

Remote Object Pose Estimation for Agile Grasping: Leveraging Cloud Computing through Wireless Communication

by

Oleksii Zamozhskyi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2025

© Oleksii Zamozhskyi 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

As industry transitions toward Industry 4.0, the demand for agile robotic systems capable of vision-guided manipulation is rapidly increasing. However, the computational limitations of onboard hardware make it challenging to support advanced perception pipelines, particularly those based on deep learning. Offloading perception to the cloud presents a promising alternative but introduces latency and reliability challenges that can compromise the real-time performance required for closed-loop robotic control. This thesis presents a robotic grasping system capable of agile, 6D pose-aware manipulation of moving objects by offloading perception to a remote inference server. RGB-D data is continuously streamed over a wireless link to the server, where a deep learning model estimates the object’s 6D pose. The estimated pose is then sent back to the robot, which uses it to generate a trajectory for executing the grasp. The system was evaluated on a conveyor-based pick-and-place task under four different wireless network types: Wi-Fi at 60 GHz, Wi-Fi 5 at 5 GHz, 5G NSA at 24 GHz, and 5G NSA at 3.5 GHz. A total of 392 trials were conducted to analyze grasping success rates and the impact of network latency and reliability on performance. The results demonstrate the feasibility of performing agile, closed-loop robotic grasping with cloud-offloaded 6D pose estimation over wireless networks. They also reveal limitations of current wireless infrastructure and deep learning models. The findings suggest that lower-latency, more reliable communication, along with more intelligent local control strategies and faster, generalizing models, are required for production deployment.

Acknowledgements

I want to thank Alexander Werner for spending countless hours helping me and for the many pieces of advice that kept things moving when I was completely lost. I am also sincerely thankful to my supervisors, Professor Marie Charbonneau and Professor William Melek, for their invaluable feedback and guidance, which greatly shaped this work.

This thesis also benefited from the use of OpenAI's ChatGPT language model for grammar correction, stylistic refinement, and academic rephrasing of selected passages. All substantive content, ideas, and analysis remain the sole work of the author.

Dedication

To my friend, Lavok Ailli Sol.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	ix
List of Tables	xii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Research Gap	3
1.4 Contributions	3
2 Theoretical Background	5
2.1 Robot Manipulation and Control	5
2.1.1 Rigid Body Motion and Robot Manipulation	5
2.1.2 Model-Based Control in Joint and Task Space	7
2.2 Core Concepts in Wireless Communication	9

3	Literature Review	10
3.1	6D Pose Estimation	11
3.1.1	6D Pose Estimation Classical Methods	11
3.1.2	6D Pose Estimation Using Deep Learning	13
3.2	Object Manipulation	16
3.2.1	Open-Loop Grasping in Static Environments	16
3.2.2	Closed-Loop Grasping in Static Environments	17
3.2.3	Closed-Loop Grasping in Dynamic Environments	18
3.2.4	Remote Processing	21
4	Experimental Setup	24
4.1	Robot Platform	25
4.2	Conveyor and Object	25
4.3	Remote Perception Server	26
4.4	Communication Setup	26
4.5	Wireless Network Configurations	27
5	Methodology	28
5.1	System Overview	28
5.2	Pose Estimation Flow	30
5.2.1	FoundationPose Deep-Learning Model	30
5.2.2	Robot-Side Sender	32
5.2.3	Server-Side Pose Estimation	33
5.2.4	Robot-Side Receiver	35
5.3	State Machine	36
5.3.1	Idle State	38
5.3.2	Registration State	39
5.3.3	Compute Approach Point State	40

5.3.4	Approach State	47
5.3.5	Visual Servoing State	47
5.3.6	Lift State	57
6	Experiment	59
6.1	System Behavior Illustration	59
6.2	Experimental Procedure	62
6.2.1	Categorization of Grasp Outcomes	63
6.2.2	Set A: High-Speed Experiments	64
6.2.3	Set B: Moderate-Speed Experiments	66
6.2.4	Network Stability Evaluation	67
6.3	Results	67
6.3.1	Results - Set A	67
6.3.2	Results – Set B	81
6.3.3	Results Summary	90
7	Conclusion	92
	References	95

List of Figures

4.1	System components: robot arm, RGB-D camera, control PC, 5G modem, conveyor belt, object of interest, and remote inference server	24
4.2	End-effector with mounted Intel RealSense D405 RGB-D camera and custom 3D-printed fingers, shown from multiple angles.	25
5.1	System pipeline illustrating	29
5.2	Flow diagram of the server-side processing pipeline. The server receives an RGB-D data and selects between registration and tracking modes. In registration mode, a color-based segmentation mask is generated to verify the presence of the object. If detected, registration inference is run, and the system switches to tracking mode. In tracking mode, pose updates are computed directly from the RGB-D input. In both cases, the estimated 6D pose is published to the robot.	34
5.3	Illustration of the robot's motion during a grasping task.	37
5.4	Diagram of the state machine that governs the grasping task. Arrows indicate transitions, labeled with the conditions that trigger them.	38
5.5	(a) Frame used for the registration. (b) Next frame captured by the camera. (c) Next frame processed by the server after registration delay. This figure illustrates the temporal gap caused by registration latency. While the object continues to move, intermediate frames are dropped, resulting in a significant visual difference between consecutive frames used by the system.	40
6.1	Object (purple) and end-effector (blue) positions along the X axis in the base frame over time. Background colors indicate the robot's control state: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown).	60

6.2	Object (purple) and end-effector (blue) positions along the Y axis in the base frame over time. Background colors indicate the robot's control state: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown).	61
6.3	Object (purple) and end-effector (blue) positions along the Z axis in the base frame over time. Background colors indicate the robot's control state: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown).	62
6.4	Initial object positions used in the experiments. The three placements represent different starting positions along the conveyor.	65
6.5	Initial object orientations used in the experiments.	66
6.6	Set A: outcomes across 50 trials using Wi-Fi at 60 GHz, categorized by success and failure reason	68
6.7	Set A: outcomes across 50 trials using Wi-Fi 5 at 5 GHz, categorized by success and failure reason	70
6.8	(a) Object X-position estimates for each frame: orange dots indicate when the robot received the estimate, and purple dots show when the corresponding camera frame was actually captured. (b) Same data as in (a), with the corresponding predicted estimates shown in red. (c) Comparison of object and end-effector X-positions during the grasping sequence, illustrating tracking and alignment behavior.	72
6.9	Set A: outcomes across 50 trials using 5G NSA at 24 GHz, categorized by success and failure reason	73
6.10	Set A: outcomes across 50 trials using 5G NSA at 3.5 GHz, categorized by success and failure reason	75
6.11	Outcome distribution for Set A experiments across four wireless network configurations. Each bar represents 50 trials, broken down by success and specific failure types.	77
6.12	Distributions of total round-trip time (RTT) and server processing time recorded for each network type in Set A.	79
6.13	Set B: outcomes across 48 trials using Wi-Fi at 60 GHz, categorized by success and failure reason	82
6.14	Set B: outcomes across 48 trials using Wi-Fi 5 at 5 GHz, categorized by success and failure reason	83

6.15	(a)–(c) show Approach (orange), Visual Servoing (green), and Lift (brown) states of a run with delayed perception where the grasp succeeded. (a) shows the estimated object X-position, with purple dots indicating capture times and orange dots indicating reception times. (b) includes the prediction from our filtering mechanism (red), which compensated for the delay. (c) compares the object and end-effector (blue) X-positions during the grasp. .	84
6.16	(a)–(c) show Approach (orange) and Visual Servoing (green) states of a run with delayed perception where the grasp failed. (a) shows the estimated object X-position, with purple dots indicating capture times and orange dots indicating reception times. (b) includes the prediction from our filtering mechanism (red), which compensated for the delay. (c) compares the object and end-effector (blue) X-positions during the grasp.	85
6.17	Set B: outcomes across 48 trials using 5G NSA at 24 GHz, categorized by success and failure reason	87
6.18	Set B: outcomes across 48 trials using 5G NSA at 3.5 GHz, categorized by success and failure reason	88
6.19	Outcome distribution for Set B experiments across four wireless network configurations. Each bar represents 48 trials, broken down by success and specific failure types.	89

List of Tables

6.1	Grasp success and failure counts, along with the average and standard deviation of RTT and server processing time for all four networks in Set A.	77
6.2	Percentage of single and five consecutive low-throughput intervals observed during the 24-hour streaming test for each network in Set A.	80
6.3	Tracking performance during the approach state in Set A, reported per network. “Tracking Lost” indicates the number of runs where tracking failed during the approach state, while “Tracking OK” refers to runs that completed successfully. FPS denotes the average number of pose estimates received per second, computed across all runs for each network.	80
6.4	The table lists the number of successful grasps, the maximum x-axis deviation observed during those successful trials, the number of failures caused by velocity constraint violations, and the corresponding maximum deviation during those failed attempts. All measurements were taken during the visual servoing state.	81
6.5	Grasp success and failure counts, along with the average and standard deviation of RTT and server processing time for all four networks in Set B.	89
6.6	Number of failures observed for each initial object position in the Set B experiments.	90
6.7	Number of failures observed for each initial object orientation in the Set B experiments.	90

Chapter 1

Introduction

1.1 Motivation

Modern industry is shifting towards smart factories and Industry 4.0, where advanced robots powered by AI play a significant role. In particular, robotic arms have become essential in manufacturing, taking over repetitive and high-load tasks with greater speed, reliability, and accuracy than humans can achieve. Among these tasks, grasping stands as a core capability. It is widely used in the majority of automation pipelines and is often critical to the success of downstream operations [7]. While many research efforts have been made to address achieving reliable and agile grasping, it remains an open challenge, especially in dynamic and unstructured environments, where object shape, lighting, and surroundings can change unpredictably [30]. This makes grasping an active direction for the research.

Recent advances in deep learning have significantly improved models' ability to perceive and reason about their surroundings. Modern neural networks can interpret images to detect [55] and classify objects [15], reason about their spatial relationships [9], estimate their 6D pose [57], and predict grasp poses [54]. These capabilities enable more adaptive and efficient robotic behavior beyond traditional hand-crafted rules. However, this power comes at a cost: state-of-the-art perception models are often too computationally heavy to run on typical robot hardware [31]. As a result, deploying high-end GPUs to a robot to support computational requirements becomes infeasible due to limitations in weight, power, or cost, especially in scalable or mobile applications.

A promising solution is to leverage cloud computing and wireless networks to offload heavy computational tasks from the robot to a remote server. In the emerging paradigm of

cloud robotics [53, 32], robots send sensor data to an edge or cloud server, where a powerful AI model processes the data and sends back the results or even full control commands. This approach enables the robots to leverage “unlimited” computation resources available in the cloud. In principle, it enables lighter, cheaper, and more intelligent robot systems, pushing their performance far beyond what current systems can achieve.

Technological developments are making this vision more feasible. High-speed wireless networks, such as 5G, promise a significant improvement for robotics. With its ultra-reliable low-latency communication (URLLC), 5G claims to support fast and robust data exchange, which is essential for real-time control. Early research has already demonstrated the feasibility of integrating 5G into robotic systems and the advantages it offers [63, 52, 23]. This motivates our work to further explore the integration of 5G and cloud computing in robotic systems. As this approach is still in its early stages, it is essential to examine the challenges, limitations, and trade-offs associated with wireless links when employed for real-time control in order to develop practical and reliable systems.

1.2 Problem Statement

As modern industrial systems increasingly depend on data-intensive components such as object detection and pose estimation [66], the choice of network technology becomes a critical factor influencing system performance. While wired connections offer low latency and high reliability, they impose deployment constraints by requiring computational resources to be collocated with each robot. Wireless connectivity presents a viable alternative, enabling remote processing while supporting more scalable and flexible deployment across manufacturing environments. However, they often suffer from latency, jitter, and packet loss [35]. These problems are most significant in real-time closed-loop systems, where the robot must react quickly and consistently to changing inputs. Delays or missing data can lead to slower responses, unstable behavior, or even failed tasks [45, 18].

Wi-Fi has traditionally been the default choice for wireless communication in robotic systems and remains widely used in manufacturing due to its accessibility and ease of integration [12]. However, it often introduces unstable latency and unreliable connections, including packet loss and interference in crowded environments [8].

5G has been introduced as a next-generation wireless technology offering significant improvements in latency, reliability, and bandwidth [14]. These characteristics make it a promising candidate for supporting remote computation in robotic systems that rely on time-critical control. However, despite its technical potential, the use of 5G in closed-loop robotic applications remains largely unexplored.

Although early demonstrations are promising [34, 21], it remains unclear whether technologies such as 5G or Wi-Fi can meet the demands of real-time robotic control in dynamic environments and how network characteristics affect system performance.

1.3 Research Gap

While robotic manipulation [59], advanced visual perception [24], and wireless communication [64] have each been studied extensively, existing research tends to isolate these components rather than explore them together in a unified system. In particular, current work does not fully address the challenge of performing closed-loop grasping with cloud-based perception over a wireless network in a dynamic scenario.

Some systems reduce computational demands by simplifying perception [26, 60, 22]. Instead of using deep learning, they rely on fiducial markers or lightweight vision algorithms. This makes the system easier to run on limited hardware, but restricts it to labeled or constrained environments. It cannot generalize to a wide variety of object types or scenes.

At the opposite end, high-performance systems demonstrate that deep learning can support closed-loop, real-time systems, such as tracking fast-moving objects [44, 20, 47]. These systems run all perception and control locally using powerful onboard hardware. While effective, this approach is expensive, power-intensive, and challenging to scale. Each robot must carry a GPU or equivalent, making it unsuitable for cost-sensitive or mobile deployments.

Other works do offload computation to remote servers, but avoid dynamic scenarios [63, 21]. They rely on one-shot perception and open-loop execution to avoid latency and reliability issues. As a result, these systems cannot adapt during execution or respond to moving objects, and do not evaluate the effect of wireless communication on feedback control.

There is a clear need for research that integrates these elements: dynamic object manipulation, continuous visual feedback, and remote perception over a wireless link. To our knowledge, no prior work has demonstrated a complete system that combines all three.

1.4 Contributions

This thesis investigates whether modern wireless networks, specifically Wi-Fi and 5G, are suitable for supporting real-time, closed-loop robotic control in dynamic environments.

The work focuses on a representative use case: 6D pose-aware pick-and-place on a moving conveyor belt. This task is both industrially relevant and particularly demanding in terms of latency and control precision, making it a suitable benchmark for evaluating cloud-based perception in closed-loop operation.

To explore this, a complete robotic system was developed around the Franka Emika Panda arm, using an Intel RealSense D405 RGB-D camera for visual input. All perception is offloaded to a remote server equipped with an NVIDIA RTX 4090 GPU. The server runs FoundationPose, a deep-learning model that estimates the 6D pose of the target object from camera images. Pose estimates are streamed back to the robot over a wireless connection, which then uses them to guide its actions in real-time. The system architecture enables full remote visual inference, highlighting the challenges of control under network constraints.

This work is set up to answer the following research questions:

- Can closed-loop robotic grasping be performed reliably when 6D object pose estimation is offloaded over a wireless network?
- How do network latency and reliability affect grasp success and system responsiveness?
- How does grasping performance compare across different wireless networks? The evaluation includes Wi-Fi at 60 GHz, Wi-Fi 5 at 5 GHz, 5G NSA at 24 GHz, 5G NSA at 3.5 GHz.
- How robust is a deep learning-based object tracking system when deployed in a dynamic, industrial-style environment?

Two sets of experiments were conducted using different conveyor belt speeds, with each trial repeated across the four network configurations to evaluate these questions in practice.

The structure of this thesis is organized to support a clear and logical presentation of the work. Chapter 2 introduces the theoretical background on robotic manipulation and control, as well as relevant concepts in wireless communication. Chapter 3 provides an overview of existing methods for 6D pose estimation and reviews prior work on object manipulation, highlighting open challenges that remain in the field. Chapter 4 explains the experimental setup used in this work. Chapter 5 describes the proposed methods and system design. Chapter 6 describes the evaluation procedure and reports the experimental results and system performance. Chapter 7 concludes the thesis with a summary of the findings and suggestions for future research.

Chapter 2

Theoretical Background

To develop a robot control system with wireless connectivity, this thesis combines knowledge from the fields of robot manipulation and control, as well as wireless communication. These topics are presented in Sections [2.1](#) and [2.2](#) below.

2.1 Robot Manipulation and Control

This section presents the core concepts used to describe and control the motion of robotic manipulators. It covers the main representations for rigid body pose, velocity, and force, along with the standard kinematics and dynamics models. The final part introduces feed-back control in both joint and task space. The formulations used follow standard notation from *Modern Robotics* by Lynch and Park [\[38\]](#).

2.1.1 Rigid Body Motion and Robot Manipulation

A rigid body in three-dimensional space has six degrees of freedom: three translational and three rotational. Its configuration relative to a fixed reference frame is represented by a homogeneous transformation matrix $T \in SE(3)$:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

where $R \in SO(3)$ is a rotation matrix describing orientation, and $p \in \mathbb{R}^3$ is a position vector describing the origin of the body frame. This matrix enables the mapping of coordinates

from one frame to another and is commonly used to represent the pose of a rigid body. In expressions such as T_A^B , the subscript A indicates the frame whose pose is being described, and the superscript B indicates the reference frame in which this pose is expressed.

Rigid body motion is characterized by the twist, a six-dimensional vector that combines angular velocity $\omega \in \mathbb{R}^3$ and linear velocity $v \in \mathbb{R}^3$:

$$V = \begin{bmatrix} \omega \\ v \end{bmatrix} \in \mathbb{R}^6$$

External forces and torques acting on the body are similarly grouped into a wrench:

$$F = \begin{bmatrix} m \\ f \end{bmatrix} \in \mathbb{R}^6$$

where $f \in \mathbb{R}^3$ is the applied force and $m = r \times f$ is the resulting torque about the reference. Here, $r \in \mathbb{R}^3$ is the position vector from the reference point to the point of force application.

A robotic manipulator with n revolute joints has n degrees of freedom (DOF), meaning it can independently control n joint variables. Such manipulators are typically structured as serial kinematic chains, where each joint contributes one rotational DOF. The pose of the end-effector relative to the robot's base frame is determined by the forward kinematics function, which maps joint configurations to a spatial transformation. Given the joint position configuration $q \in \mathbb{R}^n$, the end-effector pose $T_{ee}^b(q) \in SE(3)$, representing the rigid-body transformation from the base frame to the end-effector frame, is defined as

$$T_{ee}^b(q) = FK(q)$$

where FK denotes the forward kinematics function. This function depends on the specific geometry and structure of the manipulator, including link lengths, joint axes, and their spatial arrangement.

The relationship between joint velocities and end-effector twist is given by the Jacobian $J \in \mathbb{R}^{6 \times n}$:

$$V_{ee}^b = J(q)\dot{q}$$

This mapping is essential in differential kinematics and control, allowing motion commands in joint space to be interpreted in Cartesian space.

When a wrench $F_{ee} \in \mathbb{R}^6$ is applied at the end-effector, the resulting joint torques are computed using the Jacobian transpose:

$$\tau_{ext} = J^T(q)F_{ee}$$

This expression provides a direct link between Cartesian forces and the corresponding actuation in joint space.

The full joint-space dynamics of the manipulator, including inertia, Coriolis and centripetal effects, and gravity, is given by:

$$\tau = M(q)\ddot{q} + C(q, \dot{q}) + g(q)$$

where $M(q) \in \mathbb{R}^{n \times n}$ is the joint-space inertia matrix or mass matrix, $C(q, \dot{q}) \in \mathbb{R}^n$ includes Coriolis and centrifugal terms, and $g(q) \in \mathbb{R}^n$ represents gravitational effects. Often, terms that don't depend on acceleration are grouped together:

$$h(q, \dot{q}) = C(q, \dot{q}) + g(q)$$

If the robot applies an external wrench F_{ee} at the end-effector, this can be included in the dynamics equation:

$$\tau = M(q)\ddot{q} + h(q, \dot{q}) + J^T(q)F_{ee}$$

It is also possible to write this equation in task space. If an external wrench F_{ee} , expressed in the base frame, is applied at the end-effector, and the actuators generate no torques or forces in response, then the motion of the end-effector is solely governed by the following equations:

$$F_{ee} = \Lambda(q)\dot{V}_{ee}^b + \eta(q, V_{ee}^b)$$

where

$$\begin{aligned}\Lambda(q) &= J^{-T}M(q)J^{-1} \\ \eta(q, V_{ee}^b) &= J^{-T}h(q, J^{-1}V_{ee}^b) - \Lambda(q)\dot{J}J^{-1}V_{ee}^b\end{aligned}$$

2.1.2 Model-Based Control in Joint and Task Space

In robotic manipulation, control systems generate actuation commands to achieve desired motion despite unmodeled dynamics, disturbances, or delays. Feedback is essential, as no dynamic model is perfectly accurate. However, incorporating a model into the control loop improves performance and enables more predictable system behavior [38]. Unless stated otherwise, all task-space quantities (positions, velocities, accelerations, and wrenches) are expressed in the robot's base frame.

A common strategy is Proportional-Derivative (PD) control, which applies joint torques based on the error between desired and measured joint positions and velocities:

$$\tau = K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})$$

Here, q_d and $\dot{q}_d$ are the desired joint position and velocity, q and $\dot{q}$ are the measured values, and K_p and K_d are gain matrices that shape the system's stiffness and damping.

While simple, PD control does not account for the robot's dynamics. To improve tracking and compensate for these effects, the controller can be extended with feedforward model terms, resulting in the inverse dynamics controller:

$$\tau = M(q)\ddot{q}_d + h(q, \dot{q}) + K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}) \quad (2.1)$$

This expression combines:

- A feedforward torque based on the desired acceleration $\ddot{q}_d$ and the robot's model (M, C, g)
- A feedback term based on joint tracking errors.

The model-based terms produce the nominal torques needed to follow the desired trajectory, while the PD feedback corrects deviations and stabilizes the system.

An equivalent formulation exists in task or Cartesian space, where the control target is the end-effector pose rather than joint angles. The Cartesian control law takes the form:

$$F_{ee} = \Lambda(q)\ddot{x}_d + \eta(q, V) + K_p(x_d - x) + K_d(\dot{x}_d - \dot{x}) \quad (2.2)$$

Here, $\ddot{x}_d \in \mathbb{R}^6$ represents the full spatial acceleration of the end-effector, including both linear and angular components, and is equivalent to $\ddot{V}_d$. The corresponding joint torques are then computed using the Jacobian transpose:

$$\tau = J^T(q)F_{ee}$$

This task-space control strategy is particularly useful when the objective is defined in Cartesian coordinates. In visual servoing, for instance, the robot must adjust its motion to track an object whose pose is given in task space. Since the end-effector trajectory is defined relative to the object, expressing the control law in Cartesian space simplifies the formulation and eliminates the need to solve inverse kinematics.

2.2 Core Concepts in Wireless Communication

This section presents the core terminology used to describe wireless network behavior, focusing on how these properties affect delay and reliability in systems that depend on timely and reliable data transmission.

Wireless communications introduce several characteristics that influence the performance of downstream tasks, particularly those that rely on remote communication and real-time data exchanges. These characteristics can be grouped into two categories: delay and reliability. Among delay-related properties, **latency** refers to the time it takes for the data to travel from a client to a server. While latency often refers to the delay introduced by the communication link itself, **round-trip time(RTT)** captures the total time between a client sending and receiving the data. RTT is particularly relevant to systems that offload computation, as it measures both network delay and processing time. In addition to the average delay, the variation in delay is also important. This is known as **jitter**, and it can cause the data to arrive inconsistently, even if the average latency remains within acceptable limits. While jitter is often considered a timing issue, it also reflects underlying reliability problems in the communication link.

Reliability itself encompasses several factors that affect the consistency and predictability of data delivery. One of the main common issues is **packet loss**, where the transmitted data never reaches its destination. Even occasional losses can cause the system to operate on outdated or incomplete information, resulting in degraded performance. Another key metric is **throughput**, which measures the actual rate at which this data is transmitted. This is often lower than **bandwidth**, which is the theoretical maximum that can be sent. This often happens due to protocol overhead, interference, or retransmissions. In a wireless setting, especially when a large amount of data needs to be transmitted, for example, sensor data, limited throughput becomes a bottleneck. These issues are often worsened by network congestion, which occurs when multiple devices compete for the same communication channel. Congestion leads to queuing delays, increased jitter, and a higher risk of packet drops. To mitigate these effects, many wireless systems implement Quality of Service (QoS) mechanisms [41], which enable the prioritization of specific types of traffic. For example, time-sensitive data can be marked as high-priority to ensure it is transmitted ahead of less critical traffic. When properly configured, QoS policies can reduce latency and jitter, improve reliability, and make overall system behavior more predictable, even under high network load. In addition to software-level prioritization, it is also important to deploy the network with appropriate hardware and capacity planning, ensuring that the infrastructure can support the intended use case and expected traffic volume.

Chapter 3

Literature Review

This chapter reviews related work in two key areas relevant to this thesis: 6D pose estimation from visual data and robotic manipulation. These topics were selected to support the design of the system and to position the contribution within the broader research context.

Section 3.1 focuses on 6D pose estimation from visual input. Although this thesis does not introduce new methods in this domain, it includes a comprehensive review of existing approaches, covering a wide range of methods based on diverse underlying principles. These include classical feature-based pipelines as well as modern deep learning methods. The goal was to understand the current state of the field, identify what is practically achievable, and examine the challenges and limitations of current techniques. These insights heavily influenced the overall system design, as the control architecture and planning strategy needed to account for the specific constraints and requirements imposed by the perception module.

Section 3.2 surveys robotic manipulation, focusing on works that combine perception, specifically visual input, with control. A variety of systems were examined and grouped based on their assumptions, task settings, and how these shaped the integration of perception and action. The goal was to identify limitations in existing approaches and highlight gaps relevant to our problem. This helped to support and frame the contribution of this thesis. The review focused on works demonstrated on physical systems and excluded those based purely on simulation.

3.1 6D Pose Estimation

6D pose estimation refers to the task of determining the six-degree-of-freedom (6D) pose of an object in 3D space [1]. This involves estimating the object’s position and orientation. Together, these can be, for example, represented as the object’s rigid-body transform, which is sufficient to uniquely localize and align the object.

This capability is critical in domains such as robotic manipulation, augmented reality, and autonomous navigation, where systems must accurately perceive and interact with physical objects. In robotics, for example, knowing the 6D pose of a target object is a prerequisite for grasping, alignment, or motion planning.

However, estimating 6D pose from visual data is a challenging problem in computer vision [46]. Multiple factors contribute to this difficulty. Environmental conditions such as partial occlusions, lighting conditions, and background clutter can distort object appearance and reduce the reliability of visual features. Object-specific properties, such as textureless surfaces or symmetric shapes, can also introduce ambiguity or degrade detection performance.

Ultimately, many high-accuracy approaches [56, 57, 6, 40] rely on complex models that require substantial computational resources. This limits their use in real-time systems unless inference speed is specifically optimized.

This section reviews the main categories of 6D pose estimation methods, which utilize either RGB or RGB-D input, starting with classical approaches and progressing to modern deep learning-based techniques.

3.1.1 6D Pose Estimation Classical Methods

Before learning-based methods, 6D pose estimation relied entirely on geometric reasoning and handcrafted features. These classical pipelines were often modular, first detecting the object and then matching or aligning it with a known 3D model using visual or geometric correspondences. While interpretable and computationally efficient in controlled environments, they often struggled in real-world scenes with partial views, poor lighting, or occlusion. Several representative approaches that defined early progress in this field are described below.

To illustrate how classical methods approach 6D pose estimation, the discussion begins with methods that operate purely on RGB input and then gradually moves toward those that rely on complete 3D geometric information.

One early family of methods is based on feature matching, which relies on handcrafted 2D descriptors such as SIFT [37] or SURF [3] to extract local features from an RGB image. These key points are then matched to corresponding 3D key points on the object model. The resulting 2D–3D correspondences are used in conjunction with a Perspective-n-Point (PnP) [25] algorithm, combined with Random Sample Consensus (RANSAC) [19], to recover the full 6D pose. These methods work well on feature-rich objects and remain usable even when parts of the object are occluded, as long as enough features remain visible. However, they tend to break down on objects with low texture or under poor lighting conditions. To address these limitations, later approaches shifted to template-based methods, which leverage both RGB and depth data for greater robustness.

Template-based methods rely on comparing the input to a set of pre-rendered object views. These templates encode the global features of the object and are matched to the scene through dense comparisons. A well-known example is LINEMOD, proposed by Hinterstoisser et al. [27], which combines gradient orientation features from RGB with surface normal features from depth. For each template view, these features are precomputed and stored. At runtime, a sliding-window matching process is used to compare each template with the input image. In this approach, a fixed-size window is moved systematically across the image, one small step at a time, both horizontally and vertically. At each position, the algorithm extracts a cropped region and compares it to the template. LINEMOD can operate on RGB, depth, or RGB-D input, and it performs well on low-texture objects where traditional keypoint methods fail. However, it requires an exhaustive number of templates and does not generalize to unseen viewpoints, limiting its scalability and flexibility.

To avoid template generation altogether, some approaches rely entirely on object geometry. The Point Pair Feature (PPF) framework, initially introduced by Drost et al. [17], has since been extended and adapted in various works that build on the same core idea [5, 16, 33]. Instead of storing image-based templates for each viewpoint, PPF operates solely on depth input, representing objects as collections of point pairs and their geometric relationships, such as distances and angles between surface normals. The same features are extracted from the observed depth scene, and similar pairs are matched to generate pose hypotheses. These hypotheses are accumulated in a Hough voting space, and the pose with the highest vote count is selected. PPF is particularly useful in cluttered environments or for textureless objects, but it typically produces coarse pose estimates.

For fine-grained alignment, especially in manipulation tasks where precision is crucial, Iterative Closest Point (ICP) [4] is commonly employed as a refinement step. ICP assumes an initial pose estimate and iteratively improves it by aligning the observed depth cloud

with the object model. For each point in the scene, the algorithm finds the closest point on the model and computes the rigid transformation that minimizes the overall point-to-point distances. This process repeats until convergence. While ICP can achieve high accuracy, it relies on good initialization and overlapping geometry. It is sensitive to occlusion, partial visibility, and sensor noise, and may converge to incorrect local minima if the initial estimate is poor.

In practice, combinations of classical methods are often employed to strike a balance between robustness and precision. For example, Cui et al. [13] proposed a pipeline that utilizes PPF for coarse pose estimation, followed by ICP for refinement, in a robotic bin-picking setup. Similarly, Hu et al. [28] combined SIFT-based feature extraction and matching with ICP to overcome the robustness bottleneck of ICP while retaining its high alignment accuracy.

While classical methods laid the foundation for 6D pose estimation, their dependence on handcrafted features and predefined object models limits their flexibility. Deep learning methods offer a more scalable alternative by learning features from data and capturing non-linear relationships between appearance and pose. Although many still depend on 3D models, their adaptability and progress have made them increasingly dominant in recent pipelines.

3.1.2 6D Pose Estimation Using Deep Learning

This section provides an overview of several influential deep learning models that demonstrate distinct approaches to the 6D pose estimation task.

PoseCNN, proposed by Xiang et al. [61], is one of the earliest deep learning models explicitly designed for estimating the 6D pose of known objects from RGB images. The architecture features a convolutional backbone for feature extraction, along with three separate output heads: a semantic segmentation head for classifying object pixels, a translation head for estimating the 2D center and depth of each object, and a rotation head that regresses a quaternion to represent the object’s orientation. The segmentation output helps isolate objects in cluttered scenes, while the translation prediction utilizes a voting mechanism over segmented pixels to estimate the object center robustly, even under occlusion. To improve robustness when estimating rotations for symmetric objects, PoseCNN introduces a ShapeMatch loss that avoids penalizing rotational ambiguity. Despite its robustness, the model requires retraining for each object. Furthermore, its raw rotation estimates often benefit from refinement. In the original work, the authors utilize an Iterative Closest Point (ICP) algorithm, combined with depth data, to refine the predicted pose. The reliance

on per-object training and post-processing steps limits its scalability and suitability for real-time applications.

PViT-6D, introduced in 2023 by Stapf et al. [50], represents a transformer-based alternative to convolutional approaches for 6D pose estimation. Like PoseCNN, it belongs to the regression-based family of methods, but instead of relying on convolutional neural networks, it leverages a vision transformer architecture. The pipeline adopts a two-stage design: an object detector first identifies regions of interest (RoIs), which are cropped and resized before being passed to the transformer for pose estimation. PViT-6D utilizes a Multi-Scale Vision Transformer (MViT) backbone, which divides the image into patches and applies self-attention to capture global context and long-range dependencies, thereby addressing the limitations of conventional CNNs. A core innovation is the use of two learnable “pose tokens” for translation and rotation, which aggregate information across the image and are processed through separate prediction heads to produce 6D pose estimates. This design enables fully end-to-end regression from RGB input. PViT-6D achieves high accuracy on standard benchmarks and shows strong robustness in cluttered scenes. Despite its accuracy, the main limitation of PViT-6D lies in its reliance on external object detection to extract regions of interest. This introduces an additional design challenge: choosing a detector that is both fast and reliable. The model’s reported inference times exclude this step, so the true latency of the full pipeline is higher and not clearly quantified. Since the model itself is not particularly fast, this added delay may make it too slow for real-time robotic applications. Additionally, the model must be retrained for each new object, as it does not support zero-shot generalization from CAD models alone.

YOLO6D, proposed by Tekin et al. [51], is a real-time single-shot deep learning model for 6D object pose estimation. It builds upon the YOLO family of object detectors [48], performing both object detection and pose estimation in a single network pass. Given an RGB image, the network predicts the 2D image projections of the corners of an object’s 3D bounding box. These 2D projections are then used in conjunction with a PnP algorithm to recover the full 6D pose. The approach eliminates the need for intermediate segmentation or refinement steps, allowing for efficient inference over entire scenes. YOLO6D provides real-time performance, efficiently handles multiple objects, and requires only 3D bounding box annotations, eliminating the need for textured models. However, it is less robust to occlusion, lacks explicit handling of symmetric objects, and requires individualized training for each object.

SAM-6D is an example of recent models proposed in 2024 by Lin et al. [36]. It is a two-stage approach capable of handling 6D pose estimation for novel objects in a zero-shot manner from RGB-D data. “Zero-shot” here means the system can estimate the pose of objects it has never seen during training as long as some reference information is available.

This contrasts with methods like PoseCNN or YOLO6D, which require dedicated training for each object instance. The SAM-6D architecture consists of two main components: an Instance Segmentation Model (ISM) and a Pose Estimation Model (PEM). The ISM leverages the Segment Anything Model (SAM) to perform general-purpose object segmentation without task-specific training. This allows the system to detect candidate object regions based on visual priors learned from large-scale datasets. Once the object is segmented, the PEM estimates its pose by aligning the segmented region’s depth data (a partial 3D point cloud) with the full 3D model of the object. This is treated as a dense point cloud registration problem and is solved using a two-stage point-matching network. The method works in a zero-shot setting with previously unseen objects, does not require retraining for each new instance, and is robust to occlusion due to its reliance on geometric alignment. However, it is computationally intensive and not suitable for real-time applications.

FoundationPose, introduced by Wen et al. [57], is a recent example of a modern deep learning model designed as a unified framework for 6D object pose estimation and tracking. It supports both model-based and model-free modes and can operate in a zero-shot setting if a 3D model of the object is available. The system builds a neural representation of the object, either from the 3D model or a small number of reference images, which enables generalization to novel instances without requiring retraining. The architecture consists of two main components: a pose refinement network and a scoring network. At inference time, the model requires a segmentation mask of the object, which can be obtained from an external detector. Based on this region, it samples multiple pose hypotheses and renders corresponding images using the learned neural representation of the object. Each hypothesis is then refined using a dedicated refinement network that compares the rendered view with the actual observation. After several iterations of refinement, the scoring network evaluates the set of refined hypotheses and selects the best pose based on visual similarity. While the process of generating and evaluating multiple pose hypotheses is computationally expensive and not suitable for real-time use, the authors propose a tracking mode to address this limitation. In tracking mode, the model treats the previously estimated pose as the current single hypothesis and applies the refinement step directly, bypassing the need for hypothesis sampling. This greatly reduces computational cost and enables real-time performance in applications such as robotic manipulation.

A more comprehensive review of existing approaches to 6D pose estimation can be found in [62, 24, 46].

3.2 Object Manipulation

The previous section reviewed algorithmic approaches for estimating 6D object pose from sensor data. This section outlines how such estimates are utilized in object manipulation pipelines, highlighting various strategies tailored to different use cases. The choice of approach, which includes sensing modality, control method, and execution strategy, often depends on the task and the assumptions made about the environment. Some methods assume a static scene and operate in an open loop, while others incorporate feedback to handle uncertainty or changing conditions. For example, depth data can improve geometric accuracy when the object is close to the camera, but may become unreliable at greater distances.

3.2.1 Open-Loop Grasping in Static Environments

Robotic tasks, such as grasping stationary objects in structured or semi-structured environments, can often be accomplished reliably using open-loop strategies. In these cases, robot motion is planned once based on an initial observation and executed without any feedback or correction. Although such approaches cannot adapt to dynamic scenes or disturbances, their simplicity and robustness make them attractive for certain scenarios. This section reviews papers that explore this paradigm. Although the primary focus of these works is not on developing agile and robust grasping like in our research, they demonstrate the feasibility of local, static, open-loop systems on real hardware (not just in simulation) and highlight both the strengths and limitations of this approach.

The work by Ma et al. [39] presents a grasping estimation system that leverages point cloud fusion to complete partial object observations and refine grasp poses. The authors introduce a modular perception pipeline that first estimates object pose using a deep neural network, then aligns the observed and model point clouds to refine the grasp using an Angle-View Network. The system is evaluated in both simulation and real-world scenarios under varied background conditions. Despite achieving high grasp success rates, the execution is strictly open-loop: the grasp is computed once and executed without correction, assuming the object remains static between perception and contact.

The paper [65] targets robust grasping in cluttered industrial environments through a geometric-driven 6D pose estimation framework. The system combines RGB-D semantic segmentation with point cloud clustering and PPF-based edge matching to estimate object pose. Grasping is performed using a fixed suction grasp selected based on confidence metrics. Although the method is tested on a physical robot and shown to handle occlusions

and background complexity, it remains a one-shot open-loop pipeline. No visual feedback or trajectory correction is incorporated after execution begins.

The paper [29] is more performance-oriented and introduces YOEO, a lightweight single-stage network for category-level 6D pose estimation that operates at a rate of up to 200 Hz. The model fuses geometric and spatial features in parallel and is validated in real-world manipulation tasks using a Kinova Gen2 arm, including opening drawers and rotating handles on articulated objects. These tasks are executed in an open-loop manner, with trajectories planned based on the predicted pose and then carried out without further adjustment. While the system achieves impressive inference speed, it depends on high-end local compute to sustain that performance. This design choice aligns with the authors’ focus on fast and accurate perception. However, the reliance on powerful onboard hardware reveals a key limitation and highlights the potential value of offloading computation to the cloud, thereby reducing local resource demands while preserving fast response times.

These works do not attempt to solve dynamic or feedback-driven grasping. Instead, they focus on improving accuracy, robustness, or speed within perception modules and use real-world robotic deployments as validation. Their experimental setups consistently assume a static environment during execution, which is rarely guaranteed in real industrial deployments. Even with accurate sensing, the inability to respond to minor disturbances or changes in object pose limits their applicability. Nevertheless, these studies demonstrate that open-loop strategies can be successful in controlled scenarios and offer valuable insights into perception components. They also suggest that many tasks benefit from closed-loop control even when the scene itself remains static, where feedback can improve reliability, precision, and recovery from minor deviations.

3.2.2 Closed-Loop Grasping in Static Environments

The following works explore closed-loop strategies applied in static or quasi-static scenes, highlighting the benefits of integrating feedback, even in relatively simple tasks.

The paper [26] introduces a hybrid visual servoing strategy designed to improve grasp reliability when depth sensing becomes unreliable near the point of contact. The system initially uses RGB-D data to estimate the 6D pose of the object via SIFT features and then transitions to image-based visual servoing (IBVS), where goal features are predicted from the robot state and the grasp synthesizer’s target pose. This design allows the controller to retain the advantages of RGB-D-based grasp planning while maintaining feedback in the final approach. The method is evaluated on a 7-DOF Franka Emika Panda arm in both static and moving object scenarios. The main contribution lies in enabling robust

grasping in the absence of depth data. However, the system assumes slow object motion and does not model velocity or acceleration, which limits its performance in more dynamic environments. Additionally, its reliance on hand-engineered visual features may constrain generalization across diverse object types or scenes.

The work [60] presents a visual servoing-based pick-and-place pipeline for handling knives in a simulated slaughterhouse environment using a UR10 robot arm. The system combines basic PBVS with color-based segmentation, achieving a 90% success rate across 20 trials. Its strength lies in its simplicity: real-time control is achieved without heavy computation or complex models. However, the approach assumes a fully static scene and also depends on visual modifications, such as colored handles, to enable object detection. While effective in its target domain, the system highlights trade-offs between simplicity and adaptability.

The final work in this group [42] demonstrates the precision achievable with visual servoing in structured, static environments. The authors develop a system for accurate peg-in-hole insertion using a Franka Emika Panda robot, achieving alignment errors of under 25 mm. Their approach combines a deep learning-based visual tracker (SE(3)-TrackNet) with a force-sensitive compliant controller. The system generalizes across object geometries using only CAD models, making it suitable for flexible industrial assembly. However, to achieve the necessary tracking speed, the system relies on a high-end RTX 2080 Ti GPU. Even for static tasks, timing remains critical when high precision is required, and this reliance on substantial onboard computing limits where and how such systems can be practically deployed.

The reviewed works demonstrate that visual servoing in static or quasi-static environments can significantly improve grasp precision, robustness, and tolerance to sensing limitations. These systems successfully integrate feedback into the execution loop, enabling the correction of minor errors and supporting more flexible grasping strategies. However, their applicability remains constrained by the assumption of a stationary scene. The following section examines systems that extend these capabilities to dynamic settings, where targets move and rapid, closed-loop adaptation becomes essential for success.

3.2.3 Closed-Loop Grasping in Dynamic Environments

The following papers present attempts to perform robotic manipulation in dynamic scenes, where the target object moves or changes state over time. Each system is a closed-loop system that incorporates feedback mechanisms to correct execution in real-time. These works demonstrate that visual feedback, when combined with filtering, prediction, or planning

modules, can significantly improve performance in dynamic environments. All reviewed systems rely on local computation and vary in their perception strategies and control complexity.

Paper [2] focuses on developing a fault-tolerant system for capturing moving objects under degraded visual conditions. The authors propose a control strategy that combines point cloud-based pose estimation via ICP, a noise-adaptive Kalman filter for object state estimation, and an adaptive replanning module triggered by vision failures. Object detection relies on classical visual features, and tracking is performed using depth data, which is periodically occluded or unavailable. The system is validated on a 7-DOF Franka Emika Panda robot, which performs real-world grasping of slowly moving objects in a practical setting. Although the task is relatively constrained, and the handcrafted perception pipeline may limit generalization to more complex scenes, the results demonstrate that dynamic manipulation is achievable without requiring high-end computation. Importantly, the system’s robustness to sensor degradation makes it well-suited for wireless settings, where temporary feedback loss due to latency or packet drops must be tolerated.

Paper [22] presents a practical implementation of a mobile manipulation system designed to perform industrial fastening tasks on moving conveyor belt parts. The setup utilizes a UR5 robot mounted on an omnidirectional mobile base, employing position-based visual servoing (PBVS) to guide the end-effector relative to fiducial markers (ARUCO tags) placed on target objects. The entire system is structured around a state machine that manages task phases, including detection, motion tracking, and screw insertion. The key strength of the system lies in its integration and deployment for real-world tasks, showing that mobile manipulation with basic feedback and lightweight perception strategies can still be viable in low-dynamic environments. However, the use of ARUCO tags significantly simplifies the perception pipeline, offering high accuracy and speed that may not be achievable with more realistic, markerless perception systems. As a result, the overall reliability of the system could degrade in scenarios where perception is less structured and more prone to noise, occlusion, or delays.

The work [43] introduces GG-CNN, a lightweight convolutional network that predicts grasp quality, orientation, and gripper width directly from depth images. Unlike sampling-based methods, the network produces dense predictions over the entire image in a single forward pass, enabling low-latency execution suitable for closed-loop control. The approach is implemented on a Kinova Mico 6-DOF robot using position-based visual servoing (PBVS). Experiments are conducted in both static and semi-dynamic settings, where objects may shift slightly during execution but do not exhibit rapid or unpredictable motion. The main contribution is the design of a compact network that supports real-time grasping with minimal computational overhead. Although the model is lightweight and does not require a

high-end GPU to run, real-time performance in the original work was demonstrated using a GTX 1070, which is more powerful than typical onboard hardware. Furthermore, the control loop relies solely on the current pose estimate without modeling object velocity or acceleration, which can degrade performance in more dynamic scenarios.

Paper [44] presents a modular framework for closed-loop grasping of dynamic objects, integrating language-driven object detection, real-time 6D pose tracking, and online trajectory optimization. A vision-language model (OWLv2) segments the object based on a prompt, after which FoundationPose estimates and tracks the 6D pose at a rate of 30 Hz. A model-predictive controller replans the motion at 10 Hz based on the updated pose. The system runs on a 7-DOF KUKA robotic arm and is evaluated on dynamic pick-and-place tasks. As far as can be inferred from the paper and accompanying video, the robot tracks the object and moves to a pre-grasp pose, then initiates grasping. If the object shifts during this phase, the system recomputes the pre-grasp pose and restarts the grasping procedure rather than correcting the motion mid-execution. This limits its applicability to more dynamic environments, such as grasping a moving object. Additionally, the system relies on a high-end RTX 3080 GPU to achieve real-time performance, which may limit its practicality in resource-constrained environments.

The work [20] introduces a real-time control system for tracking fast-moving objects using 6D pose feedback from an eye-in-hand camera. It combines a learning-based object detector that identifies objects in individual frames without requiring prior observations with a fast model-based tracker connected through a time-delay correction module, which replays buffered frames to align delayed outputs. This hybrid perception pipeline operates asynchronously, providing continuous pose updates at a rate of up to 120 Hz. These estimates are fed into a model predictive controller that operates at the torque level with a 1 kHz control loop, allowing the 7-DOF Franka Emika Panda robot to maintain a fixed relative pose to the object. The work presents a tightly integrated system that enables stable, high-frequency visual tracking without the need for markers. Among the systems in this category, this work is most closely aligned with the present study in its focus on real-time object tracking with visual feedback and deep learning-based perception. However, it focuses solely on tracking and does not perform grasping. It also relies on powerful local hardware, including an RTX 3080 GPU. While it demonstrates strong performance in high-speed visual servoing, it does not address grasp execution or offloaded perception, which are key aspects of the presented system.

In summary, this line of work demonstrates the potential of closed-loop systems that can incorporate real-time visual feedback and adjust motion in response to rapid, unpredictable changes in the environment. However, they assume reliable, low-latency sensor input and require powerful local computing to sustain high-frequency, precise control. This

makes it expensive to deploy at scale, as each robot would need dedicated high-end hardware. Offloading heavy perception or planning to the cloud appears more scalable, as it reduces the need for onboard GPUs. Whether such an approach can achieve comparable responsiveness and reliability remains an open question. The following section reviews recent efforts that relax these constraints by offloading computationally intensive components such as pose estimation or trajectory planning to remote servers and explores how these systems cope with the resulting communication delays and uncertainty.

3.2.4 Remote Processing

When considering remote processing in a real-time closed-loop system, a central question is which system components, such as perception or planning, can be moved to an external computer without compromising control performance. This section reviews works that explore offloading computation to the remote server, highlighting the trade-offs in terms of latency, reliability, resource utilization, and overall system responsiveness.

Paper [10] introduces FogROS2-FT, a fault-tolerant multi-cloud framework designed to support the robust deployment of robotic systems over distributed cloud infrastructures. The framework enables robot workloads to migrate across multiple cloud providers without interrupting execution, addressing potential points of failure in cloud-based robotics. For evaluation, the authors implement a scan-pick-and-place task using a UR10e robot equipped with an Apriltag-based perception pipeline. Both perception and motion planning are offloaded to cloud resources. The task involves alternating the placement of a static object between fixed positions. Although the authors refer to the setup as a feedback loop, the system appears to follow an open-loop execution model, with no evidence of runtime corrections once the motion plan is issued. The core contribution lies in developing a robust framework for deploying cloud-offloaded robotic tasks over wireless infrastructure. However, the experiments are limited to simple, static scenarios and do not address the challenges of real-time feedback or adaptation required in dynamic or uncertain environments.

Paper [49] presents a 5G-enabled robotic part repair system that offloads perception and planning tasks to an edge or cloud server. In this setup, laser scan data of a damaged component is streamed in real time to an edge server, where the resulting point cloud is aligned with a reference CAD model to identify defects. A corrective milling trajectory is then generated on the edge server and transmitted over 5G to a local robotic arm for execution. This system effectively demonstrates the benefits of 5G for low-latency, high-bandwidth data transmission, enabling responsive remote processing in industrial tasks.

However, the robot’s execution strategy is purely open-loop: it follows a precomputed trajectory without any form of feedback or runtime correction. While this validates the feasibility of 5G-enabled cloud planning, the manipulation task is simplified and does not explore how network characteristics such as latency or jitter affect execution quality or task performance.

Paper [21] investigates the impact of wireless network conditions, particularly in 5G environments, on the performance of robotic motion control systems. In their architecture, trajectory generation is offloaded to a remote server, which transmits sparse position set-points to the robot. On the local side, a spline interpolation module reconstructs smooth, high-frequency motion trajectories from the sparse input, enabling continuous control execution despite variations in network timing. The main contribution lies in a detailed empirical analysis of how latency, jitter, and background traffic affect trajectory tracking. Through controlled experiments, the authors evaluate performance under different networking conditions, offering quantitative insights into the relationship between network quality and motion accuracy. The results confirm that, with appropriate interpolation mechanisms, offloading trajectory planning over 5G is feasible and can support high-accuracy motion execution. However, because the study is limited to position tracking without dynamic compensation, the approach may face challenges in high-speed or timing-critical tasks where velocity and acceleration play a significant role.

Paper [63] describes a robotic strawberry harvesting system that uses 5G and WiFi to offload perception tasks to a remote edge server. The robot captures RGB images of the crop and sends them over the wireless network. A remote server processes the images to detect ripe strawberries and computes a corresponding motion trajectory, which is then transmitted back to the robot for execution. The authors conduct field experiments comparing performance over 5G and WiFi. They report improvements in both task efficiency and system cost when using cloud-based perception compared to their prior local computation setup [47]. The results also show that 5G provides a more stable and lower-latency connection than WiFi, which is beneficial for time-sensitive operations. However, the task is executed in an open-loop manner. Once a trajectory is generated, it is carried out without incorporating runtime feedback. This design choice may reduce precision or robustness in real-world deployments, where environmental conditions can change between perception and execution.

The reviewed literature indicates that manipulation in dynamic environments is an active and rapidly evolving area, with steady progress being made across perception, planning, and control. Many recent works improve individual components, whether by making perception faster and more accurate, communication more robust, or planning more adaptive. Some systems demonstrate complete pipelines for dynamic grasping, but these typi-

cally rely on powerful onboard hardware and operate in controlled conditions. In general, existing approaches tend to follow one of two patterns: either they simplify the task setup to reduce computational demands, or they tackle more complex scenarios but depend on high-end local compute to do so. What remains largely unaddressed is the integration of visual feedback and dynamic manipulation with remote perception over a wireless network.

To the best of our knowledge, no prior system performs full 6D pose-aware grasping in dynamic scenes with visual feedback while offloading perception to a remote server. This gap motivates the direction taken in this thesis: to explore a complete system that operates under real-time constraints, incorporates closed-loop feedback, and runs perception remotely over wireless communication.

Chapter 4

Experimental Setup

This chapter describes the experimental setup used to evaluate the system’s real-time grasping performance on moving objects under different wireless network configurations. The setup consists of a 7-DOF robotic arm, a remote server equipped with a high-performance GPU for perception, a wireless communication link between the robot and server, and a conveyor belt that delivers the object of interest. The complete system is shown in Fig. 4.1. The goal is to assess the impact of wireless communication on closed-loop control in dynamic manipulation tasks.

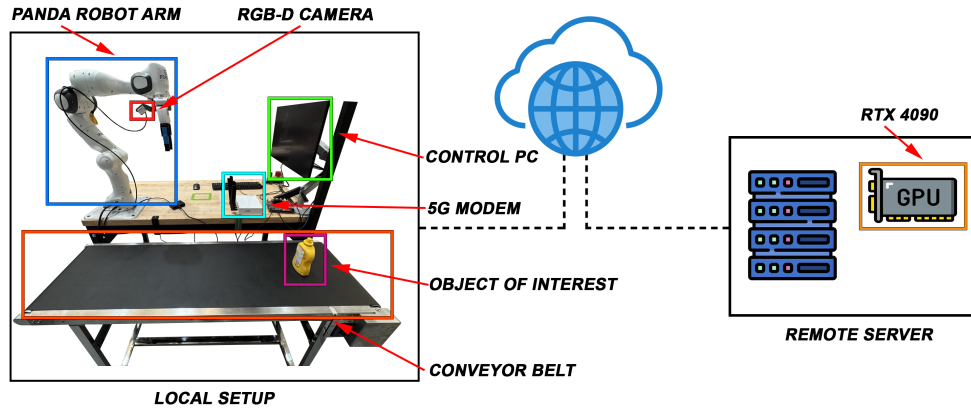


Figure 4.1: System components: robot arm, RGB-D camera, control PC, 5G modem, conveyor belt, object of interest, and remote inference server

4.1 Robot Platform

The Franka Emika Panda robotic arm is mounted on a table and controlled by a local Linux-based PC via a wired Ethernet connection. An Intel RealSense D405 RGB-D camera is mounted on the robot flange using a custom 3D-printed bracket (Fig. 4.2) and connected to the control PC via USB 3.0. The camera streams both color and depth data at a resolution of 640×480 and a frame rate of 30 Hz. A set of 3D-printed gripper fingers (Fig. 4.2), as used in [11], is employed in place of the default fingertips. These custom fingers are longer, wider, and softer, making them more suitable for grasping the target objects in this setup.

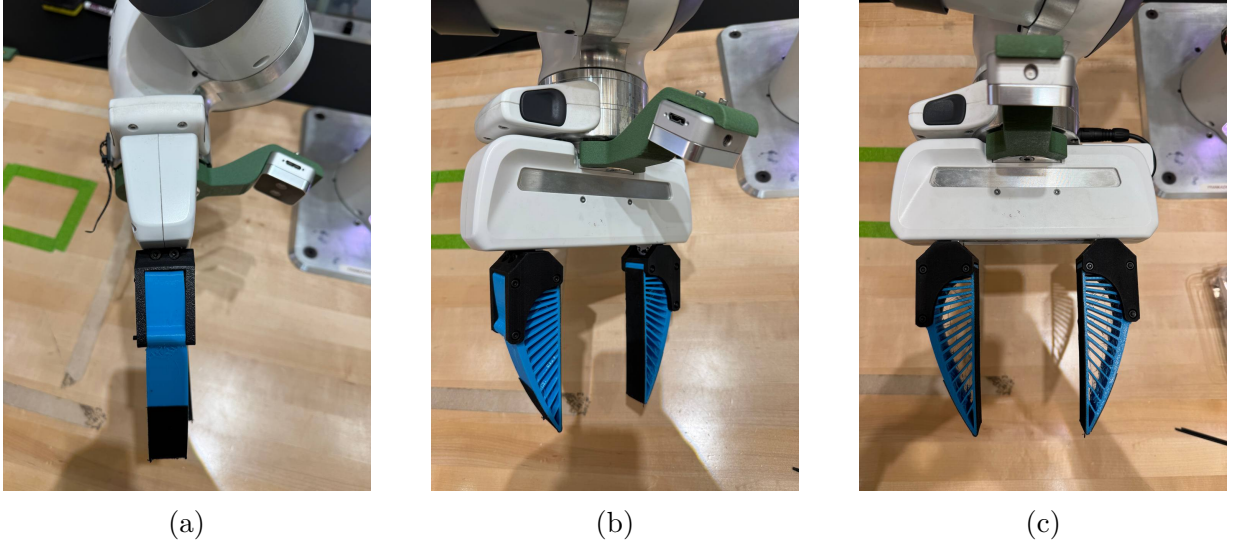


Figure 4.2: End-effector with mounted Intel RealSense D405 RGB-D camera and custom 3D-printed fingers, shown from multiple angles.

4.2 Conveyor and Object

A VEVOR 59-inch motorized conveyor belt with adjustable speed is placed adjacent to the robot table to simulate a realistic industrial setting in the lab. The mustard bottle from the YCB-Video dataset [61] is used as the target object, due to its frequent use in 6D pose estimation benchmarks and ease of acquisition.

4.3 Remote Perception Server

The perception pipeline runs on a remote Ubuntu-based PC equipped with an AMD Ryzen Threadripper Pro 7995WX CPU, 200 GB of RAM, and an NVIDIA RTX 4090 GPU. The server handles all data processing, including decoding the received RGB-D images from the robot and running inference with the NVIDIA FoundationPose model to estimate the object’s 6D pose. The predicted pose is then streamed back to the robot.

4.4 Communication Setup

Communication between the robot and the server is implemented using a PUSH/PULL socket pair over TCP with ZeroMQ. All data is transmitted over various wireless networks. A WireGuard-based VPN tunnel was also implemented to bypass firewall restrictions and ensure that the same codebase runs identically across different network configurations. The same communication protocol and software stack are used in all experiments.

4.5 Wireless Network Configurations

The system is evaluated under four distinct wireless network configurations:

- 60 GHz Wi-Fi (802.11ad):
 - Frequency: 58320 MHz
 - Channel width: 2160 MHz
 - Hardware: Mikrotik RBwAPG-60ad-SA (used on both ends)
- 5 GHz Wi-Fi (802.11ac):
 - Frequency: 5240 MHz
 - Channel width: 80 MHz
 - Hardware: Mikrotik wAP G-5HacT2HnD (access point) and Intel CNVi Wi-Fi adapter
- 5G NSA mmWave (Band n261):
 - Frequency: 24 GHz
 - Channel width: 100Mhz
 - Hardware: Quectel RM510Q-GL
 - Carrier: Rogers
- 5G NSA (Band n78):
 - Frequency: 3.5Ghz
 - Hardware: Quectel RM520N-GL
 - Carrier: Rogers

Each configuration was tested in isolation, using identical routing, perception, and control software.

Chapter 5

Methodology

This section provides an in-depth overview of how the system operates, covering its main components, processes, and the way they interact to accomplish the overall task. The aim is to give a clear understanding of the system’s design, underline its limitations, and provide the necessary context to better interpret the results presented later.

5.1 System Overview

The system consists of two main logical components: the control logic, which governs the robot’s behavior, and the pose estimation pipeline, which provides 6D object pose estimates from visual input. The overall system pipeline is illustrated in [Figure 5.1](#).

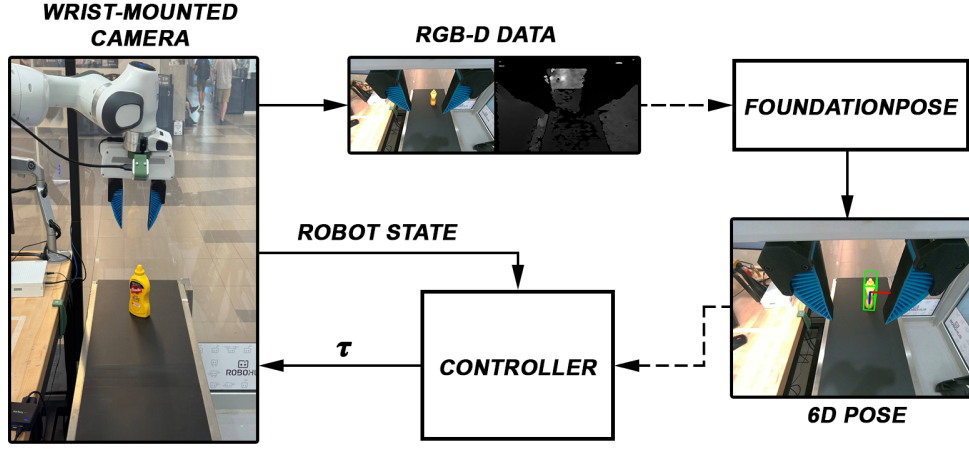


Figure 5.1: System pipeline illustrating

The control logic consists of a state machine, defined in Section 5.3, running in the main thread of the robot’s C++ program. It reads the latest 6D pose of the object of interest from shared memory and plans and executes motion to perform grasping. This process is non-blocking and operates independently of the perception pipeline, enabling continuous control even when visual feedback is delayed.

The pose estimation pipeline spans both the robot and a remote GPU server. On the robot side, this involves two additional threads within the program that runs the state machine: one continuously captures RGB-D frames from the wrist-mounted Intel RealSense D405 camera and sends them to the server, while the other receives the estimated 6D pose and stores it in shared memory. On the server side, a high-performance machine equipped with an RTX 4090 processes the RGB-D input using the FoundationPose model [57] and sends the estimated pose back to the robot. This division allows the robot to offload computationally expensive inference while maintaining local responsiveness.

The overall architecture follows a dual-loop design, separating high-frequency control from slower visual processing. This is essential because the control loop operates at 1000 Hz, while the camera provides data at 30 Hz. The decoupled design also absorbs delays introduced by network transmission and remote inference, allowing the control logic to remain reactive and consistent.

This modular architecture enables robust real-time performance, tolerates network variability, and supports system debugging and future extensions. The remainder of this chapter is divided into two parts: one describes the pose estimation flow, and the other outlines

the state machine that coordinates the task execution.

5.2 Pose Estimation Flow

The pose estimation pipeline consists of three main components: two independent threads running on the robot side and a server-side pose estimation component. The first thread captures RGB-D data from the camera and sends it to the server, while the second thread receives 6D pose estimates and stores them in shared memory. Both threads operate independently of the main state machine. The server module receives the RGB-D input, performs 6D pose estimation of the target object using the FoundationPose model, and returns the estimated position and orientation to the robot. The following sections provide a detailed description of each component and offer additional insight into the FoundationPose model.

5.2.1 FoundationPose Deep-Learning Model

For 6D pose estimation of the object of interest, the NVIDIA FoundationPose deep learning model is used. It takes both color and depth data as input. It returns the object’s pose as a vector consisting of six elements: three for translation and three for rotation in angle-axis form. The model supports two modes of operation: registration and tracking. Registration is a single-frame estimation method that requires an RGB image, a depth map, and a segmentation mask of the object of interest. Tracking, in contrast, updates the object’s pose using the current RGB-D input along with the pose estimate from the previous time step.

The FoundationPose framework introduces a unified architecture that supports both model-based (when a CAD model is available) and model-free (when a small set of reference images is provided) setups. This work focuses on the model-based configuration, which enables the system to render realistic views of the object under different poses using the provided 3D model. The framework was trained entirely on synthetic data, using a large-scale pipeline that leverages 3D object repositories, large language models (for generating texture prompts), and diffusion-based texture synthesis. The training data was rendered in NVIDIA Isaac Sim using randomized lighting, materials, object configurations, and camera placements to ensure broad generalization. Notably, the model does not require fine-tuning on target objects or datasets. The registration pipeline begins by generating a large set of coarse pose hypotheses, formed by sampling camera viewpoints from an icosphere around

the object and applying discrete in-plane rotations. For each hypothesis, the model renders a view of the object and compares it against the input image. These candidate poses are then refined using a dedicated pose refinement network, which takes as input the rendered image of the object and a pose-conditioned crop from the real observation. Unlike fixed bounding boxes, this adaptive cropping is based on the estimated pose and better captures spatial context. The network predicts minor corrections to translation and rotation, which are applied iteratively to improve alignment. Once refined, the pose candidates are passed to a hierarchical ranking module that selects the best estimate. The system embeds each hypothesis by comparing rendered and observed crops, then processes the complete set of candidates jointly using a transformer-based attention module. This enables relative scoring across all poses rather than relying on absolute alignment scores. Training is performed using a contrastive triplet loss that encourages correctly aligned poses to rank higher in the results.

While registration provides accurate single-frame pose estimation, it is computationally slow and not suitable for real-time operation. It typically takes around 0.3 to 1.3 seconds per object, depending on the number of hypotheses being reviewed. To address this, FoundationPose includes a tracking mode that avoids resampling multiple pose hypotheses. Instead, the pose from the previous frame is the only hypothesis and is directly refined using the current RGB-D input. This method significantly reduces computation time, enabling tracking at approximately 32 Hz. However, effective tracking requires temporal consistency, meaning the object should transition smoothly from one frame to the next. If there is a sudden shift in pose due to occlusion, visual ambiguity, or delay, the tracker may return incorrect results.

The model has a few practical limitations. First, registration requires a segmentation mask, so a separate object detection pipeline must be integrated to provide it. Second, registration is too slow for real-time use, especially in high-speed or time-critical tasks. While tracking is fast enough for closed-loop control, it is inherently less robust in situations where the pose changes abruptly. Third, the model does not provide a built-in confidence score. Even if tracking fails, it continues to output a pose estimate as if it were correct, which makes failure detection difficult without external validation. Another key limitation arises from the typical usage of the pose estimation model, where registration is performed on the first available frame, and tracking is used afterward due to its faster runtime. However, registration takes time to complete, and during this period, the object continues moving along the conveyor. Incoming frames are skipped while registration is still running, so by the time tracking begins, the object’s position in the new frame may differ significantly from its position in the frame used for registration. This mismatch can violate the temporal consistency expected by the tracking module, potentially leading to tracking failure. The

solution to this issue is described in detail in Section 5.3.2.

The setup uses a color-based segmentation mask to detect when an object enters the scene. This mask ensures that registration is triggered only when the object is within the camera’s field of view (i.e., visible in the captured images), and it also serves as the segmentation input required by the registration module. Registration is performed once, at the moment of first detection, then the system switches into tracking mode. To avoid repeated registration, which is time-consuming, the controller is designed to move the robot in a way that keeps the object within the camera view once it has been registered. This allows tracking to proceed continuously using each new frame without requiring re-running registration.

5.2.2 Robot-Side Sender

The robot-side sender is a separate thread within the C++ control program that runs independently of the main controller (state machine) and is triggered by the camera driver’s callback when a new frame becomes available from the Intel RealSense D405. This thread also accesses the state machine to retrieve the robot’s end-effector transformation. It is essential to obtain the current end-effector transformation, as the deep learning model outputs the object’s pose expressed in the camera frame, whereas focus is directed toward the pose expressed in the base frame. To transform the object’s pose expressed in the camera frame to the base frame, the transformation from the base to the camera is needed. The transformation base-to-camera is decomposed into base-to-end-effector and end-effector-to-camera as follows:

$$T_o^b(q) = T_c^b(q)T_o^c = T_{ee}^b(q)T_c^{ee}T_o^c$$

where $T_o^b(q)$ is pose of the target object in the base frame; $T_c^b(q)$ - the camera frame in the base frame; T_o^c - the object pose in the camera frame; $T_{ee}^b(q)$ - the end-effector frame in the base frame; T_c^{ee} - the camera frame in the end-effector frame. T_c^{ee} is fixed and is derived based on the type of camera mount used, while $T_{ee}^b(q)$ depends on the robot’s joint configuration and changes over time as the robot moves. Thus, the robot’s pose at capture time is needed to compute the object’s pose in the base frame at the correct moment. For this reason, this information is attached alongside the color and depth data. To reduce bandwidth usage, the data is then compressed: RGB images are encoded as JPGs, and depth frames as PNGs. All components are serialized to binary, packed, and sent to the server. The multi-part messaging functionality provided by ZeroMQ is utilized, allowing

separate messages (e.g., image, depth, metadata) to be sent as a single logical unit. This approach eliminates the need to serialize everything into one buffer. Data is transmitted using a PUSH/PULL socket type, where the server pulls all updates pushed by the robot.

5.2.3 Server-Side Pose Estimation

The server-side process is implemented in Python. Initially, the server loads and sets up the FoundationPose model, making it ready for inference. Then, it binds two ports: one for incoming traffic from the robot (pulls from the robot) and another for outgoing traffic (pushes to the robot). Before starting the main loop, the server is set to registration mode. After that, the server starts listening to the incoming traffic.

When a new message arrives, all its parts are retrieved, including color data, depth data, timestamp, and end-effector transformation, and stored in local variables. The color and depth data are deserialized, decoded, and converted to cv2.Mat objects. The base-to-end-effector transformation and timestamp are not deserialized because they are not used by the server and are simply passed back to the robot alongside the predicted pose.

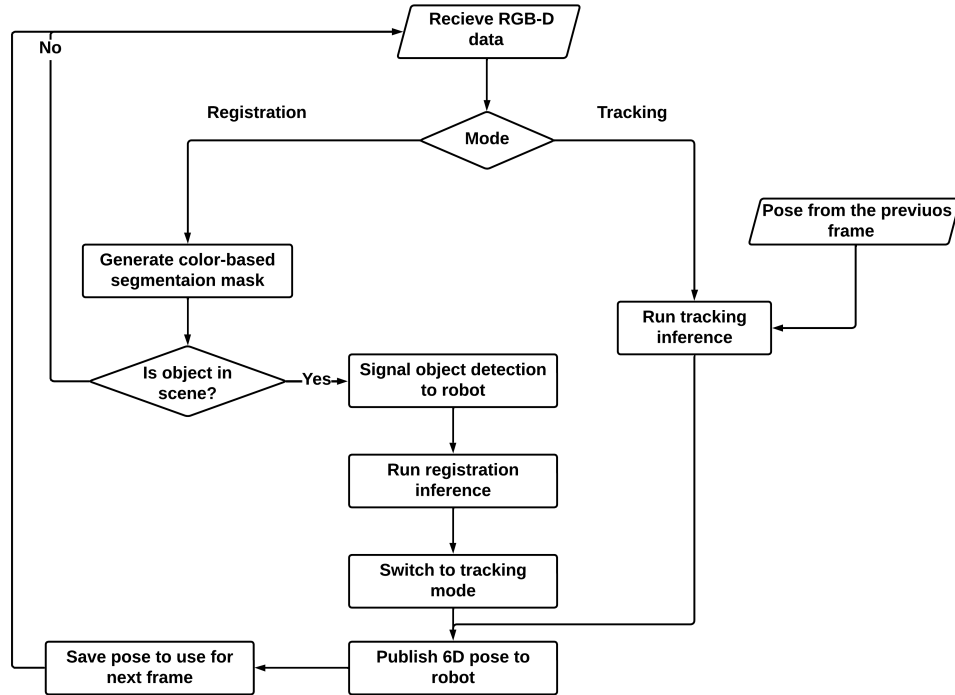


Figure 5.2: Flow diagram of the server-side processing pipeline. The server receives an RGB-D data and selects between registration and tracking modes. In registration mode, a color-based segmentation mask is generated to verify the presence of the object. If detected, registration inference is run, and the system switches to tracking mode. In tracking mode, pose updates are computed directly from the RGB-D input. In both cases, the estimated 6D pose is published to the robot.

Before running inference on this data, a segmentation mask for the target object is created. This serves two purposes: first, the FoundationPose model requires a mask when running registration; second, the model does not provide confidence metrics to assess whether the object is actually in the scene, so a separate heuristic, described below, is used to determine object presence before running registration.

To generate the mask, color masking is applied using predefined thresholds to isolate the object’s predominant color (yellow, in this case) from the black background of the conveyor belt. Contour detection (`cv2.findContours`) is then applied to the color mask, and bounding boxes are constructed around each contour. The largest contour is checked against a size threshold, determined experimentally to avoid triggering on image artifacts

and to respond only when a sufficiently large object enters the scene. This approach has proven reliable for the current use case. However, more general-purpose solutions, such as YOLO-based object detectors [55] or other deep learning models, could also be employed. If the largest contour does not exceed the threshold, the message is discarded and the system waits for the next one. If it does, it indicates that the desired object is present in the scene, and the registration process can begin. Before proceeding, a notification is sent to the robot indicating that the object has been detected and that the registration method is now executing. This step is important because, while registration is running, the object continues to move, potentially causing a discontinuity between the registration frame and subsequent tracking frames, which can lead to failure when starting tracking. Once registration is completed, the FoundationPose model returns the estimated object pose. The server then forms a multipart message consisting of the six pose values, serialized into a raw byte array, along with the timestamp and end-effector transformation captured earlier. This message is sent back to the robot using the same PUSH/PULL ZeroMQ architecture. Finally, the server switches its internal mode to tracking and flushes the buffer, discarding all messages that arrived while the server was busy processing the registration. This ensures that the next iteration operates only on the most recent data.

Once registration is completed, the server enters tracking mode. Tracking uses the previously estimated pose to refine the current estimate, reducing runtime compared to registration. Tracking works similarly to registration in terms of communication: the server waits for a new message, receives it, and decodes its parts exactly as before. However, the processing step differs slightly from registration mode because the tracking method does not require a mask and assumes that it remains visible in the scene once the object has been detected and registered. Hence, verifying the object’s presence is no longer necessary. This assumption is valid because the controller is designed to maintain continuous focus on the object. Subsequently, the server executes the tracking method of the FoundationPose model by passing in the current RGB-D data and the pose result from the previous frame. Once tracking completes, the pose is returned to the robot in the same way as in registration: the pose is serialized, combined with the original timestamp and end-effector transformation, and sent back as a multipart message using the same format and transmission mechanism as in the registration phase.

5.2.4 Robot-Side Receiver

The robot-side receiver is also a separate thread, implemented in C++, that is independent of both the main (state machine) thread and the sender thread. It starts at the same time as the sender thread, connects to the server, and waits for a message to arrive. Upon

receiving a message in ZMQ multi-part format, all of its parts are deserialized and stored in local variables. The message contains the object pose expressed in the camera frame, the corresponding base-to-camera transformation at the time of capture, and a timestamp indicating when the data was acquired, as defined in Section 5.2.2. This transformation is used to convert the object pose from the camera frame to the base frame. The result is stored in shared memory, along with the corresponding time and an `is_new` flag, which indicates whether the current data has already been processed by the state machine. After the state machine processes the current data, this flag is set to false. Since the shared variable is accessed by multiple threads, a mutex prevents data corruption and race conditions.

5.3 State Machine

The robot is controlled by a state machine, a typical structure used when a system operates in multiple states or phases. The terms “state” and “phase” are used interchangeably, as they refer to the same concept in this context. Each state or phase has different logic, timing, and required data. In this framework, the robot performs actions based on its current state and transitions to the next state when specific conditions are met, typically when a portion of the task is completed. For example, the robot may need to wait for an object to appear, move toward it, and attempt to grasp it. Each of these actions is handled in a separate state with clearly defined entry and exit conditions. The state-based design provides a clean and deterministic approach to handling complex scenarios, such as reacting to asynchronous sensory data, executing multi-step tasks, and recovering from errors. It ensures that each process part is isolated and well-defined, simplifying development and debugging. Without a formal state machine, the task logic would have to be embedded in a monolithic loop with numerous nested conditionals, making the code more difficult to maintain, test, or extend.

Figure 5.4 presents the high-level state machine diagram, showing the system states and the transition conditions between them, while Figure 5.3 illustrates how this logic maps onto the robot’s motion during a grasping task. The numbered icons represent the end-effector’s position and the corresponding object location at discrete time steps throughout the sequence. The pipeline begins with the Idle state (step 1), where the robot remains stationary while waiting for an object to appear in the camera’s field of view. Once the object is detected, the system enters the Registration state (steps 2–3), during which the robot moves alongside the conveyor belt while the remote server processes the object registration. After the pose becomes available, the robot transitions to Compute Approach Point (steps 3–4). In this phase, the robot remains still while a feasible approach trajectory

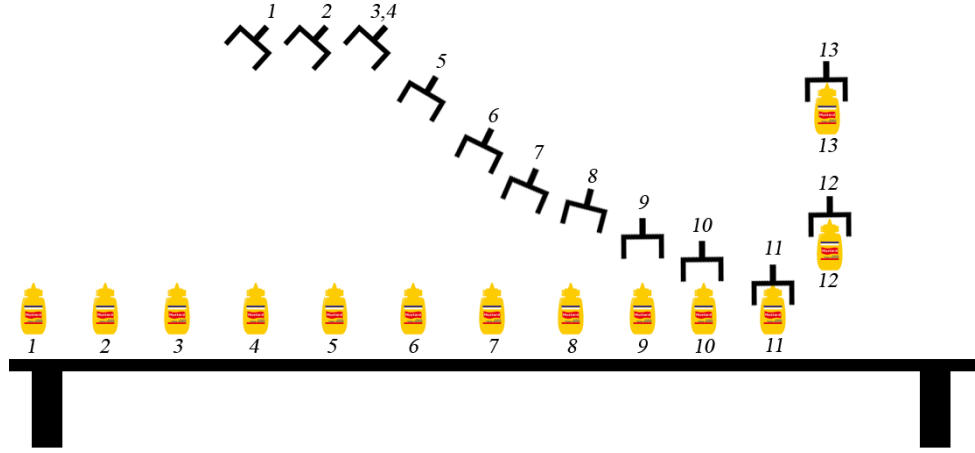


Figure 5.3: Illustration of the robot's motion during a grasping task.

is computed. It then enters the Approach state (steps 5–9), during which it executes the planned motion to move closer to the object. Once the approach trajectory is complete, the system transitions to the Visual Servoing state (steps 9–11), where the robot uses visual feedback to track the object in real time and align the end-effector for grasping. Finally, in the Lift state (steps 12–13), the robot lifts the object and begins to move away from the grasp location.

The following subsections provide detailed discussions of each state in the state machine.

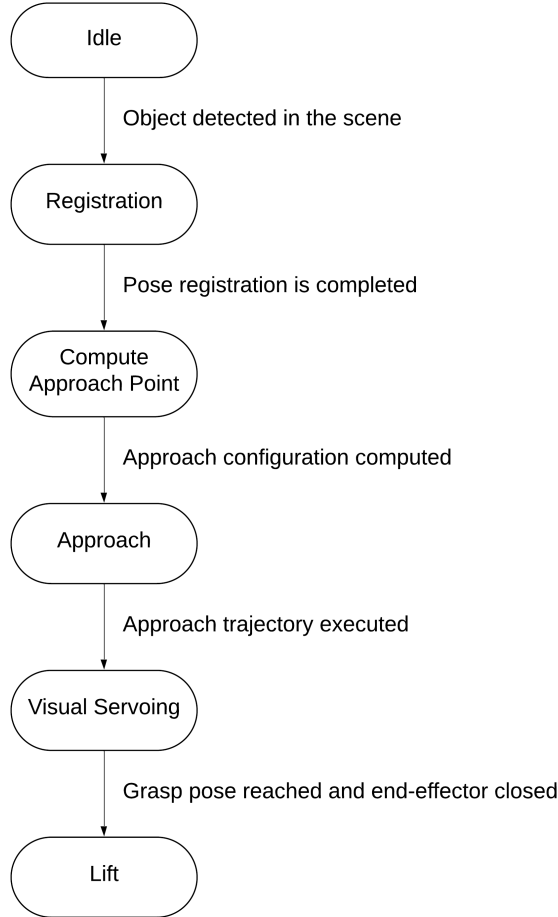


Figure 5.4: Diagram of the state machine that governs the grasping task. Arrows indicate transitions, labeled with the conditions that trigger them.

5.3.1 Idle State

The idle state is the initial state of the system. The robot starts in a predefined configuration, which provides a wide and clear view of the workspace, specifically, the start of the conveyor belt where the object is expected to appear. The goal of this state is to wait until the object enters the scene to start the later pipeline. To achieve this, the system continuously monitors the shared variable where pose data from the server is stored. As described in Section 5.2.4, this shared structure includes an `is_new` flag that indicates when

new information is available. Upon receiving a command from the server, as described in Section 5.2.3, indicating that the object has been detected and registration has started, the system transitions from the idle state to the registration state.

5.3.2 Registration State

The registration state begins immediately after the server sends the command indicating that the object has been detected in the scene and handles the robot’s motion while registration is being computed. The goal of this phase is to compensate for the offset that results from the fact that, while registration is running on a specific frame, the object is in motion. More specifically, when the server detects an object in a frame and initiates registration (5.5a), new frames continue to arrive. However, these are discarded during registration to ensure that the system operates on the most recent data. As a result, by the time registration is complete, which takes approximately 0.5 seconds, the next frame used for tracking (5.5c) is not the immediate successor (5.5b) of the frame used for registration (5.5a), but a significantly later one. For example, with a 30 FPS camera, registration may begin on a frame captured at time t_0 (5.5a), and approximately 15 frames are skipped during the 0.5-second registration period. As a result, tracking begins on a frame captured at $t_0 + 15 \cdot 33$ ms, as shown in 5.5c. This frame may differ significantly from the one (5.5b) captured at $t_0 + 33$ ms, breaking the temporal consistency that the model relies on between consecutive frames. As a result, running tracking on that frame might fail.

To overcome this, the end-effector is moved in the direction of the conveyor belt at the same speed to align the viewpoint, ensuring that the next received frame shows the object in nearly the same position relative to the frame. This compensation is computed using a constant-velocity model. When the registration state begins, the initial end-effector position $p_{ee,init}$ and orientation $R_{ee,init}$, both expressed in the base frame, are stored. Let t denote the discrete time variable, incremented at each control cycle. Let t_{init} represent the timestamp marking the beginning of the registration phase, and let v_{cb} denote the speed of the conveyor belt. The controller input consists of the desired pose and velocity at each step. A task-space PD controller is used, with Coriolis and gravity compensation. The control law is defined as:

$$\begin{aligned}
p_{\text{input}} &= p_{ee,\text{init}} + v_{cb}(t - t_{\text{init}}) \\
R_{\text{input}} &= R_{ee,\text{init}} \\
v_{\text{input}} &= v_{cb} \\
\omega_{\text{input}} &= \vec{0}
\end{aligned}$$

The angular velocity input, ω_{input} , is set to zero because the object moves along a straight line and does not change its orientation.

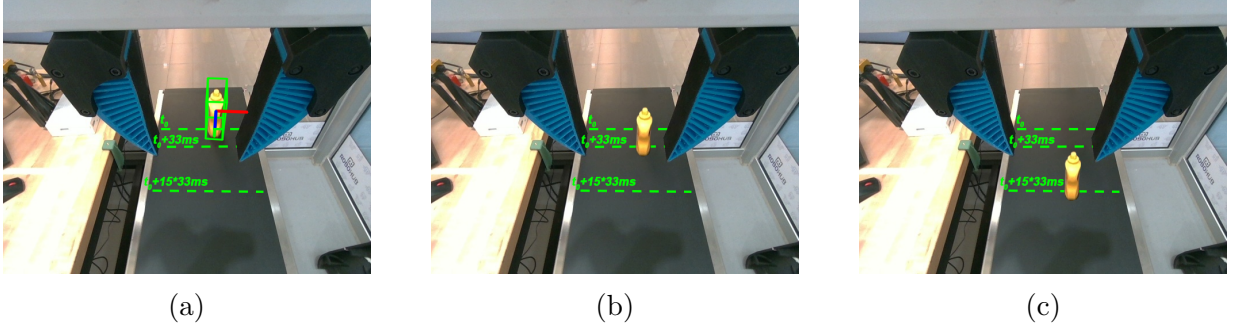


Figure 5.5: (a) Frame used for the registration. (b) Next frame captured by the camera. (c) Next frame processed by the server after registration delay. This figure illustrates the temporal gap caused by registration latency. While the object continues to move, intermediate frames are dropped, resulting in a significant visual difference between consecutive frames used by the system.

Compensation for the movement continues until new data is received from the server, indicating that registration has been completed and the actual pose of the object is now available. At that point, the system transitions to the next state.

5.3.3 Compute Approach Point State

The next state is the Compute Approach Point state, whose goal is to compute a feasible approach configuration and the trajectory required to reach it.

Visual servoing cannot start immediately after registration because it operates in task space and does not account for joint limits. Task-space control is particularly well-suited for local motion tasks, as it enables the end-effector to move directly in a specified direction. This approach avoids the complexity of solving inverse kinematics, which is especially

beneficial for manipulators with a high number of degrees of freedom. In this setup, the end-effector must travel a substantial distance from its initial configuration to the object’s position. While visual servoing can be extended to account for joint limits, it was more straightforward to implement joint-space planning for this stage, which ensured the motion remained within the robot’s reachable configuration without introducing additional drawbacks.

Another challenge in this state is keeping the object within the camera’s field of view. While the robot moves toward the object, it is important for the camera to maintain continuous visual contact; otherwise, tracking may fail. As described in Section 5.2.1, the FoundationPose model does not provide confidence metrics on object presence. If the object leaves the frame, the model continues to output pose estimates, but these are entirely incorrect. Even if the object reappears later, the model will not automatically reacquire it and will continue returning invalid data. These faulty predictions cannot be automatically rejected; therefore, it is critical to keep the object visible at all times. Since tracking cannot self-correct, the only way to recover is to rerun registration. However, registration is time-consuming and causes the system to wait for new results, during which the robot remains idle. This delay can negatively impact task performance, especially when the object continues to move.

To avoid tracking failures, the approach trajectory is planned to maintain the object within the camera’s field of view throughout the motion. The camera’s optical axis is not explicitly aligned with the object during the approach; instead, the trajectory is designed so that the end-effector gradually turns toward the object. This orientation is required by the subsequent grasping logic, as discussed in Section 5.3.5.2, and also helps preserve visibility due to the camera’s wide field of view. Passive alignment in this manner has been found sufficient to ensure the object remains visible during the approach.

The goal of the compute approach trajectory state is to generate a motion plan for a subsequent state that brings the end-effector into position to intercept the object. This state is necessary because computing the trajectory is not instantaneous and may take hundreds of milliseconds. During this time, the robot remains stationary, waiting for the motion plan to be computed.

The interception point is defined as a future position where the robot’s end-effector and the moving object are aligned in space, up to a fixed relative offset. Unless otherwise specified, all quantities in this section are expressed in the robot base frame.

Let $p_o(t) \in \mathbb{R}^3$, $R_o(t) \in SO(3)$ be the object’s position and orientation at time t , and $p_{ee}(t) \in \mathbb{R}^3$, $R_{ee}(t) \in SO(3)$ be the end-effector’s position and orientation at the same time. The object moves at a known constant speed v_o , which equals to conveyor belt speed v_{cb} ,

so its position over time is given by:

$$p_o(t) = p_{o,init} + v_o t$$

where $p_{o,init}$ is the initial object position at the time $t = 0$ for simplicity. It is assumed that the object does not rotate, i.e., $R_o(t) = R_{o,init}$, where $R_{o,init}$ is the initial orientation of the object. The goal is to find a time t such that the robot can reach a position where the offset between the end-effector and the object matches a predefined desired $\delta \in \mathbb{R}^3$, i.e.:

$$p_{ee}(t) - p_o(t) = \delta$$

This means that the robot must reach the position:

$$p_{ee}(t) = p_{o,init} + v_o t + \delta$$

Since the robot starts from a known position $p_{ee,init}$ and cannot move arbitrarily fast, the planned trajectory must ensure that the end-effector can reach $p_{ee}(t)$ within the time t , without exceeding the maximum allowed Cartesian velocity V_{max} . Ideally, the end-effector velocity would be constrained element-wise as $-V_{max} \leq V_{ee}(t) \leq V_{max}$, where $V_{ee}(t)$ is the end-effector velocity. However, in practice, it was sufficient to constrain only the magnitude of the average speed, as long as a conservative upper bound for V_{max} is used, which is lower than the theoretical limit specified by the robot manufacturer. Let $\hat{V}_{ee}(t)$ denote the average end-effector speed over the time interval $[0, t]$, and let V_{max} now refer to a conservative upper bound on velocity, reduced from the theoretical limit. This leads to the inequality:

$$\|\hat{V}_{ee}(t)\| \leq \|V_{max}\|$$

$$\|\hat{V}_{ee}(t)\| = \frac{\|p_{ee}(t) - p_{ee,init}\|}{t}$$

$$\frac{\|p_{ee}(t) - p_{ee,init}\|}{t} \leq \|V_{max}\|$$

$$\frac{\|p_{o,init} + v_o t + \delta - p_{ee,init}\|}{t} \leq \|V_{max}\|$$

Let $\lambda = p_{o,init} - p_{ee,init} + \delta$, then the equation simplifies to

$$\frac{\|\lambda + v_o t\|}{t} \leq \|V_{max}\|$$

Squaring both sides, where $\langle \cdot, \cdot \rangle$ denotes the dot product:

$$\langle \lambda + v_o t, \lambda + v_o t \rangle \leq (\|V_{max}\|t)^2$$

Expanding

$$\|\lambda\|^2 + 2\langle \lambda, v_o \rangle t + (\|v_o\|t)^2 \leq (\|V_{max}\|t)^2$$

Rewriting

$$(\|v_o\|^2 - \|V_{max}\|^2)t^2 + 2\langle \lambda, v_o \rangle t + \|\lambda\|^2 \leq 0$$

This results in a quadratic inequality in t , which can be solved analytically. If multiple solutions exist, the smallest positive t is selected, as it corresponds to the earliest feasible interception time that satisfies the robot's motion constraints.

Let $t_{\text{intercept}}$ be the solution to the previous inequality; then the corresponding interception position is given by

$$p_{ee}(t_{\text{intercept}}) = p_{o,init} + v_o t_{\text{intercept}} + \delta$$

This yields the desired position of the end-effector. The desired orientation is defined using an empirically determined grasping transformation R_{grasping} , which produces the desired end-effector orientation by post-multiplying the current object orientation as follows:

$$R_{ee}(t) = R_o(t)R_{\text{grasping}} = R_{o,init}R_{\text{grasping}}$$

Given the initial pose

$$T_{ee,init} = \begin{bmatrix} R_{ee,init} & p_{ee,init} \\ 0 & 1 \end{bmatrix}$$

and the desired pose of the end-effector

$$T_{ee,des} = \begin{bmatrix} R_{ee,des} & p_{ee,des} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{ee}(t_{\text{intercept}}) & p_{ee}(t_{\text{intercept}}) \\ 0 & 1 \end{bmatrix}$$

a nonlinear optimizer (described below) is used to compute $N = 20$ intermediate waypoints, each consisting of a joint configuration and velocity. The objective is to minimize total joint displacement while satisfying joint position and velocity limits, and to enforce a soft constraint that the end-effector should continue pointing at the object. To generate the final trajectory, a 5th-degree spline is fitted through these waypoints. Accelerations at each point are estimated using finite differences.

The trajectory is defined as a sequence of joint configurations $q_i^{opt} \in \mathbb{R}^7$ and velocities $\dot{q}_i^{opt} \in \mathbb{R}^7$ for $i \in \overline{0, N-1}$. All indexed variables in this range form the decision variables

of the optimization problem. The initial and final configurations, q_0^{opt} and q_{N-1}^{opt} , are constrained to match the initial and desired poses. The index i denotes the position of each waypoint in the sequence. Additionally, the total time to execute the trajectory is known, which is denoted as $t_{\text{intercept}}$. Let us define $\Delta t = \frac{t_{\text{intercept}}}{N-1}$ as the time step between each waypoint. The optimizer seeks to determine these variables so that the robot achieves the desired pose while satisfying both physical and task-specific constraints. A symbolic forward kinematics function FK that maps each q_i^{opt} to a position $p_{ee,i}$ and orientation $R_{ee,i}$, representing the corresponding end-effector pose. This is used to define constraints and the objective function.

Hard constraints:

- **Joint position limits:** Each q_i^{opt} must stay within bounds, i.e., $\forall i : q_i^{opt} \in [q_{i,\min}, q_{i,\max}]$. Each joint has its own set of limits.
- **Joint velocity limits:** Each $\dot{q}_i^{opt}$ must stay within bounds, i.e., $\forall i : \dot{q}_i^{opt} \in [\dot{q}_{i,\min}, \dot{q}_{i,\max}]$. Each joint has different corresponding limits. Acceleration limits in this implementation are omitted for simplicity.
- **Discrete motion mode:** This corresponds to forward Euler integration, where joint velocities are assumed constant over each interval:

$$\forall i \in \overline{1, N-1} : q_i^{opt} = q_{i-1}^{opt} + \dot{q}_{i-1}^{opt} \Delta t$$

- **Initial condition:** At the beginning of the trajectory at $i = 0$, we require $FK(q_0^{opt}) = T_{ee,init}$.

Soft constraints:

- **Terminal condition:** The final end-effector position $FK(q_{N-1}^{opt})$ is enforced to be approximately equal to $p_{ee,des}$. A large weight compared to other terms, a weight of 50 is applied to this term because it is a crucial constraint, but small errors are tolerated, i.e., $\|FK(q_{N-1}^{opt}).translation - p_{ee,des}\|^2$.
- **Alignment with object:** Each q_i^{opt} corresponds to a timestamp $t_i = i\Delta t$. At that time, object position is computed as $p_{o,i} = p_{o,init} + v_o t_i$, assuming constant object velocity v_o . Desired pointing direction $z_{des,i}$ is defined as

$$z_{des,i} = \frac{p_{o,i} - FK(q_i^{opt}).translation}{\|p_{o,i} - FK(q_i^{opt}).translation\|}$$

Let $z_{ee,i}$ denote the unit vector representing the Z-axis of the end-effector frame, expressed in the base frame. To encourage the end-effector to point toward the object, the misalignment is penalized by the following term:

$$(1 - \langle z_{ee,i}, z_{des,i} \rangle)^2$$

This penalty is added to the cost function with a weight of 0.5 for $i = 0$ to $N - 2$, since the camera's wide field of view makes strict pointing unnecessary. For $i = N - 1$, a higher weight of 1.5 is applied to make the final pose more precise.

- **Smoothness:** Both joint velocity and acceleration are penalized to ensure smooth motion across the trajectory, using the following term:

$$\|\dot{q}_i^{opt}\|^2 + \|\dot{q}_{i+1}^{opt} - \dot{q}_i^{opt}\|^2$$

A weight of 0.2 is used to ensure this term is present without dominating other more critical objectives.

Combining all the terms above, we define the final objective function as:

$$\begin{aligned} & w_{\text{ter-p}} \|FK(q_{N-1}^{opt}).translation - p_{ee,des}\|^2 + w_{\text{ter-o}} (1 - \langle z_{ee,N-1}, z_{des,N-1} \rangle)^2 \\ & + \sum_{i=0}^{N-2} w_s (\|\dot{q}_i^{opt}\|^2 + \|\dot{q}_{i+1}^{opt} - \dot{q}_i^{opt}\|^2) + w_a (1 - \langle z_{ee,i}, z_{des,i} \rangle)^2 \end{aligned}$$

where $w_{\text{ter-p}} = 50$, $w_{\text{ter-o}} = 1.5$, $w_a = 0.5$, and $w_s = 0.2$.

The solution to this optimization problem provides the waypoints interpolated in joint space using a spline to generate the trajectory. However, due to the end-effector's tool geometry, which is symmetric under a π rotation around the Z-axis, there exist two end-effector poses that are equivalent in task space. Although these poses have the same task outcome, their corresponding joint configurations differ. This setup allows the system to select a configuration that respects its constraints. The state that follows the computation and execution of the approach trajectory is visual servoing, which operates purely in task space and does not explicitly account for joint limits. Although joint-limit avoidance is not enforced during visual servoing, selecting an initial configuration that is farther from joint boundaries helps reduce the risk of exceeding those limits. To support this, a score function $configuration_score(q)$ is defined based on joint limit proximity:

$$configuration_score(q_{N-1}^{opt}) = \min_{j \in \{1, \bar{7}\}} \min(q_{\max}^j - q^j, q^{opt,j} - q_{\min}^{j,opt})$$

The trajectory whose final configuration yields the highest score is selected as the approach trajectory.

The output of the optimization process is a sequence of joint-space waypoints, where each waypoint comprises a joint position and a corresponding velocity vector. These waypoints define the motion plan but are sparse in time, typically around 20 points distributed over a few seconds. To generate a continuous trajectory suitable for real-time execution, it is necessary to interpolate between these waypoints at a high frequency, as the controller operates at 1000 Hz. The system uses a joint-space PD controller with a feedforward term to compute torque commands. This controller requires the desired joint position, velocity, and acceleration at each control cycle. To satisfy this requirement, piecewise quintic polynomials are employed to interpolate between every consecutive pair of waypoints. Quintic interpolation ensures continuity of position, velocity, and acceleration across segment boundaries.

The optimization process is not instantaneous, and during that time, the object continues to move. To ensure the planned trajectory remains valid, the solver runtime $t_{\text{computations}}$ is measured and subtracted from the original interception time to account for this delay as follows:

$$t_{\text{execution}} = t_{\text{intercept}} - t_{\text{computations}}$$

The updated $t_{\text{execution}}$ time is then used to plan the trajectory. Based on the value of $t_{\text{execution}}$, the interval $[0, t_{\text{execution}}]$ is divided into $N - 1$ segments, each of duration $\Delta t = \frac{t_{\text{execution}}}{N-1}$. For each joint $j \in \overline{1, 7}$, a 5th-degree polynomial of the form

$$q_i^j(t) = a_i^j t^5 + b_i^j t^4 + c_i^j t^3 + d_i^j t^2 + e_i^j t + f_i^j$$

is constructed over each interval $[i\Delta t, (i+1)\Delta t]$, where $i \in \overline{0, N-1}$. Each polynomial has six coefficients, determined using boundary conditions that ensure continuity of position, velocity, and acceleration between segments. Joint positions and velocities are taken from the optimizer's solution. Since accelerations are not provided directly, they are estimated using finite differences of the velocity values:

$$\ddot{q}_i^{\text{opt},j} = \frac{\dot{q}_i^{\text{opt},j} - \dot{q}_{i-1}^{\text{opt},j}}{\Delta t}$$

At the initial and final waypoints, velocity and acceleration are set to zero. The full set of conditions per segment is:

- $q_i^j(i\Delta t) = q_i^{\text{opt},j}$

- $q_i^j((i+1)\Delta t) = q_{i+1}^{opt,j}$
- $\dot{q}_i^j(i\Delta t) = \dot{q}_i^{opt,j}$
- $\dot{q}_i^j((i+1)\Delta t) = \dot{q}_{i+1}^{opt,j}$
- $\ddot{q}_i^j(i\Delta t) = \ddot{q}_i^{opt,j}$
- $\ddot{q}_i^j((i+1)\Delta t) = \ddot{q}_{i+1}^{opt,j}$

These coefficients are precomputed and stored for each joint and for each segment, allowing the trajectory to be evaluated at any timestamp during execution to obtain the desired joint position, velocity, and acceleration.

Once computations are complete, the system transitions into the approach state, which runs at the required 1000 Hz real-time frequency and executes the planned trajectory.

5.3.4 Approach State

The goal of the approach state is to execute the trajectory generated in the previous Section 5.3.3. This trajectory consists of a sequence of piecewise polynomial segments with precomputed coefficients for each joint. To execute it, the timestamp of the start of the state t_{start} is stored. At each control cycle, the current timestamp $t_{current}$ is used to compute the active segment index $\lfloor \frac{t_{current} - t_{start}}{\Delta t} \rfloor$. Using this index, the corresponding polynomial coefficients are retrieved, and polynomials along with their derivatives are evaluated at $t_{current} - t_{start}$ to obtain the desired joint position, velocity, and acceleration. These values are passed to a joint-space PD controller with a feedforward term 2.1, which returns torque commands sent to the robot.

The approach state ends when the current timestamp $t_{current}$ exceeds $t_{start} + t_{execution}$, indicating that the planned trajectory has been completed. At this point, the end-effector is close to the object, allowing for a switch to visual servoing.

5.3.5 Visual Servoing State

Visual servoing is a closed-loop control method in which a robot continuously adjusts its motion based on visual feedback. It is particularly effective for tasks involving moving targets or uncertain environments, as it allows the system to respond to real-time changes by correcting its trajectory during execution.

In this system, visual servoing is used during the final stage of the grasping pipeline. After the approach state, the robot reaches a pre-grasp pose: the end-effector is oriented correctly for grasping but remains offset from the object by a small, predefined distance. To complete the task, the robot must move from this pre-grasp configuration to the final grasping pose. However, the object continues to move during this phase, so the system must adapt its motion accordingly to maintain alignment with the target.

To achieve this, visual servoing is employed to guide the end-effector toward the object in real-time. The motion is not predefined; instead, it is continuously adapted based on the object’s estimated pose. A trajectory generator produces a smooth reference motion for the end-effector using recent object pose measurements. This reference includes the desired end-effector pose, velocity, and acceleration, and is updated at each control cycle.

The controller itself is a task-space PD controller with feedforward term and Coriolis and gravity compensation (Equation 2.2). It tracks the generated trajectory at 1000 Hz, enabling the robot to smoothly converge toward the object and reach the final grasp pose despite perception delays or object motion. Once the end-effector reaches the grasp pose, the controller continues tracking the moving object until the end-effector is fully closed. Only then does the system transition to the next state.

The remainder of this section explains in detail how the reference trajectory for the end-effector is constructed based on object pose estimates, and how grasping is performed. It is structured as follows:

- Subsection 5.3.5.1 describes how a high-frequency, smooth, and continuous estimate of the object’s pose and velocity is obtained from raw measurements.
- Subsection 5.3.5.2 defines the desired end-effector motion relative to the object during the final approach phase.
- Subsection 5.3.5.3 describes how this desired motion is used to generate a time-parameterized trajectory consisting of pose, velocity, and acceleration at each control step.
- Subsection 5.3.5.4 describes how the final grasping is performed by tracking the object with the end-effector until the gripper closes.

5.3.5.1 Object State Estimation and Signal Filtering

This section describes how the raw object pose measurements are processed using a Kalman filter, rate limiter, and low-pass filter to obtain a smooth, real-time estimate of the object

state. The filtered estimates are then used to generate reference trajectories for the robot’s motion controller.

The object state refers to its position and linear velocity, as well as its orientation and angular velocity, all of which are expressed in the base frame. For visual servoing, a task-space PD controller is employed, which requires a smooth desired pose, velocity, and, optionally, acceleration. These must be updated at 1000 Hz to ensure stable operation. However, the data obtained from the vision pipeline, based on a deep learning model (FoundationPose) and utilizing input from an RGB-D camera, does not meet these requirements. First, the camera operates at 30 Hz, which is significantly lower than the control loop frequency. This effective rate is further reduced due to processing time and potential packet loss. Additionally, when the robot receives the data, it is already outdated: after the frame is captured, it takes time to transmit it to the server, perform inference, and return the result. Furthermore, the data is often noisy due to estimation errors or hallucinations from the deep learning model. In some cases, it may exhibit abrupt, large-magnitude changes. For example, two consecutive frames may produce inconsistent estimates, or network and inference delays may cause abrupt changes in the reported pose between updates. These effects depend on network conditions and processing load, and may vary significantly over time. Finally, since only discrete pose estimates are available, velocity must be inferred indirectly.

A processing pipeline is employed to compensate for measurement noise, delays, and missing data, while producing high-frequency, smooth, and accurate estimates of the object state suitable for real-time control. The remainder of this subsection describes the components of this pipeline in detail.

Object Position Estimation

At its core, a time-delayed Kalman filter is used to estimate the object’s position and velocity at each control cycle. The filter maintains these quantities along with the timestamp of the last update, stored as part of the internal filter state:

$$filter_state = [obj_pos, obj_vel, t_last_update]$$

The filter provides two methods: `predict(t)` and `correct(measurement)`. `predict(t)` allows evaluating the object state at any desired timestamp, assuming a constant velocity model. When called, it computes a new state based on the internal state as follows:

- $obj_pos = obj_pos + obj_vel * (t - t_last_update)$
- $obj_vel = obj_vel$

- $t_{last_update} = t$

This enables the acquisition of continuous, high-frequency estimates, even when new measurements are unavailable. Although the filter only receives position measurements as input, it maintains and updates both position and velocity estimates, which is important since the controller also requires velocity input.

When a new pose measurement $T_m(t)$ arrives, the translational component is extracted and `correct( $T_m(t)$ .translation)` is called to update the internal state. The filter parameters are configured to prioritize measurement consistency, which may introduce discontinuities in the estimate but allows the filter to closely follow the actual object trajectory.

To handle delayed measurements, a time-replay mechanism is implemented. The filter stores all past states and their timestamps over the last second in a ring buffer. A ring buffer is a data structure that efficiently stores the most recent entries and discards the oldest ones when new data arrives. When a delayed measurement is received, it includes the actual timestamp of the measurement, allowing the correction to be applied at the appropriate point in time. The filter state corresponding to that timestamp is retrieved, and the filter is rolled back to that state. The correction is then applied using the `correct()` function, and the filter re-predicts forward to the current time. This procedure ensures that each correction is applied at the appropriate point in time, and that the resulting state reflects the most accurate and current estimate available.

A high-frequency estimate of the object’s position and velocity, denoted by $\hat{p}_o(t)$ and $\hat{v}_o(t)$ respectively, is now available and serves as the basis for the downstream filtering and control components. However, the raw estimates from the Kalman filter cannot be used directly, because the Kalman filter is configured to trust measurements strongly, allowing it to react quickly to changes. This is beneficial for accuracy; for example, if a previous measurement was inaccurate or if something in the object’s motion changed, the filter will adjust quickly. However, this setup has a drawback: after calling `correct()`, the filter state can jump significantly. So, although the estimates are accurate, they are not always smooth or continuous. Sudden discontinuities in the input trajectory are problematic from a control perspective. They can lead to large instantaneous torques, excite unmodeled dynamics, and degrade stability. To avoid this, additional processing is applied to make the output controller-ready. The first step is applying a rate limiter, whose goal is to prevent large instantaneous jumps and make the reference signal continuous. At time t_n , a new object position estimate $\hat{p}_o(t_n)$ is obtained from the Kalman filter. Here, n denotes discrete control cycles. The previous estimate $p_{o,ref}(t_{n-1})$, stored during the last control cycle, is also available. Here, the subscript “ref” denotes the final filtered value, after applying all corrections and smoothing. In contrast, the hat notation indicates the output

of the Kalman filter. The difference $\hat{p}_o(t_n) - p_{o,ref}(t_{n-1})$ is computed. Ideally, the reference would be updated by directly jumping to the new estimate:

$$p_{o,ref}(t_n) = p_{o,ref}(t_{n-1}) + \hat{p}_o(t_n) - p_{o,ref}(t_{n-1})$$

However, this change may be too significant and introduce discontinuities. Instead, the change is saturated elementwise:

$$\mu = \max(\min(\hat{p}_o(t_n) - p_{o,ref}(t_{n-1}), \text{threshold}), -\text{threshold})$$

and update:

$$p_{o,ref}(t_n) = p_{o,ref}(t_{n-1}) + \mu$$

Here, the absolute value of each component of the difference vector is limited to a predefined threshold. This threshold is chosen empirically based on the maximum Cartesian velocity of the end-effector. It is set large enough to allow the robot to catch up to a moving object, but small enough to maintain smooth and safe tracking behavior.

However, the signal may still have non-smooth curvature even after rate limiting. For example, if the object abruptly changes direction, the rate-limited signal will exhibit a discontinuity in its derivative at that point, resulting in an instantaneous change in velocity. This can lead to unstable or jerky robot motion. To smooth them out, a low-pass filter is applied to the output of the rate limiter. The filter is applied as:

$$p_{o,ref}(t_n) = \alpha \cdot p_{o,ref}(t_n) + (1 - \alpha) \cdot p_{o,ref}(t_{n-1})$$

A high value of α (e.g., 0.9) is used, since the signal has already been filtered and rate-limited. The goal at this stage is simply to smooth out directional transitions, rather than to reject noise.

This results in a smoothed, bounded reference object position $p_{o,ref}(t_n)$ that the end-effector can safely and stably follow at 1000 Hz.

Object Translational Velocity Estimation

The following describes how the reference velocity $v_{o,ref}(t_n)$ is obtained. A direct velocity estimate is not available, as the FoundationPose model provides only pose information. However, velocity is part of the Kalman filter state, which provides a filtered velocity estimate $\hat{v}_o(t_n)$ at high frequency. The same post-processing used for position (rate limiter and low-pass filter) was considered for velocity. However, since the object moves at a near-constant speed on the conveyor and the Kalman filter already acts as a low-pass filter,

adding these steps does not improve stability or accuracy; instead, it increases complexity and latency. So, the reference object velocity is set as

$$v_{o,ref}(t_n) = \hat{v}_o(t_n)$$

directly from the Kalman filter estimate.

Object Orientation Estimation

The reference orientation $R_{o,ref}(t_n)$ is computed based on orientation measurements $R_m(t_m)$, which arrive together with position measurements $p_m(t_m)$ at a lower frequency, typically around 30 Hz. These measurements may contain noise or occasional estimation errors. To ensure smooth behavior, the reference is not directly set to the latest measurement. Instead, an intermediate orientation estimate is constructed, and the reference orientation is updated gradually over time.

When a new measurement $R_m(t_m)$ arrives at time t_m , a filtered orientation estimate, denoted by $\hat{R}(t_m)$, is computed by interpolating between the previous reference orientation $R_{o,ref}(t_m^-)$ and the new measurement:

$$\hat{R}(t_m) = \text{SLERP}(R_{o,ref}(t_m^-), R_m(t_m), \beta)$$

Here, SLERP (Spherical Linear Interpolation) smoothly interpolates between two rotation matrices on the unit sphere, preserving constant angular velocity. The parameter $\beta \in [0, 1]$ is a fixed smoothing factor that determines the weight given to the new measurement relative to the prior estimate. A smaller β yields slower response but smoother updates. In the current implementation, $\beta = 0.3$ is used, providing a balance between stability and reactivity. The estimate $\hat{R}(t_m)$ serves as the target for reference orientation updates during the interval $t_n \in [t_m, t_{m+1})$. At each control cycle, the current reference orientation $R_{o,ref}(t_n)$ is incrementally rotated toward $\hat{R}(t_m)$, subject to a fixed maximum angular velocity. This rate-limiting step ensures that orientation changes smoothly over time, even in the presence of abrupt variation in the measurements.

Object Angular Velocity Estimation

The reference angular velocity $\omega_{o,ref}(t_n)$ is not estimated by the perception system or the Kalman filter, so a valid estimate for it is not available. Since the object's orientation is nearly constant and only changes slowly, $\omega_{o,ref}(t_n)$ is set to 0. This simplifies the control, avoids injecting noise into the torque calculation, and is consistent with the system behavior. No negative effects were observed during testing in the described setup.

Together, the components of this pipeline produce high-frequency and smooth estimates of the object’s full state, including object pose

$$T_{o,ref}(t) = \begin{bmatrix} R_{o,ref}(t) & p_{o,ref}(t) \\ 0 & 1 \end{bmatrix}$$

and twist

$$V_{o,ref}(t) = \begin{bmatrix} v_{o,ref}(t) \\ \omega_{o,ref}(t) \end{bmatrix}$$

5.3.5.2 Grasping Strategy and Converging Motion

To grasp a moving object, the robot must reach a specific pose relative to the object known as the *grasp pose*. This pose aligns the gripper in a way that ensures a stable pickup given the object’s geometry. This target is defined using a fixed transformation T_{grasp} :

$$T_{\text{grasp}} = \begin{bmatrix} R_{\text{grasp}} & p_{\text{grasp}} \\ 0 & 1 \end{bmatrix}$$

which specifies the desired orientation and position of the end-effector in the object frame. At any time t , the object’s pose estimate $T_{o,ref}(t)$, expressed in the base frame, is obtained as described in Subsection 5.3.5.1. The corresponding desired end-effector pose is:

$$T_{ee,\text{grasp}}(t) = T_{o,ref}(t) \cdot T_{\text{grasp}}$$

However, moving directly to the grasp pose risks collision, especially if the end-effector approaches from an arbitrary direction. To mitigate this, a pre-grasp pose is introduced. It is defined by applying an offset from the grasp pose along a predefined direction, specifically the Z-axis of the end-effector expressed in the base frame. This offset ensures that the end-effector approaches the object along a straight, aligned, and collision-free path. The approach direction is defined as:

$$d(t) = R_{ee,\text{grasp}}(t) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The pre-grasp pose is then defined by placing the end-effector at a fixed distance λ_0 behind the grasp pose along this direction, while maintaining the same orientation:

$$p_{ee,\text{pre-grasp}}(t) = p_{ee,\text{grasp}}(t) - \lambda_0 d(t)$$

$$R_{ee,pre-grasp}(t) = R_{ee,grasp}(t)$$

Once the end-effector reaches this pre-grasp pose, the system begins a smooth converging motion along the end-effector's Z-axis, gradually reducing the offset $\lambda(t)$ from λ_0 to zero. At any time t , the desired pose is:

$$p_{des}(t) = p_{ee,grasp}(t) - \lambda(t)d(t)$$

$$R_{des}(t) = R_{ee,grasp}(t)$$

The offset $\lambda(t)$ is decreased at each control cycle by a small fixed step $\Delta\lambda$, chosen based on the control frequency and a maximum allowed Cartesian velocity:

$$\lambda(t + \Delta t) = \lambda(t) - \Delta\lambda$$

This approach generates a straight-line motion toward the grasp pose. Because the grasp pose is defined relative to the object, which is tracked in real time, this converging motion keeps the end-effector aligned with the object as it moves. Tracking continues until the gripper is fully closed, ensuring a reliable pickup.

The following section explains how this converging behavior is implemented as a smooth trajectory that the controller can follow.

5.3.5.3 Trajectory Generator

The purpose of this section is to explain how a reference trajectory is constructed for real-time tracking. The trajectory includes the desired end-effector pose, velocity, and acceleration. The section first describes how the desired pose and velocity are derived from the object state estimate and grasping strategy, and then explains how acceleration is incorporated using short-horizon prediction.

To compute the desired end-effector pose and twist, the grasping logic described in Section 5.3.5.2 is applied. The grasping strategy is defined by a time-varying transformation that maps the object pose to a desired end-effector pose. This includes both the grasping orientation and an offset along the end-effector's local Z-axis, which is gradually reduced over time. By applying this logic to the current reference object state $T_{o,ref}(t)$, the target end-effector pose $T_{des}(t_n)$ is obtained, composed of rotation $R_{des}(t_n)$ and position $p_{des}(t_n)$. For the twist $V_{des}(t)$, it is set directly to the reference twist of the object $V_{o,ref}(t)$.

This pose and twist define the target motion for the end-effector and could be passed to the task-space controller to generate appropriate torque commands; however, they are

insufficient to ensure stable and precise control, especially when the object moves quickly and timing is tight. In simple scenarios, a PD controller without a feedforward term is often sufficient, especially when the object moves at a constant speed. This assumption is nearly valid in the current system, where the object moves steadily on a conveyor belt. However, at the start of visual servoing, the object is already in motion, while the end-effector remains stationary. At that moment, the robot must catch up quickly to align with the moving target. This is where the limitation becomes apparent: the PD controller does not explicitly account for changes in the rate of motion. It tries to compensate using only position and velocity errors, which results in a slower response and reduced tracking accuracy. In practice, the robot lags behind the object during the initial phase.

To address this, a PD controller with the feedforward term (Equation 2.2) is used. This allows the system to anticipate motion rather than merely react to it. However, anticipating motion requires knowledge of how it changes over time, specifically the acceleration. Since direct access to acceleration data is not available, acceleration is computed by generating a short-horizon trajectory based on the current end-effector state. This provides a sequence of desired positions, velocities, and accelerations that the controller can use to apply more accurate torque commands.

A short-horizon trajectory is generated whenever a new measurement is received to compute a smooth desired acceleration for the feedforward term. The horizon duration is set to 200 ms. Upon receiving a new measurement, the data is first processed to obtain the desired end-effector state $T_{des}(t)$ and $V_{des}(t)$. A constant velocity model is then used to predict the end-effector's position 200 ms into the future. Using the current and predicted states, along with the last commanded acceleration as boundary conditions, a 5th-order spline trajectory is generated to connect these points. At time t_m , $p_{des}(t_m)$ and $v_{des}(t_m)$ are available; then boundary conditions are computed as follows:

- $p_{\text{initial}} = p_{ee}(t_m)$
- $p_{\text{predicted}} = p_{des}(t_m) + \Delta t \cdot v_{des}(t_m)$
- $v_{\text{initial}} = v_{ee}(t_m)$
- $v_{\text{predicted}} = v_{des}(t_m)$
- $a_{\text{initial}} = a_{\text{input}}(t_m^-)$
- $a_{\text{predicted}} = 0$

The final acceleration is set to zero based on the assumption that, by the end of the trajectory, the end-effector will have caught up with the object and will be moving at a constant velocity. Then, for each Cartesian coordinate (x, y, z), a 6-equation system is solved using these boundary conditions, which results in a valid trajectory for translational component:

- $p_{\text{traj}}(t) \in \mathbb{R}^3$
- $v_{\text{traj}}(t) \in \mathbb{R}^3$
- $a_{\text{traj}}(t) \in \mathbb{R}^3$

This spline provides new desired positions, velocities, and accelerations for the next 200 ms. If a new measurement arrives before the horizon ends, which is typically the case given data rate of 20–25 Hz, the previous trajectory is discarded and a new one is generated. Suppose no new measurement arrives by the end of the current horizon, such as when network delays occur. In that case, the most recent desired end-effector state is used, and the desired acceleration is set to zero. The controller inputs are then defined as follows:

- if $t_n \in [t_m, \min(t_m + \Delta t, t_{m+1}))$, the current predicted trajectory is used:

$$p_{\text{input}}(t_n) = p_{\text{traj}}(t_n - t_m)$$

$$v_{\text{input}}(t_n) = v_{\text{traj}}(t_n - t_m)$$

$$a_{\text{input}}(t_n) = a_{\text{traj}}(t_n - t_m)$$

- Otherwise, the most recent desired end-effector state is used:

$$p_{\text{input}}(t_n) = p_{\text{des}}(t_n)$$

$$v_{\text{input}}(t_n) = v_{\text{des}}(t_n)$$

$$a_{\text{input}}(t_n) = \vec{0}$$

For orientation, the following definitions are used:

$$R_{\text{input}}(t_n) = R_{\text{des}}(t_n)$$

$$\omega_{\text{input}}(t_n) = \vec{0}, \quad \dot{\omega}_{\text{input}}(t_n) = \vec{0}$$

These values are passed to the task-space PD controller (Equation 2.2), which includes feedforward terms, to compute the required torques for following the reference motion. This allows the end-effector to follow the object smoothly while reducing the spatial offset until it reaches the grasp pose.

5.3.5.4 Grasping

The end-effector continues tracking the object until it reaches the final grasp pose. In practice, the following condition is used to determine grasp readiness:

$$\|p_{ee}(t_n) - p_{ee,grasp}(t_n)\| < \epsilon$$

where $\epsilon = 0.008$ m, chosen empirically as a balance between being precise enough to ensure reliable grasping yet not so strict that the condition would rarely be satisfied. This ensures the end-effector is sufficiently close to the desired pose before initiating the grasp. Even after this condition is met, the trajectory generator continues to issue commands that maintain alignment with the moving object, keeping the end-effector positioned correctly for a stable grasp.

The gripper API used in this system performs the closing operation synchronously, meaning it blocks code execution until the gripper is fully closed. This behavior makes it unsuitable for the main control thread, which must run continuously at 1000 Hz. To avoid interrupting the control loop, the grasping operation is executed in a separate thread. Once the grasp condition is met, this thread is launched to trigger the gripper asynchronously, allowing the main loop to remain active. During this time, the end-effector maintains the grasp pose and continues to correct any misalignments. When the gripper finishes closing, it sends a signal back to the main thread. At that point, the visual servoing state concludes: the robot arm stops, and the system transitions to the next state, where the robot lifts the object from the conveyor.

5.3.6 Lift State

The lift state consists of two sequential substates: a vertical lifting motion followed by a transition to a predefined drop-off location. Unlike the previous stages, no visual feedback is required during either of the substates. This state occurs after the grasping action is completed and does not affect the experimental evaluation presented in this thesis. It was added to complement the system and enable execution of a complete pick-and-place cycle.

In the first substate, the goal is to lift the object vertically to a predefined height. The lifting motion begins from the end-effector pose reached at the end of the visual servoing state. This pose is stored as $T_{lift,init}$. The target pose for the first phase of lifting, referred to as $T_{lift,final}$, is defined by taking $T_{lift,init}$ and replacing the z-coordinate of its translation with a fixed predefined value (0.45 m in this setup), while keeping the x and y positions and the orientation unchanged. To execute this motion, a time-varying desired pose $T_{lift,des}$ is

used. It is initialized to $T_{lift,init}$ and updated incrementally at each control cycle by adding a small vertical offset to its translation component:

$$T_{lift,des}(t_n).translation = T_{lift,des}(t_{n-1}).translation + \begin{bmatrix} 0 \\ 0 \\ \Delta z \end{bmatrix}$$

where $\Delta z = 0.00007$ m. This results in an effective vertical speed of approximately 0.07 m/s, assuming a control rate of 1000 Hz. The motion is tracked using a task-space controller (Equation 2.2) with zero desired velocity and acceleration. Although this can reduce tracking accuracy in general, it is acceptable here due to the slow and unconstrained nature of the motion and because it has no impact on grasp success. This substate continues until the end-effector pose $T_{ee}(t_n)$ reaches $T_{lift,final}$ within a specified tolerance, defined by the condition:

$$\|T_{ee}(t_n).translation - T_{lift,final}.translation\| < \epsilon$$

where $\epsilon = 0.008$ m. Once this condition is met, the system transitions to the second substate.

In the second substate, the goal is to move the arm to a predefined drop-off location $q_{dropoff,final}$, specified in joint space. To generate the motion, the current joint configuration $q_{dropoff,init}$ (recorded at the end of the previous substate) is interpolated to the target configuration $q_{dropoff,final}$, using a 6th-order polynomial trajectory similarly to Section 5.3.3. To compute the polynomial coefficients, the execution time of the motion must be determined first. The task-space distance between the current and target end-effector poses is calculated and then divided by a conservative velocity bound (0.2 m/s in this setup), ensuring that the resulting trajectory respects joint velocity limits. For each joint j , the trajectory is defined with the following boundary conditions: the initial position is set to $q_{dropoff,init}^j$ and the final position to $q_{dropoff,final}^j$. Initial and final velocities and accelerations are set to zero. Once the trajectory is generated, the motion is executed using a joint-space PD controller feedforward term and Coriolis and gravity compensation (Equation 2.1). At each control cycle, the desired joint position, velocity, and acceleration are computed from the polynomial, and the controller applies torque commands accordingly.

When the desired drop-off location is reached, the robot opens the gripper to release the object, completing the pick-and-place pipeline.

Chapter 6

Experiment

This chapter presents the experiments conducted to evaluate the proposed system. It begins with Section 6.1, which provides example test runs to illustrate how the system operates during key phases of the grasping pipeline. Section 6.2 describes the experimental procedure, including the conditions under which the system was tested and the data collected. Finally, Section 6.3 reports the quantitative results of the experiments and analyzes system performance.

6.1 System Behavior Illustration

This section presents an illustrative example from a single test run to clarify how the robot interacts with the object throughout the grasping sequence. Figures 6.1, 6.2, 6.3 show the X, Y, and Z positions of the object (purple) and the end-effector (blue) plotted over time expressed in the base frame. Background shading indicates the robot's current control state, with the following color coding: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown). The sequence of states follows the progression described in Section 5.3. This example illustrates system behavior.

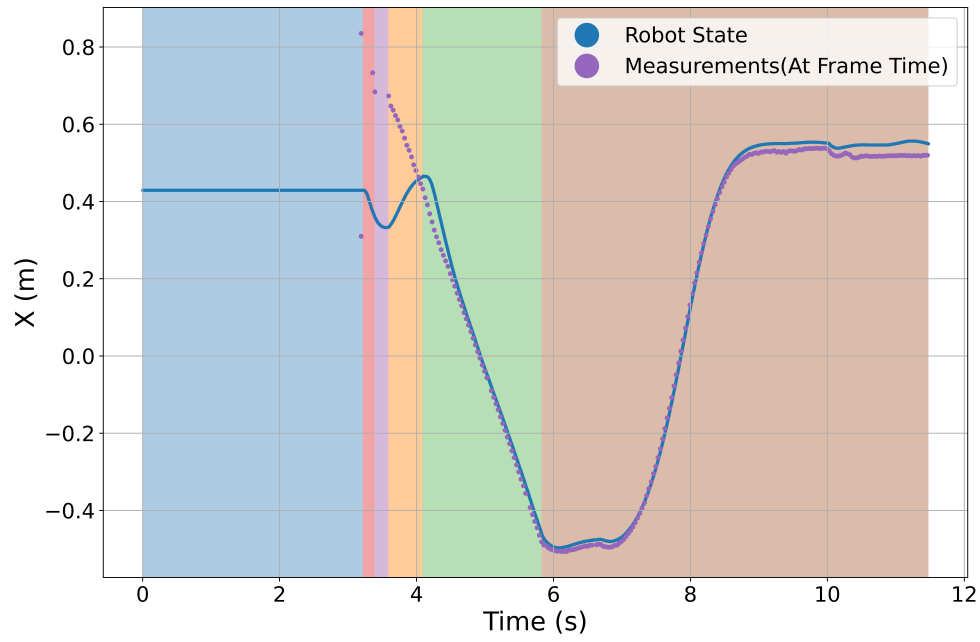


Figure 6.1: Object (purple) and end-effector (blue) positions along the X axis in the base frame over time. Background colors indicate the robot's control state: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown).

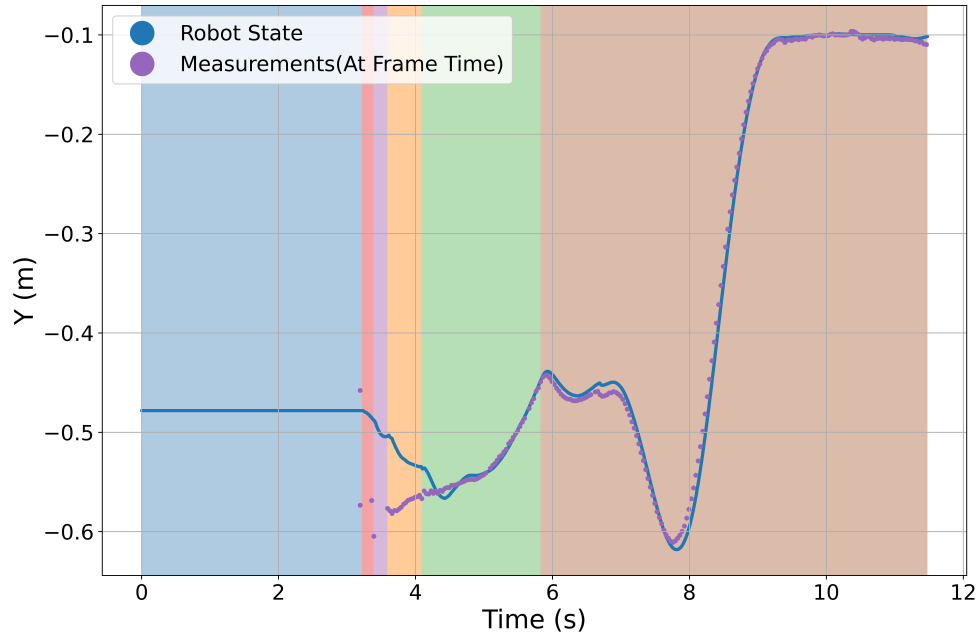


Figure 6.2: Object (purple) and end-effector (blue) positions along the Y axis in the base frame over time. Background colors indicate the robot's control state: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown).

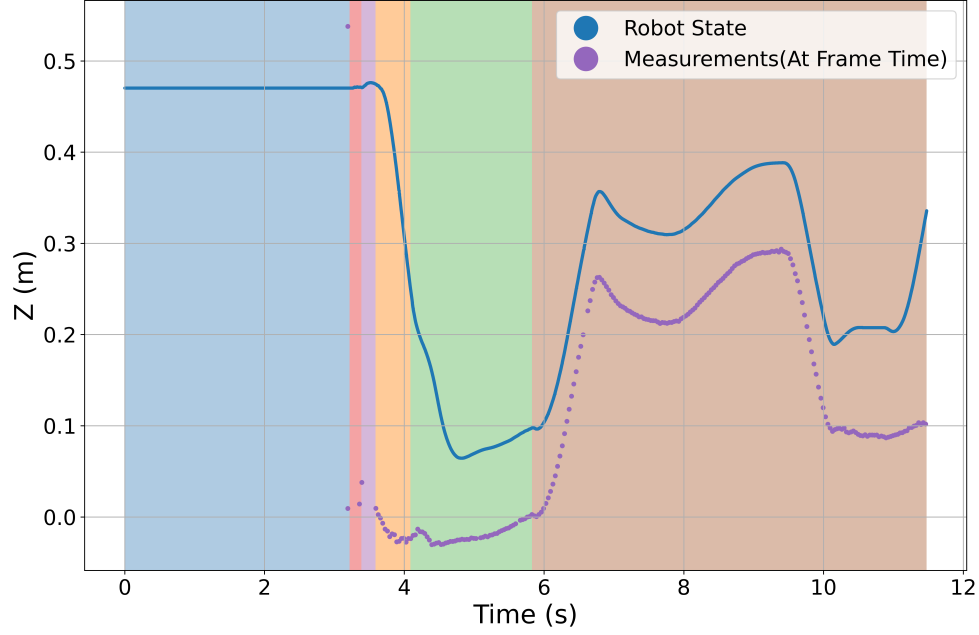


Figure 6.3: Object (purple) and end-effector (blue) positions along the Z axis in the base frame over time. Background colors indicate the robot’s control state: Idle (blue), Registration (red), Compute Approach Point (purple), Approach (orange), Visual Servoing (green), and Lift (brown).

6.2 Experimental Procedure

The experiments were conducted to evaluate the effect of the wireless network on grasping performance. The primary metric was grasp success rate. In each trial, a mustard bottle was placed at the beginning of the moving conveyor belt, and the system was allowed to detect, track, and attempt to grasp the object using 6D pose estimates received from the server. A trial was considered successful if the robot grasped and lifted the object without colliding with or dropping it. In addition to success or failure labels, each failed trial was further categorized according to specific failure reasons, as described in Section 6.2.1. After each attempt, the bottle was manually reset.

In all experiments, the object was placed upright, as shown in Figure 6.5. This reduced the problem from full 6D to 4D pose, since out-of-plane rotations were not considered. This constraint was introduced to ensure that a consistent grasp and pre-grasp pose was

feasible across all initial positions along the conveyor, allowing each orientation–position pair to form a valid test case. Importantly, neither the pose estimation pipeline nor the controller logic relied on this assumption, so the system did not exploit prior knowledge of the object’s orientation.

The experiments consisted of two sets, referred to as Set A and Set B. Set A was conducted under high conveyor belt speed to expose system limitations and observe the effect of different network conditions. Set B used a moderate conveyor speed and served as validation of the control framework under realistic, but less challenging, conditions. Details of each set are provided in Sections 6.2.2 and 6.2.3.

To support deeper analysis of the results, the following data were recorded: round-trip time (RTT) for each pose estimation cycle (including both network delay and server processing time), server-side processing time, end-effector pose at every control cycle, raw object pose measurements, filtered object state estimates produced by the filtering pipeline, and the input sent to the controller.

6.2.1 Categorization of Grasp Outcomes

After the experiments were conducted, each trial was analyzed and labeled based on its outcome. This included both successful and failed grasp attempts. Failed trials were further categorized based on the observed failure mode. Outcomes were grouped into the following seven categories:

- [F0] Success
- [F1] Loss of tracking during the approach phase
- [F2] Velocity constraint violation during the approach phase
- [F3] Velocity constraint violation during visual servoing
- [F4] Initial registration failure
- [F5] Collision with the object during visual servoing
- [F6] End-effector did not close in time

Each failure type is discussed in context within the network-specific analysis.

6.2.2 Set A: High-Speed Experiments

The goal of this set is to understand how different network conditions influence the performance of the grasping system. To expose the effects of communication delays and instability, the conveyor belt speed was set to a high value of 0.5 m/s. At this speed, timing becomes more critical, and the system becomes more sensitive to variations in network behavior. This heightened sensitivity makes the impact of network-related issues more evident and easier to study. However, the increased speed also exposed a limitation of the control logic used during the Approach state (Section 5.3.4), whose goal is to move the end-effector to the pre-grasp pose. Some object orientations result in pre-grasp poses that are unreachable within the available time unless joint velocity constraints are violated. These cases correspond to the failure type labeled [F2]. To avoid early failures caused by this limitation and to better isolate the effect of network conditions during the visual servoing phase, in which the robot responds to real-time visual feedback, the object was always placed upright in a fixed pose with minor placement variations. For this test, the object was placed at the initial position labeled “Right (3)” (Fig. 6.4c) and orientation labeled “0” (Fig. 6.5a). These labels correspond to predefined placements used throughout the experiments. Although the object orientation was fixed, the robot still estimated its pose online and generated the corresponding motion plan without prior knowledge of the object’s pose. This setup ensured consistent testing conditions while allowing the system to operate within its hardware constraints.

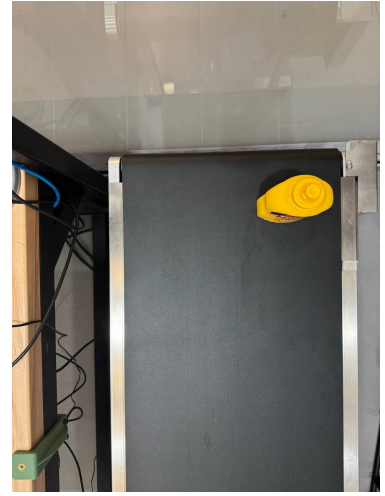
Each of the four network types was tested in 50 grasping trials, resulting in a total of 200 trials for Set A.



(a) Left (1)



(b) Center (2)



(c) Right (3)

Figure 6.4: Initial object positions used in the experiments. The three placements represent different starting positions along the conveyor.

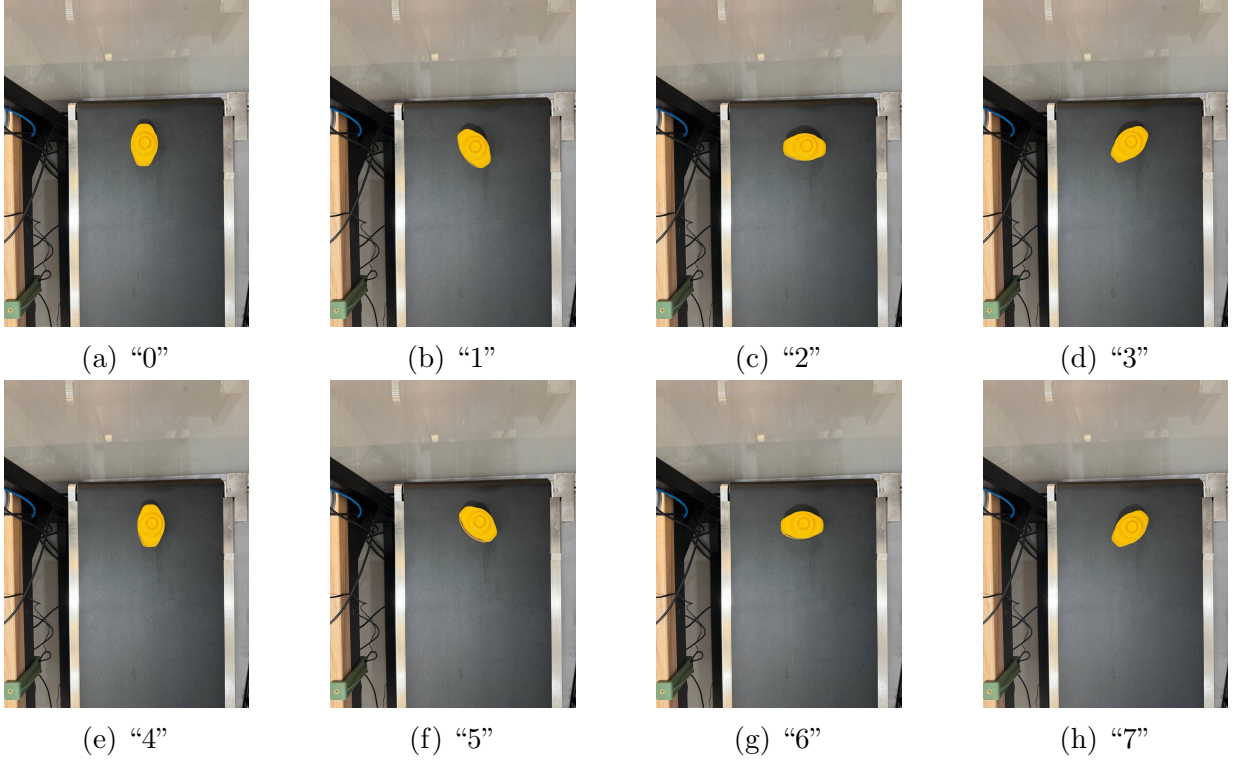


Figure 6.5: Initial object orientations used in the experiments.

6.2.3 Set B: Moderate-Speed Experiments

The goal of Set B is to validate the control framework and demonstrate that the system performs as intended. The conveyor belt speed was reduced to 0.25 m/s to ensure all pre-grasp poses were feasible within the available time, enabling evaluation across multiple initial object configurations. For each of the four network types, the object was placed upright in three different initial positions, as shown in Fig. 6.4. Additionally, for each position, eight different orientations were tested, shown in Fig. 6.5. These orientations were generated by starting from orientation label “0” (Fig. 6.5a) and incrementally rotating the object by 45° around its Z-axis. Each unique combination of position and orientation was repeated twice, resulting in $3 \times 8 \times 2 = 48$ grasping trials per network. This yielded a total of 192 trials across all four networks.

Additionally, the experiments for Set B were conducted earlier in the day, between 15:00 and 16:00. At the same time, Set A trials were performed later in the evening, around

19:00. As a result, there may have been increased interference on shared spectrum bands such as 5 GHz and 3.5 GHz during Set B, due to higher network activity from other users. This timing difference may have contributed to some of the variability observed in network performance.

6.2.4 Network Stability Evaluation

To complement the grasping experiments and provide a more comprehensive assessment of network reliability, an additional long-duration throughput test was conducted. Since the Set A and Set B trials spanned only a few hours each, network reliability was evaluated over an extended 24-hour period. For each network configuration, a constant 70 Mbps data stream was transmitted, and throughput was measured at 100 ms intervals.

A low-throughput interval was defined as a 100 ms interval where the measured throughput fell below 35 Mbps. In addition, occurrences of five consecutive low-throughput intervals were counted to indicate more severe reliability issues. Results from this test are reported alongside the grasping performance outcomes for each network in the corresponding sections.

6.3 Results

This section presents the experimental results and analysis. Section 6.3.1 reports the findings from Set A, which evaluates system performance under high-speed task execution. Section 6.3.2 covers Set B, which assesses the system’s ability to perform 6D grasping under moderate conveyor belt speed.

6.3.1 Results - Set A

Set A evaluates how different network conditions affect grasping performance during high-speed task execution. The conveyor belt was set to 0.5 m/s to increase the system’s sensitivity to communication delays and instability. To ensure consistent testing and focus on the impact of network behavior during visual servoing, the object was always placed upright in a fixed pose. Each of the four network types was tested in 50 trials. The following subsections present the results for each network individually.

6.3.1.1 Wi-Fi at 60 GHz

Among all network types, the Wi-Fi at 60 GHz setup yielded the highest grasp success rate, with 47 successful grasps out of 50 trials (Fig. 6.6). The system demonstrated consistent and stable performance, with a mean round-trip time (RTT) of 41.65 ± 3.46 ms and an average server processing time of 21.45 ± 1.65 ms. Table 6.1 shows the distribution of RTT and processing times across all trials, highlighting low variation and consistent network performance.

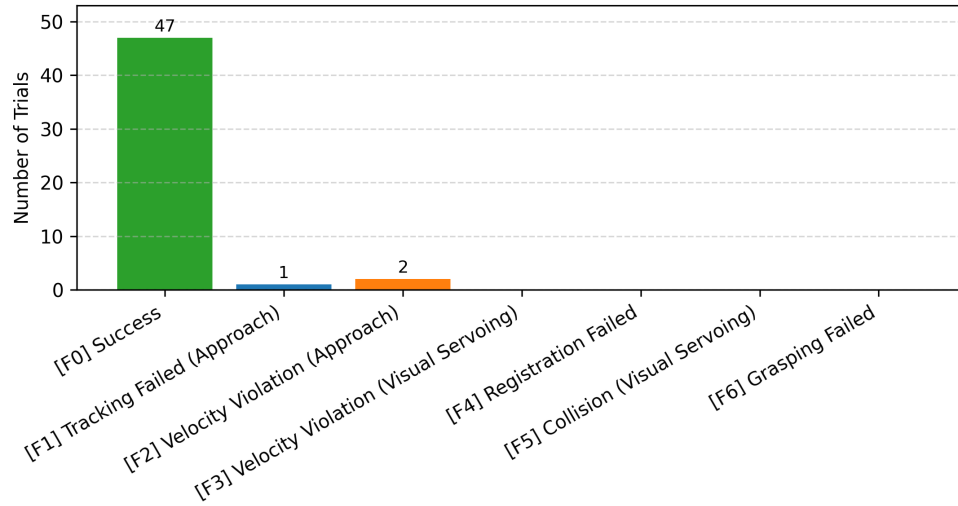


Figure 6.6: Set A: outcomes across 50 trials using Wi-Fi at 60 GHz, categorized by success and failure reason

Only three failures were observed. One failure was caused by [F1] loss of object tracking during the approach phase. The FoundationPose model, which estimates 6D object pose, relies on both the current RGB-D frame and the previous pose estimate, assuming temporal consistency between frames. During fast motion, this assumption can be violated: as the robot moves rapidly toward the object, the object’s appearance changes significantly between frames. In such cases, the model may generate inaccurate pose estimates that worsen over time, especially if continuous motion prevents it from self-correcting. When these estimates drift beyond expected bounds, a safety mechanism halts execution to avoid unsafe behavior.

Two were due to [F2] velocity constraint violations during the approach phase. These issues arose due to a known limitation in the controller logic, previously discussed in Sec-

tion 6.2.2. When selecting a pre-grasp pose, the controller chooses between two symmetric candidates and prefers the one farther from joint limits. Even though the object was placed in a pose that should result in a reachable pre-grasp configuration, this was achieved in the vast majority of trials. However, occasional violations still occurred when multiple contributing factors aligned. These include small placement inaccuracies, since the object was positioned manually without precise measurement, as well as network delay and the stochastic behavior of the optimization solver. Together, these factors sometimes caused the system to select an alternative symmetric pre-grasp pose that required more motion, ultimately violating the velocity constraint. Such violations were rare, occurring once or twice per 50 trials, and appeared across all network types, indicating that the root cause lies in the controller rather than the network itself.

The reliability test results reinforce the observations made during the grasping experiments. The Wi-Fi at 60 GHz link exhibited high stability, with only 0.0083 % of throughput samples falling below the 35 Mbps threshold and no occurrences of five consecutive low-throughput intervals. These findings confirm that the Wi-Fi at 60 GHz link consistently provides reliable performance, supporting its use as a robust baseline for real-time, vision-based robotic control.

6.3.1.2 Wi-Fi 5 at 5 GHz

Under Wi-Fi 5 at 5 GHz, the system completed 44 successful grasps out of 50 trials (Fig. 6.7). The average round-trip time (RTT) was 44.05 ms, close to the Wi-Fi at 60 GHz baseline. However, the standard deviation was significantly higher at 32.96 ms, indicating greater variability in network latency. In contrast, the server processing time remained consistent at 20.04 ± 1.70 ms, which closely matches the performance observed in the Wi-Fi at 60 GHz experiments. This suggests that the perception module itself remained stable and that the latency variation came solely from the network.

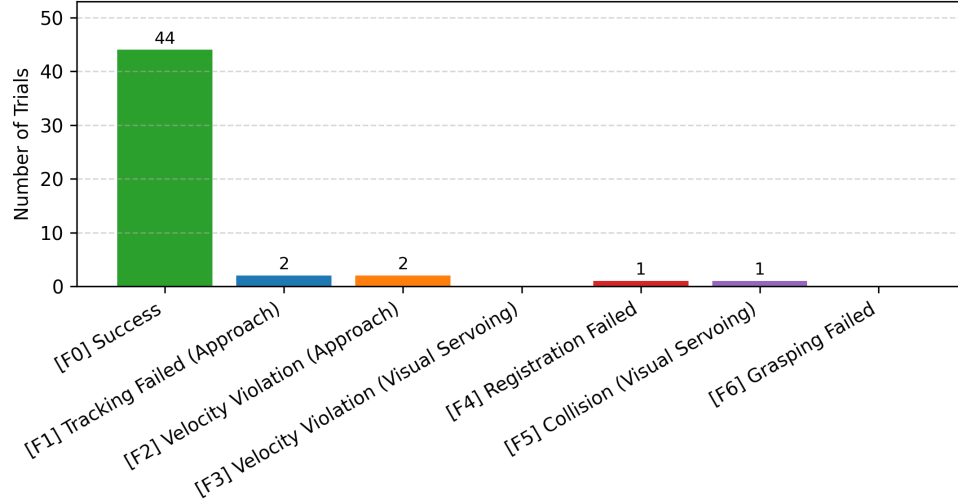


Figure 6.7: Set A: outcomes across 50 trials using Wi-Fi 5 at 5 GHz, categorized by success and failure reason

Six attempts to grasp failed during this experiment. Two failures were due to [F1] tracking loss during the approach phase. As discussed in Section 6.3.1.1, these errors happened when FoundationPose produced incorrect pose estimates due to rapid motion. Since the model assumes consistency between frames, fast movement can cause its predictions to drift. If the estimated pose exceeds the safe limits, the system halts execution to prevent unsafe behavior.

Two additional failures were caused by [F2] velocity constraint violations during the approach phase, resulting from a known limitation in our controller. As explained earlier, the controller selects between two symmetric pre-grasp poses and tends to choose the one farther from joint limits. While this typically helps avoid joint limit issues during visual servoing, it can sometimes result in a longer trajectory that cannot be executed within the time constraints at high conveyor speeds.

Another failure occurred due to [F4] incorrect initial pose estimate during the registration process. On the first frame of the run, FoundationPose returned a significantly incorrect pose, causing the robot to move in the entirely wrong direction. The trial was manually stopped to avoid damage. This issue was observed only once out of 200 total runs recorded in Set A.

The final failure was [F5] collision with the object during visual servoing. The robot was tracking the object and began closing the end-effector, but the end-effector bumped

into the object instead of grasping it cleanly. Logs from that run were examined in detail, but no apparent cause was identified. There were no RTT spikes, control anomalies, or out-of-range pose estimates. This suggests that the error likely resulted from a combination of minor inaccuracies in perception and control that accumulated over time. It represents a rare corner case in which no individual component failed, but the overall system drifted just enough to cause a misalignment.

Figure 6.12 presents the distributions of total round-trip time (RTT) and server processing time measured across all network types during the Set A experiments. The y-axis is displayed on a logarithmic scale, indicating the number of pose estimation events recorded at each time interval. Subfigure 6.12b reveals occasional spikes exceeding one second that occurred during execution. However, analysis of all six failed runs confirmed that none of these high-latency events took place during those trials. This suggests that the failures were not caused by excessive network delay.

To better understand why such delays did not lead to failure, two runs were identified in which RTT briefly exceeded one second. A representative example is shown in Figure 6.8. In Subfigure 6.8a, the object’s X-position estimates are shown with purple dots indicating when each frame was captured, and orange dots showing when the estimates were received. The horizontal gap between these markers illustrates the network and processing delay. Subfigure 6.8b includes the same data along with the predicted estimates, which are computed despite the missing updates. Subfigure 6.8c compares the object’s X-position with the end-effector’s position during the grasping sequence. In this case, the delay occurred near the end of visual servoing, when the end-effector had already reached the grasp pose. The Kalman filter maintained sufficient accuracy to enable the successful completion of the grasp. If the delay had occurred earlier, during the motion from the pre-grasp to the grasp pose, when visual feedback plays a more critical role, the outcome would likely have been a failed grasp.

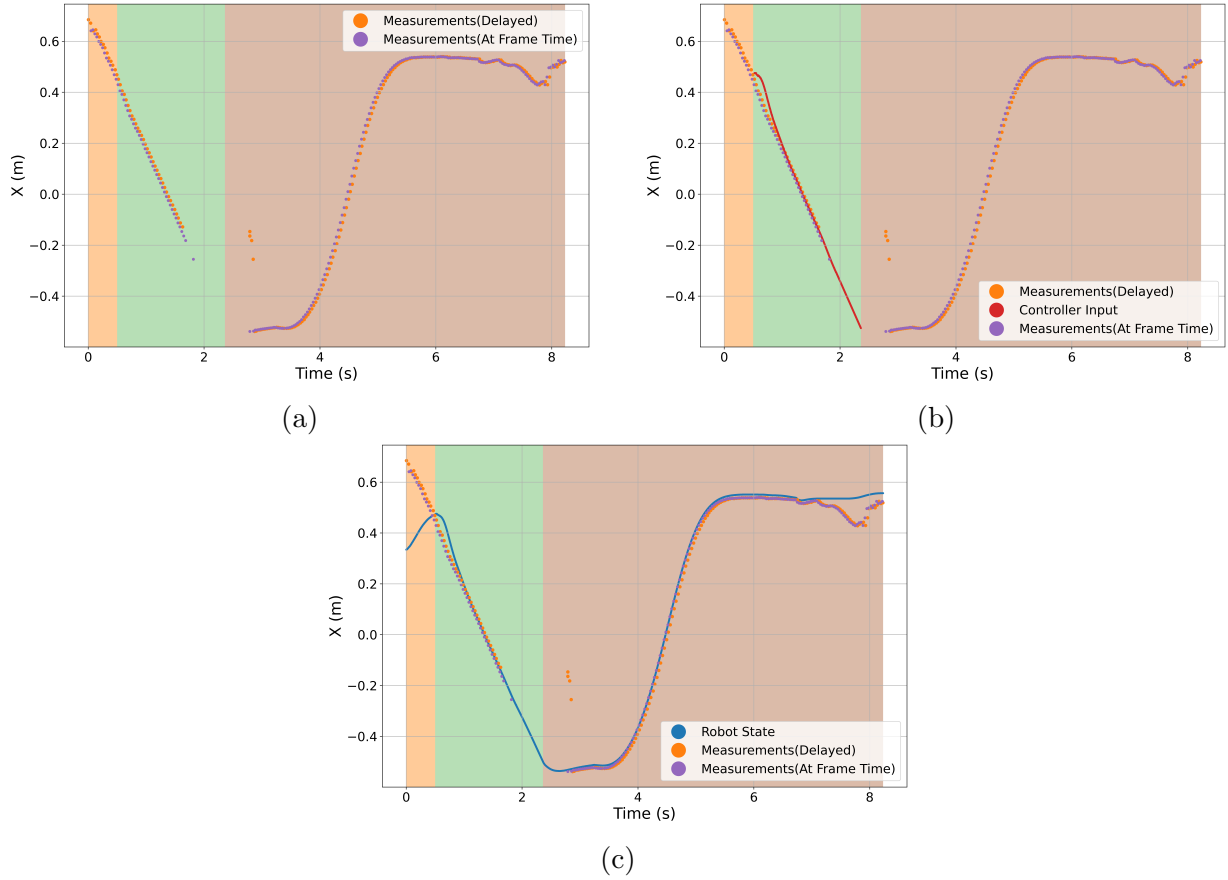


Figure 6.8: (a) Object X-position estimates for each frame: orange dots indicate when the robot received the estimate, and purple dots show when the corresponding camera frame was actually captured. (b) Same data as in (a), with the corresponding predicted estimates shown in red. (c) Comparison of object and end-effector X-positions during the grasping sequence, illustrating tracking and alignment behavior.

Results from the throughput test showed that 1.36 % of intervals experienced a drop below 35 Mbps, and 1.21 % of the time included five or more consecutive low-throughput intervals. These findings align with the reliability issues observed during the grasping experiments, where occasional performance degradation was noted. Compared to the Wi-Fi at 60 GHz, this represents a substantial difference, indicating that while Wi-Fi 5 at 5 GHz may support low-latency operation in principle, its instability poses a clear risk for continuous, real-time robotic applications.

6.3.1.3 5G NSA at 24 GHz

Under 5G NSA at 24 GHz, the system achieved 43 successful grasps out of 50 trials (Fig. 6.17). The average round-trip time was 58.26 ms, with a standard deviation of 6.95 ms. Although latency was higher than in both Wi-Fi setups, the variability was significantly lower than under Wi-Fi 5 at 5 GHz, indicating a more stable connection. The server processing time remained consistent at 20.23 ms, again confirming that perception performance performed consistently across different runs.

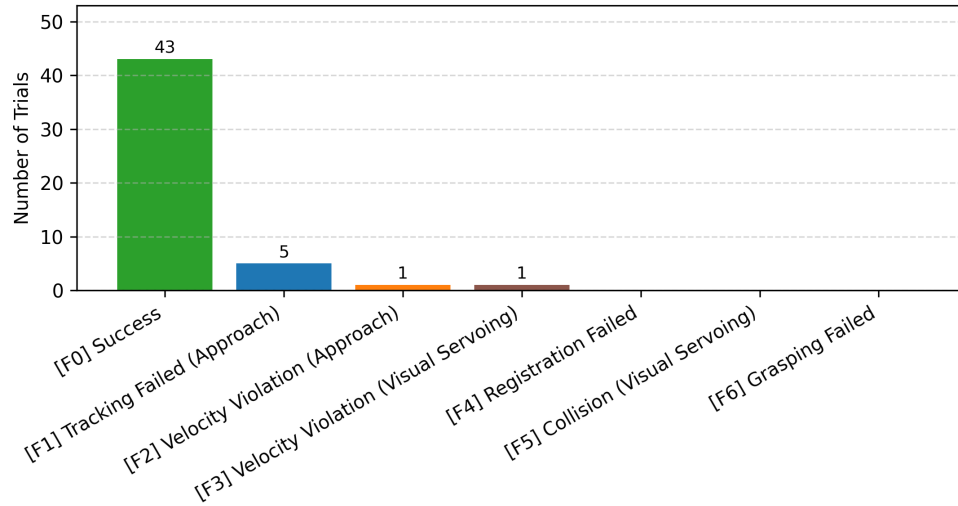


Figure 6.9: Set A: outcomes across 50 trials using 5G NSA at 24 GHz, categorized by success and failure reason

A higher number of tracking losses was observed under this setup, with five trials failing due to [F1] loss of tracking during the approach phase. The underlying cause is difficult to pinpoint precisely because FoundationPose is a deep learning model with high-dimensional internal states and nonlinear behavior; it is inherently challenging to determine the factors that lead to failure in a given trial. As a result, it is not possible to determine exactly what triggered the failure in each case. Only indirect evidence and behavioral trends can be analyzed. To support this analysis, the effective number of pose updates received during the approach phase was measured. In successful runs, where tracking did not fail during the approach state, the average update rate was 29.55 FPS; in failed runs, it dropped to 27.09 FPS (Table 6.3). Although tracking can still succeed at lower frame rates, some successful trials occurred even with a 26 FPS update frequency; a reduced update frequency makes

tracking less stable. A slower update rate weakens the model’s assumption of temporal consistency between frames, which likely increases the risk of prediction drift and tracking failure.

One failure was due to [F2] velocity constraint violation during the approach phase, consistent with the behavior described in previous sections. As before, this stems from the controller’s strategy of selecting the pre-grasp pose farther from joint limits, which occasionally results in an infeasible trajectory under tight timing constraints.

This set introduced a new failure mode that had not been previously observed. One grasp attempt failed due to [F3] velocity constraint violation during the visual servoing state. This occurred at the beginning of the visual serving state, after transitioning from the approach state when the end-effector began accelerating to catch up with the moving object. This behavior is attributed to increased network latency. The delayed start of motion left less time to correct the trajectory, increasing the likelihood of violating control constraints. This issue did not occur under either Wi-Fi setup, suggesting that the added latency reduced the system’s temporal margin of error.

Throughput test results showed that only 0.03 % of intervals fell below 35 Mbps, and no sequences of five consecutive low-throughput intervals were observed. This indicates that the 5G NSA at 24 GHz link was more stable than Wi-Fi 5 at 5 GHz, although not as consistent as the Wi-Fi at 60 GHz baseline. Overall, the system demonstrated reliable performance under this link, but the increased latency introduced new challenges. In fast-paced environments, the reduced timing margin may require more robust controller logic or additional prediction mechanisms to ensure safe and accurate operation.

6.3.1.4 5G NSA at 3.5 GHz

Under 5G NSA at 3.5 GHz the system achieved 23 successful grasp attempts out of 50 trials (Fig. 6.10), which is the lowest success rate among all evaluated network types. The average round-trip time (RTT) was 88.51 ms with a standard deviation of 14.74 ms. Although this standard deviation is smaller than that of Wi-Fi 5 at 5 GHz, the overall latency is significantly higher than that of all other methods. Server-side processing time remained stable at 20.36 ms, indicating consistent performance across trials.

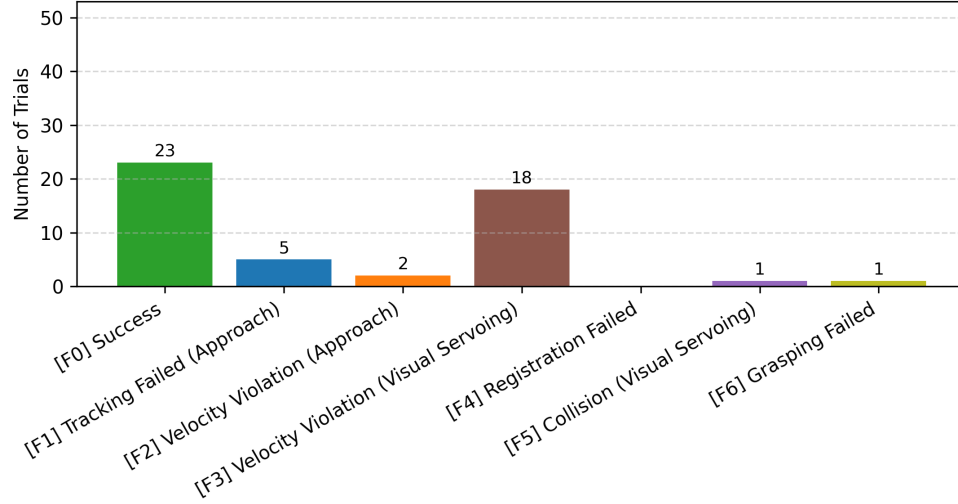


Figure 6.10: Set A: outcomes across 50 trials using 5G NSA at 3.5 GHz, categorized by success and failure reason

Failures were diverse, covering several known and new categories. Five trials failed due to [F1] object tracking loss during the approach phase. This failure mode was previously discussed in Section 6.3.1.3, and the results presented here confirm the same trend. Although the average frame rate for successful runs under the 5G NSA at 3.5 GHz link was relatively low at 26.70 FPS, it was still higher than the frame rate observed in failed runs within the same network configuration. Specifically, trials that experienced tracking failure had an average update rate of 24.56 FPS. While absolute FPS values differ across network types, the trend remains clear: when the update rate drops within a given condition, the temporal consistency that FoundationPose relies on becomes increasingly difficult to maintain. This increases the risk of tracking failure.

Two attempts failed due to [F2] velocity constraint violations during the approach phase; this failure mode was already discussed in earlier sections. These occurrences are consistent with the known controller limitation and typically appear one to two times per 50 trials, regardless of the network.

The most prominent failure mode under this network was [F3] velocity constraint violations during visual servoing, which occurred in 18 out of the 27 failed trials. Although a similar failure was previously observed under 5G NSA at 24 GHz, the frequency observed here represents a notable increase. To better understand the cause, the maximum deviation along the X-axis between the object’s estimated pose and the end-effector’s position during

visual servoing was analyzed (Table 6.4). The deviation between the end-effector and the object was found to be slightly larger in failed runs, averaging 0.129 meters, compared to 0.118 meters in successful ones. This indicates that the end-effector began farther behind the object in those cases. Due to the added delay introduced by network latency, the controller needed to accelerate more aggressively to catch up, which brought it closer to exceeding its velocity constraints. This trend suggests that the system is already operating near its limits, and even modest delays can cause it to exceed its limits.

One instance of [F5] was observed, in which the end-effector collided with the object during visual servoing. This failure was attributed to increased latency, resulting in a mismatch between the predicted and actual object positions. The example illustrates how higher latency can narrow the task’s timing margins and lead to failure, even when tracking and control perform as expected.

Another unique failure [F6] occurred when the end-effector did not close in time during the grasp. In that trial, the object was being tracked correctly, and the grasp command was issued at the right time. However, the system’s high latency caused a slight delay in executing the end-effector command. As a result, the object reached the end of the conveyor and fell before the fingers could close around it.

Results from the reliability test showed that only 0.08 % of intervals were marked as low-throughput, and five consecutive low-throughput intervals occurred in just 0.01 % of the time. While overall performance was not the strongest within the experimental conditions tested, the 5G NSA at 3.5 GHz network demonstrated solid reliability and outperformed the Wi-Fi 5 at 5 GHz setup in terms of connection stability. Although throughput was sufficient to support continuous perception data transfer, the increased latency noticeably reduced system responsiveness during time-critical phases of the task. These results suggest that additional control adaptations may be necessary to ensure safe operation under such conditions.

6.3.1.5 Summary

The results of Set A highlight the importance of network latency and stability in fast-paced, dynamic environments. Table 6.1 summarizes the grasp success rates, failure counts, average RTT, and server processing times for all four network types.

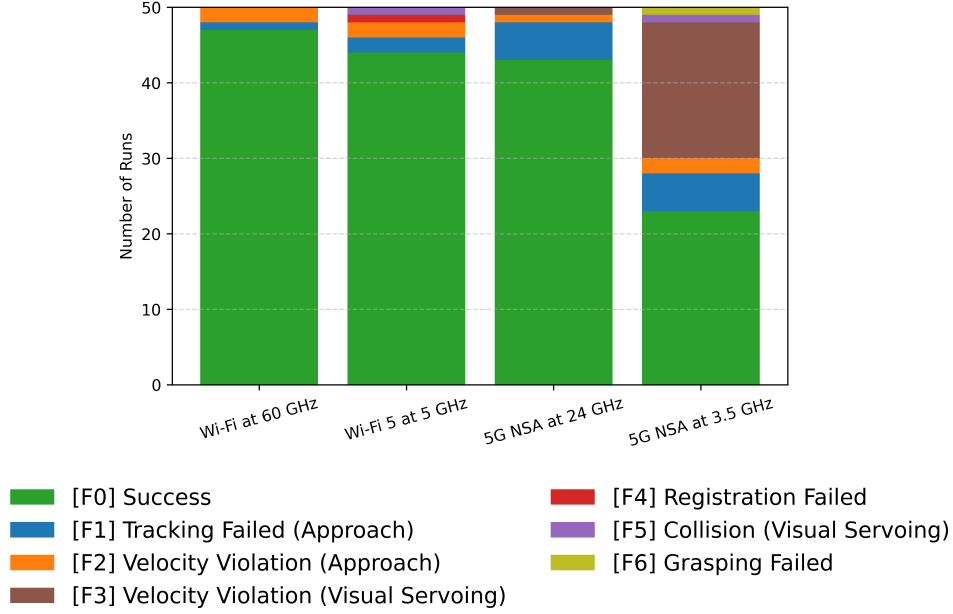


Figure 6.11: Outcome distribution for Set A experiments across four wireless network configurations. Each bar represents 50 trials, broken down by success and specific failure types.

Experiment	Success	Failures	Mean RTT [ms]	Mean Proc [ms]
Wi-Fi at 60 GHz	47	3	41.16 ± 3.46	21.45 ± 1.65
Wi-Fi 5 at 5 GHz	44	6	44.05 ± 32.96	20.04 ± 1.70
5G NSA at 24 GHz	43	7	58.26 ± 6.95	20.23 ± 1.77
5G NSA at 3.5 GHz	23	27	88.51 ± 14.74	20.36 ± 1.80

Table 6.1: Grasp success and failure counts, along with the average and standard deviation of RTT and server processing time for all four networks in Set A.

Among the four tested networks, Wi-Fi at 60 GHz achieved the highest success rate of 94 %, with the lowest latency and most stable connection. This makes it especially well-suited for real-time robotic control tasks.

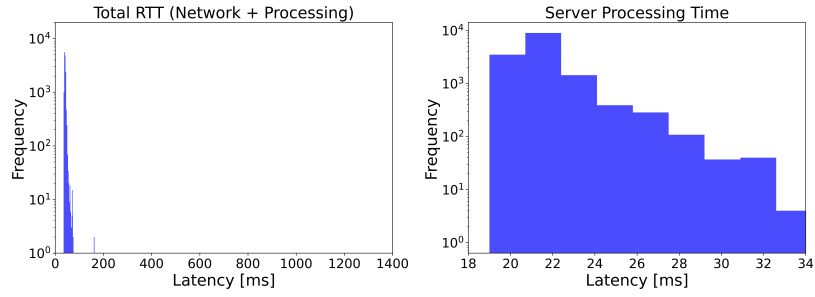
Wi-Fi 5 at 5 GHz also achieved a high success rate of 88 %, with the second-lowest average latency. However, it showed considerable variability, likely due to the widespread use

of this frequency in consumer-grade devices. Although this variability did not consistently lead to failures, it underscores the potential risk, suggesting that proper quality-of-service management is crucial when deploying robotic applications over commonly used frequencies.

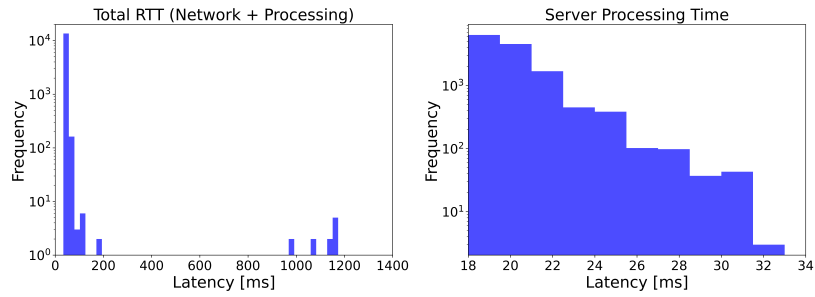
The 5G NSA at 24 GHz network also demonstrated strong performance, achieving an 86 % success rate. Although its latency was higher than both Wi-Fi setups, it demonstrated superior reliability, outperforming the Wi-Fi at 60 GHz setup in terms of stability. Assuming latency requirements are met, 5G NSA at 24 GHz could be a better choice due to its stronger reliability.

Finally, the 5G NSA at 3.5 GHz network demonstrated the worst performance, with only 46 % of trials succeeding. Its high latency often led to timing issues that made it impossible for the robot to complete the task without exceeding velocity constraints. Although it is more reliable in terms of connection stability than Wi-Fi 5 at 5 GHz, it highlights that reliability alone is insufficient if latency is too high.

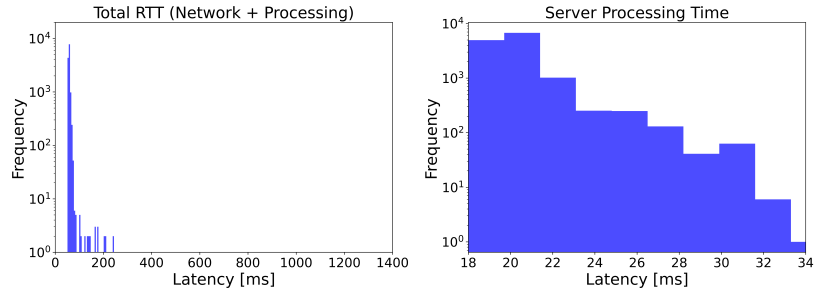
Table 6.2 reports the percentage of low-throughput intervals observed during a 24-hour streaming test, while Figure 6.12 shows the distributions of round-trip time (RTT) and server processing time recorded during Set A. Although Set A consists of short-duration task executions, the network behavior observed in these runs closely matches the longer-term reliability patterns captured in the 24-hour test. This consistency across both time scales supports the validity of the conclusions drawn from the Set A experiments.



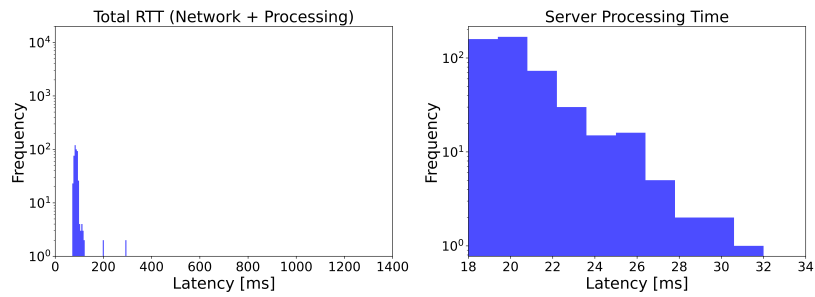
(a) Wi-Fi at 60 GHz



(b) Wi-Fi 5 at 5 GHz



(c) 5G NSA at 24 GHz



(d) 5G NSA at 3.5 GHz

Figure 6.12: Distributions of total round-trip time (RTT) and server processing time recorded for each network type in Set A.

Experiment	Low Throughput Intervals [%]	5× Low Throughput Intervals [%]
Wi-Fi at 60 GHz	0.008	0
Wi-Fi 5 at 5 GHz	1.36	1.21
5G NSA at 24 GHz	0.03	0
5G NSA at 3.5 GHz	0.08	0.01

Table 6.2: Percentage of single and five consecutive low-throughput intervals observed during the 24-hour streaming test for each network in Set A.

Table 6.3 presents the tracking performance in Set A by comparing the average effective pose estimates per second between runs where tracking failed ([F2] outcome) and runs where it did not, for each network. The results show a consistent trend across all networks: runs with tracking failure are associated with lower effective FPS. This generalizes the earlier observation and confirms that reduced update frequency is a common factor contributing to tracking loss, regardless of network type.

Experiment	Tracking OK	FPS (OK)	Failures (F2)	FPS (Fail.)
Wi-Fi at 60 GHz	47	29.92 ± 0.76	1	28.68 ± 0
Wi-Fi 5 at 5 GHz	46	29.53 ± 0.98	2	28.42 ± 0.75
5G NSA at 24 GHz	44	29.55 ± 1.36	5	27.09 ± 0.89
5G NSA at 3.5 GHz	41	26.7 ± 1.91	5	24.56 ± 2.6

Table 6.3: Tracking performance during the approach state in Set A, reported per network. “Tracking Lost” indicates the number of runs where tracking failed during the approach state, while “Tracking OK” refers to runs that completed successfully. FPS denotes the average number of pose estimates received per second, computed across all runs for each network.

Table 6.4 reports, for each network, the number of successful grasps and the corresponding maximum X-axis deviation during those trials, as well as the number of failures due to [F3] velocity constraint violations and the associated deviations. Although [F3] failures were not present under Wi-Fi at 60 GHz and Wi-Fi 5 at 5 GHz, and occurred only once under 5G NSA at 24 GHz, the results consistently show that successful runs exhibited lower maximum deviations compared to those that resulted in violations. The standard deviations indicate some overlap, suggesting that large deviations do not always result in

failure; however, the overall trend supports the conclusion that higher X-axis misalignment increases the risk of violating velocity constraints.

Experiment	Succ. VS	Max Δx (Succ.) [m]	Failures (F3)	Max Δx (Fail.) [m]
Wi-Fi at 60 GHz	47	0.102 ± 0.009	0	-
Wi-Fi 5 at 5 GHz	44	0.991 ± 0.009	0	-
5G NSA at 24 GHz	43	0.101 ± 0.009	1	0.119 ± 0
5G NSA at 3.5 GHz	23	0.118 ± 0.007	18	0.129 ± 0.012

Table 6.4: The table lists the number of successful grasps, the maximum x-axis deviation observed during those successful trials, the number of failures caused by velocity constraint violations, and the corresponding maximum deviation during those failed attempts. All measurements were taken during the visual servoing state.

Overall, these results show that different networks can produce vastly different outcomes, even when the rest of the system remains unchanged. These results underscore the importance of carefully aligning task requirements and control logic with the specific characteristics of the wireless infrastructure.

6.3.2 Results – Set B

Set B focuses on evaluating grasping performance at a moderate conveyor speed of 0.25 m/s. While the primary evaluation metric remains grasp success rate, this configuration no longer pushes the robot to its physical limits. Instead, the goal is to assess whether the system can reliably perform agile grasps across a wide range of object orientations and positions, under different network conditions.

6.3.2.1 Wi-Fi at 60 GHz

Under the Wi-Fi at 60 GHz connection, the system successfully completed all 48 grasping trials (Fig. 6.13), achieving a 100 % success rate. The average round-trip time remained consistent at 38.57 ms, matching the results observed in Set A (Section 6.3.1.1). The standard deviation was also low at 3.75 ms, reinforcing the observation that consumer wireless devices rarely use the 60 GHz band and remain largely free from interference. This contributes to stable communication and consistent system performance. These results indicate that when the network is reliable, all system components function as intended. It

can handle agile and repeatable grasping without failures. This setup confirms that under good communication conditions, the system performs as expected without encountering any operational constraints.

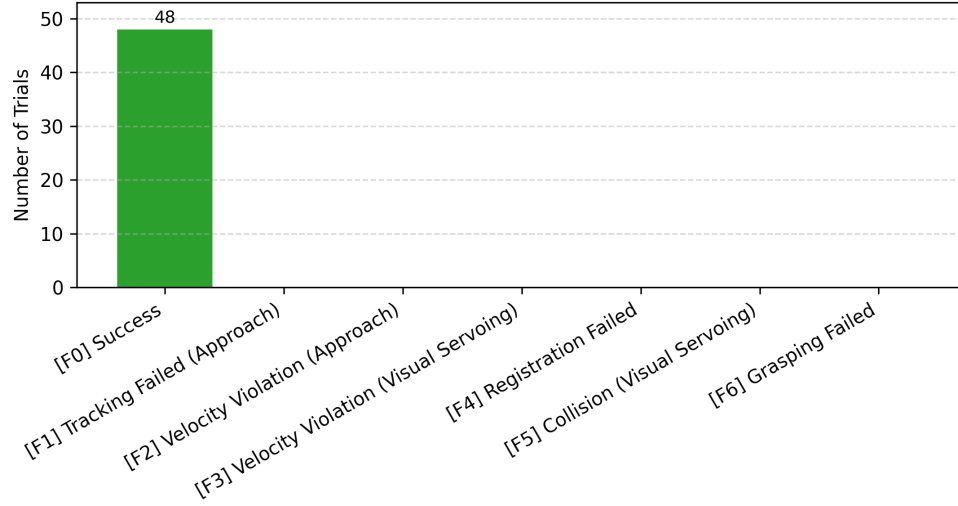


Figure 6.13: Set B: outcomes across 48 trials using Wi-Fi at 60 GHz, categorized by success and failure reason

6.3.2.2 Wi-Fi 5 at 5 GHz

Under the Wi-Fi 5 at 5 GHz connection, the system achieved 45 successful grasps out of 48 trials (Fig. 6.13). The average round-trip time increased to 51.02 ms, with a significantly higher standard deviation of 65.11 ms compared to 44.05 ± 32.96 ms in Set A (Section 6.3.1.2). This indicates that the network was less stable during this experiment, most likely due to increased interference from other users. Such a significant standard deviation suggests severe variability, including the possibility of substantial network outages.

Despite the unstable connection, the system maintained a high success rate. To better understand this outcome, RTT logs were analyzed, revealing that 42 pose estimate messages out of 14,139 experienced delays exceeding one second. These spikes occurred across nine different runs. Notably, none of those runs failed.

Further analysis revealed that the spikes occurred either near the end of the visual servoing phase, where the prediction component of the control pipeline was able to mitigate

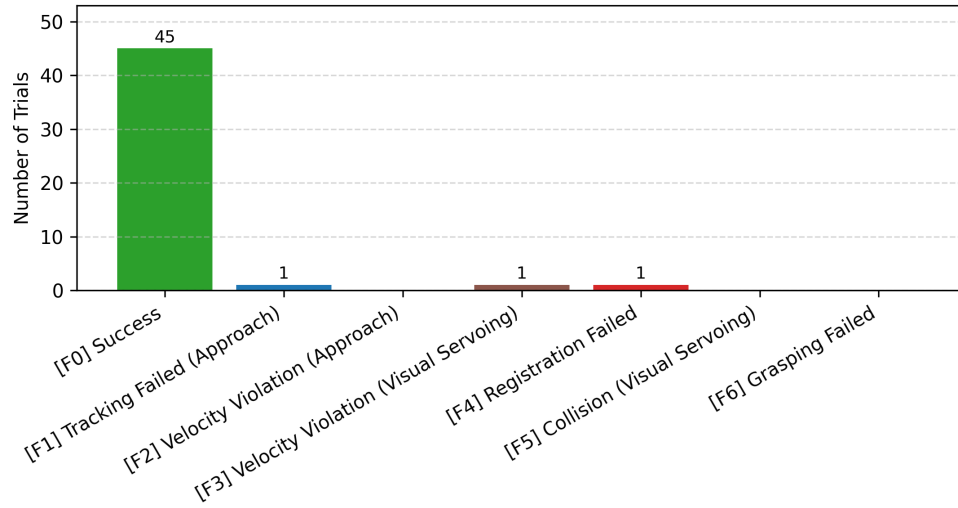


Figure 6.14: Set B: outcomes across 48 trials using Wi-Fi 5 at 5 GHz, categorized by success and failure reason

their impact, or during states where visual feedback was not required, such as the approach or lift phases. As a result, the system remained unaffected. However, if such delays had occurred during phases where pose estimation data was critical, successful task execution would have been less likely. To test this hypothesis, the experiment was extended by running additional trials during the same time period. Earlier observations had shown that the network was unreliable and prone to unpredictable degradation, making it likely that a delay would eventually disrupt a phase where visual feedback was critical. Indeed, during one stretch, 8 out of 10 trials failed.

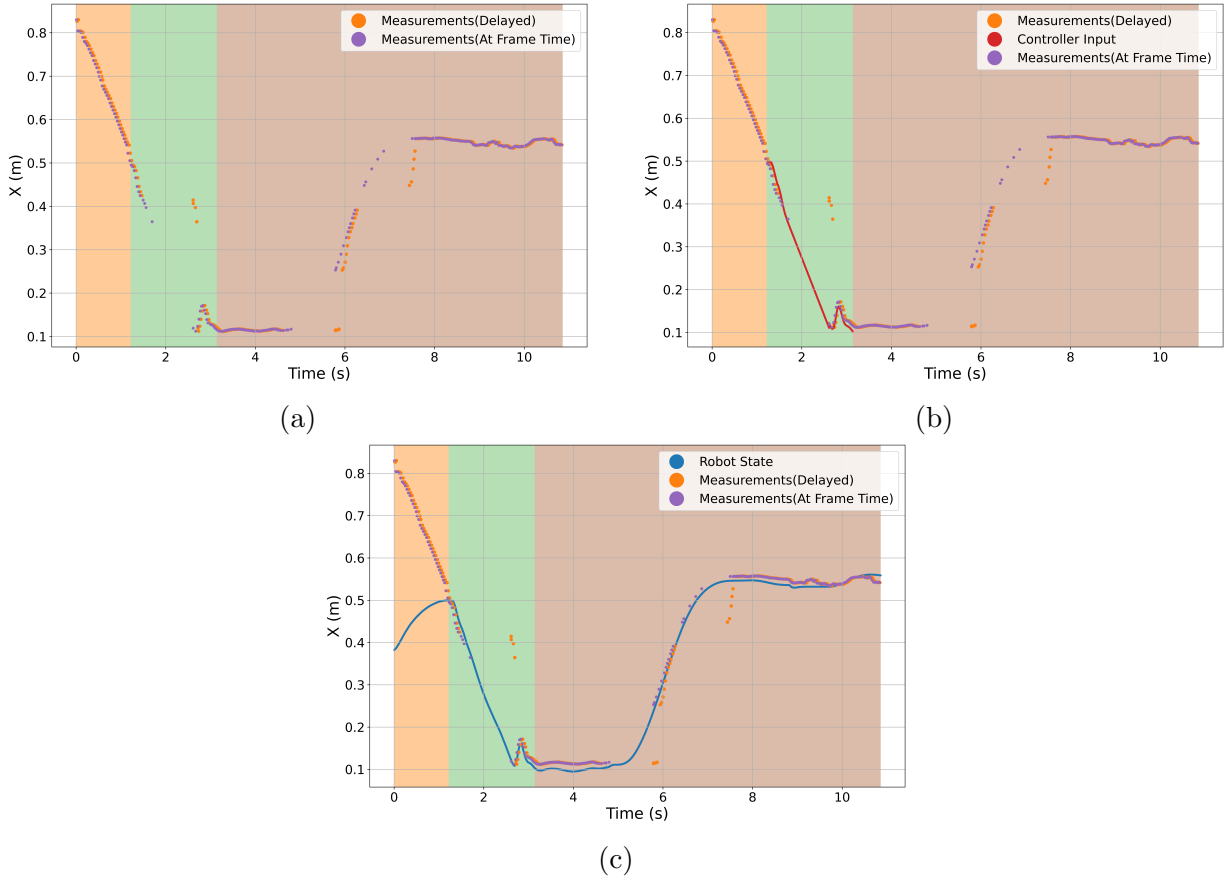


Figure 6.15: (a)–(c) show Approach (orange), Visual Servoing (green), and Lift (brown) states of a run with delayed perception where the grasp succeeded. (a) shows the estimated object X-position, with purple dots indicating capture times and orange dots indicating reception times. (b) includes the prediction from our filtering mechanism (red), which compensated for the delay. (c) compares the object and end-effector (blue) X-positions during the grasp.

Figure 6.15 illustrates a successful example from this batch, where delays were present but did not impact the outcome. During this run, multiple delays occurred; however, they were either compensated for by the prediction mechanism or occurred during the lifting phase, where visual feedback is no longer used. In the plots, purple dots indicate when each camera frame was captured, and orange dots show when the robot received the corresponding estimate. The shaded background marks the current control state: orange for approach, green for visual servoing, and brown for lifting.

Figure 6.16 shows a contrasting failed run from the same batch. In this case, a delay occurred during the transition from the approach phase to visual servoing. The Subfigure 6.16b shows that the controller input to the robot was not issued until the delayed pose estimate arrived. As a result, the object had already moved too far ahead, and the controller attempted to catch up by issuing a high-velocity command, which ultimately led to a [F3] velocity constraint violation.

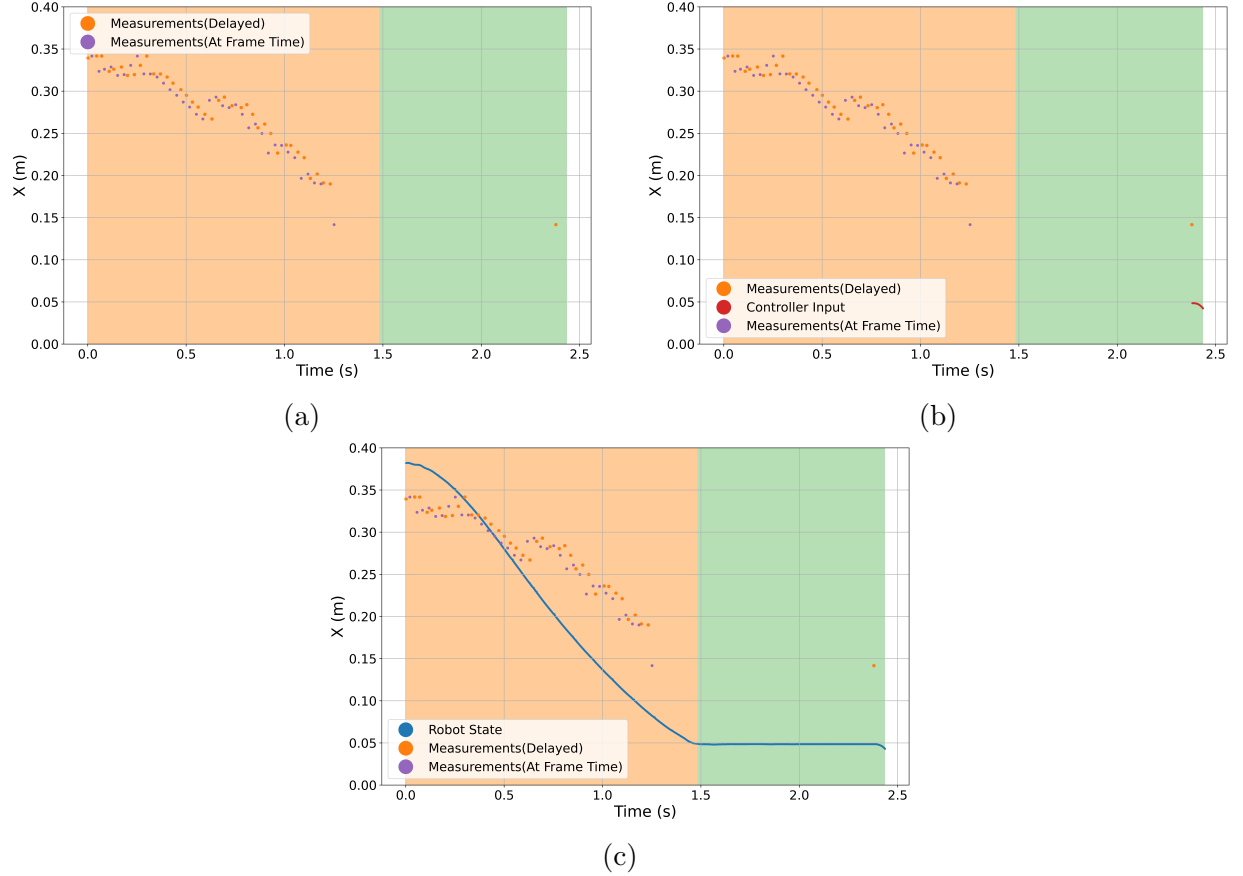


Figure 6.16: (a)–(c) show Approach (orange) and Visual Servoing (green) states of a run with delayed perception where the grasp failed. (a) shows the estimated object X-position, with purple dots indicating capture times and orange dots indicating reception times. (b) includes the prediction from our filtering mechanism (red), which compensated for the delay. (c) compares the object and end-effector (blue) X-positions during the grasp.

This suggests that the failure is not due to a deterministic flaw in the system but instead

emerges when multiple devices compete for bandwidth on the same frequency band. It emphasizes the importance of deploying networks properly and ensuring quality-of-service guarantees when utilizing wireless infrastructure for time-sensitive robotic applications. Without some level of traffic isolation or bandwidth reservation, the system becomes vulnerable to interference and contention.

Among the three failed trials in the main 48-trial batch, one was caused by [F1] tracking loss during the approach phase. Another failure was due to [F3] velocity constraint violation during visual servoing. It occurred at the initial position Left (1), where the object is placed close to the robot base. This is attributed to the combined effect of increased latency, which reduces the available reaction time, and the fact that reaching near the base requires higher joint velocities. This combination likely pushed the controller beyond its limits. The final failure was [F4] incorrect initial pose estimate during the registration step. This is a known but rare failure mode that have been encountered before.

6.3.2.3 5G NSA at 24 GHz

Under the 5G NSA at 24 GHz network, the system successfully completed 47 out of 48 grasp attempts (Fig. 6.17). The average round-trip time was 61.03 ms with a standard deviation of 6.25 ms, closely matching the results observed in Set A (Section 6.3.1.3). This low variability suggests that the network remained stable throughout the experiment. The consistency reinforces our assumption that the 24 GHz mmWave band is not widely used by other devices and is therefore less prone to interference from nearby users.

Only one failure [F4] occurred during the experiment. It took place at the initial position Left (1), where tracking was lost shortly after the initial registration step. Since this issue appeared only once across all 48 trials, it is reasonable to consider it a statistical outlier rather than evidence of a systematic problem.

Notably, there were no tracking failures during the approach phase. This contrasts with Set A, where five trials failed due to lost tracking in that stage. A likely reason for this improvement is the lower conveyor speed used in Set B, which allowed the robot to move more slowly. This increased the temporal consistency between incoming frames, making the perception system more reliable overall.

Overall, the system demonstrated strong and consistent performance on this network. The high grasp success rate, combined with stable network conditions, shows that the system is capable of performing agile grasping reliably. These results further support the conclusion that 5G networks, when properly configured, can support time-sensitive robotic applications that rely on offboard perception.

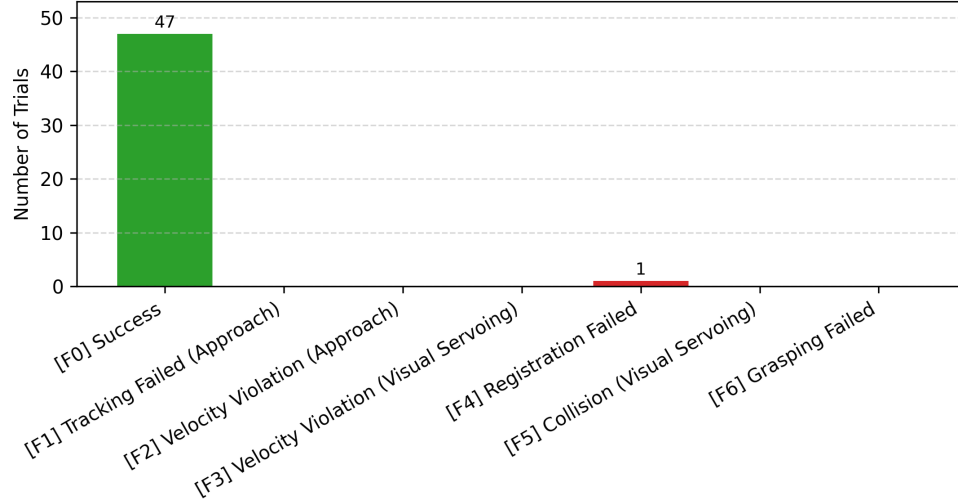


Figure 6.17: Set B: outcomes across 48 trials using 5G NSA at 24 GHz, categorized by success and failure reason

6.3.2.4 5G NSA at 3.5 GHz

Under the 5G NSA at 3.5 GHz network, the system successfully completed 46 out of 48 grasping trials (Fig. 6.18), achieving a 95.8 % success rate. This represents a significant improvement compared to Set A (Section 6.3.1.4), where the same network yielded only 46 % success. The average round-trip time was 84.09 ms with a standard deviation of 18.77 ms, slightly better than the 88.51 ± 14.74 ms observed in Set A. Although 3.5 GHz is a band commonly used in consumer devices, this result suggests that the network was either more resistant to interference or benefited from better service quality in this second test. A definitive claim cannot be made, as competing traffic was not directly measured. However, informal observation indicated that more users were present during Set B (around 16:00 on Tuesday) compared to Set A (around 19:00 on Friday).

The two failures in this batch were caused by [F1] tracking loss during approach and [F3] violation of velocity constraints during visual servoing. Both occurred at the initial position Left (1), which places the object close to the robot base. This region requires higher joint velocities to achieve the same Cartesian motion, making it more sensitive to timing delays. Since this network had the highest average latency among all the tested networks, such failures remain possible, although rare under the slower conveyor speed used in this case.

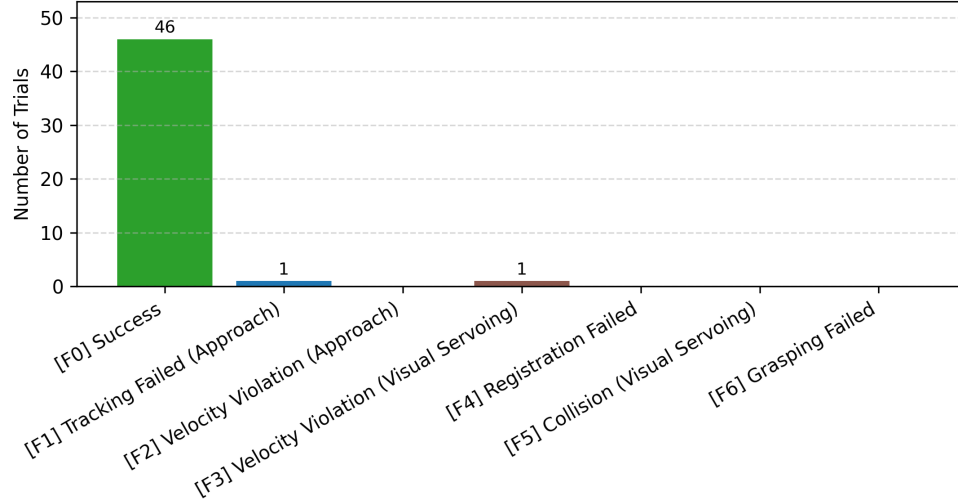


Figure 6.18: Set B: outcomes across 48 trials using 5G NSA at 3.5 GHz, categorized by success and failure reason

Compared to Set A, the overall improvement is evident. [F1] tracking loss during the approach phase dropped from five cases in Set A to just one in Set B. This reinforces our earlier observation that a lower conveyor speed improves temporal consistency between image frames, thereby increasing the robustness of the FoundationPose tracking.

Similarly, in Set A, the leading cause of failure was [F3] velocity constraint violations during visual servoing, accounting for 18 out of 27 failed trials. In this test, with the conveyor speed reduced, those constraints were no longer a limiting factor. Even with high network latency, the controller had sufficient time to execute the required motion, resulting in reliable and repeatable grasps.

6.3.2.5 Summary

All four networks tested in Set B successfully executed 6D pose-aware grasping. Despite apparent differences in latency and reliability, the system showed that, under moderate conveyor speed, it can achieve agile and repeatable grasping across a wide range of wireless conditions. Table 6.5 summarizes the grasp success rates, failure counts, average RTT, and server processing times for all four network types.

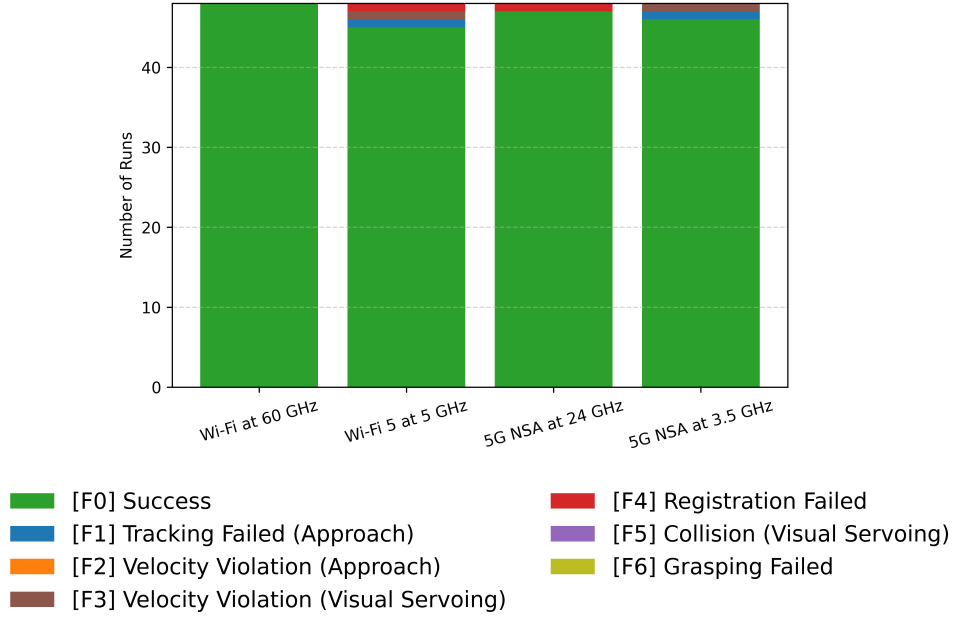


Figure 6.19: Outcome distribution for Set B experiments across four wireless network configurations. Each bar represents 48 trials, broken down by success and specific failure types.

Experiment	Success	Failures	Mean RTT [ms]	Mean Proc [ms]
Wi-Fi at 60 GHz	48	0	38.57 ± 3.75	19.12 ± 0.99
Wi-Fi 5 at 5 GHz	45	3	51.99 ± 65.07	19.40 ± 1.31
5G NSA at 24 GHz	47	1	61.03 ± 6.25	19.21 ± 1.03
5G NSA at 3.5 GHz	46	2	84.09 ± 18.77	19.39 ± 1.21

Table 6.5: Grasp success and failure counts, along with the average and standard deviation of RTT and server processing time for all four networks in Set B.

The Wi-Fi at 60 GHz and 5G NSA at 24 GHz networks provided the most stable communication, with minimal variability and nearly flawless performance. These results reflect the advantage of operating on less congested frequency bands that are rarely used by consumer devices. In contrast, the Wi-Fi 5 at 5 GHz and 5G NSA at 3.5 GHz networks exhibited higher variability and occasional failures, which were most often attributed to external interference or contention. Still, the success rates remained above 93 %, showing

that the system can operate robustly even under more constrained and unpredictable conditions.

Starting Position	Left (1)	Center (2)	Right (3)
Failures	5	1	0

Table 6.6: Number of failures observed for each initial object position in the Set B experiments.

A detailed breakdown of failures by initial object position is shown in Table 6.6. Most of them occurred at the initial position Left (1), where the object is located closer to the robot base. This configuration requires higher joint velocities to maintain the same Cartesian motion, which increases the chances of hitting velocity limits or missing tight timing margins. This pattern was consistent across networks, suggesting that future work could focus on enhancing the controller’s ability to handle challenging kinematic zones. Failures by initial orientation, presented in Table 6.7, were more evenly distributed, and

Starting Orientation	0	1	2	3	4	5	6	7
Failures	1	0	0	1	2	1	0	1

Table 6.7: Number of failures observed for each initial object orientation in the Set B experiments.

no consistent link was found between orientation and outcome. This indicates that all eight orientations tested were handled with comparable reliability, and failures were more likely due to other factors, such as tracking loss or latency spikes.

Overall, these results confirm that 6D pose-aware grasping can be performed reliably over wireless networks, provided the system is designed with awareness of wireless communication constraints and the task requirements are matched with the capabilities of both the network and the robot. This supports the feasibility of deploying robotic manipulation systems in real-world wireless environments, provided that network conditions are sufficiently understood and taken into account.

6.3.3 Results Summary

This section compared system performance across two experimental conditions: Set A, which emphasized speed and latency stress, and Set B, which emphasized task generality.

As expected, success rates were consistently higher in Set B, where slower conveyor speed relaxed timing constraints. However, Set B introduced more diverse object positions and orientations, showing that the system could generalize to a broader range of initial configurations without modification. This contrast illustrates how speed and generality trade off in real deployments and how system performance varies accordingly.

The absence or reduced frequency of specific failure modes in Set B reflects the system’s increased tolerance to delay under lower-speed conditions. [F1] Tracking loss was also less common, likely due to better temporal consistency between frames under slower motion. [F2] violations were absent since all pre-grasp poses remained reachable within the relaxed timing constraints. Failures such as [F3] (velocity constraint violation during visual servoing), [F5] (collision), and [F6] (end-effector delay) were less frequent or did not occur. These trends demonstrate how task speed significantly impacts the system’s vulnerability to both network delays and control limitations.

Finally, the experiments provide important insights about the network technologies themselves. Wi-Fi at 60 GHz consistently offered the best combination of low latency and stable throughput. Wi-Fi 5 at 5 GHz had similarly low latency but exhibited high variability, making it a potential risk factor for closed-loop control systems. 5G NSA at 24 GHz showed moderate latency but superior stability, suggesting it could be a strong candidate for reliable wireless deployment. In contrast, more stable than Wi-Fi 5 at 5 GHz but suffered from high average latency, which limited its suitability for time-critical tasks.

FoundationPose performed robustly across all conditions. Initial registration failed only 3 times out of 392 trials, and tracking loss occurred in 15 trials despite rapid robot motion and frequent object pose changes. This indicates that the model is sufficiently stable for use in dynamic real-world applications.

Overall, these findings demonstrate that wireless networks can support real-time robotic control with offboard computation, provided that certain conditions are met. Reliable performance depends on aligning task requirements with both the network’s capabilities and the control logic’s design. The outcomes of Set A and Set B offer practical insights into how latency, stability, and task structure interact and can help inform future system design choices in real-world deployments.

Chapter 7

Conclusion

This thesis addressed the challenge of performing agile and reliable 6D pose-aware grasping in dynamic industrial environments using wireless communication. The key difficulty lies in closing the visual feedback loop when perception is offloaded to a remote server over potentially unstable or high-latency links. In such scenarios, robotic actions must still be responsive and accurate despite delayed and asynchronous observations. This is particularly demanding in real-time manipulation tasks involving fast-moving objects, where precise timing and alignment are crucial. Developing such systems is critical for enabling scalable, flexible, and cloud-integrated robotic automation in future smart factories.

To investigate this challenge, a complete robotic manipulation system was developed to simulate a realistic industrial use case. The system combines 6D pose estimation using FoundationPose, which runs remotely on a GPU server, with a high-frequency local controller that utilizes visual feedback to execute closed-loop grasping of moving objects on a conveyor belt. Two test scenarios were designed: one with a fast-moving conveyor to expose the impact of latency and test system limits, and another with moderate speed to validate system performance and assess reliability. This setup enabled us to systematically evaluate performance across four wireless technologies: Wi-Fi at 60 GHz, Wi-Fi 5 at 5 GHz, 5G NSA at 24 GHz, and 5G NSA at 3.5 GHz.

The results of this work demonstrate that it is possible to close a visual servoing loop over a wireless network using remote computer vision, provided that network latency and variability remain within acceptable bounds. However, the success and stability of such a system are highly dependent on the characteristics of the wireless link. In our experiments, increased latency directly limited the maximum feasible conveyor speed. Moreover, more stable networks consistently enabled better control performance, highlighting the impor-

tance of communication reliability for closed-loop robotic systems. The experiments also offer practical insights into the wireless technologies themselves. Wi-Fi at 60 GHz consistently delivered the best combination of low latency and stable throughput. Wi-Fi 5 at 5 GHz offered low latency but exhibited high variability, making it less suitable for consistent closed-loop control. 5G NSA at 24 GHz provided moderate latency with excellent stability, suggesting strong potential for wireless robotic deployment. In contrast, 5G NSA at 3.5 GHz, although stable, suffered from high latency, which limited its suitability for time-critical tasks.

FoundationPose performed reliably across all network conditions, with only 3 registration failures and 15 tracking losses across 392 trials. However, its long initialization time and lack of built-in confidence metrics present challenges for use in unstructured or production environments.

Several limitations of the current system arise from both the perception pipeline and the control architecture. On the perception side, the pose estimation model requires a segmentation mask, which can be time-consuming to generate in general settings. To simplify integration, a color-mask-based segmentation scheme tailored to a single object type was used, which limits generalizability. Additionally, the model’s registration phase is relatively slow, requiring compensation for object motion during initialization. To enable this, the conveyor belt speed is hardcoded, introducing a dependency that may not generalize to variable-speed systems.

On the control side, the system did not account for joint-limit constraints during visual servoing or reachability during pre-grasp pose selection. Because the conveyor speed was already known, it was also used to compute the pre-grasp pose along a fixed approach trajectory.

The experiments were conducted with the object (a bottle) placed in an upright orientation, which effectively reduced the problem from full 6D to 4D pose estimation. Neither the pose estimation pipeline nor the controller logic relied on this assumption. It was introduced to ensure that all tested orientations were feasible across all initial positions, allowing every orientation and position pair to form a valid test case. The grasp transformation was explicitly configured for the upright pose, which limited the range of configurations the system could handle.

Another simplification was that the object moved along a conveyor belt in a straight line at constant speed. This assumption made it easier to compute the interception point during the approach phase and to compensate for the object’s motion during registration.

Finally, while the laboratory setup was designed to mimic industrial environments, it cannot fully capture all factors present in a factory setting. These constraints and

simplifications supported the development of a complete end-to-end pipeline. However, the core component of interest, visual servoing with remote perception, was not dependent on these assumptions and therefore remains valid under more general conditions.

Future work can begin by improving the control pipeline to address limitations observed in the current system, such as velocity constraint violations during the approach phase. The two-phase structure could be replaced with a unified controller capable of planning and adapting motion in real time while accounting for joint limits and task constraints. This could involve model predictive control or learning-based methods. Such a system would be better equipped to handle nonlinear object motion and generate safer, more responsive behavior. A dedicated grasping module could also be incorporated to predict optimal grasp poses across a wider range of object orientations, removing the current assumption of upright alignment. These extensions would increase the system’s flexibility and enable deployment in less structured environments. On the perception side, a more complex object detection pipeline could be integrated to support multiple object classes in diverse settings. In addition, newer pose estimation models could be incorporated to provide faster and more robust performance without relying on segmentation masks or strong temporal consistency between frames. These improvements would make the system more suitable for real-world applications, where conditions are less controlled and unpredictable.

Overall, this work demonstrates the feasibility of closing a visual servoing loop over a wireless network using remote perception for real-time robotic manipulation. While practical deployment in production environments still requires both lower-latency and more reliable communication, as well as more intelligent local control strategies and faster, more generalizing models, the results presented here confirm that such systems are not only viable but also promising. As network, control, and perception technologies continue to advance, cloud-augmented robotic architectures are expected to play an increasingly central role in the future of industrial automation.

References

- [1] 6d pose estimation using rgb. <https://paperswithcode.com/task/6d-pose-estimation>, 2024. Accessed: 2025-06-27.
- [2] Farhad Aghili. Fault-tolerant and adaptive visual servoing for capturing moving objects. *IEEE/ASME Transactions on Mechatronics*, 27(3):1773–1783, 2022.
- [3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: speeded up robust features. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV’06, page 404–417, Berlin, Heidelberg, 2006. Springer-Verlag.
- [4] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.
- [5] Tolga Birdal and Slobodan Ilic. Point pair features based object detection and pose estimation revisited. In *2015 International Conference on 3D Vision*, pages 527–535, 2015.
- [6] Matteo Bortolon, Theodore Tsesmelis, Stuart James, Fabio Poiesi, and Alessio Del Bue. Iffnerf: Initialisation free and fast 6dof pose estimation from a single image and a nerf model, 2024.
- [7] Boyuan Cao, Xinyu Zhou, Congmin Guo, Baohua Zhang, Yuchen Liu, and Qianqiu Tan. Nbmod: Find it and grasp it in noisy background, 2023.
- [8] Gianluca Cena, Stefano Scanzio, and Adriano Valenzano. Seamless link-level redundancy to improve reliability of industrial wi-fi networks. *IEEE Transactions on Industrial Informatics*, 12(2):608–620, 2016.
- [9] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Danny Driess, Pete Florence, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities, 2024.

- [10] Kaiyuan Chen, Kush Hari, Trinity Chung, Michael Wang, Nan Tian, Christian Juetten, Jeffrey Ichnowski, Liu Ren, John Kubiatoicz, Ion Stoica, and Ken Goldberg. Fogros2-ft: Fault tolerant cloud robotics, 2024.
- [11] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots, 2024.
- [12] Pietro Chiavassa, Stefano Scanzio, and Gianluca Cena. On the suitability of wi-fi for interconnecting moving equipment in industrial environments, 2025.
- [13] Xining Cui, Menghui Yu, Linqigao Wu, and Shiqian Wu. A 6d pose estimation for robotic bin-picking using point-pair features with curvature (cur-ppf). *Sensors*, 22:1805, 02 2022.
- [14] Ramraj Dangi, Praveen Lalwani, Gaurav Choudhary, Ilsun You, and Giovanni Pau. Study and investigation on 5g technology: A systematic review. *Sensors*, 22(1), 2022.
- [15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
- [16] Bertram Drost and Slobodan Ilic. 3d object detection and localization using multi-modal point pair features. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 9–16, 2012.
- [17] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. pages 998–1005, 07 2010.
- [18] Mark Eisen, Santosh Shukla, Dave Cavalcanti, and Amit S. Baxi. Communication-control co-design in wireless edge industrial systems, 2022.
- [19] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
- [20] Mederic Fourmy, Vojtech Priban, Jan Kristof Behrens, Nicolas Mansard, Josef Sivic, and Vladimir Petrik. Visually guided model predictive robot control via 6d object pose localization and tracking, 2023.

- [21] David Ginthoer and Henrik Klessig. Robot motion control offloading in 5g network using trajectory interpolation. In *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, pages 1–6, 2023.
- [22] Javier Gonzlez Huarte and Aitor Ibarguren. Visual servoing architecture of mobile manipulators for precise industrial operations on moving objects. *Robotics*, 13(5), 2024.
- [23] Andreas Ingo Grohmann, Christopher Lehmann, Thomas Höschele, and Frank H.P. Fitzek. Bakeryrobot: 5g connected robot for smes. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, pages 925–926, 2022.
- [24] Jian Guan, Yingming Hao, Qingxiao Wu, Sicong Li, and Yingjian Fang. A survey of 6dof object pose estimation methods for different application scenarios. *Sensors*, 24(4), 2024.
- [25] R.M. Haralick, D. Lee, K. Ottenburg, and M. Nolle. Analysis and solutions of the three point perspective pose estimation problem. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 592–598, 1991.
- [26] Jesse Haviland, Feras Dayoub, and Peter Corke. Control of the final-phase of closed-loop visual grasping using image-based visual servoing, 2020.
- [27] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 International Conference on Computer Vision*, pages 858–865, 2011.
- [28] Lihe Hu, Yi Zhang, Yang Wang, Gengyu Ge, and Wei Wang. Salient preprocessing: Robotic icp pose estimation based on sift features. *Machines*, 11:157, 01 2023.
- [29] Jingshun Huang, Haitao Lin, Tianyu Wang, Yanwei Fu, Yu-Gang Jiang, and Xiangyang Xue. You only estimate once: Unified, one-stage, real-time category-level articulated object 6d pose estimation for robotic grasping, 2025.
- [30] Yiqi Huang, Travis Davies, Jiahuan Yan, Jiankai Sun, Xiang Chen, and Luhui Hu. Spatial robograsp: Generalized robotic grasping control policy, 2025.
- [31] Yiqi Huang, Travis Davies, Jiahuan Yan, Jiankai Sun, Xiang Chen, and Luhui Hu. Spatial robograsp: Generalized robotic grasping control policy, 2025.

- [32] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015.
- [33] Eunyoung Kim and Gerard Medioni. 3d object recognition in range images using visibility context. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3800–3807, 2011.
- [34] Hyung Joo Lee, Ajith Krishnan, and Sigrid Brell-Cokcan. 5g in construction: from deployment to evaluation for robotic applications. *Construction Robotics*, 8, 04 2024.
- [35] Kang B. Lee, Rick Candell, Hans-Peter Bernhard, Dave Cavalcanti, Zhibo Pang, and Inaki Val. Reliable, high-performance wireless systems for factory automation, 2020-09-17 04:09:00 2020.
- [36] Jiehong Lin, Lihua Liu, Dekun Lu, and Kui Jia. Sam-6d: Segment anything model meets zero-shot 6d object pose estimation. *arXiv preprint arXiv:2311.15707*, 2023.
- [37] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.
- [38] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. Available at <http://modernrobotics.org>.
- [39] Haofei Ma, Gongcheng Wang, Hua Bai, Zhiyu Xia, Weidong Wang, and Zhijiang Du. Robotic grasping method with 6d pose estimation and point cloud fusion. *The International Journal of Advanced Manufacturing Technology*, 134:5603–5613, 09 2024.
- [40] Debapriya Maji, Soyeb Nagori, Manu Mathew, and Deepak Poddar. Yolo-6d-pose: Enhancing yolo for single-stage monocular multi-object 6d pose estimation. In *2024 International Conference on 3D Vision (3DV)*, pages 1616–1625, 2024.
- [41] Sulata Mitra and Mosa Ali Abu-Rgheff. Quality of service (qos) issues in multimedia wireless network: a survey. *J. Mob. Multimed.*, 5(3):181–202, September 2009.
- [42] Andrew Morgan*, Bowen Wen*, Junchi Liang, Abdeslam Boularias, Aaron Dollar, and Kostas Bekris. Vision-driven compliant manipulation for reliable; high-precision assembly tasks. In *Robotics: Science and Systems XVII*, RSS2021. Robotics: Science and Systems Foundation, July 2021.

- [43] Douglas Morrison, Peter Corke, and Jürgen Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach, 2018.
- [44] Huy Hoang Nguyen, Minh Nhat Vu, Florian Beck, Gerald Ebmer, Anh Nguyen, and Andreas Kugi. Language-driven closed-loop grasping with model-predictive trajectory replanning, 2024.
- [45] Mahsa Noroozi and Lorenz Kies. Performance analysis of universal robot control system using networked predictive control. In *2022 7th International Conference on Robotics and Automation Engineering (ICRAE)*, pages 227–232, 2022.
- [46] Kostas Ordoumpozanis and George A Papakostas. Reviewing 6d pose estimation: Model strengths, limitations, and application fields. *Applied Sciences*, 15(6), 2025.
- [47] Soran Parsa, Bappaditya Debnath, Muhammad Arshad Khan, and Amir Ghalamzan E. Modular autonomous strawberry picking robotic system. *Journal of Field Robotics*, 41(7):2226–2246, 2024.
- [48] Leo Thomas Ramos and Angel D. Sappa. A decade of you only look once (yolo) for object detection, 2025.
- [49] ElHussein Shata, Baihui Chen, Liwen Hu, Ivan Seskar, Yuebin Guo, Charif Mahmoudi, Shashank Shekhar, and Qingze Zou. 5g-cloud-based real-time robotic part repairing for advanced manufacturing via computer vision. *Manufacturing Letters*, 41:1398–1404, 2024. 52nd SME North American Manufacturing Research Conference (NAMRC 52).
- [50] Sebastian Stapf, Tobias Bauernfeind, and Marco Riboldi. Pvit-6d: Overclocking vision transformers for 6d pose estimation with confidence-level prediction and pose tokens, 2023.
- [51] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction, 2018.
- [52] Mehdi Testouri, Gamal Elghazaly, Faisal Hawlader, and Raphael Frank. 5g-enabled teleoperated driving: An experimental evaluation, 2025.
- [53] Jiafu Wan, Shenglong Tang, Hehua Yan, Di Li, Shiyong Wang, and Athanasios Vasilakos. Cloud robotics: Current status and open issues. *IEEE Access*, 4:1–1, 01 2016.
- [54] An-Lan Wang, Nuo Chen, Kun-Yu Lin, Li Yuan-Ming, and Wei-Shi Zheng. Task-oriented 6-dof grasp pose detection in clutters, 2025.

- [55] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection, 2024.
- [56] Yulin Wang, Yizeng Han, Chaofei Wang, Shiji Song, Qi Tian, and Gao Huang. Computation-efficient deep learning for computer vision: A survey, 2023.
- [57] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects, 2024.
- [58] Wikipedia contributors. Computer network — wikipedia, the free encyclopedia, 2024. Accessed: 2025-06-27.
- [59] Rosa Wolf, Yitian Shi, Sheng Liu, and Rania Rayyes. Diffusion models for robotic manipulation: A survey, 2025.
- [60] Haiyan Wu, Thomas Timm Andersen, Nils Axel Andersen, and Ole Ravn. Application of visual servoing for grasping and placing operation in slaughterhouse. In *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pages 457–462, 2017.
- [61] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *CoRR*, abs/1711.00199, 2017.
- [62] Chen Yuanwei, Mohd Hairi Mohd Zaman, and Mohd Faisal Ibrahim. A review on six degrees of freedom (6d) pose estimation for robotic applications. *IEEE Access*, 12:161002–161017, 2024.
- [63] Usman A. Zahidi, Arshad Khan, Tsvetan Zhivkov, Johann Dichtl, Dom Li, Soran Parsa, Marc Hanheide, Grzegorz Cielniak, Elizabeth I. Sklar, Simon Pearson, and Amir Ghalamzan-E. Optimising robotic operation speed with edge computing via 5g network: Insights from selective harvesting robots. *Journal of Field Robotics*, 41(8):2771–2789, 2024.
- [64] Kouros Zambouri, Md. Noor-A-Rahim, Jobish John, Cormac J. Sreenan, H. Vincent Poor, and Dirk Pesch. A comprehensive survey of wireless time-sensitive networking (tsn): Architecture, technologies, applications, and open issues. *IEEE Communications Surveys Tutorials*, pages 1–1, 2024.
- [65] Qinglei Zhang, Cuige Xue, Jiyun Qin, Jianguo Duan, and Ying Zhou. 6d pose estimation of industrial parts based on point cloud geometric information prediction for robotic grasping. *Entropy*, 26(12), 2024.

- [66] Chungang Zhuang, Shaofei Li, and Han Ding. Instance segmentation based 6d pose estimation of industrial objects using point clouds for robotic bin-picking. *Robotics and Computer-Integrated Manufacturing*, 82:102541, 2023.
-