled bounding boxes. Figure 14 illustrates the labeling process used for data training to generate the corresponding XML annotation files.

Figure 14: Labeling for data training



This process demonstrated the typical workflow for preparing and training a YOLO object detection model. The conversion scripts played a crucial role in transforming human-labeled data into a format that could be efficiently processed by the network. Although the limited dataset size (50 images) restricted the final model's accuracy, the training pipeline established a functional baseline for object detection and provided valuable insight into the preparation and conversion stages required for YOLO-based vision applications in robotics.

5.3.3. System Architecture

The core runtime is a single ROS 2 Python node, my_RGB_detector, which integrates camera acquisition, color-based perception, manual joint/gripper teleoperation, and contact-aware grasp logic. Communication is realized via ROS 2 topics, an action client, and services discovered at startup.

Node: my_RGB_detector (package: my_robot_controller)

Primary responsibilities:

- Subscribe to the front camera stream and run real-time color detection.
- Publish joint trajectories for the 7-DOF Panda arm and the two finger joints.
- Maintain a "hold" loop to stabilize pose and reassert gripper commands.
- Monitor finger contact topics and trigger auto-attach via Gazebo link-attacher services.
- Provide a lightweight Tkinter GUI for manual joint jogging, homing, gripper open/close, and camera preview.

Quality of Service (QoS): The image subscriber uses a BEST_EFFORT profile with KEEP_LAST (depth=1) to minimize latency and avoid queuing stale frames.

Table 2: Types of Messages

Interface	Name	Type	Direction	Purpose
Topic	/panda_arm_controller/joint_trajectory	trajectory_msgs/JointTrajectory	Publish	Sends arm joint goals (7 joints).
Topic	/panda_hand_controller/joint_trajectory	trajectory_msgs/JointTrajectory	Publish	Sends finger joint goals (two fingers).
Topic	/panda_hand_controller/commands	std_msgs/Float64MultiArray	Publish	Gripper finger positions mirror (for compatibility/monitoring).
Topic	/joint_states	sensor_msgs/JointState	Publish	Publishes current arm + finger states (for visualization/record).
Topic	/panda/front_camera/image_raw	sensor_msgs/Image	Subscribe	RGB image stream for detection and GUI preview.
Action (optional)	/panda_gripper_controller/gripper_cmd	control_msgs/action/GripperCommand	Client	Secondary gripper interface (position/effort).
Service	(discovered Attach/Detach)	…/AttachLink, …/DetachLink	Client	Used if canonical service names are unavailable.

Control timing/guards:

- Hold loop publishes a short-horizon trajectory at 5 Hz to keep the arm/gripper steady (/panda_*_controller/joint_trajectory).
- Smooth gripper motion: timer-driven interpolation for grasping objects.With help of IFRA_LinkAttacher .
- Debounce / windows: attach attempts are rate-limited (e.g., 0.05 s debounce; 0.6–0.8 s retry windows).

GUI (Tkinter):

- Joint jogging (± buttons per joint), smooth homing (cosine easing over ~1.5 s), Open/Close gripper, camera preview pane (~30 FPS), and manual attach/detach fields for testing.
- ROS spinning is integrated via a Tk after () pump to keep the UI responsive.

5.4.4. Pick and place with YOLO and Color

This section describes the experimental implementation of the pick-and-place task using two different visual perception approaches: color-based HSV segmentation and deep-learning-based YOLO object detection. It explains how the perception and control subsystems were integrated within the ROS 2 framework to achieve autonomous object manipulation. The experiment begins by launching the simulation environment, which automatically spawns the robot, camera, and workspace components.

Once the system is initialized, the control node and perception modules are executed, enabling the robot to detect, grasp, and place objects in predefined target locations. The following subsections detail the operation of the trajectory controller and the vision pipelines, outlining how sensory data from the camera is processed and translated into motion commands for the Panda arm.

Trajectory Controller

The trajectory controller for the Panda manipulator operates in joint space, where the seven actuated joints are expressed as a vector $q(t) = [q_1,q_2,q_3,q_4,q_5,q_6,q_7]$. Each element represents the angular position of a revolute joint in terms of radians. The gripper's opening is represented as w(t), which defines the total width between the two symmetric fingers, each commanded as f(t) = 0.5 w(t). The desired joint trajectory $q_{\text{target}}(t)$is transmitted through a single-point JointTrajectory message containing the target position and the motion duration. The internal PID loops in the low-level controllers minimize the position error:

$$e(t) = q_{\text{target}} - q(t) \text{ [rad]} \quad (1)$$

where:

- *e(t)* = joint position error vector (rad)
- $q_{\text{target}}(t)$ = desired joint positions (rad)
- *q(t)* = measured joint positions (rad)

so that the actual joint angles smoothly converge toward their commanded positions. This proportional–integral–derivative control ensures stable convergence and compensates for small system delays or external disturbances.

A periodic hold mechanism operates at 5 Hz to maintain the robot's current posture and prevent drifting caused by uncommanded forces or model inaccuracies. The controller republishes the most recent configuration to serve as a keep-alive command, enforcing zero steady-state error in the absence of new inputs. When returning to the home configuration, a cosine interpolation function is used to create a smooth, jerk-limited motion. The interpolation is defined as:

$$q(t) = q_0 + (q_{\text{home}} - q_0) \times (0.5 - 0.5\cos(\pi t/T)) \text{ [rad]} \quad (2)$$

where:

- q_0 = initial joint vector (rad)
- q_{home}= desired home configuration (rad)
- t = time (s)
- T = total motion duration (s)

where q_0 is the current joint vector, q_{home} is the desired home pose, and T is the total duration of the motion. This function gradually accelerates and decelerates the motion, ensuring continuity in both velocity and acceleration profiles. The motion is discretized into 120 steps over 1500 ms, achieving a natural and safe transition without mechanical shocks.

Gripper motion follows a similar concept, employing a sine ease-out function for its position trajectory:

$$w(t) = w_0 + \left(w_{\text{target}} - w_0\right) \times \sin\left(\frac{\pi t}{2T}\right) \text{ [m]} \quad (3)$$

where:

- w_0= initial gripper width (m)
- w_{target}= final gripper width (m)
- t = time (s)
- T = total motion duration (s)

where w_0 and w_target denote the initial and desired finger openings. This formulation produces a gradual slowdown near the target width, avoiding sudden contact or overshooting during grasping. The combination of cosine interpolation for arm motion and sine easing for the gripper generates continuous, smooth trajectories that respect the manipulator's kinematic limits and mechanical safety constraints. Together, these mathematical models ensure the arm and gripper exhibit coordinated, repeatable, and dynamically stable motion suitable for precise manipulation and visual-feedback experiments. Figure 15 shows the ROS2 communication while the controller is running.

Figure 15: Rqt_graph for GUI

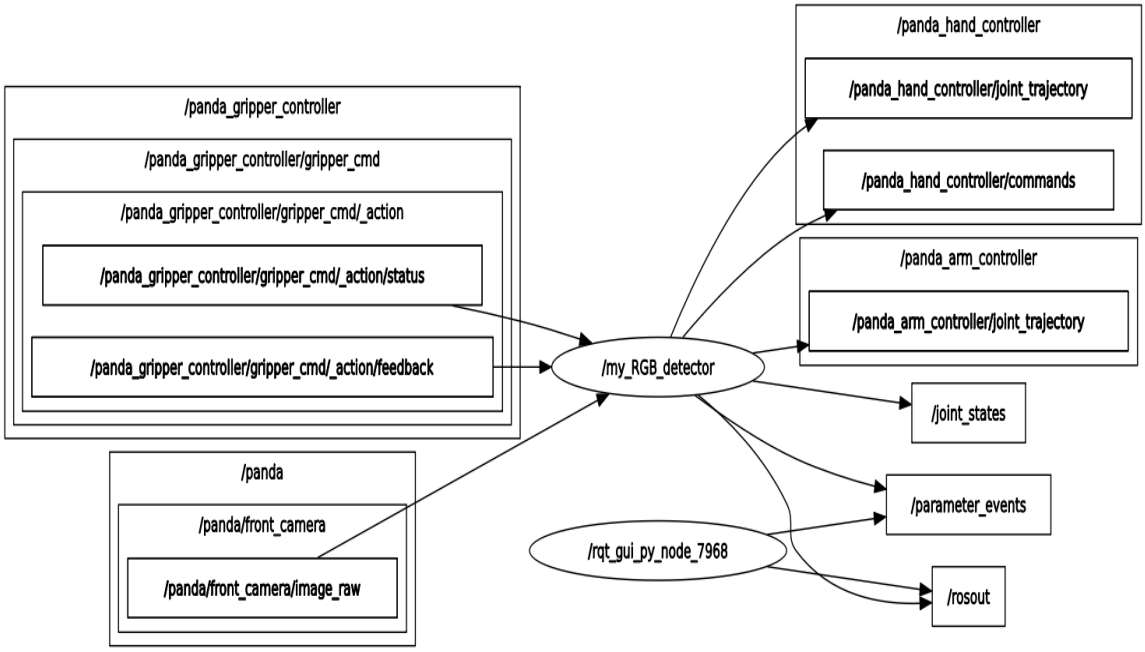


Image Processing and Detection

Two distinct image-processing pipelines were implemented for visual detection: one based on color segmentation in HSV space, and another using a deep-learning YOLOv8 model. Both pipelines subscribed to the same topic, /panda/front_camera/image_raw, using a low-latency BEST_EFFORT QoS profile, and processed frames in real time to produce 2D detections used by the control logic.

The color-based (HSV) detection system converted each incoming frame from BGR to HSV color space, decoupling color from illumination. It then applied predefined thresholds for each target color: two hue ranges for red to account for hue wrap-around and one range for blue. Morphological opening and closing with an elliptical kernel (size 5) were applied to remove noise and close small gaps in the detected regions. Contours were extracted, and only those exceeding 800 pixels in the area were considered valid detections. For each color blob, the node computed an axis-aligned bounding box, a centroid, and an area-based quality metric. These detections were visualized in the GUI with color-coded boxes and centroids and logged to ~/vision_metrics.csv with columns including timestamp, method ("HSV"), latency, FPS, detection count, centroid coordinates, bounding-box size, and quality. The HSV approach delivered consistent, high-speed performance under controlled lighting and proved computationally efficient, though it was sensitive to shadows and background colors similar to the targets.

The YOLO-based detector used the same camera feed but performed inference using Ultralytics YOLOv8n weights (yolov8n.pt) with a confidence threshold of 0.45, IoU of 0.45, and a maximum of ten detections per frame. Each frame was converted to RGB and passed to the neural network, which produced bounding boxes, class labels, and confidence scores. The node drew green rectangles and confidence labels for each detection, marking centroids as

reference points, and logged the same key statistics (timestamp, method = “YOLO,” latency, FPS, detection count, centroid, width, height, and confidence) to the same CSV file, allowing a direct comparison with HSV performance. Although YOLO was more robust to lighting variations and textures, it required greater computational resources and performed inconsistently due to the limited training set of approximately fifty images.

Both pipelines ultimately produced equivalent outputs in the form of bounding boxes and centroids for each frame, thereby ensuring identical data flow within the control stack and allowing a fair, unbiased comparison between the HSV-based color detection and YOLO-based object detection methods under the same experimental conditions.

The color-based detector provided a faster, simpler, and more hardware-efficient solution, whereas the YOLO-based detector exhibited greater potential for generalized object recognition, but at the cost of reduced processing speed and stability when operating under limited data conditions.

5.4.5. Experimental Procedure

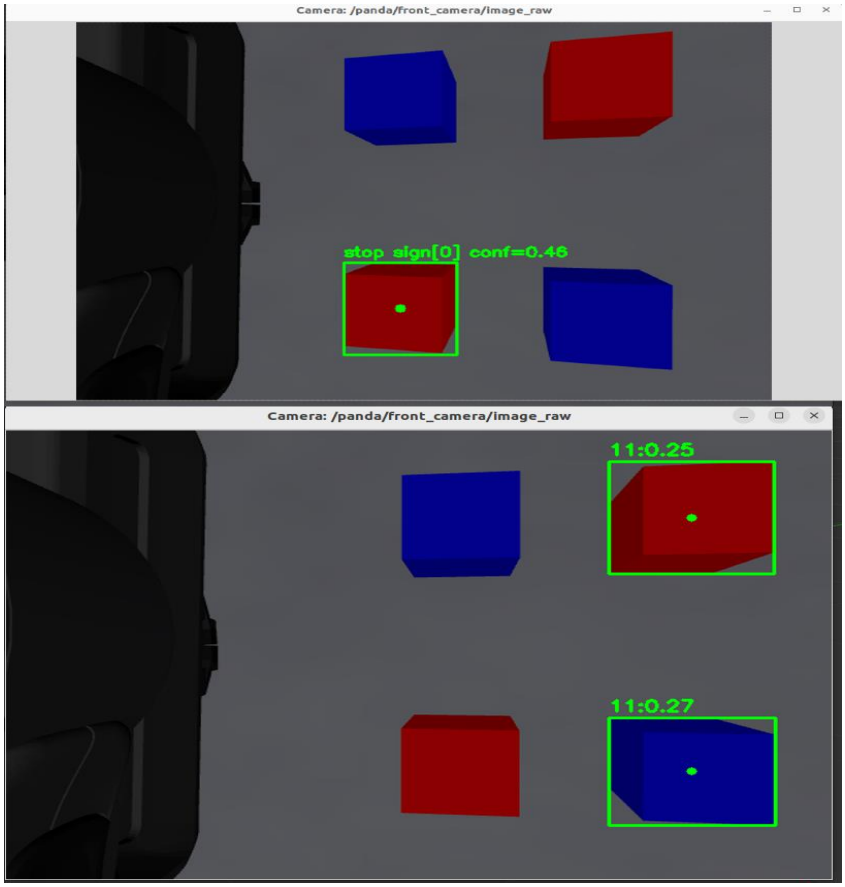
The experiment was conducted through a series of iterative trial-and-error runs. After launching the simulation environment, the task involved detecting, grasping, and placing objects into their correctly colored bins. Each trial followed the same sequence of perception, motion planning, and execution steps to ensure consistency.

The system performance was observed over multiple repetitions to evaluate the stability and reliability of the visual detection methods. Data were collected regarding how each image processing approaches color-based segmentation and YOLO detection—behaved under identical conditions. The outcomes were analyzed from different viewing perspectives within the simulation to verify detection accuracy and ensure that the robot’s movements corresponded precisely to the visual feedback.

6. RESULTS AND DISCUSSION

The two vision pipelines produced distinct detection behaviors under the same simulation setup. The color-based HSV method consistently returned stable masks and centroids for the red and blue cubes and bins, which translated into reliable bounding boxes across repeated runs. In contrast, the YOLO-based pipeline performed poorly in differentiating between different colored cubes and showed greater variability in bounding box placement and detection confidence. These inconsistencies were mainly due to the limited amount of training data available for this project. In practical terms, the HSV outputs were steady and predictable from frame to frame, while YOLO occasionally misplaced bounding boxes or assigned incorrect class labels for visually similar objects. Figure 16 shows the YOLO camera trying to detect the red cubes.

Figure 16: YOLO camera detection



These differences extended beyond the detection of accuracy to computational performance as well. Running the YOLO detection pipeline placed a significant load on the GPU, which caused the laptop to slow down and sometimes crash during simulation. This made it difficult to collect consistent data and monitor the experiment in real time. Furthermore, the heavy computational demand also complicated the colcon build process for the ROS 2 workspace, as the large dataset and model weights required considerable memory and processing resources. These challenges highlight how hardware limitations can directly affect

experimental reliability when working with deep learning-based methods. Figure 17 is the difference in hardware usage while the YOLO and the color-based detection are running. While in Figure 18 we can see a successful grab using the color-based detection.

Figure 17: The hardware usage (top – YOLO, bottom – color-based)

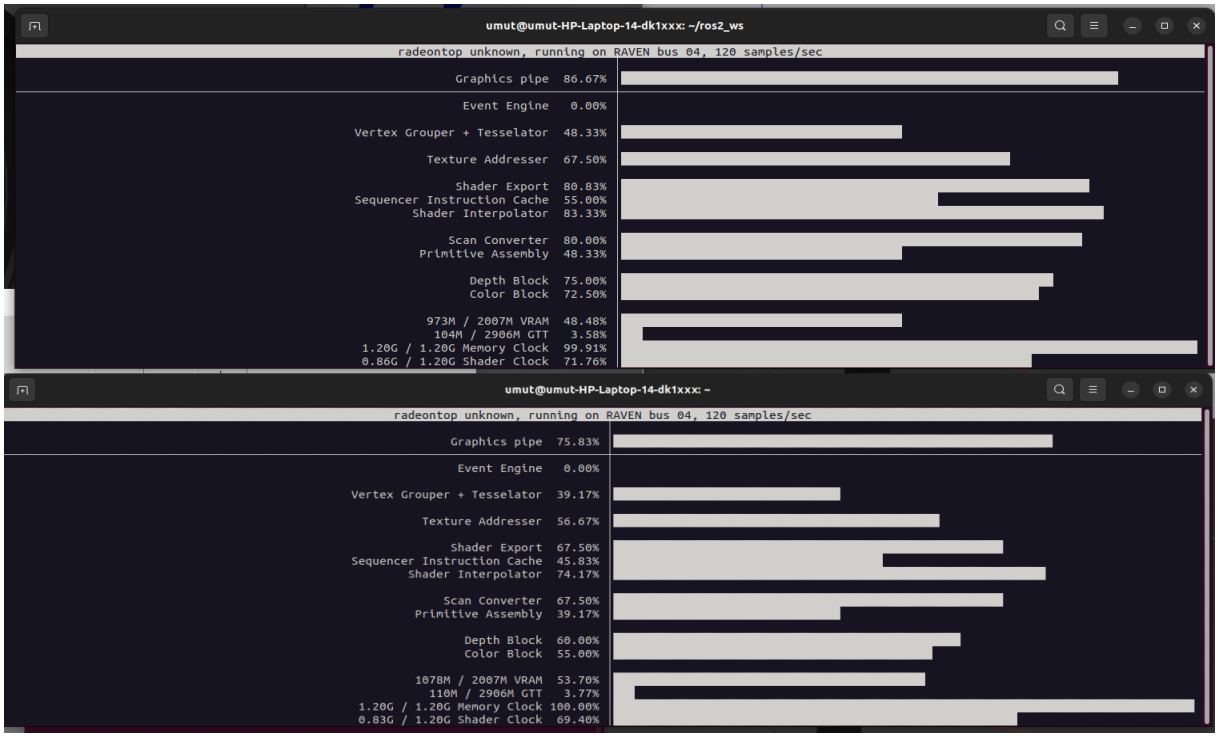
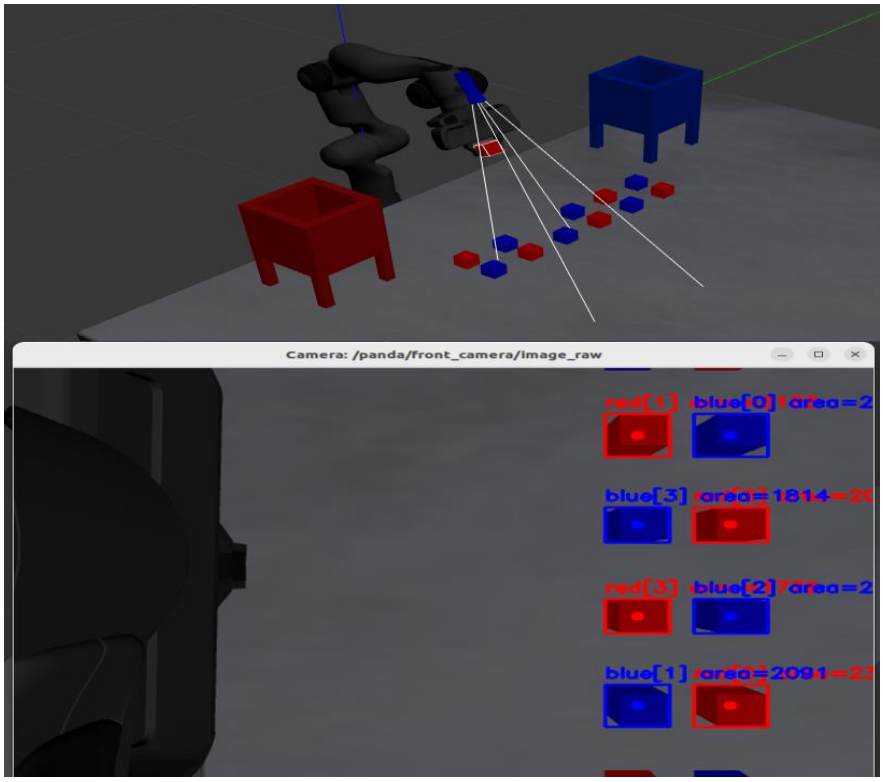
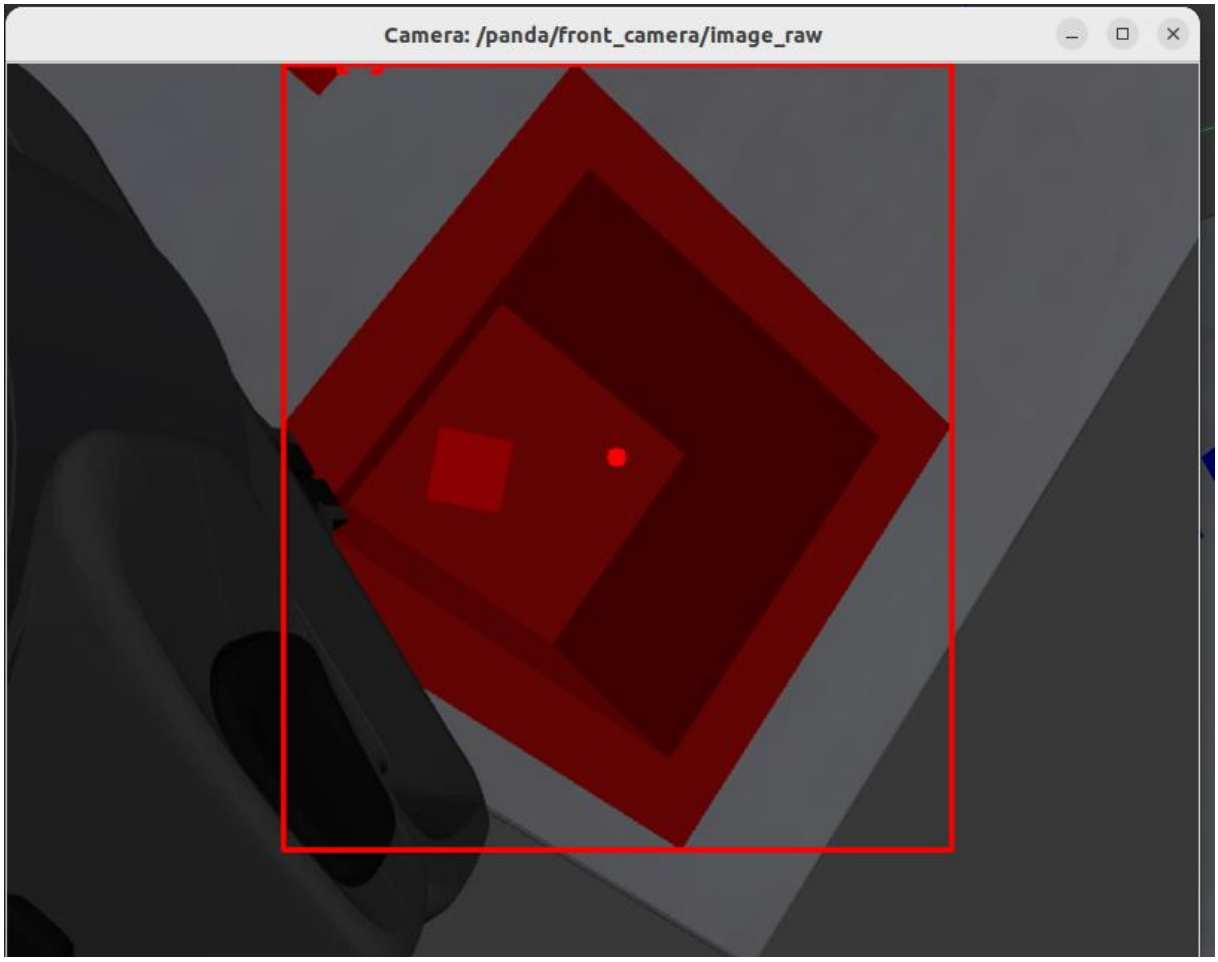


Figure 18: Grasping red cube with the help of color-based detection



From a practical standpoint, the HSV-based color detection method provided a lower-effort and highly stable solution, as long as the objects were color-coded. YOLO, on the other hand, offered a more general framework capable of recognizing different object types but demanded significantly more training data, computational power, and setup time to achieve similar stability. Given the project constraints, color-based detection proved to be the more efficient and dependable approach for simple sorting and binning tasks, whereas YOLO would be more suitable for complex environments where object identity cannot be determined by color alone. Figure 19 shows a cube successfully placed inside the correct bin.

Figure 19: After repositioning the camera, the system could no longer detect the object because the bin and cube share the same red color,



7. CONCLUSION AND FUTURE PLANS

In conclusion, even though the YOLO-based object detection method is generally recognized as an accurate approach, the results obtained in this project were limited due to hardware constraints and the small size of the training dataset. The 50 pictures I used for training were not enough. The model required more data and more extensive training to reach its full potential. In contrast, the color-based HSV detection system provided faster and more consistent performance under the given conditions. It proved to be an efficient and low-cost solution for simpler, well-controlled environments or for tasks where objects can be distinguished by color, such as basic pick-and-place operations.

For the future, combining both approaches could create a more robust visual servoing system. A mixed solution to this problem would create a project that can detect both the object and also its other characteristics, resulting in a more reliable and robust solution.

In this project, the YOLO model was trained only on red cubes, but it still counted blue cubes as the same model since it could not tell the difference and was also not confident in the detections it made. Expanding the training dataset to include multiple colors and shapes, together with optimized hardware, would significantly improve the system's overall detection of accuracy and real-world usability. For now, the color-based system has proved to be the better and simpler choice that can be used for similar tasks.

8. SUMMARY

In this research, I built a ROS2 workspace and used Gazebo Classic to emulate a workspace and compare the YOLO object detection model with color-based recognition. The goal of this experiment was to evaluate these two different methods for visual servoing and analyze their respective advantages and disadvantages.

After creating the necessary workspace, I ran the simulation multiple times to gather data and observe the system from different angles. A camera was used to grab an object and place it inside a designated bin.

With HSV color-based recognition, the system successfully detected the workpiece and allowed me to place the cubes into the correct locations. Because it required less hardware power, the FPS and overall system performance were stable, enabling smoother robot control. However, HSV detection was less accurate than object detection because it relies on pixel-based segmentation around the cube. As a result, it could not detect cubes inside the bin since they were all the same color.

When using the object-detection approach based on my trained data, it did not work as intended. As mentioned previously, the training dataset was too small, and the system quickly became overloaded, leading to performance issues such as FPS drops. Although the model performed better when it successfully detected objects, it requires more powerful hardware and a larger dataset to function properly.

In the end, I concluded that training data collection is quite an expensive and time-consuming task, and until more efficient methods for data processing become available, integrating object detection widely will remain challenging. Therefore, HSV color-based recognition can be considered a suitable alternative—it is easier to implement, requires fewer computational resources, and still provides usable performance for basic manipulation tasks.

9. BIBLIOGRAPHY

Blank, D.S., Meeden, L. and Kumar, D. (2003) 'Python robotics', ACM SIGCSE Bulletin, 35(1), p. 317. Available at: https://doi.org/10.1145/611993.611996

Chiang, Y.-H., Ni, C.E., Sung, Y., Hou, T.-H., Chang, T.-S. and Jou, S.J. (2022) 'Hardware-Robust In-RRAM-Computing for Object Detection', arXiv preprint, May. Available at: https://doi.org/10.48550/arXiv.2205.03996

Elsner, J. (2023) 'Taming the Panda with Python: A powerful duo for seamless robotics programming and integration', SoftwareX, 24(LBNL-53142), 101532. Available at: https://doi.org/10.1016/j.softx.2023.101532

Fraanje, R., Koreneef, T., Le Mair, A. and de Jong, S. (2016) 'Python in robotics and mechatronics education', in Proceedings of the 2016 11th France-Japan & 9th Europe-Asia Congress on Mechatronics (MECATRONICS) / 17th International Conference on Research and Education in Mechatronics (REM), June. Available at: https://doi.org/10.1109/MECATRONICS.2016.7547108

Gode, C., Kakkeri, R.B., Thoutam, N. and Jaya, — (2025) 'Autonomous Suturing in Robotic Surgery Using Reinforcement Learning and 3D Visual Feedback', Journal of Neonatal Surgery, 14(10S), pp. 24–35. Available at: https://doi.org/10.52783/jns.v14.2754

González Dorantes, A.N. and Orozco-Soto, S.M. (2024) Robust Visual Feedback for the Control of a Robot Manipulator with Position Actuator. 45(1), pp. 38–50. Available at: https://www.researchgate.net/publication/381461174_Robust_Visual_Feedback_for_the_Control_of_a_Robot_Manipulator_with_Position_Actuator (Accessed: 27 October 2025).

Haresh, S., Dijkman, D., Bhattacharyya, A. and Memisevic, R. (2024) 'ClevrSkills: Compositional Language and Visual Reasoning in Robotics', arXiv preprint, November. Available at: https://doi.org/10.48550/arXiv.2411.09052

Harrington, K. (2025). Reinforcement Learning and Its Applications in Robotics. March 2025. Available at: https://www.researchgate.net/publication/391866613_Reinforcement_Learning_and_Its_Applications_in_Robotics (Accessed: 27 October 2025).

Kee, E., Chong, J.J., Choong, Z.J. and Lau, M. (2024) 'Object Detection with Hyperparameter and Image Enhancement Optimisation for a Smart and Lean Pick-and-Place Solution', Signals, 5(1), pp. 87–104. Available at: https://doi.org/10.3390/signals5010005

Kumar, R.V.N. and Sreenivasulu, R. (2019) 'Inverse Kinematics (IK) Solution of a Robotic Manipulator using PYTHON', Journal of Mechatronics and Robotics, 3(1), pp. 542–551. Available at: https://doi.org/10.3844/jmrsp.2019.542.551

Lawrence, E. (2024) Reinforcement Learning for Adaptive Robotic Control: From Simulation to Real-World Deployment. Available at: https://www.researchgate.net/publication/390107375_Reinforcement_Learning_for_Adaptive_Robotic_Control_From_Simulation_to_Real-World_Deployment(Accessed: 27 October 2025).

Moghadam, M., Christensen, D.J., Brandt, D. and Schultz, U. (2013) 'Towards Python-based Domain-specific Languages for Self-reconfigurable Modular Robotics Research', February. Available at: https://www.researchgate.net/publication/235696893_Towards_Python-based_Domain-specific_Languages_for_Self-reconfigurableModular_Robotics_Research (Accessed: 27 October 2025).

Nedelcu, D. and Latinovic, T. (2025) Programming engineering applications in Python, wxPython and Streamlit., February. Available at: https://www.researchgate.net/publication/389264609_Programming_engineering_applications_in_Python_wxPython_and_Streamlit (Accessed: 27 October 2025).

Platt, J. and Ricks, K. (2022) 'Comparative Analysis of ROS-Unity3D and ROS-Gazebo for Mobile Ground Robot Simulation', Journal of Intelligent & Robotic Systems, 106, 80. Available at: https://doi.org/10.1007/s10846-022-01766-2

Song, P. (2025) 'Object Detection based on HSV in OpenCV', Applied and Computational Engineering, 121(1), pp. 116–122. Available at: https://doi.org/10.54254/2755-2721/2025.19740

Soto, G., Maina, F.I. and Byiringiro, J.B. (2024) 'Nonlinear Parameter Identification of the Franka Emika PANDA Robot: A Comparative Analysis of Friction Models', Andalas Journal of Electrical and Electronic Engineering Technology, 4(2), pp. 45–57. Available at: https://doi.org/10.25077/ajeeet.v4i2.101

Thaker, R. (2024) ‘Reinforcement Learning in Robotics: Exploring Sim-to-Real Transfer, Imitation Learning, and Transfer Learning Techniques’, 10(5). Available at: https://doi.org/10.5281/zenodo.14001716

Wang, X., Pan, H., Zhang, H., Li, M., Hu, S., Zhou, Z., Xue, L., Guo, P., Wang, Y., Wan, W., Liu, A. and Zhang, L.Y. (2024) ‘TrojanRobot: Backdoor Attacks Against Robotic Manipulation in the Physical World’, arXiv preprint, November. Available at: https://doi.org/10.48550/arXiv.2411.11683

Weinberg, A.I., Shirizly, A., Azulay, O. and Sintov, A. (2024) ‘Survey of learning-based approaches for robotic in-hand manipulation’, Frontiers in Robotics and AI, 11, 1455431. Available at: https://doi.org/10.3389/frobt.2024.1455431

Wen, K., Wu, L., Wang, C., Zhang, Q., Zeng, J. and Huang, G. (2024) ‘High-precision positioning system for composite robots based on visual feedback’, Journal of Physics: Conference Series, 2897(1), 012008. Available at: https://doi.org/10.1088/1742-6596/2897/1/012008

Weng, W.-T., Huang, H.-P., Zhao, Y.-L. and Lin, C.-Y. (2022) ‘Development of a Visual Perception System on a Dual-Arm Mobile Robot for Human-Robot Interaction’, Sensors, 22(23), 9545. Available at: https://doi.org/10.3390/s22239545

Ye, Y., Nie, Z., Liu, X., Xie, F., Li, Z. and Li, P. (2023) ‘ROS2 Real-time Performance Optimization and Evaluation’, Chinese Journal of Mechanical Engineering, [online] Available at: https://doi.org/10.1186/s10033-023-00976-5

10. ANNEXES

10.1. Annex A – URDF and Configuration Files

This annex contains the robot's URDF XACRO and YAML configuration files used for setting up the Panda manipulator in the ROS 2 environment. The URDF defines the robot's kinematic and dynamic structure, while the YAML files specify the controller parameters and joint limits used during simulation and testing. All files were created as XACRO components and merged into a single urdf.xacro file to ensure a more dynamic, and easy-to-debug setup. The final configuration is shown in *Figure A.1*.

Figure A.1: The robot.urdf.xacro file connecting all Panda components (panda.xacro, camera.xacro, and gazebo_joint_controller.xacro) into a unified robot description

10.2. Annex B – External Packages and Dependencies

This annex lists the external repositories and libraries integrated into the project. The *ifra_linkattacher*

This package was used to simulate dynamic link attachments between the Panda manipulator and objects within Gazebo. It enables runtime attachment and detachment of links using ROS2 services.
Repository:https://github.com/IFRA-Cranfield/IFRA_LinkAttacher
Ultralytics

Used to run real-time object detection on the robot's camera stream for visual feedback, on-screen overlays, and logging of detection confidence/centroids. Integrated via ultralytics import YOLO, loading the lightweight yolov8n.pt checkpoint for fast inference.

Repository:https://github.com/ultralytics/ultralytics

Panda Robot

The Franka Emika Panda is an open-source, seven-degree-of-freedom collaborative robotic arm widely used in research and education.

Repository:https://github.com/noshluk2/ROS2-Ultimate-guide-for-Custom-Robotic-Arms-and-Panda-7-DOF/tree/master/franka_description/meshes

Components

The bin and table used in the simulation environment were reproduced from standard Gazebo model assets available through the official Gazebo online model library: https://app.gazebosim.org/dashboard

10.3. Annex C – Declarations

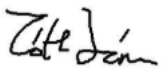
DECLARATION

Ilbay Umut (EKE8AH) as a consultant, I declare that I have reviewed the final thesis and that I have informed the student of the requirements, legal and ethical rules for the correct handling of literary sources.

I recommend / do not recommend[1] the final thesis / dissertation / portfolio to be defended in the final examination.

The thesis contains a state or official secret: yes no*[2]

Date: _2025_year _10_ month _28_ day



insider consultant

Declaration of Students and Doctoral Candidates on the Use of Artificial Intelligence (AI)"

1. general information:

Name of the student:	Umut Ilbay
Neptun ID:	EKE8AH
Level of program (mark with X):	■BSc/BA ☐ MSc/MA ☐ Doctoral School (PhD) ☐ Other: ……
Name and code of the subject*:	Thesis Work 2 /MUSZK340N
Title of the work:	Design and control of robotic arms for industrial applications

* Not required to be completed in the case of a doctoral dissertation.

2. Declaration on the Use of AI

I, the undersigned, fully aware of my ethical responsibility, make the following declaration:

(Please choose one of the options below!)

☐ A) I have not used any artificial intelligence system or service.

(If you selected this option, completing the subsequent tables is not required.)

■ B) I have used an artificial intelligence system or service.

(Please fill in the relevant tables!)

3. Details of Artificial Intelligence Usage

TABLE I: Assistant or Minor Usage (e.g., translation, language proofreading, brainstorming, etc.)

(For these uses, attaching the specific prompts and responses is not required.)

Purpose of Use	**Name and Version of the AI Tool Used**	**Affected Section (if not applicable to the entire text)**
Text grammar and Flow Correction	ChatGPT 5	General texts

TABLE II: Significant Content Contribution (e.g., generating an entire figure or a longer text section)

(In these cases, documenting the key prompts used and the raw responses provided by the AI, and attaching them as an appendix to the work, is required.)

Purpose of Use	Name, Version, and Access Information of the AI Tool Used	Exact Number of the Affected Chapter / Figure / Table	Entry Number of the Appendix Containing the Prompt Log

3/A. Additional Rules Prescribed by the Lecturer (if any)

If the instructor or supervisor of the course has established specific rules or expectations regarding the use of AI tools, please summarize them in the field below:

For example: prohibition of AI use for certain types of tasks; only specific tools are permitted; different citation requirements; documentation format, etc.

Rules Prescribed by the Lecturer or Supervisor

.....

.....

.....

.....

4. Declaration Applicable to All Students:

I declare that I have critically reviewed, edited, and incorporated any content potentially generated by AI in all cases. I take full responsibility for every element of the submitted work, including its originality and scientific validity. I acknowledge that the Hungarian University of Agriculture and Life Sciences may check the submitted work with an artificial intelligence detector and may initiate proceedings if my declaration is found to be false or incomplete.

Place and Date: Hungary, Gödöllő, 2025/10/27

.....

Signature of the Student

.....

Signature of the Advisor/Supervisor

MATE Organizational and Operational Regulations
III. Requirements for Students
III.1. Study and Examination Regulations
Appendix 6.13: The MATE Uniform Thesis /thesis / final thesis / portfolio guidelines
Annex 4.2: Declaration of public access and authenticity of the thesis/thesis/dissertation/portfolio

DECLARATION

the public access and authenticity of the thesis/dissertation/portfolio[1]

Student's name:	Umut Ilbay
Student's Neptun code:	EKE8AH
Title of thesis:	Design and control of robotic arms for industrial application
Year of publication:	2025
Name of the consultant's institute:	Institute of Technology
Name of consultant's deparment:	Mechanical Engineering

I declare that the final thesis/thesis/dissertation/portfolio[2] submitted by me is an individual, original work of my own intellectual creation. I have clearly indicated the parts of my thesis or dissertation which I have taken from other authors' work and have included them in the bibliography. Furthermore, I declare that the artificial intelligence tools (e.g. text generation, linguistic correction, translation, data analysis) used during the preparation of the thesis did not substitute my own research and creative work; their use was indicated either in the list of sources or in the methodology section, and I acted in accordance with professional and ethical expectations.

If the above statement is untrue, I understand that I will be disqualified from the final examination by the final examination board and that I will have to take the final examination after writing a new thesis.

I do not allow editing of the submitted thesis, but I allow the viewing and printing, which is a PDF document.

I acknowledge that the use and exploitation of my thesis as an intellectual work is governed by the intellectual property management regulations of the Hungarian University of Agricultural and Life Sciences.

I acknowledge that the electronic version of my thesis will be uploaded to the library repository of the Hungarian University of Agricultural and Life Sciences. I acknowledge that the defended and

- not confidential thesis after the defence
- confidential thesis 5 years after the submission

will be available publicly and can be searched in the repository system of the University.

Date: 2025/10/27

Student's signature

[1] While keeping the appropriate thesis type, all other types are to be removed.
[2] While keeping the appropriate thesis type, all other types are to be removed.0