



FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS ENGINEERING DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Jaqueline de Souza Ribeiro

**Marker-Triggered Pick-and-Place for CNC Machine Tending Using an
ArUco-Based Vision Interface in ROS**

Florianópolis
2026

Jaqueline de Souza Ribeiro

**Marker-Triggered Pick-and-Place for CNC Machine Tending Using an
ArUco-Based Vision Interface in ROS**

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Henrique Simas, Dr.
Co-supervisor: Lucas Manassés, Eng.

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

<http://portalbu.ufsc.br/ficha>

Jaqueline de Souza Ribeiro

**Marker-Triggered Pick-and-Place for CNC Machine Tending Using an
ArUco-Based Vision Interface in ROS**

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, July 8th, 2026.

Prof. Marcelo de Lellis Costa de Oliveira, Dr.
Course Coordinator

Examining Board:

Prof. Henrique Simas, Dr. Eng.
Advisor

UFSC/CTC/EMC

Lucas Manassés
Pinheiro de Souza

 Digital unterschrieben von Lucas
Manassés Pinheiro de Souza
Datum: 2026.07.17 09:54:56 +02'00'

Lucas Manassés Pinheiro de Souza, MSc.
Supervisor

Laboratory for Machine Tools and Production Engineering (WZL) of RWTH Aachen
University

Gabriel Thaler, MSc.
Evaluator
UFSC/CTC/EAS

Prof. Hector Bessa Silveira, Dr. Eng
Board President
UFSC/CTC/EAS

This work is dedicated to to the entire EAS department,
my dear parents, and my sister.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my parents, Jackson and Lucilene. Since long before I could comprehend the world, they have been my greatest advocates, nurturing my identity and supporting every one of my endeavors. I owe every opportunity life has presented to me to their unwavering devotion; if I have had the privilege of stepping through the doors that opened before me, it was they who turned every handle.

I am profoundly grateful to my sister, Paula, for her steadfast support, even when my choices were difficult to grasp. She embraced me with maternal care at the start of my academic journey and has remained a constant presence by my side, regardless of time or distance.

To my friends who offered a helping hand during challenging times, who patiently listened to my frustrations, and who shared in the laughter of lighter moments: thank you. I am also grateful to the colleagues who walked this academic path with me, after all, no one graduates alone. To those who have crossed my path and, for reasons of fate, are no longer present, I offer my thanks, for I would not be who I am today without the impact they left upon my life.

I extend my heartfelt thanks to my supervisor, Lucas Manassés. Without his dedication, this final project would not have been possible. Thank you for agreeing to mentor me despite the frantic pace, the limited time, and the many obstacles we encountered. I am truly grateful for his guidance, his belief in my potential, and for being such a supportive partner during our time in the cold lands of Germany.

I would also like to thank my academic advisor, Prof. Henrique Simas, for his understanding, his encouraging words, and his unwavering support throughout this non-linear journey. Special thanks are due to all the professors and staff of the Department of Automation and Systems Engineering at UFSC for their support throughout my time at the university.

Last, but certainly not least, I offer a special acknowledgment to myself. We are the only ones who truly know the difficulty of living inside our own heads. Thank you, Jaqueline, for not giving up herself, for overcoming every crisis, for enduring what seemed unbearable, and for holding onto the deep-seated belief that she would prevail.

"Those who pass by us do not go alone. They leave a little of themselves and take a little of us." — Antoine de Saint-Exupéry

*“If one’s aim is to design something useful,
one should avoid designing something
useless because unused.”
(DIJKSTRA, 1974)*

DISCLAIMER

Aachen, July 1st, 2026.

As representative of the Laboratory for Machine Tools and Production Engineering (WZL) of RWTH Aachen University in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation (“*Trabalho de Conclusão de Curso*”), in accordance with the *Resolução Normativa n° 126/2019/CUn*.

Lucas Manassés
Pinheiro de Souza

Digital unterschrieben von Lucas
Manassés Pinheiro de Souza
Datum: 2026.07.17 09:55:22
+02'00'

Lucas Manassés Pinheiro de Souza
Laboratory for Machine Tools and Production Engineering (WZL) of RWTH Aachen
University

Abstract

This work presents the development and validation of a deterministic software architecture for automated Machine Tending, enabling a repurposed assistive manipulator to perform high-precision Computer Numerical Control (CNC) loading and unloading. The research addresses the challenge of adapting legacy robotic systems for flexible manufacturing environments, where dynamic 6-Degree-of-Freedom (DoF) pose-based grasping often fails due to hand-eye calibration sensitivities, sensor depth noise on reflective metallic surfaces, and coordinate transformation jitter. To bypass these perceptual vulnerabilities, the proposed solution implements an ID-to-Recipe control framework that relies on discrete topological identification rather than continuous spatial tracking. The implementation integrates a 6-DoF Kinova Jaco2 manipulator and an Intel RealSense L515 depth camera within the Robot Operating System (ROS 1) Noetic middleware. The visual perception pipeline utilizes ArUco Markers exclusively to extract integer identifiers, which act as triggers to query a modular YAML database. This database maps each task to a specific sequence of pre-recorded, collision-free joint-space waypoints and gripper commands. By routing Trajectory Execution directly to the low-level hardware drivers and avoiding online sampling-based planners, the system circumvents mathematical singularities and computational latency. The methodology includes strict trajectory shaping with intermediate waypoints and explicit temporal dwell times to dissipate kinetic oscillations before grasping. Experimental validation demonstrated that the architecture achieves safe, collision-free, and highly repeatable short-term execution within the tight mechanical tolerances of a CNC workcell. Although the open-loop nature of the system introduces susceptibility to spatial drift over extended continuous cycles, the approach successfully neutralizes the high-frequency Cartesian jitter that typically plagues vision-guided robotics. The results validate the Technological Retrofit of legacy assistive hardware for advanced manufacturing research, providing a sustainable, vendor-independent, and easily reconfigurable blueprint for high-mix, low-volume production lines.

Keywords: Machine Tending. ROS 1. ArUco Markers. Trajectory Execution. Technological Retrofit.

RESUMO

Este trabalho apresenta o desenvolvimento e a validação de uma arquitetura de software determinística para o Abastecimento de Máquinas automatizado, permitindo que um manipulador assistivo reaproveitado realize operações de carga e descarga de alta precisão em tornos de Controle Numérico Computadorizado (CNC). A pesquisa aborda o desafio de adaptar sistemas robóticos legados para ambientes de manufatura flexível, onde o controle dinâmico baseado em pose de 6 Graus de Liberdade (DoF) frequentemente falha devido a sensibilidades na calibração olho-mão, ruído de profundidade do sensor em superfícies metálicas reflexivas e instabilidade nas transformações de coordenadas. Para contornar essas vulnerabilidades perceptivas, a solução proposta implementa uma estrutura de controle baseada na conversão de identificadores para receitas, que depende da identificação topológica discreta em vez do rastreamento espacial contínuo. A implementação integra um manipulador Kinova Jaco2 de 6 DoF e uma câmera de profundidade Intel RealSense L515 por meio do middleware Robot Operating System (ROS 1) Noetic. O pipeline de percepção visual utiliza Marcadores ArUco exclusivamente para extrair identificadores inteiros, que atuam como gatilhos para consultar um banco de dados YAML modular. Este banco de dados mapeia cada tarefa para uma sequência específica de pontos de passagem pré-gravados no espaço das juntas e comandos de garra, garantindo rotas livres de colisões. Ao rotear a Execução de Trajetórias diretamente para os drivers de hardware de baixo nível e evitar planejadores baseados em amostragem em tempo real, o sistema evita singularidades matemáticas e latência computacional. A metodologia inclui a modelagem rigorosa do caminho com pontos intermediários e tempos de espera explícitos para dissipar oscilações cinéticas antes da preensão. A validação experimental demonstrou que a arquitetura atinge uma execução de curto prazo segura e altamente repetível dentro das restritas tolerâncias mecânicas de uma célula de trabalho CNC. Embora a natureza de malha aberta do sistema introduza suscetibilidade ao desvio espacial ao longo de ciclos contínuos prolongados, a abordagem neutraliza com sucesso a instabilidade cartesiana de alta frequência que tipicamente afeta a robótica guiada por visão. Os resultados validam o Retrofit Tecnológico de hardware assistivo legado para pesquisa avançada em manufatura, fornecendo um modelo sustentável, independente de fornecedor e facilmente reconfigurável para linhas de produção de alta variação e baixo volume.

Palavras-chave: Abastecimento de Máquinas. ROS 1. Marcadores ArUco. Execução de Trajetórias. Retrofit Tecnológico.

LIST OF FIGURES

Figure 1 – Kinematic diagram and DH frames for Kinova Jaco2.	27
Figure 2 – Standard ArUco detection pipeline: from binarization to pose estimation.	30
Figure 3 – Perception jitter effect: sensor noise induces subpixel uncertainty in marker vertex detection.	31
Figure 4 – Uncertainty propagation: 2D subpixel noise yields a 3D ellipsoid elongated along the depth axis.	32
Figure 5 – TF architecture connecting the robot base, camera, and TCP.	34
Figure 6 – System architecture: decoupled sensor pipelines and MoveIt/RViz monitoring.	35
Figure 7 – RViz diagnostic layer: RealSense L515 ArUco feed (left) and digital twin configuration (right).	36
Figure 8 – Control architectures: open-loop predictive vs. closed-loop feedback.	37
Figure 9 – Hardware-constrained robotic retrofit: mapping legacy assets to open-source middleware.	38
Figure 10 – Comparison: (a) manual clamping fixtures vs. (b) vision-guided Jaco2 cell.	41
Figure 11 – Overview of the four core research chairs at WZL, RWTH Aachen University.	43
Figure 12 – Continuous 6D pose estimation vs. deterministic ID-to-Recipe pipeline.	45
Figure 13 – Open-loop spatial decay: error accumulation across pick-and-place cycles.	45
Figure 14 – ROS TF Coordinate Frame inheritance tree detailing the static transformation mapping.	47
Figure 15 – TCP tolerance envelope and angular constraints for open-loop insertion.	48
Figure 16 – Original assistive wheelchair configuration of the Kinova Jaco2.	49
Figure 17 – Retrofitted CNC tending cell at WZL using a vision-guided Jaco2.	50
Figure 18 – Hardware topology: USB 3.0, power, and peripherals linking sensors, manipulator, and workstation.	51
Figure 19 – ROS software architecture: the sensor-to-actuator pipeline.	54
Figure 20 – Kinematic chain (Eq. 11): red cloud denotes propagated Cartesian jitter.	57
Figure 21 – Data extraction: from RealSense RGB to binarized grid and canonical bit matrix.	59
Figure 22 – Step response: <code>wait_after</code> dwell time allows residual oscillations to settle before tool actuation.	61
Figure 23 – Resolving IK multiplicity: absolute joint angles prevent unpredictable configuration flips inside the confined CNC.	63

Figure 24 – Trajectory divergence: unconstrained joint interpolation yields collision-prone Cartesian arcs.	64
Figure 25 – Proprioceptive joint closed-loop vs. exteroceptive task-space open-loop.	67
Figure 26 – DFMA-based passive compliance: chamfers mitigate open-loop placement drift.	68
Figure 27 – Underactuated enveloping grasp vs. precision pinch for TCP centrality.	69
Figure 28 – Coulomb friction drop ($\mu_s \rightarrow \mu_k$): residual TCP oscillations induce workpiece slippage.	70
Figure 29 – ID-to-Recipe pipeline: mapping ROS nodes to Python scripts.	71
Figure 30 – Metric PnP ($SE(3)$ jitter) vs. topological classification (Z^+ extraction).	72
Figure 31 – Separation of Concerns: static Python engine decoupled from YAML task definitions.	73
Figure 32 – Waypoint tuple $\rightarrow$ YAML parameter mapping per operational step. .	74
Figure 33 – Probabilistic paths (RRT/PRM) vs. deterministic joint-space interpolation.	75
Figure 34 – Polynomial joint-space interpolation: $\mathcal{C}^1$ and $\mathcal{C}^2$ continuity ensures smooth actuator transitions.	76
Figure 35 – Abbe Error: angular inaccuracy ($\Delta\theta$) causing magnified TCP displacement (Δx).	79
Figure 36 – Specular reflection artifacts: infrared scattering causing depth "holes" and pose instability.	80
Figure 37 – Data flow diagram of the ID-to-Recipe execution pipeline.	81
Figure 38 – Tracking error accumulation ($e(t)$): protective halt triggered upon exceeding safety thresholds.	84
Figure 39 – Average cycle time and success rate for pick-and-place tasks, illustrating high temporal repeatability.	85
Figure 40 – Deterministic YAML execution ($O(1)$): eliminating latency jitter to ensure consistent industrial OEE.	85
Figure 41 – FSM logic: 'Home' configuration as a zero-velocity boundary attractor.	86
Figure 42 – Consecutive grasp cycles: progression of lateral drift ($a \rightarrow b$) leading to failure (c).	87
Figure 43 – Stochastic drift accumulation: variance expansion ($\sqrt{n}$) violating gripper tolerances.	87
Figure 44 – Hierarchical control: decoupled probabilistic perception from deterministic joint-space execution.	88
Figure 45 – Collision-free swept volume: deterministic joint-space routing ensures workspace safety.	89

Figure 46 – Admittance control in peg-in-hole insertion: compliant resolution of contact forces. 90

Figure 47 – Proposed IBVS loop: robust 3D pose estimation via 2D feature space minimization. 91

LIST OF TABLES

Table 1 – Complete Hardware and Software Specifications	51
Table 2 – Summary of System Requirements for the CNC Machine Tending Cell.	53
Table 3 – Architectural comparison between Continuous 6D Pose Estimation and the Proposed ID-to-Recipe Strategy.	58
Table 4 – ROS Core Interfaces utilized by the Execution Node for state monitor- ing and motor control.	63
Table 5 – Functional anatomy of the deterministic waypoint sequence used to mimic Cartesian constraints and prevent kinematic collisions.	65
Table 6 – Task performance: high repeatability in cycle time and success rate. .	84
Table 7 – Index of supplementary multimedia materials.	103
Table 8 – Third-party dependencies utilized in this research.	126

List of Abbreviations

API	Application Programming Interface
CNC	Computer Numerical Control
CMOS	Complementary Metal-Oxide-Semiconductor
DoF	Degrees of Freedom
FCL	Flexible Collision Library
FSM	Finite State Machine
HMLV	High-Mix, Low-Volume
ID	Identification
IK	Inverse Kinematics
LiDAR	Light Detection and Ranging
OEE	Overall Equipment Effectiveness
OMPL	Open Motion Planning Library
PFC	Projeto de Fim de Curso
PID	Proportional-Integral-Derivative
PnP	Perspective-n-Point
PRM	Probabilistic Roadmaps
RMS	Reconfigurable Manufacturing Systems
ROS	Robot Operating System
RRT	Rapidly-exploring Random Trees
TCP	Tool Center Point
TF	Transform
UFSC	Federal University of Santa Catarina
URDF	Unified Robot Description Format
VGR	Vision-Guided Robotic

WZL	Werkzeugmaschinenlabor (Laboratory for Machine Tools and Production Engineering)
YAML	YAML Ain't Markup Language

CONTENTS

1	INTRODUCTION	19
1.1	BACKGROUND AND CONTEXT	19
1.2	MOTIVATION	19
1.3	PROBLEM STATEMENT	20
1.4	RESEARCH OBJECTIVES AND RESEARCH QUESTIONS	21
1.5	SCOPE AND LIMITATIONS	22
1.6	SIGNIFICANCE OF THE WORK	23
1.7	STRUCTURE OF THE DOCUMENT	23
2	THEORETICAL FOUNDATION	25
2.1	ROBOTIC MANIPULATION	25
2.1.1	Kinova Jaco2: Degrees of Freedom and Kinematics	25
2.1.2	Joint-Space vs. Cartesian-Space Execution	27
2.2	COMPUTER VISION IN INDUSTRIAL AUTOMATION	29
2.2.1	ArUco Markers: Detection and Identification (ID)	29
2.2.2	Pose Estimation Challenges (Jitter and Noise)	30
2.3	SOFTWARE FRAMEWORKS	32
2.3.1	ROS Noetic and Coordinate Transformations (TF)	33
2.3.2	RealSense L515 Integration and MoveIt/RViz Ecosystem	34
2.4	INDUSTRIAL CONTROL AND AUTOMATION	36
2.4.1	Open-Loop vs. Closed-Loop Control and Error Accumulation	36
2.4.2	Technological Retrofit of Legacy Robotic Hardware	38
3	PROBLEM DESCRIPTION AND TECHNICAL REQUIREMENTS	40
3.1	CNC WORKCELL CONTEXT AND MOTIVATION	40
3.2	INSTITUTIONAL CONTEXT	42
3.3	PROBLEM DESCRIPTION AND THE OPEN-LOOP DRIFT CHALLENGE	43
3.4	TECHNICAL CONSTRAINTS	45
3.4.1	Perception Pipeline Sensitivity	46
3.4.2	Repeatability and Tolerance Demands	47
3.5	HARDWARE AND ENVIRONMENT SETUP	48
3.6	TECHNICAL REQUIREMENTS	52
3.6.1	Functional Requirements (FR)	53
3.6.2	Non-Functional Requirements (NFR)	55
4	PROPOSED SOLUTION AND METHODOLOGY	56
4.1	SYSTEM ARCHITECTURE	56
4.1.1	Data Flow: From ArUco ID to Joint Execution	58
4.1.2	Modular YAML Recipe System	60

4.2	WAYPOINT AUTHORIZING AND PATH SHAPING	62
4.2.1	Manual Joint Configuration Recording	62
4.2.2	Intermediate Waypoints: Approach, Pre-grasp, and Retreat . . .	64
4.3	RECIPE EXECUTION PIPELINE	65
4.3.1	Direct Joint Command Interface	66
4.3.2	Gripper and Finger Control Integration	68
5	SYSTEM IMPLEMENTATION	71
5.1	DEVELOPED COMPONENTS AND DELIVERABLES	71
5.1.1	Perception and Topological Classification	71
5.1.2	Decision Logic and the YAML Recipe System	73
5.1.3	Deterministic Kinematic Execution	75
5.2	TROUBLESHOOTING AND ENCOUNTERED ISSUES	77
5.2.1	Software and Configuration Fixes	77
5.2.2	Transition from Pose-Based to ID-Driven Workflow	78
5.2.3	Execution Workflow	80
6	RESULTS AND EXPERIMENTAL VALIDATION	83
6.1	EVALUATED SCENARIOS AND EXPERIMENTAL SETUP	83
6.2	QUANTITATIVE PERFORMANCE AND REPEATABILITY	84
6.3	OPEN-LOOP DRIFT AND FAILURE ANALYSIS	86
6.4	CURRENT VALIDATED CAPABILITIES	88
6.5	PROPOSED SOLUTIONS FOR FUTURE WORK	89
7	CONCLUSION	92
7.1	SUMMARY OF WORK	92
7.2	RESEARCH OBJECTIVES AND LIMITATIONS	93
7.3	TECHNICAL LESSONS LEARNED	94
7.4	IMPACTS OF THE RESEARCH	96
7.5	FUTURE WORK	97
	References	98
	APPENDIX A – SUPPLEMENTARY MULTIMEDIA CONTENT . . .	103
A.1	DIAGNOSTIC NOTES ON KINEMATIC LIMITATIONS	103
	APPENDIX B – SOFTWARE IMPLEMENTATION AND CONFIGU- RATION	104
B.1	CONTROL ORCHESTRATOR: MARKER_TO_JACO.PY	104
B.2	TASK RECIPES: MARKER_POSES_SEQUENCIAL.YAML	112
B.3	EXECUTION ORCHESTRATION: VISION_JACO_ARUCO_DEMO.LAUNCH	119
B.4	SUPPORTING CONFIGURATION FILES	120
B.4.1	aruco_l515_multi.launch	120
B.4.2	rs_camera_quiet.launch	121
	APPENDIX C – THIRD-PARTY DEPENDENCIES	126

C.1	SUMMARY OF DEPENDENCIES	126
C.2	INSTALLATION AND CONFIGURATION	126

1 INTRODUCTION

1.1 BACKGROUND AND CONTEXT

The evolution toward highly responsive and technologically advanced manufacturing systems has necessitated higher levels of flexibility, particularly in high-precision environments such as Computer Numerical Control (CNC) machining (Koren *et al.*, 1999). Traditionally, industrial automation relied on fixed robotic cells with rigid programming, which are highly efficient for mass production but struggle with the "high-mix, low-volume" demands of modern customized manufacturing (Perzylo *et al.*, 2019).

In this context, the integration of collaborative manipulators and perception systems has emerged as a solution to bridge the gap between human-like adaptability and machine-like repeatability (Siciliano *et al.*, 2010). Modern robotic workflows often utilize vision-based sensing, such as ArUco marker detection, to allow robots to identify workpieces and determine their spatial coordinates within a workspace (Romero; Munoz; Medina, 2018). By combining the Robot Operating System (ROS) with specialized motion planning frameworks like MoveIt, researchers and engineers can develop pipelines that integrate perception, pose estimation, and execution in a unified environment (Quigley *et al.*, 2009; Chitta; Sucan; Cousins, 2012).

This project was developed at the *Werkzeugmaschinenlabor* (WZL) of RWTH Aachen University, a leading institute in production engineering. The WZL focuses on developing synchronized, highly productive, and sustainable manufacturing systems for global industry partners. This work is situated within a broader research initiative at the lab aimed at optimizing "pick-and-place" workflows for CNC machine tending, where the automation of loading and unloading workpieces is essential to increasing the efficiency of the manufacturing cell. This study focuses on the retrofitting process, which consists of repurposing an obsolete and underutilized robot for novel applications.

1.2 MOTIVATION

The motivation behind this project is twofold: it addresses a critical technical challenge in automated machine tending while promoting the technological retrofitting of legacy hardware. The Werkzeugmaschinenlabor (WZL) possessed an underutilized Kinova Jaco2 manipulator originally manufactured in 2016. This specific robotic arm was initially engineered as an assistive medical device designed to be mounted on wheelchairs to facilitate daily activities for people with disabilities (Campeau-Lecours *et al.*, 2019). By repurposing this hardware for industrial research, this work establishes a sustainable framework for academic asset maximization through engineering intervention.

To adapt this assistive manipulator for advanced manufacturing tasks specifically the loading and unloading of workpieces in a Computer Numerical Control (CNC) machine, a comprehensive retrofit strategy was required. In industrial automation, retrofitting serves as a cost-effective and efficient method to integrate modern computational capabilities into older hardware platforms (Lins; Oliveira, 2020). In this project, the retrofit is achieved by leveraging the Robot Operating System (ROS 1) as the middleware wrapper (Quigley *et al.*, 2009). This allows the 2016 legacy arm to communicate with sensors, such as the Intel RealSense L515 camera, which also operates within the ROS 1 ecosystem.

Furthermore, this project is motivated by the operational need to develop a robust, repeatable workflow that simplifies complex human-robot-machine interactions within the laboratory. There is a significant operational advantage in utilizing an ID-driven recipe system. By using ArUco markers for executing pre-validated, joint-space trajectories, it is possible to achieve the high repeatability required for industrial cycles while maintaining the flexibility to handle different workpieces based on their unique IDs. This approach aims to provide a stable, safe, and efficient retrofit solution for automated material handling in a laboratory-scale industrial environment, transforming an assistive tool into a viable platform for manufacturing research.

1.3 PROBLEM STATEMENT

Repurposing legacy robotic hardware for high-precision industrial applications presents a severe integration and software compatibility challenge. Modern manufacturing environments, such as Computer Numerical Control (CNC) machine tending, require high repeatability and tight tolerances. However, legacy platforms, such as the 2017 Kinova Jaco2, which was originally designed for low-torque assistive medical tasks, lack native, modern industrial control interfaces (Campeau-Lecours *et al.*, 2019).

To bridge this gap, developers often attempt to implement advanced, dynamic 6D pose-estimation pipelines using online frameworks inside modern middleware (Quigley *et al.*, 2009). In practice, relying on open-loop perception-to-execution architectures (such as the traditional look-and-move approach) makes the robotic system highly sensitive to positioning uncertainties. Furthermore, position-based visual control is highly susceptible to sensor noise and camera-to-robot calibration errors. In these cases, uncertainties in the extrinsic and intrinsic calibration parameters produce errors in the operational space estimation, which can negatively affect the accuracy of automated grasping (Siciliano *et al.*, 2010).

Furthermore, deploying a deterministic, open-loop alternative to mitigate this perception jitter introduces its own physical boundaries. While executing pre-recorded joint-space trajectories ensures immediate safety and baseline repeatability, the system lacks real-time sensory feedback during runtime. Consequently, minor geometric

displacements of the workpiece accumulate over continuous, repeated operations, a phenomenon known as open-loop drift, where the controller cannot compensate for external perturbations or structural misalignments without a closed-loop feedback pipeline (Spong; Hutchinson; Vidyasagar, 2006). This progression eventually degrades alignment quality, causing subsequent cycle failures.

Therefore, the core problem addressed by this work stems from the absence of closed-loop feedback control and inaccuracies in both camera and robot calibration. These limitations precluded the use of the MoveIt framework for automated trajectory planning, necessitating the manual mapping of waypoints and prompting a shift toward an alternative approach based on .yaml configuration files.

1.4 RESEARCH OBJECTIVES AND RESEARCH QUESTIONS

The main objective of this work is to develop, implement, and validate a ROS-based automation workflow for Computer Numerical Control (CNC) pick-and-place machine tending using an ArUco-ID-driven recipe system. The specific objectives of this work include:

- Integrate an Intel RealSense L515 depth camera and a legacy 6-DoF Kinova Jaco2 manipulator within a unified ROS Noetic environment.
- Implement an optimized ArUco vision node dedicated to stable marker identification and discrete task selection.
- Develop a modular YAML-based recipe system that dynamically maps specific marker IDs to structured sequences of joint-space targets and end-effector commands.
- Author and shape collision-free trajectories using manual joint-space waypoints and mandatory dwell times to guarantee operational safety inside a CNC workcell.
- Evaluate the operational reliability and physical limitations of the open-loop execution compared to non-deterministic, 6D pose-based grasping strategies.

Together, these objectives define the practical engineering steps required to deploy and evaluate the proposed system. To guide the investigation at a conceptual level, this work formulates a main research question that captures the core problem addressed by the thesis: *how an ArUco-ID-to-recipe joint-space control architecture can improve short-term repeatability and operational simplicity in CNC machine tending tasks compared to online 6D pose-based planning when using legacy manipulation hardware.*

The following sub-questions further structure the inquiry by clarifying how the system handles legacy hardware interfaces and how open-loop design choices impact long-run cycle performance:

- *How can ROS 1 middleware be effectively utilized to retrofit a legacy assistive manipulator into a deterministic industrial workcell, bypassing the uncertainties of dynamic motion planners?*
- *What are the systemic bottlenecks (e.g., TF jitter, hand-eye calibration errors) that render continuous 6D pose estimation via MoveIt unviable for high-precision CNC machine tending using retrofitted hardware?*

1.5 SCOPE AND LIMITATIONS

This work focuses on developing and validating a ROS-based software pipeline to enable reliable part handling and machine tending operations using a legacy Kinova Jaco2 (6-DoF) manipulator. The scope includes designing a software architecture that serves as an interface between discrete vision-based identification and direct joint-space execution, allowing the system to detect workpieces via ArUco markers and execute sequential pick-and-place tasks within a CNC workcell.

The system architecture integrates components for data acquisition, coordinate transformations (*TF*), and trajectory execution. Integration with Robot Operating System (ROS 1 Noetic) packages, specifically `aruco_ros` for marker identification, `realsense2_camera` for data streaming, and the native Kinova ROS driver interface, constitutes a core component of this development. Operational safety and baseline trajectories are authored and monitored through MoveIt and RViz visualization tools before direct hardware deployment.

The key limitations of this work include the following:

- The system operates strictly in an open-loop manner; it does not utilize real-time computer vision updates or force-torque feedback to correct workpiece displacement after the initial marker ID detection.
- MoveIt is utilized exclusively for monitoring, safety visualization, and joint configuration readout during the manual waypoint authoring phase, rather than for online, real-time trajectory planning or dynamic collision avoidance.
- Experimental validation is limited to a single robotic manipulator, as a comparative multi-robot implementation was obstructed because the second available arm unit presented a hardware limitation regarding insufficient joint torque.

- The workflow assumes an ideal environment where the workpiece remains within tight geometric tolerances at its designated placement areas, leaving the architecture susceptible to cumulative errors over extended operational loops.
- Validation is restricted to laboratory-scale industrial scenarios at the institute, without evaluating communication network latencies, protocol performance, or full-scale industrial production line disruptions.

1.6 SIGNIFICANCE OF THE WORK

This work addresses a prominent practical challenge in industrial automation: engineering highly reliable part-handling and machine-tending workflows using legacy manipulation hardware. It proposes and validates an open-source software framework that effectively bridges the gap between discrete vision-based perception and deterministic robot execution, bypassing the instabilities commonly associated with real-time 6D online planning in unstructured or noisy laboratory environments.

By integrating robust ArUco-ID recognition with a custom, joint-space YAML recipe system within ROS Noetic, this architecture significantly improves operational simplicity and short-term execution repeatability. This method establishes an empirical blueprint for industrial retrofitting, demonstrating how older robotic assets, such as the 2017 assistive Kinova Jaco2 arm, can be successfully rehabilitated and repurposed for manufacturing research.

In the long term, the insights and structural limitations documented in this study, particularly regarding the degradation of cycles due to open-loop drift, provide foundational data for future research initiatives at the Werkzeugmaschinenlabor (WZL). This implementation serves as a benchmark for developing hybrid control architectures, where low-overhead open-loop routines can be augmented with sparse, discrete closed-loop validation steps, ultimately advancing the deployment of flexible, cost-effective automation cells in high-mix manufacturing.

1.7 STRUCTURE OF THE DOCUMENT

The remainder of this document is organized as follows. Chapter 2 presents the theoretical foundation encompassing robotic kinematics, computer vision for industrial automation, and the applied software frameworks. Chapter 3 details the problem description, the institutional context, and the strict technical requirements necessary for the CNC machine tending workcell. Chapter 4 outlines the proposed solution and methodology, detailing the architectural shift toward a deterministic “ID-to-Recipe” control scheme and the manual waypoint authoring process. Chapter 5 discusses the practical implementation of the system, including the developed components and the

troubleshooting of technical bottlenecks such as software configuration and TF alignment. Chapter 6 presents the experimental validation, providing a critical performance analysis, quantitative repeatability metrics, and proposed technical solutions for identified limitations. Finally, Chapter 7 concludes the work, summarizing the technical impacts and outlining recommendations for future research.

2 THEORETICAL FOUNDATION

This chapter establishes the theoretical and technological foundations required to engineer a reliable vision-guided robotic cell for CNC machine tending. Rather than exploring isolated concepts, it examines the critical intersection of robotic kinematics, discrete computer vision, and distributed software architectures necessary to retrofit legacy hardware. By analyzing the mathematical constraints of spatial transformations, the vulnerabilities of pose estimation, and the physical realities of open-loop control, this section provides the fundamental scientific groundwork to justify the architectural shift from dynamic trajectory planning to a deterministic, state-machine-driven execution.

2.1 ROBOTIC MANIPULATION

Industrial robotic manipulation fundamentally relies on the capacity to position and orient a tool in three-dimensional space to interact with the environment in a deterministic manner. As assumption by Spong, Hutchinson, and Vidyasagar (2006), mathematically formalizing this interaction requires a rigorous mapping of the manipulator's state across distinct coordinate spaces, a process intrinsically governed by its serial topological structure and geometric constraints. This section presents the structural and kinematic characteristics of the utilized 6-DoF Kinova Jaco2 manipulator and discusses the mathematical, computational, and practical implications of executing movements in the joint configuration space $\mathcal{Q} \subset \mathbb{R}^n$ compared to the Cartesian operational space $\mathcal{X} \subset SE(3)$.

By analyzing the differential kinematics via the manipulator analytical Jacobian matrix $J(\theta)$, this section establishes how kinematic singularities, non-linear inversions, and perceptual uncertainties propagate through the system architecture. Ultimately, this theoretical contrast provides the mathematical and operational justification for replacing dynamic, pose-based Cartesian trajectory planning with a deterministic, joint-space recipe framework tailored for highly repeatable CNC machine tending operations.

2.1.1 Kinova Jaco2: Degrees of Freedom and Kinematics

The degrees of freedom (DoF) of an open serial manipulator define the minimum number of independent joint variables required to uniquely and completely determine the physical configuration and spatial posture of all its mechanical links (Craig, 2005). The Kinova Jaco2 manipulator employed in this project (specifically the *j2n6s300* industrial model) features $n = 6$ revolute joints. This specific mechanical topology provides exactly 6 DoF, which represents the necessary threshold to endow the end-effector with full three-dimensional positioning capabilities ($\mathbb{R}^3$) coupled with arbitrary angular orientation ($SO(3)$) within its accessible workspace.

The internal state of the manipulator is represented by the joint configuration vector $\theta \in \mathbb{R}^6$, explicitly defined as:

$$\theta = [\theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6]^T. \quad (1)$$

The static geometric relationship between this joint configuration vector θ and the resulting end-effector pose x in the operational space is governed by forward and inverse kinematics mappings (Siciliano *et al.*, 2010). The forward kinematics mapping, denoted by the continuous non-linear geometric function $f(\theta) : \mathcal{Q} \rightarrow \mathcal{X}$, computes the Cartesian pose of the tool from the angular positions measured by the absolute optical encoders of the joints:

$$x = f(\theta). \quad (2)$$

To systematically establish coordinate frames between consecutive links, the standard Denavit-Hartenberg (DH) convention parameterizes each joint-link pair using four geometric parameters: link length (a_i), link twist (α_i), link offset (d_i), and joint angle (θ_i). The homogeneous transformation matrix $A_i^{i-1}(\theta_i)$ that maps the coordinate frame of link i back to the frame of link $i - 1$ is formulated as follows:

$$A_i^{i-1}(\theta_i) = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

As shown in Figure 1, the cumulative multiplication of these individual matrices from the fixed inertial base of the robot (Frame 0) to the final tool mounting flange (Frame 6) yields the global homogeneous transformation matrix $T_6^0(\theta)$:

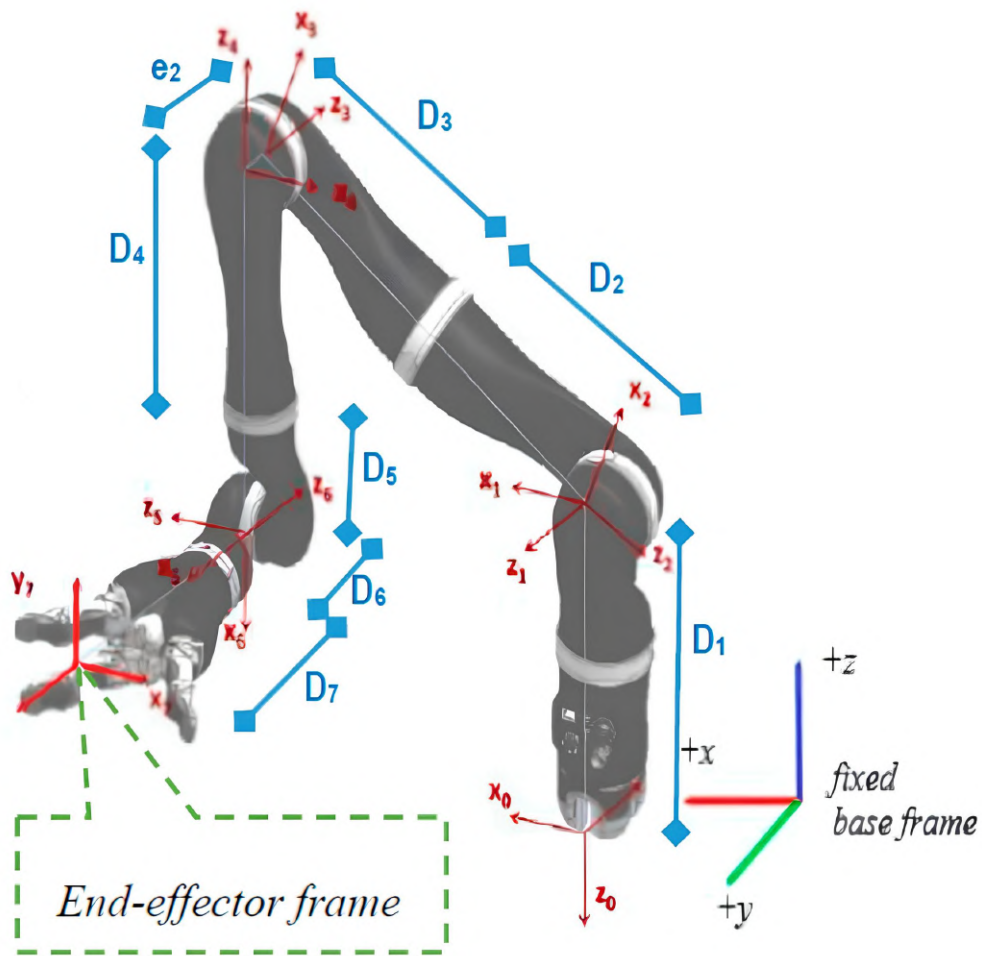
$$T_6^0(\theta) = A_1^0(\theta_1)A_2^1(\theta_2)A_3^2(\theta_3)A_4^3(\theta_4)A_5^4(\theta_5)A_6^5(\theta_6). \quad (4)$$

Conversely, the inverse kinematics problem aims to determine the vector of joint variables necessary to colocalize the Tool Center Point (TCP) at a target operational pose:

$$\theta = f^{-1}(x). \quad (5)$$

For a 6-DoF serial architecture, the inverse mapping $f^{-1}(x)$ constitutes a highly non-linear problem composed of coupled transcendental equations that typically admit multiple distinct discrete analytical or numerical solutions (Craig, 2005). This mathematical multiplicity represents different physical arm assembly modes (e.g., elbow-up versus elbow-down configurations reaching the exact same spatial pose).

Figure 1: Kinematic diagram and DH frames for Kinova Jaco2.



Source:(Arif; Fu, 2022)

In a tightly confined industrial environment, such as a CNC machine tending workspace, this kinematic ambiguity introduces unpredictable trajectory variations during dynamic online path generation. If the trajectory planning algorithm abruptly switches between alternative assembly modes during continuous path tracking, the physical manipulator will attempt an instantaneous structural reconfiguration. This behavior causes dangerous self-collisions or catastrophic structural impacts against the CNC enclosure unless individual joint configurations are strictly restricted.

2.1.2 Joint-Space vs. Cartesian-Space Execution

Trajectory generation and motion execution within industrial cells can be formulated in two competing operational domains, each presenting distinct computational complexities and structural trade-offs:

- **Cartesian-Space (Task-Space) Planning:** The trajectory of the TCP is explicitly restricted along a continuous geometric path in three-dimensional space. The displacement between the initial and final points is interpolated linearly or circularly regarding Cartesian coordinates (x, y, z) and orientation parameters, such as Hamilton quaternions $(q \in \mathcal{S}^3)$, relative to a fixed global reference frame.
- **Joint-Space (Configuration-Space) Planning:** The trajectory is calculated without restricting the specific geometric path traversed by the end-effector in the physical workspace. The motion profile focuses exclusively on the smooth, synchronized interpolation (typically utilizing cubic or quintic polynomials to bound velocity, acceleration, and jerk profiles) of individual actuator positions over time from an initial vector θ_{start} directly to a target vector θ_{goal} .

The Cartesian tracking accuracy of the TCP is intensely sensitive to differential and structural perturbations. The linear mapping that translates joint velocities $\dot{\theta}$ into instantaneous linear and angular operational velocities $\dot{x} = \begin{bmatrix} v^T & \omega^T \end{bmatrix}^T$ is defined by the Manipulator Jacobian matrix $J(\theta) \in \mathbb{R}^{6 \times 6}$ (Spong; Hutchinson; Vidyasagar, 2006):

$$\dot{x} = J(\theta)\dot{\theta}. \quad (6)$$

Consequently, computing Cartesian-space tracking commands continuously in real-time forces the system control loop to calculate the inverse of the Jacobian matrix at high frequencies:

$$\dot{\theta} = J^{-1}(\theta)\dot{x}. \quad (7)$$

The critical vulnerability of this method lies in the inevitable existence of kinematic singularities. When the manipulator falls into a geometric configuration where the axes of two or more joints become colinear, the determinant of the Jacobian matrix drops to zero ($\det(J(\theta)) \rightarrow 0$), causing the matrix to become rank-deficient. In these singular regions, the analytical inverse $J^{-1}(\theta)$ becomes mathematically undefined and numerically unbounded. Forcing a finite Cartesian velocity $\dot{x}$ near a singularity requires mathematically infinite joint velocities $\dot{\theta}$, which instantly saturates the actuators, violates physical current limits, and triggers an immediate overcurrent safety trip (*driver fault*) in the robot controller.

Furthermore, static and dynamic errors originating from the perception pipeline, such as hand-eye extrinsic calibration misalignments, stochastic sensor depth noise, and transport delays in coordinate transform calculations (TF), propagate directly into the Cartesian inverse calculation (Siciliano *et al.*, 2010). For high-precision applications like loading raw blocks into narrow CNC pneumatic clamps, a sub-millimeter perception error distorts the inverse kinematic computation. This distortion translates into an inac-

curate target joint state, inducing critical part jams or mechanical damage to the tool and clamping fixtures.

Direct tracking via the **Joint Space** completely circumvents the Jacobian inversion problem, eliminating mathematical singularities and numerical instability. By recording deterministic joint angles manually via a joystick (*teach pendant method*) and organizing them structurally into offline YAML recipes, the execution pipeline eliminates real-time Cartesian spatial computation. This architecture shifts the system's operational paradigm from a flexible but error-prone dynamic planner to a deterministic, highly repeatable state-machine execution. The practical result is the optimization of the local mechanical repeatability of the legacy manipulator, increasing the reliability of collision-free and stable pick-and-place cycles.

2.2 COMPUTER VISION IN INDUSTRIAL AUTOMATION

While the integration of visual sensors theoretically provides robotic systems with precise spatial awareness (Hartley; Zisserman, 2004), translating raw pixel data into reliable kinematic commands remains a significant challenge in precision manufacturing. In optically harsh environments, bridging the gap between perception and execution requires algorithms that prioritize determinism over continuous dynamic estimation.

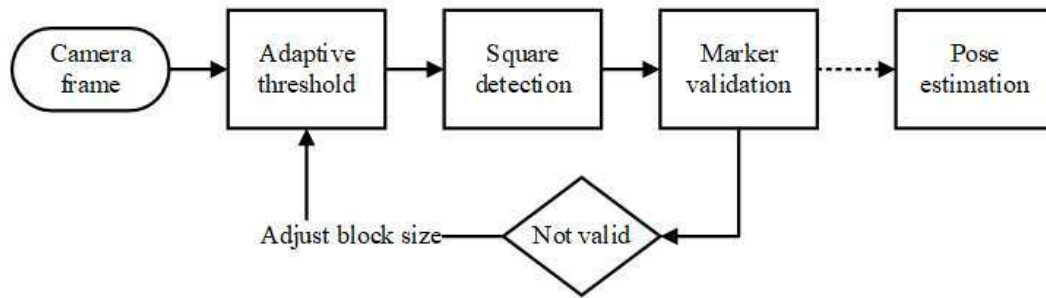
This section explores the methodologies required to establish this robust visual interface. It begins by examining the algorithmic foundation of fiducial marker systems, highlighting how synthetic binary matrices can be exploited for discrete object identification and reliable task triggering. Subsequently, it critically addresses the physical limitations of 3D pose estimation, specifically characterizing sensor noise and the phenomenon of perception jitter, to illustrate why continuous visual tracking often degrades the stability of industrial workcells.

2.2.1 ArUco Markers: Detection and Identification (ID)

ArUco markers, originally proposed by (Garrido-Jurado *et al.*, 2014), are square fiducial systems composed of an inner synthetic binary matrix surrounded by a thick, continuous black outer border. The primary function of this external black border is to facilitate rapid contour localization within complex visual backgrounds, while the internal binary matrix encodes a unique numerical identifier (ID) utilizing error-immune coding schemes, such as modified Hamming codes. This structural design ensures that the system can robustly validate and decode the marker even under adverse industrial conditions, including partial occlusions, perspective distortions, or non-uniform illumination profiles.

As illustrated in Figure 2, the standard computational pipeline for detecting ArUco markers operates through a sequence of deterministic digital image processing stages:

Figure 2: Standard ArUco detection pipeline: from binarization to pose estimation.



Source:(Tentone, 2021)

1. **Adaptive Thresholding:** The raw grayscale image captured by the CMOS sensor is subjected to a local adaptive thresholding algorithm. This converts the image into a binary matrix (black and white pixels), isolating the marker boundaries from global ambient illumination variations.
2. **Contour Extraction and Polygonal Approximation:** Contour-finding algorithms isolate closed geometric boundaries. Those contours that can be mathematically approximated as four-cornered polygons (quadrilaterals) are selected as potential marker candidates.
3. **Canonical Sampling and Decoding:** A perspective homography transformation is applied to remove geometric distortions induced by the camera's perspective projection. This process projects the internal matrix into a regular, rectified pixel grid, which is subsequently discretized into a matrix of black and white bits.
4. **Dictionary Validation:** The extracted binary code is evaluated against a pre-indexed dictionary database. If the calculated Hamming distance between the extracted code and a valid dictionary pattern falls within the error-correction thresholds, the marker is successfully validated, and its numerical ID is returned.

When the automation architecture leverages the fiducial marker strictly for the discrete recognition of the object's ID (an ID-to-task mapping workflow), instead of continuously estimating its geometric spatial pose, the computational complexity and overall vulnerability of the system decrease substantially. This discrete perception paradigm bypasses the dependency on continuous, real-time inverse projection equations for trajectory generation. Consequently, it eliminates software planning exceptions induced by dynamic occlusions or sensor instabilities during critical motion steps.

2.2.2 Pose Estimation Challenges (Jitter and Noise)

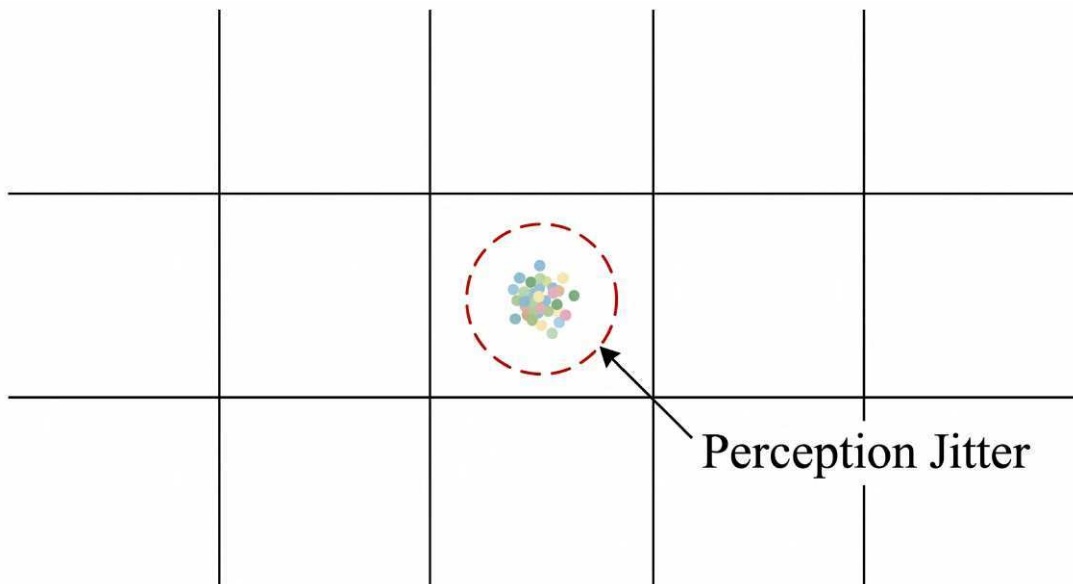
Although fiducial markers can be utilized to compute the full 6D pose (translation and rotation) of an object by solving the classical Perspective-n-Point (PnP) problem,

the continuous extraction of these metric coordinates introduces severe stochastic instabilities. The phenomenon designated as *perception jitter* manifests as high-frequency numerical oscillations in the estimated pose parameters, even when both the visual sensor and the target object remain perfectly static and rigidly mounted on industrial fixtures.

The physical and mathematical mechanisms underlying this behavior result from a combination of multiplicative factors:

- **Sensor Thermal and Electronic Noise:** Solid-state CMOS image sensors operate under the influence of inevitable electronic read noise, photon shot noise, and thermal dark currents. These intrinsic phenomena introduce random intensity fluctuations in the grayscale values of adjacent pixels over time.
- **Vertex Subpixelization Uncertainty:** Spatial pixel intensity noise directly corrupts edge-detection algorithms operating at the subpixel level. Because the localized corners of the ArUco marker fluctuate by subpixel fractions between consecutive frames, the spatial control points provided to the PnP solver vary continuously, as illustrated in Figure 3.

Figure 3: Perception jitter effect: sensor noise induces subpixel uncertainty in marker vertex detection.

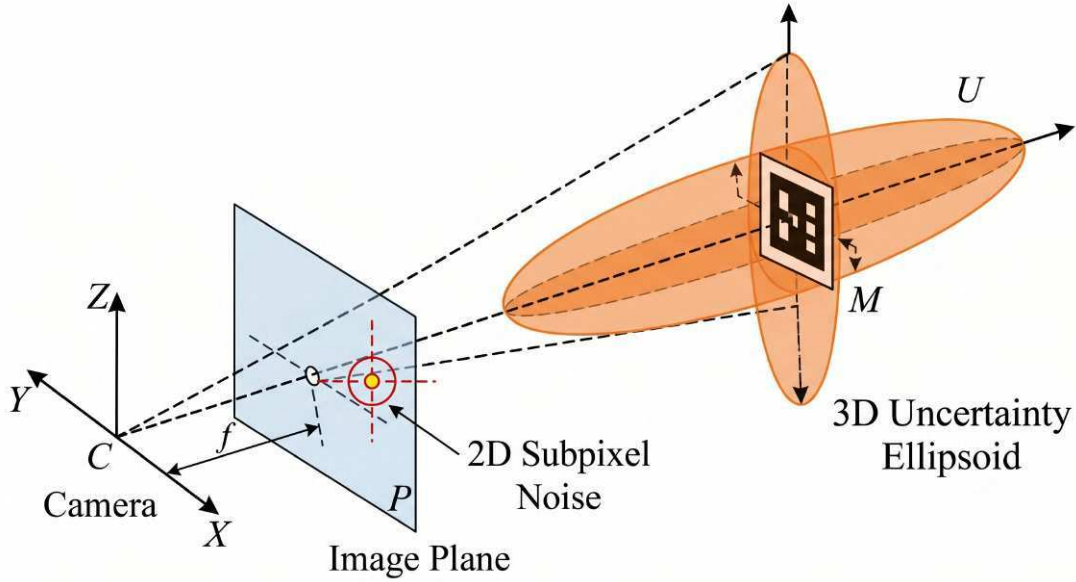


Source: Created by the Author.

- **PnP Conditioning and Sensitivities:** Solving the 6D pose requires inverting mathematical projection equations incorporating the camera's intrinsic parameters. Due to the non-linear nature of projective geometry, minor subpixel variations in corner inputs are amplified non-linearly in the three-dimensional output space.

This amplification is particularly severe along the optical depth axis (Z), where the estimation error scales quadratically with respect to distance (see Figure 4).

Figure 4: Uncertainty propagation: 2D subpixel noise yields a 3D ellipsoid elongated along the depth axis.



Source: Created by the Author.

The propagation of this stochastic noise through the system's coordinate transformation tree (in topics) introduces intolerable spatial deviations at the robot's tool center point (TCP). In repetitive CNC machine tending applications characterized by millimeter-level geometric clearances within pneumatic clamping mechanisms, direct vision-based closed-loop control becomes unfeasible. The accumulation of these geometric fluctuations causes critical part misalignment, thereby justifying the development of a stable joint-space recipe framework triggered discretely by ID identification.

2.3 SOFTWARE FRAMEWORKS

The synchronous coordination of multiple hardware processes, high-frequency sensor data streams, and motion control algorithms demands a standardized and modular communication infrastructure (Quigley *et al.*, 2009). In advanced robotic cells, middleware software serves as an abstraction layer that unifies heterogeneous peripherals under a common data bus.

This section presents the architectural foundations of the Robot Operating System (ROS 1 Noetic), detailing the operation of its distributed coordinate transformation system (TF). Furthermore, it addresses the hardware integration of the Intel RealSense L515 depth sensor and describes the structural role of the MoveIt ecosystem and

the RViz graphical interface during the static geometric verification of the industrial workspace.

2.3.1 ROS Noetic and Coordinate Transformations (TF)

The Robot Operating System (ROS 1) is an open-source meta-operating system that provides a structured communication framework over a distributed computing architecture (Quigley *et al.*, 2009). Individual computational processes are abstracted as **Nodes**, which specialized tasks in isolation and communicate asynchronously via a peer-to-peer messaging model based on the publication and subscription of **Topics**. This decoupled topology ensures that low-level robotic drivers and high-level computer vision nodes can operate concurrently without direct compilation dependencies.

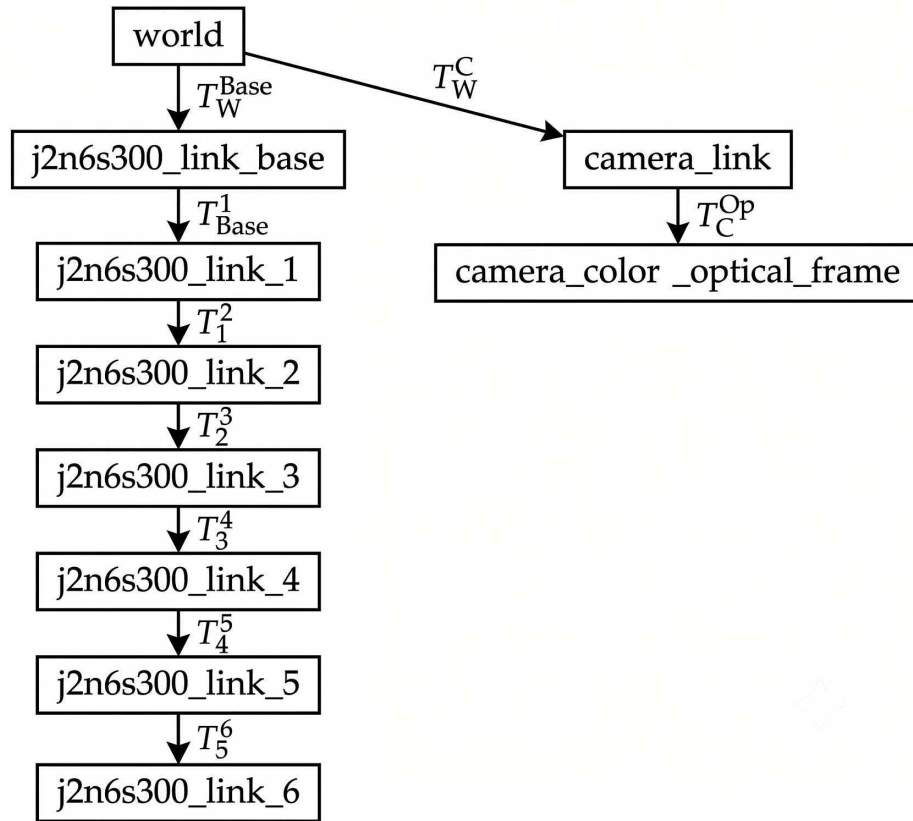
Spatial and temporal consistency across the heterogeneous physical and virtual frames of the manufacturing cell is centrally managed by the coordinate transform package, designated as **TF** (Foote, 2013). The TF framework dynamically constructs and maintains a directed acyclic tree of spatial relationships among localized coordinate systems, as illustrated in Figure 5. The spatial transformation of a three-dimensional point measured within the local camera frame ($\{C\}$) to the global origin reference frame at the robot base ($\{B\}$) is computed via a homogeneous transformation matrix $\mathbf{T}_C^B \in SE(3)$ (Spong; Hutchinson; Vidyasagar, 2006; Craig, 2005), mathematically formulated as:

$$\mathbf{T}_C^B = \begin{bmatrix} \mathbf{R}_{3 \times 3} & \mathbf{P}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}, \quad (8)$$

where $\mathbf{R}$ represents the 3×3 orthogonal rotation matrix describing the relative spatial orientation between the two coordinate frames, and $\mathbf{P}$ defines the 3×1 translation vector locating the origin of frame $\{C\}$ with respect to frame $\{B\}$.

As established by the kinematic and visual control theory in (Siciliano *et al.*, 2010), uncertainties within static calibration parameters produce significant errors in the operational space estimation. In practice, minor numerical inaccuracies within $\mathbf{T}_C^B$, stemming from hand-eye extrinsic calibration errors or minor mechanical flexions in the camera mounting fixture, propagate multiplicatively during subsequent matrix chain-multiplications. This geometric distortion corrupts the translation of raw parts detected by the camera into the robot's workspace, inducing positioning offsets at the Tool Center Point (TCP). Such deviations severely compromise rigid mechanical engagement if the system relies exclusively on open-loop, vision-guided online calculation.

Figure 5: TF architecture connecting the robot base, camera, and TCP.



Source: Created by the Author.

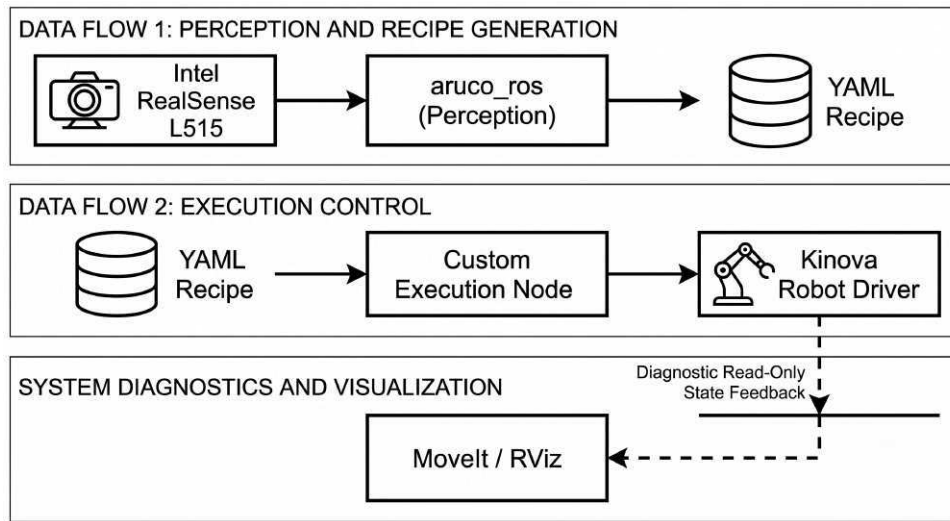
2.3.2 RealSense L515 Integration and MoveIt/RViz Ecosystem

The Intel RealSense L515 visual sensor utilizes solid-state Light Detection and Ranging (LiDAR) laser scanning technology to output high-resolution depth maps paired with aligned RGB color streams. Within the ROS workspace, the `realsense2_camera` node functions as the low-level hardware abstraction layer. This driver decodes the hardware transport protocol and stream-publishes standardized point cloud data structures (`sensor_msgs/PointCloud2`) along with intrinsic camera calibration parameters over synchronous topics.

To evaluate and manage the volumetric and geometric state of the manipulator, the **MoveIt** motion planning framework (Chitta; Sucas; Cousins, 2012) is combined with the **RViz** interactive graphical visualization tool, as structurally mapped in Figure 6. MoveIt operates as an integrated software suite combining advanced kinematic plugins, motion planners, and collision detection libraries (such as the Flexible Collision Library - FCL), modeling the industrial workcell through unified geometric representation files (URDF/SRDF).

Within the software architecture implemented for this project, the operational role of MoveIt and RViz intentionally diverges from conventional dynamic application

Figure 6: System architecture: decoupled sensor pipelines and MoveIt/RViz monitoring.

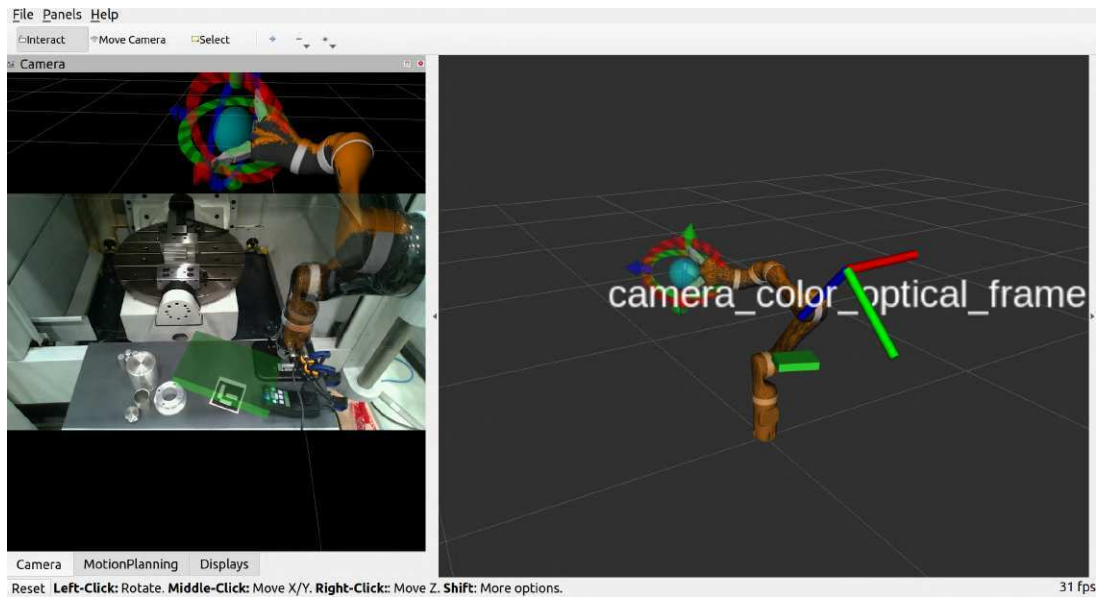


Source: Created by the Author.

pipelines. Instead of utilizing the framework to plan paths online in real-time, which would introduce unacceptable computational latencies, collision risks near fixtures, and algorithmic convergence failures due to camera noise, the tools are deployed **strictly as a static engineering and diagnostic layer**.

This practical deployment is empirically demonstrated in Figure 7. The interfaces provide structural safety monitoring, 3D environment visualization, and precise joint-state angle extraction (`joint_states`) during the manual teaching and trajectory recording process, thereby prioritizing absolute operational determinism via direct joint commands.

Figure 7: RViz diagnostic layer: RealSense L515 ArUco feed (left) and digital twin configuration (right).



Source: Created by the Author.

2.4 INDUSTRIAL CONTROL AND AUTOMATION

The operational efficiency and reliability of repetitive manufacturing processes critically depend on the stability and robustness of the utilized control architecture when subjected to environment-induced disturbances and uncertainties (Groover, 2015). In the specific context of automated machine tending, the choice of control strategy dictates how the system handles minor geometric variations and mechanical clearances over extended multi-cycle loops.

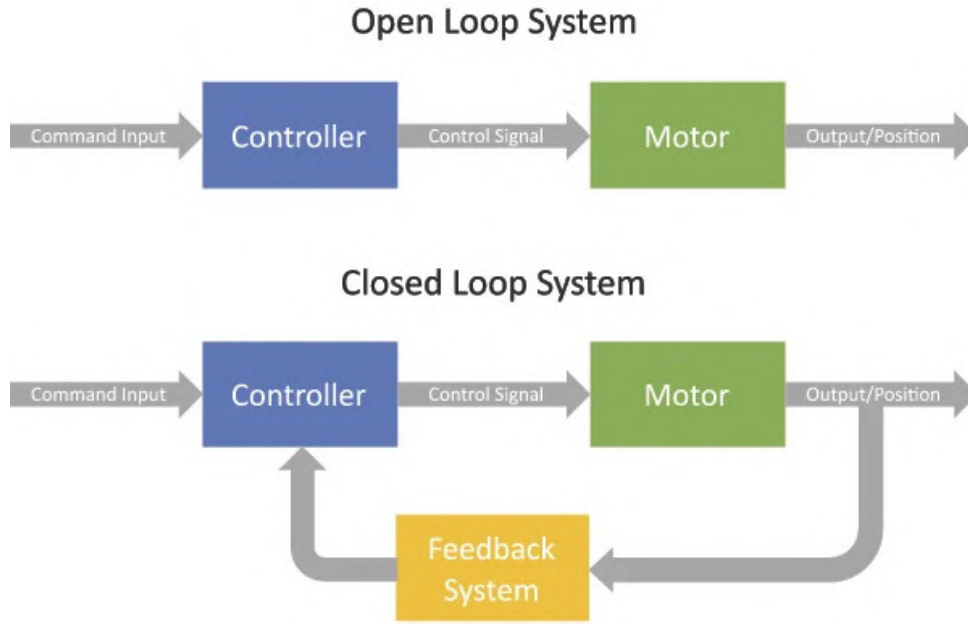
This section establishes the mathematical and conceptual contrast between open-loop and closed-loop control systems, analyzing the mechanics of error propagation and accumulation (*open-loop drift*) in purely repetitive automation cycles. Additionally, it addresses the methodology of technological retrofit as an engineering approach to modernize and integrate legacy robotic hardware into contemporary open-source software ecosystems.

2.4.1 Open-Loop vs. Closed-Loop Control and Error Accumulation

The fundamental distinction among automation control architectures lies in the application of feedback mechanisms to govern the actuation variables (Ogata, 2010), as structurally compared in Figure 8.

- **Closed-Loop Control Systems:** The system's output signal $y(t)$ (such as the actual spatial position of a part or the tool center point) is continuously monitored by sensors and fed back for comparison against a desired reference command

Figure 8: Control architectures: open-loop predictive vs. closed-loop feedback.



Source:(NTCHIP, 2026)

$r(t)$. The resulting error signal, mathematically defined as $e(t) = r(t) - y(t)$, is processed by a control law to dynamically adjust the system inputs in real-time, thereby actively suppressing external disturbances.

- **Open-Loop Control Systems:** The control action is strictly determined by initial system calibration and pre-established parameters. The controller issues predictive actuation commands under the operational assumption that both the internal hardware dynamics and the external environment will remain invariant, without executing any physical verification of the actual output following the action.

In cyclic, purely repetitive manipulation tasks executed under an open-loop paradigm, the overall process is highly vulnerable to the phenomenon designated as *open-loop drift*. Mathematically, if an external perturbation or physical release mechanic introduces a residual spatial error ϵ_k at the conclusion of part-handling cycle k , this displacement remains unmeasured by the controller. In subsequent iterations lacking intermediate sensory re-localization, the cumulative positioning error at cycle N can be modeled as the summation of the historical stochastic deviations:

$$E_{\text{total}} = \sum_{k=1}^N \epsilon_k. \quad (9)$$

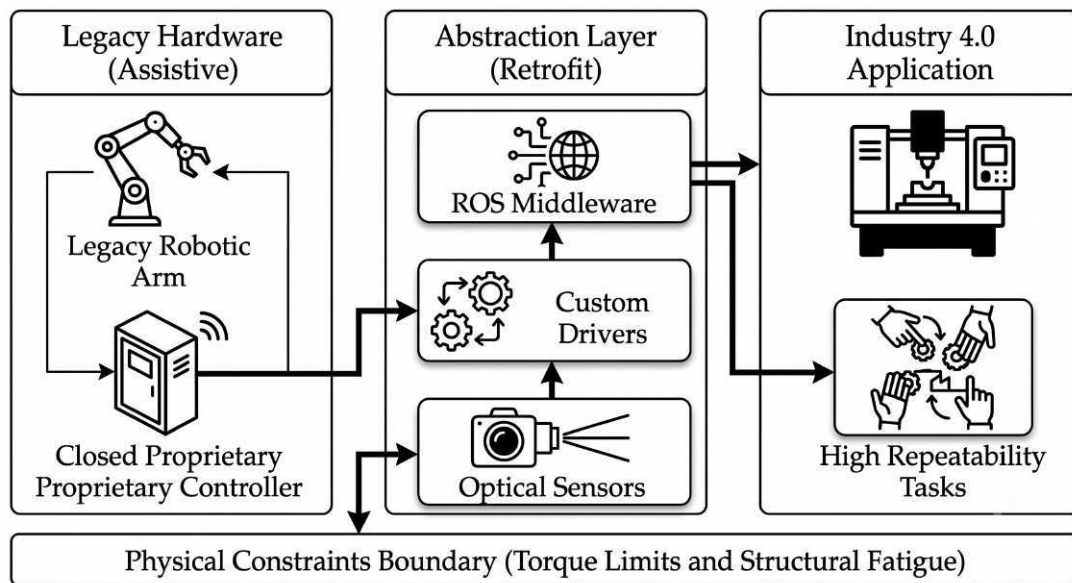
In long-term manufacturing operations, if the mean value of these micro-displacements ϵ_k is non-zero, often caused by systematic gravity-induced sliding or uniform gripper-jaw friction surfaces, the boundary error E_{total} increases progressively.

This continuous geometric degradation eventually exceeds the tight mechanical tolerance clearances of the pneumatic CNC fixtures, leading to critical part jamming, fixture damage, or severe operational routine failures.

2.4.2 Technological Retrofit of Legacy Robotic Hardware

The term technological retrofit in industrial automation refers to the modification, engineering adaptation, or modernization of obsolete or legacy electromechanical systems through the targeted integration of contemporary software frameworks, advanced sensors, and modern computational interfaces (Lins; Oliveira, 2020). Within the framework of Industry 4.0 paradigms, retrofitting operates as a highly sustainable and cost-effective methodology. It extends the functional lifecycle of mechanically robust hardware assets while avoiding the significant capital expenditure associated with complete infrastructure replacement, as illustrated by the architectural blueprint in Figure 9.

Figure 9: Hardware-constrained robotic retrofit: mapping legacy assets to open-source middleware.



Source: Created by the Author.

In the field of robotics, retrofitting allows the repurposing of manipulators originally developed for restricted applications, such as assistive or healthcare robotic arms designed in previous technological generations, into research platforms for advanced smart manufacturing. This integration requires the implementation of custom software driver abstraction nodes capable of translating legacy hardware bus protocols into standardized, high-performance distributed middlewares like the Robot Operating System (ROS).

However, implementing a robust robotic retrofit requires a strict mapping of the asset's original physical and kinematic boundaries. Under modern industrial workflows characterized by continuous, high-frequency repetitive stress profiles, critical operating factors such as actuator thermal limits, joint torque ratings, and structural load constraints must be rigorously monitored to prevent accelerated component wear or catastrophic hardware failures.

3 PROBLEM DESCRIPTION AND TECHNICAL REQUIREMENTS

This chapter establishes the core technological framework, engineering constraints, and architectural requirements governing the development of the robotic machine tending workcell. In accordance with the academic guidelines established by the course coordination, this section transitions from the foundational literature review and theoretical paradigms previously discussed into the concrete, practical deployment of the system.

To bridge the gap between abstract robotic kinematic theories and applied manufacturing automation, this methodology delineates the implementation of a localized, deterministic **ID-to-Recipe** control architecture. This software-defined strategy leverages discrete perception algorithms to drive real-time joint-space command sequences, systematically addressing the physical constraints of legacy hardware assets within a high-repeatability industrial manipulation environment.

3.1 CNC WORKCELL CONTEXT AND MOTIVATION

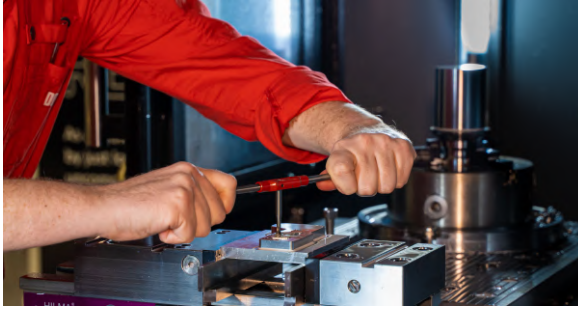
The automation of machine tending operations, specifically the cyclical loading and unloading of workpieces into Computer Numerical Control (CNC) milling or turning stations, constitutes a foundational pillar of modern manufacturing processes (Groover, 2015). Historically, in mass production paradigms, robotic cells relied heavily on highly rigid, permanent mechanical fixtures, proximity sensors, and hardcoded pneumatic indexing tables. These mechanisms guarantee that a workpiece is consistently located at an identical spatial coordinate before manipulation (Koren *et al.*, 1999; Groover, 2015). While highly effective for massive production volumes of a single part geometry, these traditional methods introduce severe cost and operational bottlenecks when applied to High-Mix, Low-Volume (HMLV) production environments. In the context of Industry 4.0 and Reconfigurable Manufacturing Systems (RMS), production lines must rapidly adapt to varying part geometries and small-batch runs without requiring costly, single-purpose mechanical overhauls (Koren *et al.*, 1999).

To inject the necessary flexibility into modern production lines, contemporary manufacturing engineering heavily integrates Vision-Guided Robotic (VGR) manipulation systems (Kragic; Christensen, 2002). By pairing an articulated arm with an optical sensor, such as depth cameras or LiDAR, the automation cell can dynamically detect, align, and handle workpieces regardless of minor initial misplacements.

This project is contextualized within the development of such a localized, vision-guided robotic manipulation cell at the Werkzeugmaschinenlabor (WZL). The primary operational task involves executing a continuous, cyclic pick-and-place routine: fetching a metallic workpiece from a staging workspace, safely transferring and feeding it into a rigid CNC machine tool interface for subtractive processing, and subsequently retrieving

Figure 10: Comparison: (a) manual clamping fixtures vs. (b) vision-guided Jaco2 cell.

a) Traditional rigid mechanical fixturing.



Source:(Smyth, 2026)

b) Vision-guided robotic (VGR) setup.



Source: Created by the Author.

the finished product back to the staging area. Furthermore, the engineering approach involves the technological retrofit of legacy hardware, specifically a Kinova Jaco2 6-DoF manipulator originally designed for assistive tasks, into an advanced manufacturing application.

Initially, the integration scheme aimed to implement a fully dynamic, closed-loop framework, coupling online 6-Degree-of-Freedom (DoF) pose estimation with the MoveIt motion planning ecosystem (Chitta; Sucas; Cousins, 2012). In that theoretical model, an online perception pipeline would calculate the continuous spatial position (x, y, z) and orientation (ϕ, θ, ψ) of the object in real time, prompting a localized obstacle-avoidance algorithm to generate collision-free trajectories on the fly (Siciliano *et al.*, 2010).

However, when deploying such advanced software frameworks on actual industrial shop floors or utilizing legacy equipment, severe constraints emerge. Theoretical kinematic models often assume ideal hardware conditions; in practice, systems suffer from hand-eye calibration sensitivity, where minute angular offsets at the camera mount translate into large lateral deviations at the Tool Center Point (TCP) (Tsai; Lenz, 1989). Additionally, depth and image noise inherent to optical sensors under real industrial lighting, coupled with coordinate transformation (TF) jitter, create unacceptable spatial instability (as theoretically modeled in Figure 4). In precision machine tending, linear deviations exceeding minimal tolerances prevent proper gripper engagement and risk catastrophic collisions with the CNC enclosure.

Consequently, this project focuses on a structural technological retrofit, pivoting from computationally expensive and highly sensitive online trajectory planning to a deterministic, high-repeatability approach known as the **ID-to-Recipe** architecture. In this framework, the visual perception pipeline is heavily constrained: ArUco fiducial markers are utilized purely for discrete identification (extracting an integer ID) rather than continuous 6D pose tracking (Garrido-Jurado *et al.*, 2014). This ID acts as a trigger to execute a pre-recorded, collision-free joint-space recipe, ensuring safe and repeatable cycles while bypassing the systemic errors of dynamic path planning.

3.2 INSTITUTIONAL CONTEXT

This Final Course Project (*Projeto de Fim de Curso* - PFC) was developed at the **Laboratory for Machine Tools and Production Engineering (WZL)** (*Werkzeugmaschinenlabor*) of **RWTH Aachen University**, Germany, in formal cooperation with the **Federal University of Santa Catarina (UFSC)**, Brazil.

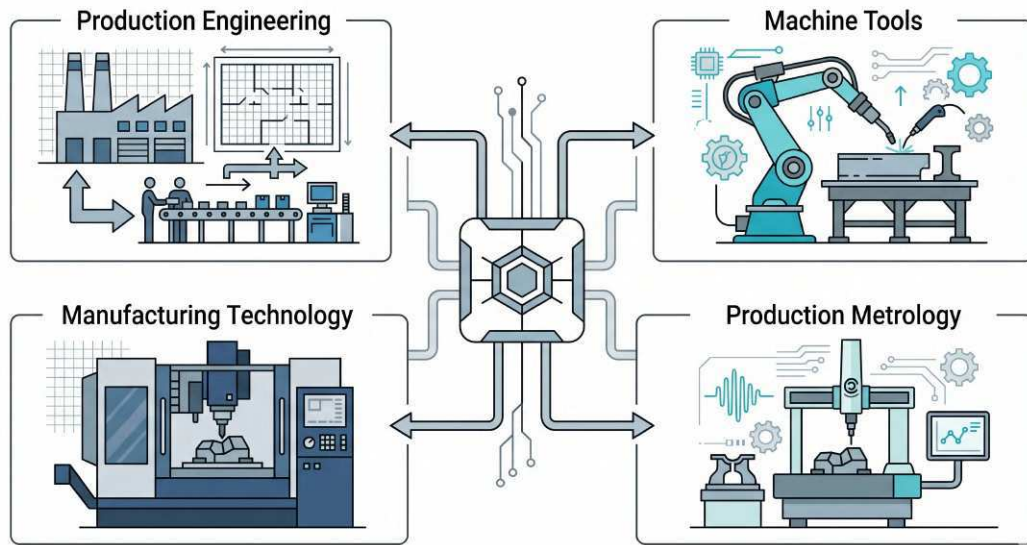
For over a century, the WZL has stood as a globally preeminent research institution at the forefront of production engineering, manufacturing optimization, and industrial automation. The institute operates as a central hub for the “Internet of Production” (IoP) excellence cluster, driving the digital transformation of manufacturing technologies. Its massive research infrastructure is divided into four highly synergistic core chairs, as outlined in Figure 11:

- **Production Engineering:** Focused on global supply chain scaling laws, factory layout planning, assembly line organizational ergonomics, and agile enterprise architectures.
- **Machine Tools:** Specializing in structural machine dynamics, advanced control loop systems, physical drive mechanisms, and the seamless integration of heavy industrial robotics.
- **Manufacturing Technology:** Dedicated to the advanced analytical modeling of subtractive processes (cutting, grinding), metal forming, and additive manufacturing metallurgy.
- **Production Metrology and Quality Management:** Concentrated on inline computer vision, coordinate measuring machines (CMM), sensor fusion, and data-driven quality control loops.

The specific research conducted in this PFC is directly embedded within a broader, long-term global automation initiative at WZL. A persistent challenge in the transition toward Industry 4.0, particularly for small and medium-sized enterprises (SMEs), is the rapid obsolescence of legacy manufacturing equipment (Stock; Seliger, 2016). Furthermore, traditional industrial robots operate on proprietary, closed-source controllers, making custom sensor integration or software upgrades prohibitively expensive and technically restricted due to vendor lock-in (Quigley *et al.*, 2009).

To overcome these barriers, this institutional project aims to extend the economic and operational lifecycles of legacy robotic assets through the concept of technological retrofitting. By applying software-defined architectures and integrating open-source robotic middlewares, most notably the Robot Operating System (ROS), researchers can decouple the physical hardware limitations from advanced control and perception algorithms (Quigley *et al.*, 2009).

Figure 11: Overview of the four core research chairs at WZL, RWTH Aachen University.



Source: Created by the Author.

The ultimate target of this research branch is to create reliable, easily deployable, and vendor-independent machine tending modules. This approach allows agile automation cells to interact seamlessly with standard CNC machine environments without requiring the cost-prohibitive, closed-source hardware overhauls typically demanded by traditional industrial integrators.

3.3 PROBLEM DESCRIPTION AND THE OPEN-LOOP DRIFT CHALLENGE

The core engineering problem addressed in this project centers on the inherent trade-off between flexible perception frameworks and strict spatial repeatability within industrial environments. Modern machine tending requires systems capable of adapting to workpiece misalignments (Kragic; Christensen, 2002), yet CNC interfaces demand strict, sub-millimeter insertion tolerances (Groover, 2015).

When implementing the original online 6D grasping pipeline, where the workpiece pose was estimated dynamically via camera streams and fed into MoveIt's randomized planning algorithms, the end-to-end spatial accuracy was found to be highly unreliable for consistent industrial operations. In theoretical robotics, dynamic path planning assumes ideal sensory inputs; however, in applied scenarios, the cumulative propagation of systemic errors corrupts the Tool Center Point (TCP) precision (Siciliano *et al.*, 2010). The failure of this dynamic strategy was driven by four distinct vectors:

1. **Hand-Eye Calibration Sensitivity:** In vision-guided robotics, the relationship between the camera sensor and the manipulator's base is governed by the hand-eye

calibration matrix. Due to the geometric nature of transformation matrices, minute physical or angular offsets at the camera's mount are amplified as they project outward into the workspace (Tsai; Lenz, 1989). A fraction of a degree in angular calibration error translates into large, millimeter-level lateral TCP deviations at the grasping point.

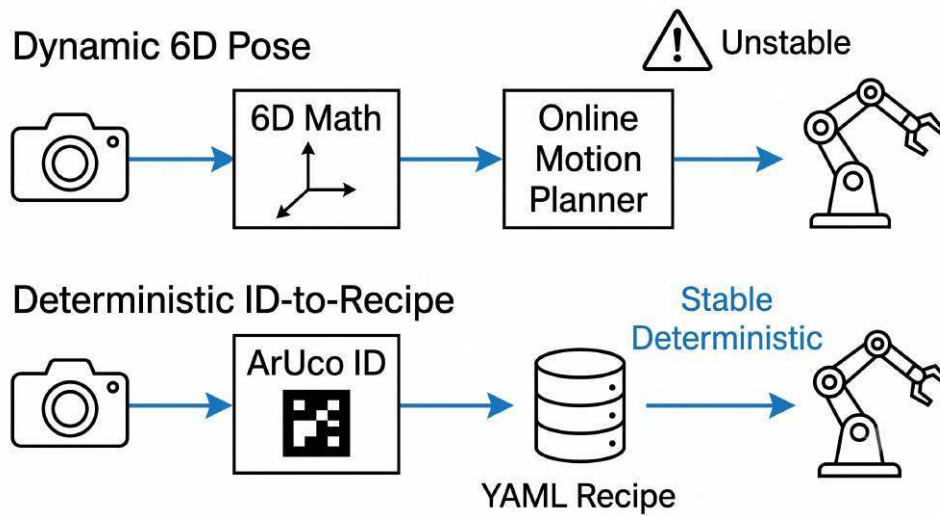
2. **Sensor Signal Degradation:** Industrial environments are optically hostile. The depth and image data inherent to solid-state LiDARs and RGB-D sensors degrade significantly when interacting with the highly reflective, machined metallic surfaces of CNC workpieces (Khoshelham; Elberink, 2012). This introduces severe spatial noise and point-cloud scattering, making continuous 6D pose estimation unstable.
3. **Coordinate Transformation (TF) Jitter:** In ROS-based architectures, spatial frames are broadcasted and managed dynamically through the *tf* system (Quigley *et al.*, 2009). However, due to its asynchronous nature, high-frequency noise and computational latency cause the coordinate frames to dynamically oscillate (jitter) in the virtual environment. If the motion planner samples a pose during a jitter spike, the generated trajectory will be spatially corrupted.
4. **Multiple Kinematic Solutions:** A 6-Degree-of-Freedom (DoF) serial link manipulator possesses multiple inverse kinematics (IK) mathematical solutions to reach a single Cartesian spatial point (Craig, 2005). Online path planners relying on numerical IK solvers may calculate wildly different joint-space configurations for nearly identical consecutive poses. This results in unpredictable, non-repeatable sweeping trajectories that risk catastrophic collisions with the physical CNC enclosure.

To completely isolate the system from these perceptual and mathematical variations, a deterministic **ID-to-Recipe** approach was conceptualized and deployed, as structurally compared in Figure 12. Instead of continuously calculating spatial transforms across $SE(3)$ (the mathematical space of 3D poses), the perception system uses fiducial tracking algorithms solely to extract a discrete, binary numerical identifier (e.g., ArUco marker ID 19) attached to the workpiece (Garrido-Jurado *et al.*, 2014). This discrete ID acts as a database key to pull a pre-recorded, collision-free joint-space recipe structurally archived in a local YAML configuration file.

While this localized strategy successfully isolates the system from perception jitter and eliminates motion-planning latency, it inherently introduces a secondary, critical technical challenge: **Open-Loop Drift**, illustrated in Figure 13.

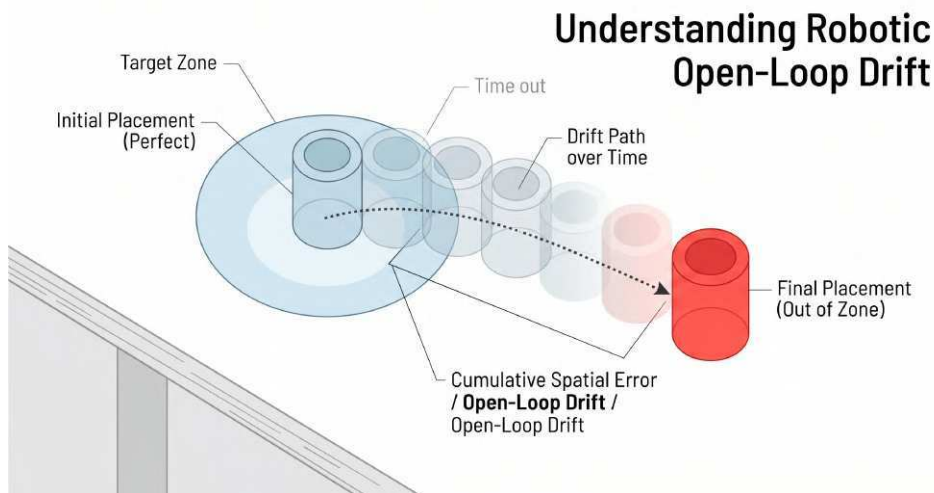
Because the manipulator executes pre-recorded joint angles blindly without real-time visual servoing or force-torque feedback, the system operates in an open control loop during the execution phase. Consequently, any microscopic physical displacement,

Figure 12: Continuous 6D pose estimation vs. deterministic ID-to-Recipe pipeline.



Source: Created by the Author.

Figure 13: Open-loop spatial decay: error accumulation across pick-and-place cycles.



Source: Created by the Author.

mechanical slipping, or tolerance shifting of the workpiece upon its release by the gripper is left entirely uncorrected by the software. Over successive, repeated cycles of loading and unloading the CNC table, these millimetric mechanical misalignments stack up. In long-run continuous execution, this open-loop spatial decay degrades the localized grasping tolerance, eventually leading to part misplacement, failure to engage the CNC fixtures, or a total breakdown of the automation cycle.

3.4 TECHNICAL CONSTRAINTS

The operational success of an open-loop, deterministic architecture relies entirely on the stability of its initial conditions and the precision of its execution limits.

Consequently, the system is governed by two major categories of technical constraints: the sensitivity of the visual perception pipeline and the strict mechanical tolerances demanded by the CNC interface.

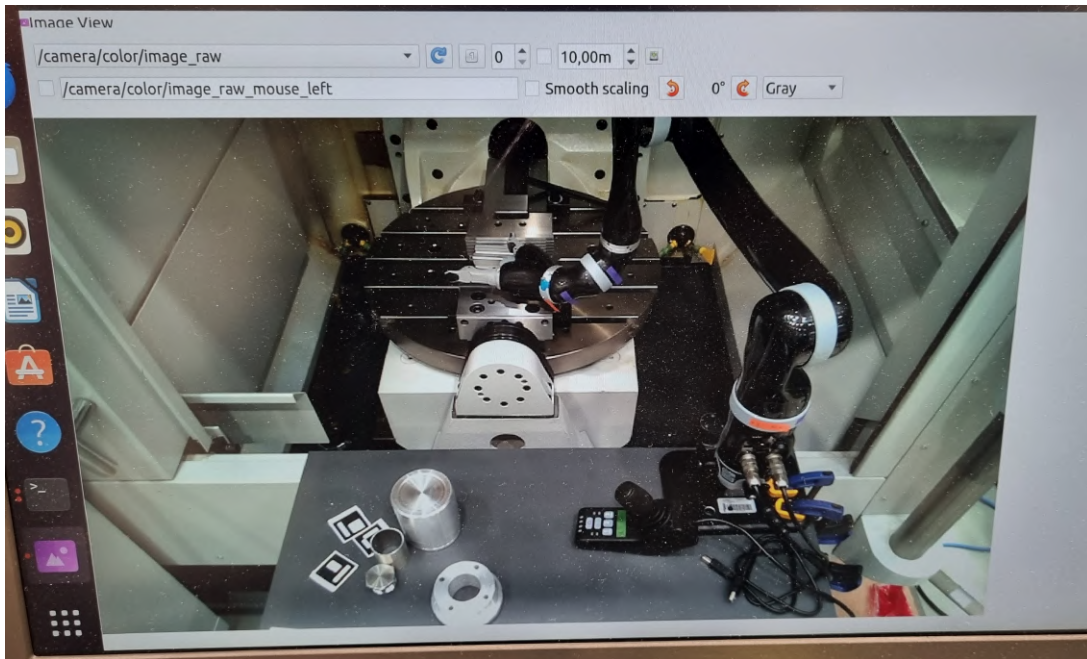
3.4.1 Perception Pipeline Sensitivity

The perception subsystem must process high-resolution visual data to guarantee correct workpiece identification while remaining entirely resilient against the harsh environmental hazards typical of a machining facility (Golnabi; Asadpour, 2007). Because the robot executes its trajectory blindly after the initial identification, any failure in the perception layer propagates critically through the entire cycle. The primary technical constraints governing this pipeline include:

- **Identification Reliability:** The system cannot tolerate misidentification errors (e.g., mistaking ID 19 for a different part variant). This requires highly constrained binary decoding thresholds within the `aruco_ros` package to filter out false positives. The fiducial dictionary must be robust enough to guarantee correct recipe selection even in suboptimal conditions.
- **Environmental Occlusions and Illumination:** Industrial shop floors present optically hostile environments. The fiducial tracking algorithms must remain stable under severe lighting variations and specular glare reflections bouncing off machined metallic surfaces. Furthermore, the vision system must successfully decode markers (previously demonstrated in Figure 7) despite partial occlusions caused by stray industrial cutting fluids, dust, or metallic chips.
- **Rigidity of Spatial Coordinate Frames:** To ensure that the spatial relationship between the camera and the workpiece is interpreted correctly, the localized transformation tree must maintain absolute geometric rigidity without dynamic drift. In ROS, this is managed by the TF (Transform) library, which tracks multiple coordinate frames over time (Foote, 2013). The system relies on a strict, static coordinate inheritance tree (as mapped in Figure 5) that links structural transforms from the global world baseline to the camera's optical core:

$$\text{world} \longrightarrow \text{base_link} \longrightarrow \text{camera_link} \longrightarrow \text{camera_color_optical_frame} \quad (10)$$

Figure 14: ROS TF Coordinate Frame inheritance tree detailing the static transformation mapping.



Source: Created by the Author.

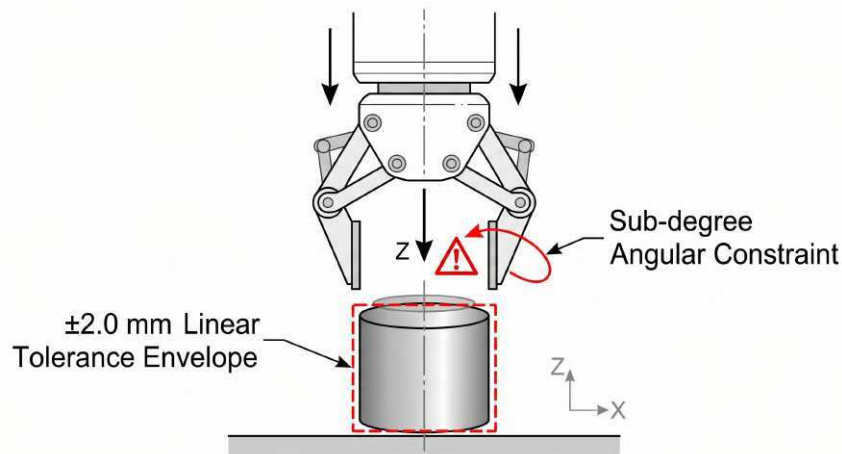
3.4.2 Repeatability and Tolerance Demands

Interfacing with high-precision CNC machining centers demands strict physical and spatial operational boundaries. Because the Kinova Jaco2 executes the pre-recorded YAML recipes without active force-torque feedback (compliance control), the mechanical tolerances dictate the success of the loading phase:

- **Grasping Tolerance Envelope:** The mechanical gripper jaw assembly possesses a narrow physical compliance window. Based on empirical project data, linear deviations at the Tool Center Point (TCP) exceeding ± 2.0 mm prevent proper finger engagement, as modeled in Figure 15. As established in flexible manufacturing theory, exceeding such tolerance envelopes generates catastrophic part drops or cross-threading during fixture insertion (Bi; Zhang, 2001).
- **Angular Alignment Constraints:** Rotational drift along the manipulator's spherical wrist joint axes (J_5 and J_6) must be heavily restricted. As established in classical robotics theory, small variations in relative position during constrained insertions can cause mechanical jamming and generate excessive contact forces (Craig, 2005). In the specific context of this project, even sub-degree angular misalignment when embedding the metallic part into the rigid machine chocks result in structural torque spikes that trigger the robot's emergency safety stops.

- **Trajectory Shaping:** Automated sampling-based planners often produce erratic or unnecessarily complicated paths that are risky in confined industrial spaces (LaValle, 2006). To mitigate kinematic redundancy and prevent these unpredictable arm sweeps, the operation was structured following a formal manipulation planning framework, decoupling the motion into distinct transit and transfer phases. Within this architecture, multiple intermediate waypoints, specifically approach, pre-grasp, retreat, and wrist unwinding poses, were manually authored. This deterministic shaping explicitly dictates the intended route and guarantees a collision-free envelope within the confined CNC structure.
- **Global Home Synchronization:** The robot must initiate and terminate every automated loop at a predefined, globally calibrated kinematic **Home** joint configuration. This environmental resetting ensures that any joint-space drift accumulated during the open-loop execution does not carry over into the subsequent cycle, maintaining a repeatable baseline.

Figure 15: TCP tolerance envelope and angular constraints for open-loop insertion.



Source: Created by the Author.

3.5 HARDWARE AND ENVIRONMENT SETUP

The physical and virtual deployment environment relies on an integrated, custom-retrofitted system architecture. Bridging the gap between legacy physical mechanisms and modern perception algorithms requires a robust, deterministic hardware-software interface. The detailed technical inventory and operational breakdown of the developed machine tending cell are organized in Table 1.

The primary physical actuator of the cell is a Kinova Jaco2 6-Degree-of-Freedom (DoF) serial link articulated manipulator (Model: j2n6s300). Originally developed primarily for assistive robotics and lightweight human-robot interaction (Campeau-Lecours

Figure 16: Original assistive wheelchair configuration of the Kinova Jaco2.



Source:(Kinova Robotics, 2026)

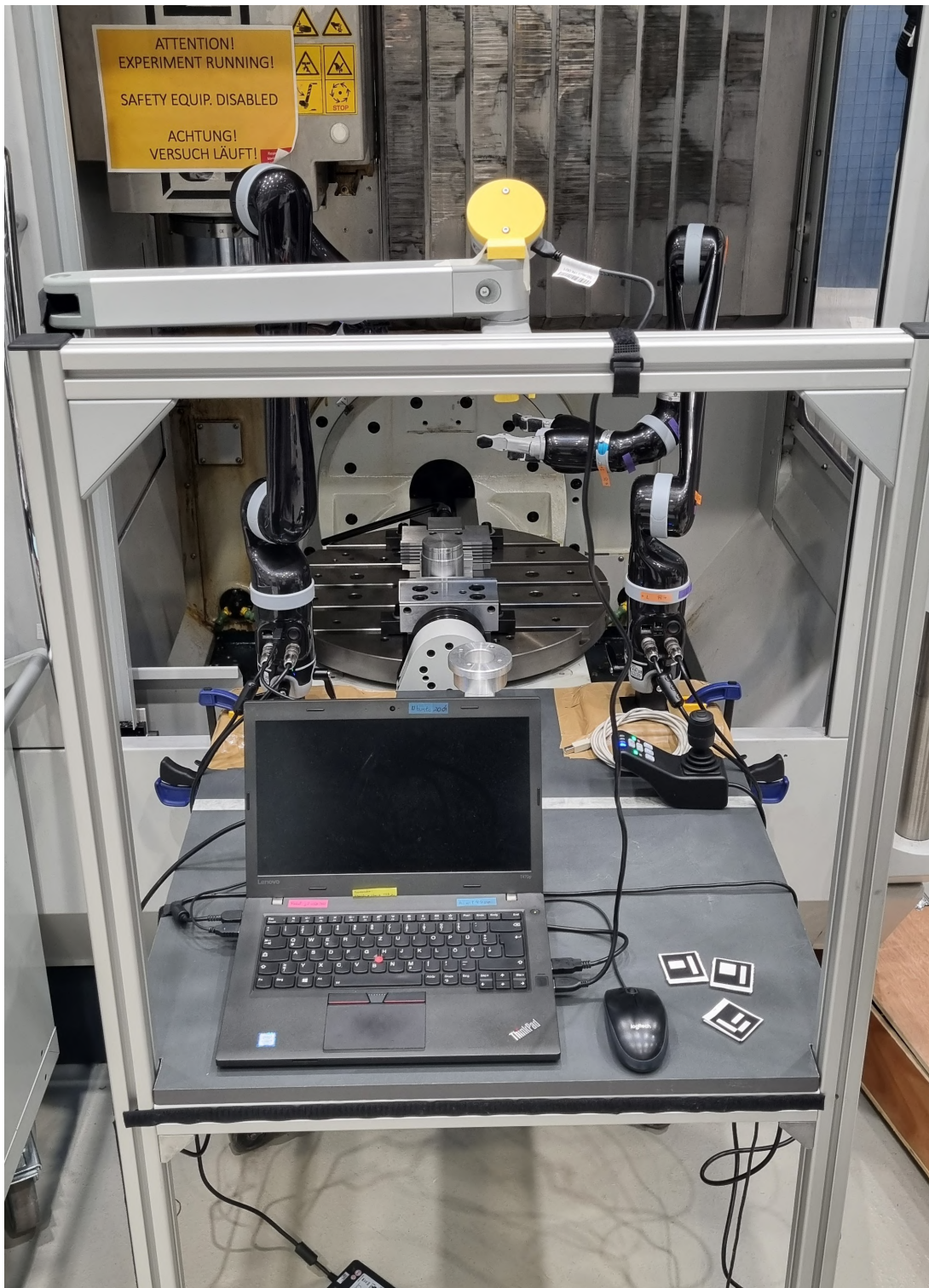
et al., 2017), this specific legacy arm was retrofitted for an industrial advanced manufacturing application. The contrast between its original assistive design paradigm (Figure 16) and the actual vision-guided CNC machine tending cell deployment at WZL (Figure 17) highlights the extent of this structural adaptation. To capture the workspace environment, the system utilizes an Intel RealSense L515 depth camera. Unlike traditional stereo vision cameras, the L515 utilizes solid-state LiDAR technology, which provides highly accurate, low-noise point clouds and RGB streams at close ranges, making it ideal for precision fiducial tracking in structured environments (Curto; Araujo, 2022).

Both the manipulator and the vision sensor interface with the host workstation, an industrial x86_64 PC running a native Linux Ubuntu 20.04 LTS operating system, via a dedicated USB 3.0 bus topology to prevent data bottlenecks and ensure high-bandwidth transmission, as schematically mapped in the hardware architecture blueprint in Figure 18.

To decouple the hardware from proprietary constraints and enable flexible perception-to-action pipelines, the system relies on the Robot Operating System 1 (ROS 1 Noetic Ninjemys) as its core middleware (Quigley *et al.*, 2009). ROS provides a standardized, graph-based communication framework that connects the isolated hardware drivers. Python 3.8 serves as the primary interpreter for the custom ID-to-Recipe control nodes.

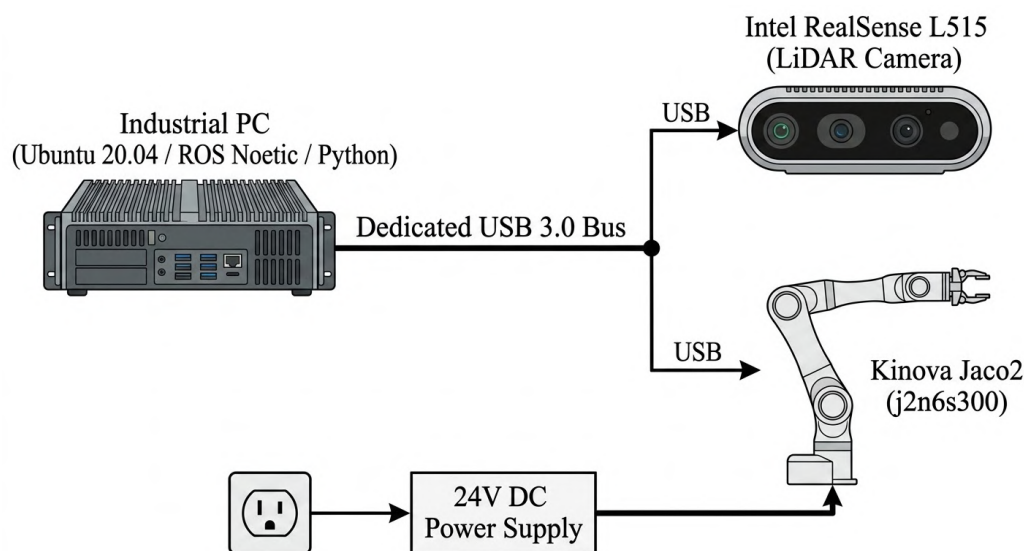
Within this ecosystem, the `realsense2_camera` node handles the continuous streaming of RGB and depth fields, feeding data directly to the `aruco_ros` tracking node (Garrido-Jurado *et al.*, 2014), which is responsible for geometric extraction and discrete ID publishing. Bypassing the latency of online motion planners, the custom execution

Figure 17: Retrofitted CNC tending cell at WZL using a vision-guided Jaco2.



Source: Created by the Author.

Figure 18: Hardware topology: USB 3.0, power, and peripherals linking sensors, manipulator, and workstation.



Source: Created by the Author.

scripts bypass Movelt (Chitta; Sucan; Cousins, 2012), which is strictly relegated to static visualization and safety debugging via RViz, and send targeted positional arrays directly to the low-level Kinova ROS Driver for immediate joint execution.

Table 1 – Complete Hardware and Software Specifications

Layer	Component Package /	Technical Specifications and Operational Role
Hardware	Manipulator Arm	Kinova Jaco2 6-DoF Serial Link (Model: j2n6s300).
	Vision Sensor	Intel RealSense L515 Solid-State LiDAR Depth Camera.
	Host Workstation	Industrial PC running native Linux Ubuntu 20.04 LTS (x86_64).
	Physical Interface	Dedicated USB 3.0 Bus Topology for high-bandwidth data.
Software	Middleware Core	Robot Operating System 1 (ROS 1 Noetic Ninjemys).
	Interpreter	Python v3.8 Environment.
	Vision Driver	realsense2_camera node (RGB + PointCloud2 fields).
	Tracking Node	aruco_ros (Fiducial extraction and discrete ID publishing).
	Diagnostics / UI	Movelt + RViz (Strictly for static visualization and safety debugging).
	Control Driver	Kinova ROS Driver (Low-level joint position command injection).

Critical Laboratory Restriction Note: In industrial automation research, the deployment of multi-arm collaborative systems is often explored to increase throughput. AI-

though a secondary, identical Kinova Jaco2 6-DoF manipulator arm was physically available within the laboratory workspace to execute parallel evaluations, it had to be completely excluded from the project's operational scope.

Empirical diagnostic torque validation testing conducted during the early phases of this project revealed severe internal hardware degradation within the unit. Specifically, the secondary arm exhibited an insufficient torque and force generation capacity inside one of its mid-link integrated joint actuators. In legacy robotic systems subjected to prolonged operational lifecycles, actuator wear and gear fatigue can severely degrade payload capacity (Bittencourt, 2014). This mechanical limitation rendered the secondary arm entirely incapable of safely executing or holding the high-load angular trajectories required to lift and manipulate heavy metallic CNC workpieces, restricting the scope of the experimental validation strictly to a single, fully functional manipulator arm.

3.6 TECHNICAL REQUIREMENTS

As established in classical software engineering, technical requirements translate abstract operational problems into measurable and testable specifications (Sommerville, 2011). Applied to robotic systems engineering, establishing these baselines ensures that the automated cell behaves predictably under the strict constraints of a machining facility. Following fundamental engineering principles, the system requirements are categorized into Functional Requirements (FR), which dictate the specific behaviors and data transformations the software must execute, and Non-Functional Requirements (NFR), which govern the quality metrics, environmental constraints, and architectural standards of the operation (Sommerville, 2011).

A synthesized overview of all functional and non-functional requirements governing the proposed automation architecture is presented in Table 2.

Table 2 – Summary of System Requirements for the CNC Machine Tending Cell.

ID	Type	Brief Description
FR1	Functional	Extract discrete integer IDs from ArUco markers within a 1.5m range.
FR2	Functional	Parse YAML configuration files using the extracted ID to retrieve joint and gripper sequences.
FR3	Functional	Bypass Cartesian planners to send joint-space position vectors directly to the Kinova driver API.
FR4	Functional	Enforce manually authored intermediate waypoints to guarantee deterministic, collision-free motion paths.
FR5	Functional	Synchronize gripper actuation with the completion of specific positional milestones.
FR6	Functional	Implement explicit dwell times (<code>wait_after</code>) at waypoints to physically dissipate kinetic oscillations.
NFR1	Non-Functional	Ensure strict path repeatability by initiating and concluding all sequences at a calibrated Home baseline.
NFR2	Non-Functional	Maintain native backward compatibility with ROS 1 Noetic and legacy hardware drivers.
NFR3	Non-Functional	Adhere strictly to PEP8 formatting standards and 4-space indentation to prevent runtime errors.
NFR4	Non-Functional	Synchronize distributed networks using hardware-level time clocks to prevent transformation (TF) errors.
NFR5	Non-Functional	Limit systemic processing latency from ID detection to initial joint actuation to a maximum of 500 ms.
NFR6	Non-Functional	Execute immediate autonomous halting during power failures to mitigate laboratory security vulnerabilities.

3.6.1 Functional Requirements (FR)

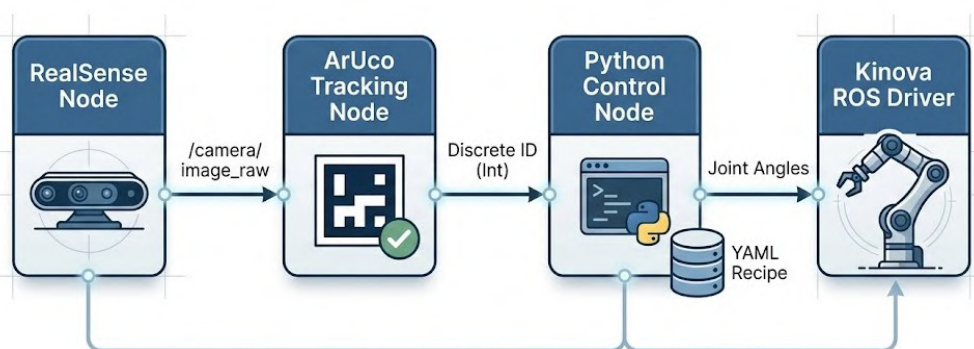
Functional requirements define the core operational capabilities that the developed system must actively perform to complete the machine tending cycle:

- **FR1 - ArUco Tag Decoding:** The perception system must process high-frequency RGB video frames to isolate ArUco markers and successfully decode their discrete integer IDs. To ensure operational safety, this tracking pipeline must maintain reliable detection within a physical range of up to 1.5 meters from the camera lens.
- **FR2 - Recipe Parsing and Mapping:** The control software must use the extracted numerical ID as an index key to parse a local structured YAML configuration file. The software must translate this data structure into an ordered sequential list of joint-space targets (in degrees) and auxiliary end-effector triggers.

- **FR3 - Direct Joint-Space Position Control:** The execution node must deliberately bypass external online Cartesian planning libraries. Instead, it must inject the targeted positional vectors directly into the low-level motor drivers via the Kinova ROS driver API, thereby minimizing processing jitter and planning latency.
- **FR4 - Trajectory Safety Profiling:** Drawing on formal manipulation planning frameworks that structurally subdivide operations into distinct motion phases, the control script must dictate mandatory, manually authored intermediate waypoints, specifically approach, pre-grasp, retreat, and wrist unwinding steps. This deterministic, step-by-step profiling guarantees collision-free motion paths around the rigid CNC chassis and mitigates kinematic singularity zones.
- **FR5 - Synchronized End-Effector Triggering:** The automation node must actuate the manipulator's fingers (triggering binary open/close states) in strict temporal coordination with the completion of specific positional recipe milestones.
- **FR6 - Mechanical Damping Buffers:** Legacy manipulator links suffer from kinetic oscillation upon suddenly reaching a target due to structural compliance and joint backlash. The control sequence must actively halt motor execution at each recorded waypoint for a user-defined dwell time (`wait_after`). This explicit time buffer allows the physical dissipation of vibrations before the next movement is initiated, ensuring TCP stability during precision tasks.

The orchestration of these functional requirements forms a deterministic, unidirectional software pipeline. As illustrated in the high-level ROS data flow diagram (Figure 19), the architecture ensures that high-bandwidth visual data is immediately distilled into a lightweight, actionable control vector, bypassing computationally heavy motion planners.

Figure 19: ROS software architecture: the sensor-to-actuator pipeline.



Source: Created by the Author.

3.6.2 Non-Functional Requirements (NFR)

Non-functional requirements dictate the systemic behaviors, operational boundaries, environmental safety, and software quality metrics of the automation cell:

- **NFR1 - Determinism and Path Repeatability:** Following the principles of industrial repeatability criteria (International Organization for Standardization, 1998), the control architecture must enforce an exact geometric path match across identical recipe executions. The system must obligatorily initiate and close every routine at a fixed, globally calibrated **Home** baseline vector to reset potential joint drift.
- **NFR2 - Software Legacy Compatibility:** To preserve strict backward compatibility with the manufacturer's original 2017 hardware driver libraries, the entire software ecosystem must compile and execute seamlessly within a native Robot Operating System (ROS) architecture (Quigley *et al.*, 2009), specifically utilizing the ROS 1 Noetic distribution.
- **NFR3 - Syntactical Uniformity and Code Quality:** All developed Python automation scripts must adhere strictly to PEP8 formatting standards (Rossum; Warsaw; Coghlan, 2001). The codebase must eliminate mixed tab-and-space indentations, enforcing a strict 4-space indent rule to prevent unexpected interpreter runtime crashes during active production loops.
- **NFR4 - Time and Clock Synchronization:** The distributed communication network must run on unified system hardware time clocks (`use_sim_time = false`). This hardware-level synchronization ensures that coordinate transform (TF) data packets maintain valid temporal stamps, preventing stale look-up drops and extrapolation errors during the perception phase (Foote, 2013).
- **NFR5 - Operational Processing Latency:** The systemic latency—measured from the instant a valid ArUco ID tag is decoded to the mechanical triggering of the initial joint movement leaving the Home position—must not exceed 500 milliseconds.
- **NFR6 - Environmental Security and Failsafe Halting:** In the event of a facility power failure, the robotic cell must immediately execute a software halt and freeze all joint commands. This localized failsafe is mandatory because the laboratory's security infrastructure relies on backup batteries with low autonomy, which causes the main access doors to unlock during outages, thereby compromising the physical perimeter of the workcell. The system must not resume autonomous operation until manually reset by an operator.

4 PROPOSED SOLUTION AND METHODOLOGY

This chapter details the proposed solution to the machine tending automation problem, outlining the shift from a continuous 6D pose estimation strategy to a deterministic “ID-to-Recipe” architecture. It technically justifies how this methodology bypasses perception-induced coordinate jitter and kinematic redundancy, fulfilling the project’s technical requirements for the CNC workcell.

4.1 SYSTEM ARCHITECTURE

The initial approach of employing a pose-based grasping strategy, where the workpiece pose is estimated online from ArUco markers and the robot plans motions directly to that 6D pose, was proven empirically unreliable for this specific project. As established in classical manipulator kinematics (Craig, 2005), small inaccuracies in the perception-to-robot pipeline, including camera-to-robot calibration sensitivity and transformation (TF) alignment offsets, mathematically propagate into noticeable Tool Center Point (TCP) errors. Because the CNC tending task requires tight tolerances and repeatability over many cycles, the system architecture abandoned this absolute coordinate reaching in favor of a simpler, discrete ID-driven approach.

To systematically justify this structural transition, the failure modes of the continuous 6D pose-based strategy must be analyzed through the lens of homogeneous transformation compounding. In a traditional vision-guided robotic (VGR) closed-loop pipeline, the target pose of the workpiece relative to the manipulator’s base frame, denoted as T_{base}^{object} , is computed via an explicit spatial transformation chain, visually represented in Figure 20:

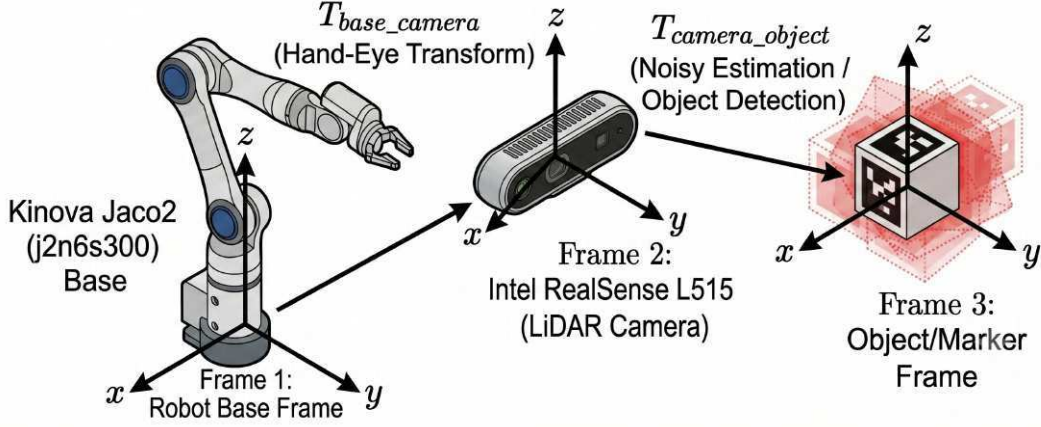
$$T_{base}^{object} = T_{base}^{camera} \cdot T_{camera}^{marker} \cdot T_{marker}^{object} \quad (11)$$

where T_{base}^{camera} represents the hand-eye calibration matrix, T_{camera}^{marker} is the extrinsic pose matrix estimated online by the fiducial marker tracking algorithm, and T_{marker}^{object} is a static geometric offset matrix.

In practical industrial settings, each matrix in this kinematic chain introduces stochastic and systematic variances. The hand-eye calibration matrix (T_{base}^{camera}), typically solved through mathematical optimizations such as the Tsai-Lenz formulation (Tsai; Lenz, 1989), suffers from residual alignment errors due to subtle mechanical misalignments or focal length approximations. Concurrently, the online estimation of T_{camera}^{marker} is highly susceptible to high-frequency Cartesian jitter caused by pixel quantization, changing lighting conditions within the CNC workcell, and solid-state LiDAR depth noise originating from the Intel RealSense L515 sensor. When these transformation matrices are multiplied as structured in Equation 11, their associated covariance

Figure 20: Kinematic chain (Eq. 11): red cloud denotes propagated Cartesian jitter.

$$T_{base_object} = T_{base_camera} \times T_{camera_object}$$



Source: Created by the Author.

matrices compound non-linearly. This mathematical propagation yields spatial variations at the end-effector that frequently exceed the tight physical insertion tolerances mandated by the CNC clamping jaws.

Furthermore, converting this noisy Cartesian target T_{base}^{object} into reachable actuator coordinates requires solving the inverse kinematics (IK) problem. For a 6-DoF manipulator with kinematic redundancy or intersecting axes like the Kinova Jaco2, the inverse differential kinematics mapping relies on the analytical or numerical inversion of the robot Analytical Jacobian matrix (J). Near numerical singularities or regions of high kinematic sensitivity, the inversion operation:

$$\Delta\theta = J^{-1}(\theta)\Delta x \quad (12)$$

amplifies minor Cartesian tracking noises (Δx) into massive, unpredictable joint velocity fluctuations ($\Delta\theta$) (Siciliano *et al.*, 2010). This structural instability results in erratic trajectories, dynamic joint oscillation, and unsafe planning failures within the Movelt framework, rendering continuous online path generation unviable for high-duty-cycle industrial manufacturing.

To circumvent these mathematical and physical limitations, this work implements a paradigm shift named the “ID-to-Recipe” architecture. This design completely divorces the computer vision tracking layer from the real-time motion planning layer. Instead of treating the visual feed as a continuous geometric tracking resource, the perception pipeline is utilized strictly as a discrete state classification interface. The spatial coordinates of the transformation matrix are discarded during execution; only the discrete ArUco marker identifier (ID) is robustly decoded and preserved (Garrido-Jurado *et al.*, 2014). This ID acts as a deterministic key to fetch a pre-optimized, collision-free joint-

space pathway (the “recipe”) from a local database, guaranteeing absolute tracking stability and removing coordinate jitter entirely.

Table 3 – Architectural comparison between Continuous 6D Pose Estimation and the Proposed ID-to-Recipe Strategy.

Feature	Continuous 6D Pose (MoveIt)	ID-to-Recipe (Proposed)
Perception Role	Continuous spatial tracking (SE3)	Discrete topological ID matching
Trajectory Generation	Online, dynamic (OMPL/RRT)	Offline, static (Manual Waypoints)
Execution Domain	Task-space (Cartesian $\rightarrow$ IK)	Joint-space (Absolute encoders)
Error Propagation	High (Calibration + Depth Noise)	None (Bypasses Cartesian matrices)
Kinematic Risk	High (Singularities, elbow collisions)	Zero (Deterministic absolute paths)

4.1.1 Data Flow: From ArUco ID to Joint Execution

The proposed architecture utilizes the perception pipeline strictly for discrete state classification rather than continuous spatial tracking. The Intel RealSense L515 camera provides high-resolution RGB frames to the core perception node, leveraging the OpenCV-based `aruco_ros` package. Instead of relying on the camera’s intrinsic matrix to solve the Perspective-n-Point (PnP) problem—which iteratively estimates the 6D pose of the marker relative to the optical center and was empirically found to be highly sensitive to depth noise—the algorithm is simplified to a purely topological detection phase. The ArUco node applies adaptive thresholding, extracts square contours, and decodes the inner binary grid of the marker (Garrido-Jurado *et al.*, 2014). Utilizing inter-marker distance for error correction, the node outputs a robust, unique integer identifier (e.g., ID 19). Consequently, the perception output collapses from a noisy $SE(3)$ transformation matrix into a stable, discrete variable $ID \in \mathbb{Z}^+$.

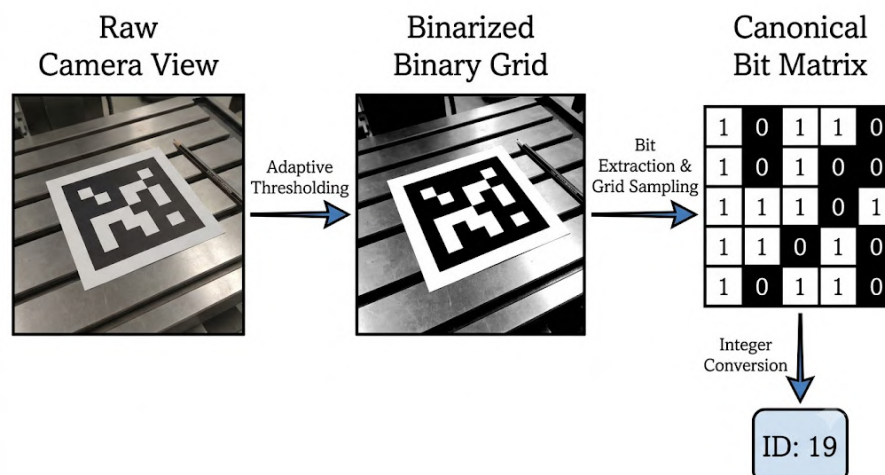
This detected ID acts as a deterministic trigger within the system’s control logic. It is immediately used as a lookup key to query a corresponding YAML recipe file. Mathematically, instead of utilizing inverse kinematics to map a Cartesian target to a joint configuration $f_{IK} : \mathbf{x} \rightarrow \boldsymbol{\theta}$, the system establishes a discrete mapping function. The YAML file provides an ordered set of pre-validated joint configurations, $\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_n\}$, where each $\boldsymbol{\theta}_i \in \mathbb{R}^6$ represents a specific waypoint in the machine tending cycle, accompanied by specific end-effector states.

By severing the continuous feedback loop between the vision system and the motion planner, the system categorically eliminates the Cartesian jitter inherent to asynchronous sensor updates in the ROS `/tf` tree (Foote, 2013). Furthermore, this data flow intentionally bypasses dynamic trajectory planning during runtime. In standard ROS 1 architectures, point-to-point navigation typically invokes the MoveIt framework, relying on algorithms like the Open Motion Planning Library (OMPL) for real-time collision checking and Time-Optimal Trajectory Generation (TOTG) (Chitta; Sucan; Cousins, 2012). While powerful, online planning introduces computational latency and non-deterministic path execution, as randomized planners (e.g., RRT*) may yield slightly different joint trajectories for the same Cartesian goal in successive cycles.

To ensure the strict repeatability required for loading and unloading the CNC machine, the custom execution node routes the target arrays directly to the Kinova ROS driver interface. The driver receives these direct joint commands—often structured as standard `trajectory_msgs/JointTrajectory` messages or via specific action servers—and handles the low-level motor control via embedded spline interpolation. Relying on formal trajectory generation models, such as quintic polynomials (Biagiotti; Melchiorri, 2008), this internal interpolation guarantees smooth velocity and acceleration profiles between the taught waypoints. This methodology restricts the high-level ROS environment to a purely supervisory and sequencing role, delegating the physical execution to the deterministic, closed-loop joint controllers embedded in the Kinova Jaco2 hardware.

The visual processing pipeline transitioning from the raw camera feed to the discrete integer ID is illustrated in Figure 21. By applying adaptive thresholding and decoding the canonical bit matrix, the system completely avoids depth noise calculation, yielding the highly stable identifier utilized by the deterministic control loop.

Figure 21: Data extraction: from RealSense RGB to binarized grid and canonical bit matrix.



Source: Created by the Author.

4.1.2 Modular YAML Recipe System

To ensure modularity and maintainability, the sequence data is strictly isolated from the execution logic, adhering to the fundamental software engineering principle of Separation of Concerns (SoC). In traditional robotic deployments, hardcoding spatial waypoints directly into the source code (e.g., within C++ or Python ROS nodes) creates a monolithic architecture. This requires software recompilation and node restarts every time a robotic trajectory needs adjustment, which is highly inefficient in Reconfigurable Manufacturing Systems (RMS) where production batches change frequently (Koren *et al.*, 1999).

To solve this, the proposed system utilizes YAML (YAML Ain't Markup Language), a human-readable data serialization standard. Each ArUco ID acts as a primary key associated with a YAML entry containing a named sequence. By moving the configuration to an external YAML database, shop-floor operators without advanced programming expertise can teach or modify the robot's pick-and-place routines simply by editing a text file, fulfilling the flexibility requirements of the modern CNC manufacturing cell.

Each step within this YAML sequence is defined by three critical parameters: the target joint angles (`joints_deg`), optional end-effector commands (`fingers`), and a discrete dwell time (`wait_after`), as demonstrated in Listing 4.1.

```

1 markers:
2   "19":
3     type: sequence
4     description: "Optimized Sequence Robo2 - PICK UP Small round piece"
5     sequence:
6       - name: "1 - Home"
7         joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
8         fingers:    [0.0, 0.0, 0.0]
9         wait_after: 1.0
10
11      - name: "1.5 - Approximation"
12        joints_deg: [286.0, 190.0, 71.0, 240.0, 81.0, 77.0]
13        fingers:    [0.0, 0.0, 0.0]
14        wait_after: 0.1

```

Listing 4.1 – Snippet of the ID-to-Recipe YAML structure, detailing the initial approach sequence for ArUco ID 19.

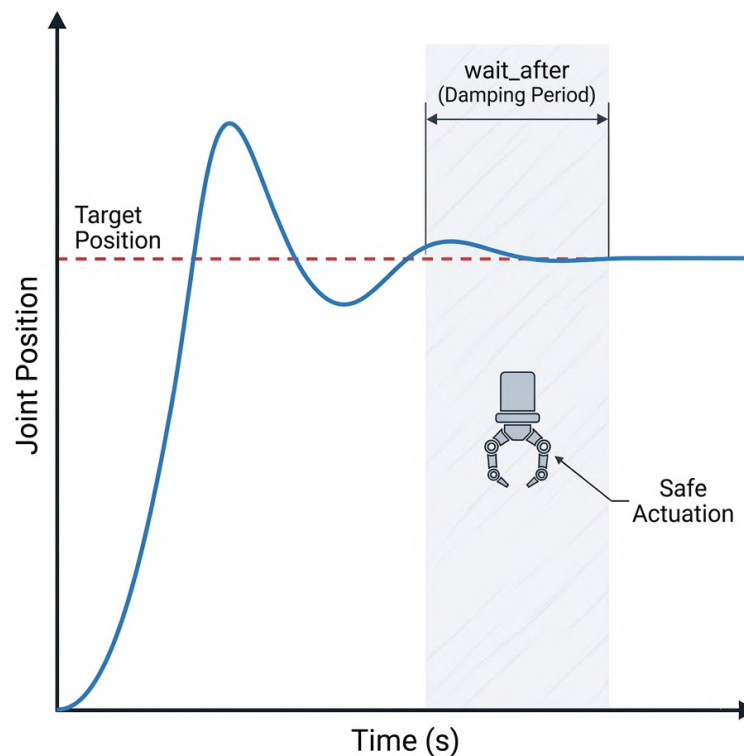
The `joints_deg` parameter explicitly defines the target in the joint-space domain. While the Robot Operating System (ROS) natively operates using radians internally for standard message structures like `sensor_msgs/JointState` (Quigley *et al.*, 2009), the YAML configuration exposes these variables in degrees. This deliberate design choice maximizes human readability and simplifies the manual transcription of coordinates from the teach-pendant or RViz interface into the recipe file. The custom Python execu-

tion node automatically handles the mathematical conversion from degrees to radians before populating the ROS trajectory messages.

The `wait_after` parameter serves a critical electromechanical function. The Kinova Jaco2 manipulator utilizes harmonic drive gear reducers, which, while offering zero backlash and high reduction ratios, inherently introduce joint elasticity. When a manipulator with flexible joints executes a fast point-to-point motion and suddenly stops, the kinetic energy of the links causes residual mechanical vibrations, a phenomenon known as kinetic oscillation or overshoot (Spong; Hutchinson; Vidyasagar, 2006).

If the gripper is commanded to close on a CNC workpiece while the end-effector is still oscillating, the dynamic slip can cause severe misalignment or part dropping. As illustrated in the step response graph in Figure 22, the `wait_after` parameter acts as a programmable settling time (e.g., 0.1 or 1.0 seconds), guaranteeing that the control system waits for the mechanical vibrations to naturally dampen out before transitioning to the next state or actuating the fingers.

Figure 22: Step response: `wait_after` dwell time allows residual oscillations to settle before tool actuation.



Source: Created by the Author.

Through this modular structure, the YAML recipe system not only abstracts the complexity of ROS trajectory generation but also provides a configurable mechanism to safely manage the physical dynamics of the manipulator within tight CNC tolerances.

(The full implementation of the execution node architecture, including driver initialization and action server parsing, is available in the Appendices).

4.2 WAYPOINT AUTHORIZING AND PATH SHAPING

Because the cycle relies on fixed, pre-recorded joint targets, small placement errors after each release may accumulate over repeated loops. To mitigate this open-loop drift and ensure a collision-free operation within the dense CNC environment, the trajectory is rigidly shaped through taught waypoints. In the absence of closed-loop visual servoing to continuously correct the end-effector pose, deterministic path shaping becomes the primary safeguard against mechanical collisions and task failure.

4.2.1 Manual Joint Configuration Recording

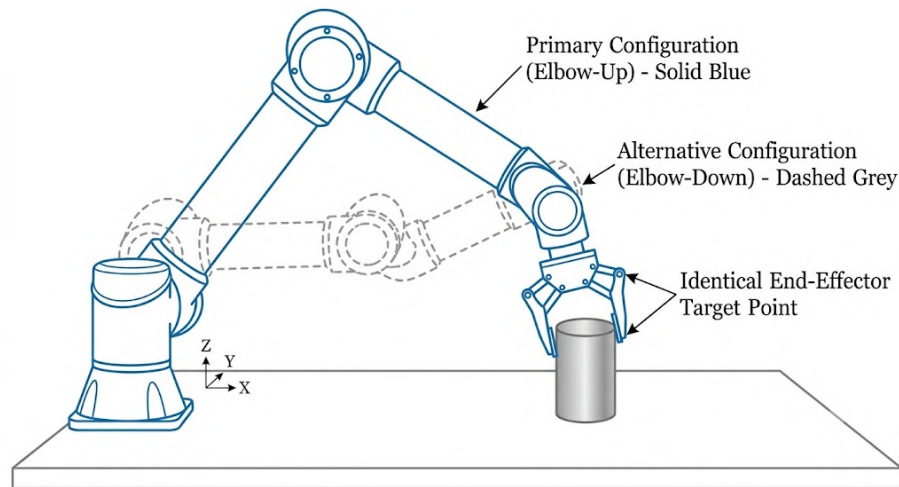
In standard industrial robotics, the methodology of manually driving the manipulator to desired poses and saving the joint coordinates is known as Teach Programming or Lead-through Programming (Groover, 2015). For this project, waypoints were authored manually to enforce a specific and highly constrained motion path. The manipulator was teleoperated using a joystick, allowing the operator to rely on human spatial intuition to navigate the dense, obstacle-rich environment of the CNC workcell.

The fundamental engineering justification for this approach lies in the circumvention of the Inverse Kinematics (IK) problem during runtime. For a 6-DoF manipulator like the Kinova Jaco2, mapping a Cartesian task-space target ($\mathbf{x} \in SE(3)$) to a joint-space configuration ($\theta \in \mathbb{R}^6$) is a non-linear mapping that often yields multiple valid mathematical solutions (e.g., “elbow-up” versus “elbow-down” configurations), as illustrated in Figure 23 (Siciliano *et al.*, 2010). In a confined CNC environment, allowing an online motion planner to dynamically select an alternative IK solution can cause the manipulator’s elbow or shoulder links to sweep unexpectedly, leading to catastrophic collisions. By recording and commanding absolute joint angles ($\theta_1, \dots, \theta_6$), the system enforces a unique, deterministic kinematic posture, entirely eliminating the risk of unpredicted structural reconfigurations.

Furthermore, relying on human teleoperation for path generation bypasses the need for high-fidelity 3D environmental modeling. Dynamic motion planners require continuous collision checking against a simulated environment (e.g., using OctoMaps or detailed URDF collision meshes). By manually shaping the path, the human operator intuitively accounts for unmodeled physical constraints, such as the dangling cables of the RealSense camera or the exact geometric boundaries of the CNC door, which are computationally expensive to model and maintain in real-time.

During the teleoperation phase, the current joint configuration was continuously monitored via ROS visualization tools. Once a safe and functionally correct pose was

Figure 23: Resolving IK multiplicity: absolute joint angles prevent unpredictable configuration flips inside the confined CNC.



Source: Created by the Author.

achieved physically, the true encoder values published by the hardware were transcribed into the YAML sequence file. This approach guarantees that the robot exactly replicates the taught configuration during execution. By locking the trajectory to absolute joint-space coordinates, the system leverages the intrinsic mechanical repeatability of the hardware, which adheres to standard industrial performance criteria (International Organization for Standardization, 1998), rather than relying on the non-deterministic output of stochastic motion planners like RRT or PRM (Kavraki *et al.*, 1996). Consequently, MoveIt and RViz were utilized exclusively as static safety verification interfaces during waypoint creation, not for active runtime execution.

To guarantee precise extraction of the encoder states and reliable command injection during execution, the custom control architecture interfaces directly with the native Kinova ROS drivers via specific action servers and topics, as detailed in Table 4.

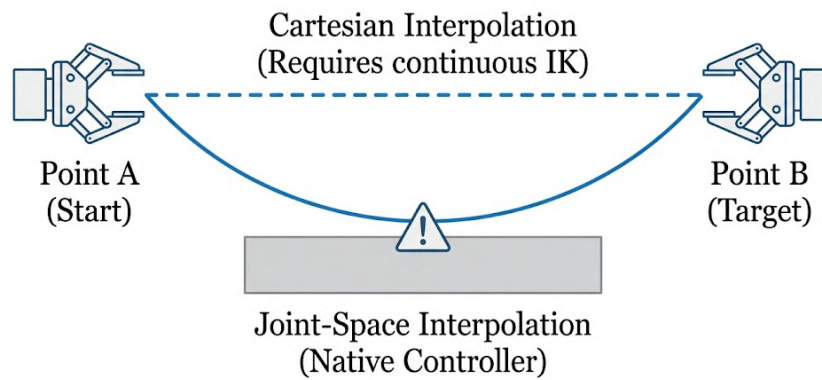
Table 4 – ROS Core Interfaces utilized by the Execution Node for state monitoring and motor control.

ROS Interface / Name	Data Type	Functional Role
/j2n6s300_driver/out/joint_angles	kinova_msgs/JointAngles	Reads real-time hardware encoders during the teaching phase.
/j2n6s300_driver/joints_action/joint_angles	kinova_msgs/ArmJointAnglesAction	Sends deterministic absolute degree vectors to the arm controllers.
/j2n6s300_driver/fingers_action/finger_positions	kinova_msgs/SetFingersPositionAction	Commands gripper opening/closing via normalized values.

4.2.2 Intermediate Waypoints: Approach, Pre-grasp, and Retreat

A fundamental challenge in controlling a 6-DoF manipulator like the Kinova Jaco2 in a confined Cartesian space is kinematic ambiguity. When interpolating directly between two distant joint configurations (e.g., from the Home position directly to the Grasp position), the underlying controller relies on joint-space trajectory generation. While this ensures smooth actuator velocities, the forward kinematics mapping $\mathbf{x}(t) = \text{FK}(\boldsymbol{\theta}(t))$ is highly non-linear. Consequently, a linear interpolation in joint space translates into a complex, unpredictable arcing trajectory for the end-effector in Cartesian space (Lynch; Park, 2017), as illustrated in Figure 24. This phenomenon can easily result in the manipulator's body colliding with the CNC table or the camera mount.

Figure 24: Trajectory divergence: unconstrained joint interpolation yields collision-prone Cartesian arcs.



Source: Created by the Author.

To achieve a strictly linear Cartesian motion, such as inserting a workpiece into a CNC fixture, controllers typically require Cartesian trajectory planning. However, this necessitates the continuous computation of the inverse manipulator Jacobian ($\mathbf{J}^{-1}(\boldsymbol{\theta})$). This mathematical approach introduces the critical risk of encountering kinematic singularities, where the robot effectively loses one or more degrees of freedom and the controller commands infinite joint velocities (Siciliano *et al.*, 2010). In dense manufacturing environments, this theoretical instability inevitably causes severe safety faults and tracking failures.

To resolve these spatial ambiguities without relying on risky online Jacobian inversions, multiple intermediate waypoints (e.g., approach, pre-grasp, retreat, and wrist “unwinding” poses) were intentionally added to shape the trajectory. This reduces ambiguity caused by kinematic redundancy, where many possible joint motions exist from point A to B, and helps ensure the robot follows the intended route without collisions with the CNC workcell. By explicitly defining “Approach,” “Pre-grasp,” and “Retreat” joint

states, the system forces a specific, deterministic joint-space route that approximates strict Cartesian constraints, summarized in Table 5.

Table 5 – Functional anatomy of the deterministic waypoint sequence used to mimic Cartesian constraints and prevent kinematic collisions.

Trajectory Phase	YAML Recipe Equivalent	Kinematic Purpose
Global Rest	1 - Home	Synchronizes the arm outside the CNC enclosure, resetting potential inter-cycle drift.
Arc Mitigation	1.5 - Aproximacao, 2.5 - Giro J4	Intermediate nodes inserted to break long joint interpolations, preventing the elbow or cables from swinging outward.
Orthogonal Alignment	2 - Pre-posicao	Aligns the Z-axis of the end-effector orthogonally to the workpiece geometry.
Pseudo-Linear Insertion	4 - Descida Final, Pega	5 - Actuates primarily the distal wrist joints to descend vertically, mimicking a linear Cartesian insertion.
Vertical Extraction	6 - Recuo Pequeno	Clears the rigid CNC holding fixtures vertically before allowing the base joint (J_1) to initiate rotation.

The “Approach” and “Pre-grasp” waypoints act as geometric constraints that align the gripper’s Z-axis orthogonally to the workpiece just millimeters above it. Approaching a workpiece directly without this proper alignment can cause the gripper to strike the object laterally, displacing it before the grasp is secured. The transition from “Pre-grasp” to the final “Grasp” configuration is engineered as a differential motion. By keeping the proximal links (base, shoulder, elbow) virtually static and commanding minor angular displacements strictly to the distal wrist joints, the resulting spatial arc radius is drastically minimized. This effectively mimics a linear Cartesian insertion without requiring complex inverse kinematics.

Similarly, the “Retreat” waypoint ensures that the manipulator extracts the workpiece purely vertically. This orthogonal extraction clears the CNC fixture before initiating the gross, non-linear motion back to the Home configuration. This rigid path shaping minimizes the risk of dynamic collisions, guarantees safe extraction, and creates a highly predictable execution loop for continuous machine tending.

4.3 RECIPE EXECUTION PIPELINE

The recipe execution pipeline is the deterministic software layer responsible for translating the static YAML data into physical electromechanical actuation. Once the appropriate pick-and-place sequence is loaded into the operational memory based on the detected ArUco ID, this pipeline assumes full control of the manipulator and its

end-effector, coordinating the timing, motion, and physical interaction with the CNC environment.

4.3.1 Direct Joint Command Interface

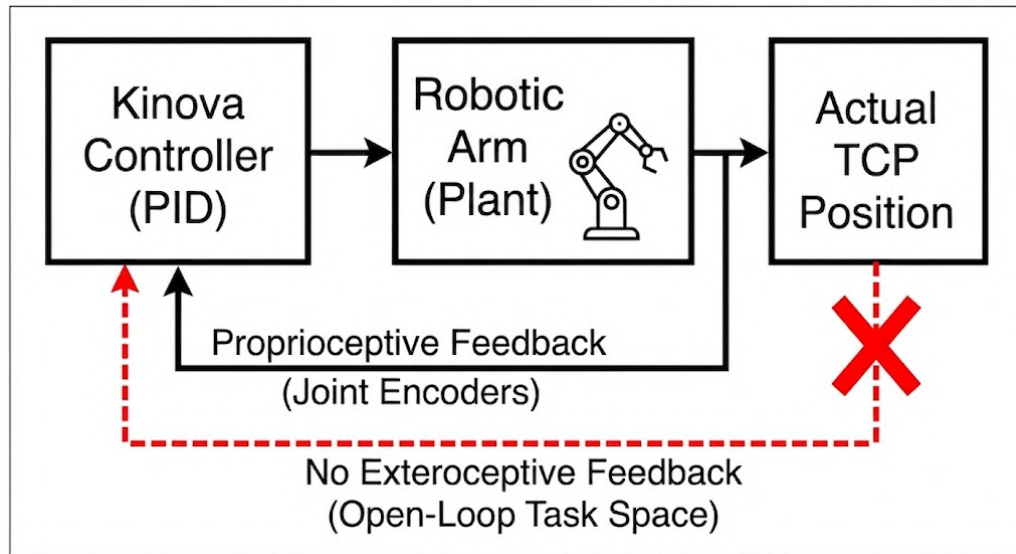
The execution phase operates entirely in the joint-space domain, effectively bypassing the MoveIt trajectory planning framework during runtime. A custom ROS node acts as a central sequencer, iterating through the YAML recipe and publishing direct joint commands to the low-level Kinova ROS driver. The driver interface (e.g., via the `/j2n6s300_driver/in/*` action servers) receives these discrete joint targets and utilizes its embedded controllers to govern the six actuators.

However, a manipulator cannot be commanded using a sheer mathematical step function, as an instantaneous change in position would require infinite acceleration, leading to actuator saturation and severe mechanical shock. To mitigate this, the Kinova embedded driver translates the discrete joint waypoints into continuous time-parameterized trajectories. It employs polynomial spline interpolation (typically cubic or quintic splines) to calculate intermediate setpoints. This ensures that the angular velocity ($\dot{\theta}$) and acceleration ($\ddot{\theta}$) profiles remain continuous and bounded (Biagiotti; Melchiorri, 2008). Consequently, this mathematical approach guarantees smooth dynamic transitions between the taught waypoints, without requiring the high-level ROS environment to compute the time-optimal derivatives online.

From a control systems perspective, this architecture operates as a strictly proprioceptive closed-loop system, mapped conceptually in Figure 25. The Kinova driver ensures that the manipulator reaches the interpolated setpoints by constantly reading internal optical joint encoders and computing the error vector $e(t) = \theta_{\text{target}} - \theta_{\text{actual}}(t)$. The embedded Proportional-Integral-Derivative (PID) control law (Ogata, 2010), which is factory-tuned by the manufacturer to prevent hardware-damaging torque overshoot, then calculates the necessary torque (τ) for each joint motor to drive this error to zero. Yet, while closed-loop at the joint level, the system is entirely open-loop regarding the Cartesian task space. It completely lacks exteroceptive feedback (such as continuous vision or force-torque sensing) regarding the actual physical state of the workpiece during manipulation.

To maintain a repeatable cycle for continuous machine tending, the sequence is rigidly designed to start and end at the exact same “Home” configuration. While this pure joint-command method is highly repeatable in the short term due to the hardware’s intrinsic precision, it introduces a systemic vulnerability known as open-loop drift. Without closed-loop re-localization to compensate for micro-collisions or dynamic slipping during the robot’s rapid movements, any small placement errors after a workpiece release will persist.

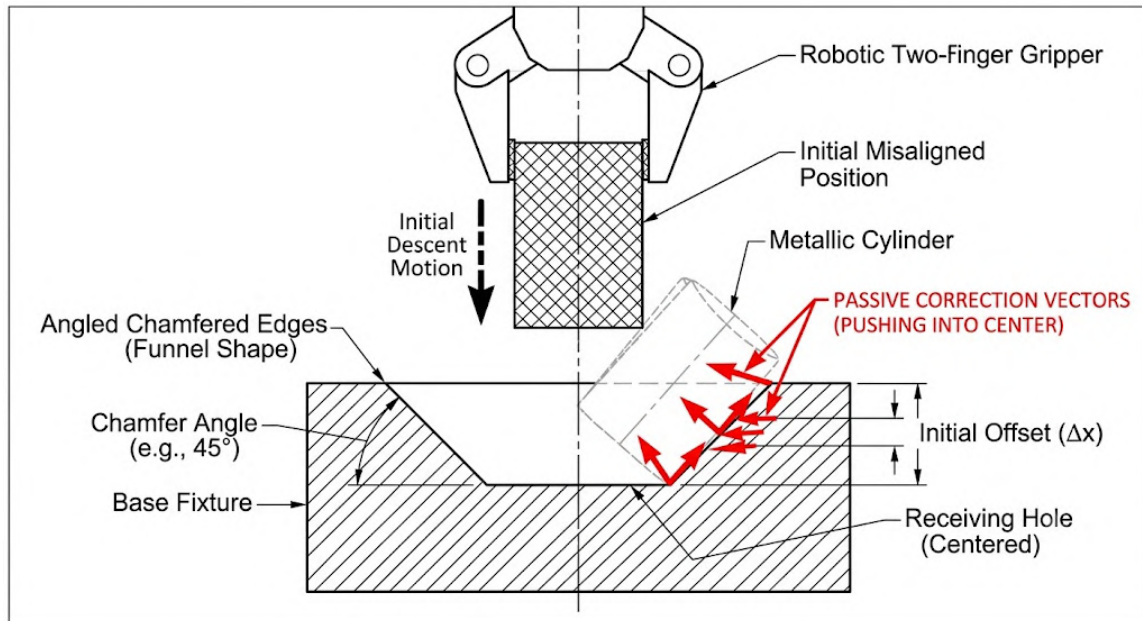
Figure 25: Proprioceptive joint closed-loop vs. exteroceptive task-space open-loop.



Source: Created by the Author.

In looping scenarios (e.g., continuous CNC table cycles), these residual spatial shifts can accumulate, leading to degraded grasps or complete misalignment over time. Therefore, this execution methodology relies heavily on the mechanical design of the workcell to bridge the software's sensing gap. Inspired by the principles of Design for Manufacture and Assembly (DFMA), it assumes the use of passive mechanical compliance and self-aligning geometries to facilitate insertion tasks (Boothroyd; Dewhurst; Knight, 2010). In practice, this means the CNC clamping fixtures must possess physical features such as chamfered edges, V-blocks, or pneumatic self-centering jaws (illustrated in Figure 26). These features forcefully re-center the workpiece upon clamping, actively absorbing and neutralizing the robot's slight positioning variances, thus resetting the spatial error to zero before the next cycle begins.

Figure 26: DFMA-based passive compliance: chamfers mitigate open-loop placement drift.



Source: Created by the Author.

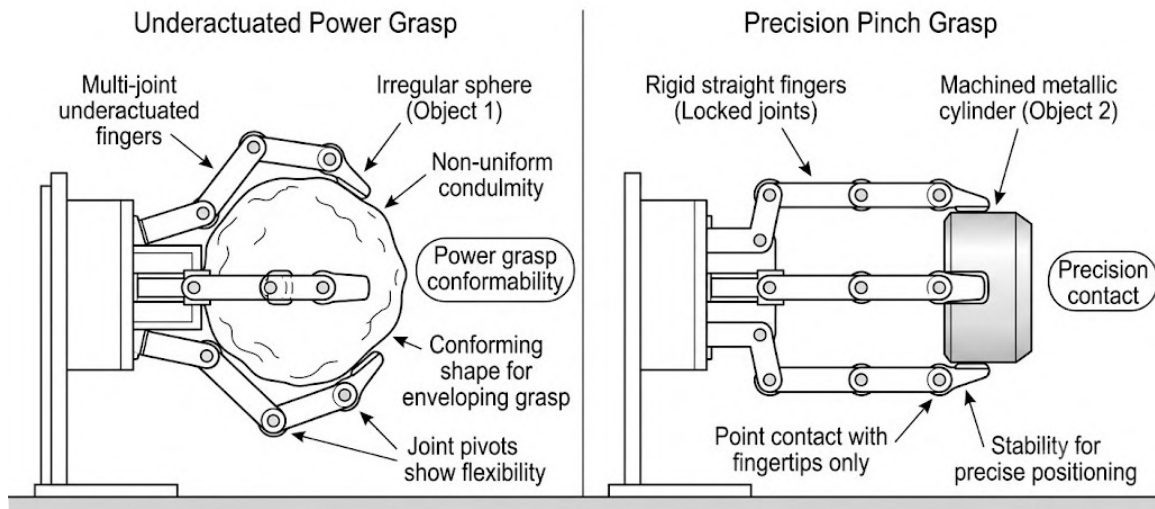
4.3.2 Gripper and Finger Control Integration

Parallel to the joint manipulation, the recipe system synchronizes the end-effector actuation. The Kinova Jaco2 features an underactuated three-finger gripper. In robotic manipulation theory, an underactuated hand possesses fewer actuators than total degrees of freedom (DoF), utilizing mechanical linkages or tendon-driven mechanisms to passively adapt to the shape of an object (Birglen; Laliberté; Gosselin, 2008). While this adaptive enveloping grasp is highly advantageous for handling unstructured or biological objects, precision machine tending requires deterministic, repeatable positioning.

To overcome the unpredictability of passive adaptation, the gripper's state is explicitly constrained and commanded via the `fingers` array in the YAML sequence (e.g., fully open at `[0.0, 0.0, 0.0]` or closed at a specified threshold like `[0.87, 0.87, 0.87]`). By defining an exact kinematic stroke threshold rather than relying purely on torque-based force limits, the system forces a rigid, predictable pinch-grasp, as compared in Figure 27. This ensures the CNC workpiece is centralized consistently relative to the Tool Center Point (TCP).

The custom execution node coordinates these state transitions, ensuring that gripper actuation only triggers at the exact specified procedural steps. This synchronization prevents the software from prematurely initiating the next arm trajectory before the physical grasping action is fully resolved, a critical safeguard against workpiece jamming or tearing during extraction from the CNC clamping fixture.

Figure 27: Underactuated enveloping grasp vs. precision pinch for TCP centrality.



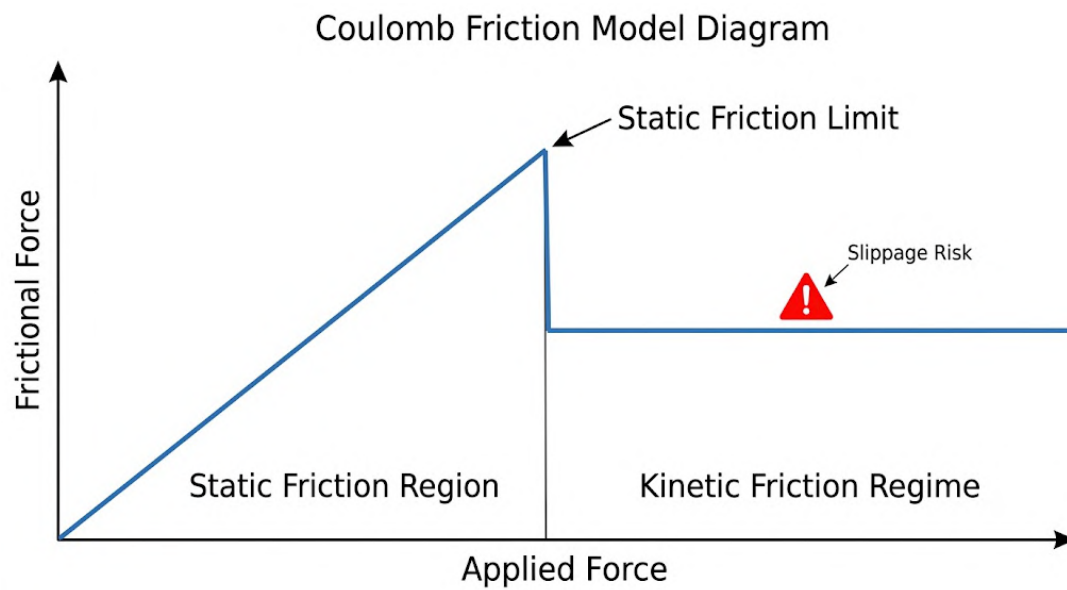
Source: Created by the Author.

A critical component of this integration is the implementation of the `wait_after` parameter defined in the recipe. The Kinova Jaco2 is constructed with carbon-fiber links and harmonic drive reducers, which introduce inherent mechanical compliance and joint elasticity into the kinematic chain. In the context of mechanical control theory, when a robotic link performs a fast point-to-point motion and decelerates to a stop, the system's kinetic energy induces a transient step response characterized by damped oscillations (Ogata, 2010).

If the fingers are commanded to close while the end-effector's TCP is still experiencing these kinetic vibrations, the grasping physics are severely compromised. A vibrating TCP creates relative velocity between the gripper's contact pads and the stationary workpiece. According to Coulomb friction models (Mason, 2001), this introduces dynamic friction conditions. Because the coefficient of kinetic friction (μ_k) is inherently lower than the coefficient of static friction (μ_s), as modeled in Figure 28, the resulting dynamic slip can easily misalign the part, cause a failed grasp, or induce unpredicted rotational torque on the object.

To counteract this, the `wait_after` parameter enforces a programmable dwell time (e.g., 1.0 or 1.5 seconds) that strictly respects the manipulator's mechanical settling time (t_s). This programmatic pause guarantees that all residual kinetic oscillations have naturally dissipated through structural damping before the fingers make physical contact with the rigid CNC workpiece. By guaranteeing zero relative velocity at the moment of contact, the system secures a pure static-friction grasp, thereby drastically improving the success rate and spatial repeatability of the grasping phase.

Figure 28: Coulomb friction drop ($\mu_s \rightarrow \mu_k$): residual TCP oscillations induce workpiece slippage.



Source: Created by the Author.

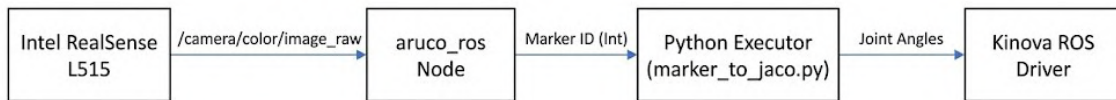
5 SYSTEM IMPLEMENTATION

This chapter details the practical implementation of the proposed "ID-to-Recipe" architecture, outlining the software deployment, troubleshooting procedures, and a critical analysis of the system's performance. The objective is to demonstrate how the theoretical framework established in Chapter 4 was translated into a functional automated machine-tending cell, evaluating its efficacy, advantages, and inherent limitations.

5.1 DEVELOPED COMPONENTS AND DELIVERABLES

The implementation of the robotic cell relies on a decentralized, node-based software architecture deployed within the Robot Operating System (ROS 1 Noetic) environment using Python 3.8. Figure 29 illustrates the implementation flowchart, detailing the data progression from visual acquisition to actuator execution. The physical hardware architecture is composed of a Kinova Jaco2 6-DoF manipulator (model: j2n6s300) and an Intel RealSense L515 LiDAR camera, both interfaced to the host machine via high-bandwidth USB.

Figure 29: ID-to-Recipe pipeline: mapping ROS nodes to Python scripts.



Source: Created by the Author.

To ensure modularity and high cohesion, the system was divided into three primary functional blocks: Perception, Decision/Mapping, and Kinematic Execution.

5.1.1 Perception and Topological Classification

The primary input stream originates from the RealSense driver (realsense2_camera), which publishes raw RGB frames to the /camera/* topic space. These frames are continuously subscribed to by the aruco_ros node, an implementation based on the robust squared planar marker detection proposed by Garrido-Jurado et al. (Garrido-Jurado *et al.*, 2014).

Initially, the system architecture considered a continuous 6D pose estimation strategy. This approach relies on the Perspective-n-Point (PnP) algorithm (Lepetit; Moreno-Noguer; Fua, 2009) to compute the full metric pose, encompassing translation $t \in \mathbb{R}^3$ and rotation $R \in SO(3)$, of the workpiece relative to the camera frame. The PnP problem seeks to minimize the reprojection error between the known 3D points

of the marker model and their corresponding 2D projections on the image plane, a process that heavily depends on the accuracy of the camera's intrinsic matrix.

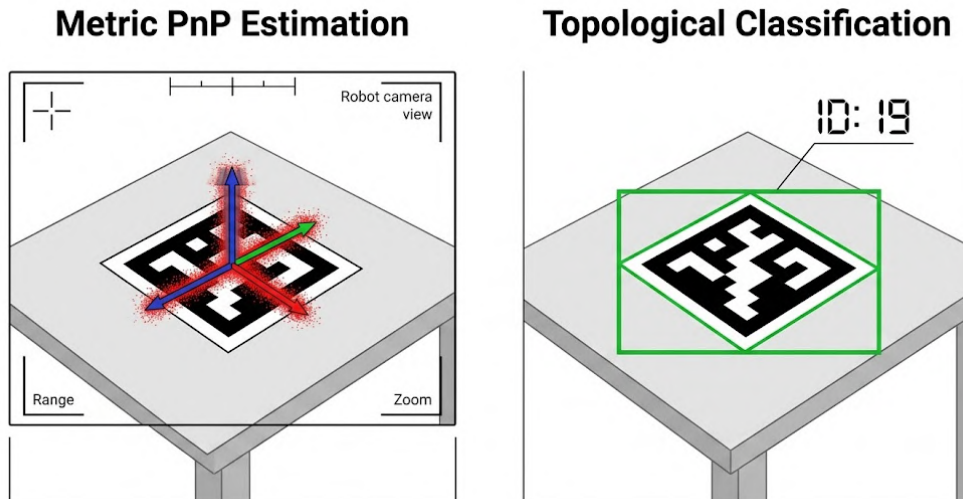
However, in a tight-tolerance CNC machine tending environment, relying on continuous PnP solving proved highly susceptible to systemic inaccuracies. The primary point of failure is the error propagation within the transformation tree (TF). To grasp an object based on visual data, the spatial transformation from the robot base to the marker, denoted as T_{base}^{marker} , must be computed using the hand-eye calibration matrix (T_{base}^{camera}) and the visually estimated camera-to-marker pose (T_{camera}^{marker}), such that:

$$T_{base}^{marker} = T_{base}^{camera} \cdot T_{camera}^{marker} \quad (13)$$

According to the classic $AX = XB$ hand-eye calibration problem formulated by Tsai and Lenz (Tsai; Lenz, 1989), any static angular misalignment in the extrinsic calibration T_{base}^{camera} is amplified proportionally by the operational distance to the target. This lever-arm effect leads to severe positional deviations at the manipulator's Tool Center Point (TCP). Furthermore, the depth sensor noise and RGB-D alignment inaccuracies inherent to the Intel RealSense L515 introduce dynamic *jitter* to the T_{camera}^{marker} estimation. Feeding this noisy metric data into a runtime motion planner results in erratic and unstable inverse kinematics (IK) solutions.

To circumvent this compounding error, the perception pipeline was structurally constrained to function strictly as a topological classifier, as visually compared in Figure 30.

Figure 30: Metric PnP ($SE(3)$ jitter) vs. topological classification (Z^+ extraction).



Source: Created by the Author.

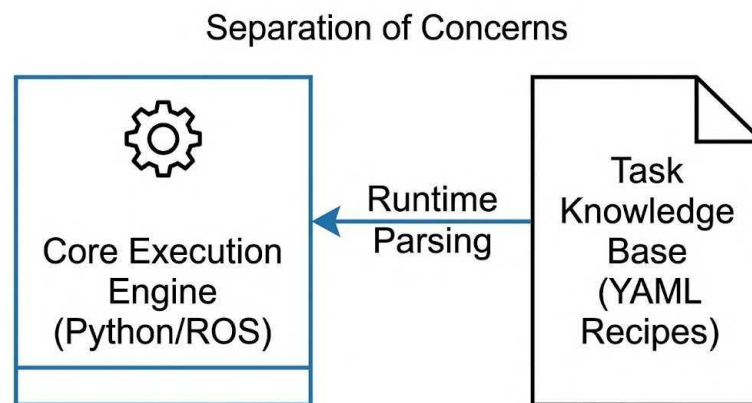
Rather than solving for continuous spatial geometry within the Special Euclidean group $SE(3)$, the node processes the visual stream to extract only a discrete, integer-based marker identifier (e.g., ID: 19). By discarding the continuous metric estimation

and mapping the visual input to a finite discrete state space $\mathbb{Z}^+$, the system effectively severs the propagation of coordinate matrix uncertainties into the manipulator's kinematic chain. The visual system thus acts as a robust topological trigger, completely decoupling perception noise from the physical execution loop and enabling the high repeatability required for industrial pick-and-place operations.

5.1.2 Decision Logic and the YAML Recipe System

The core decision logic of the ID-to-Recipe architecture resides in a custom Python execution orchestrator, designated as the `marker_to_jaco.py` node. Upon receiving a discrete ID via the ROS messaging system from the perception pipeline, this node transitions the system's operational state by cross-referencing a modular YAML database. From a control perspective, this architecture models the robotic workcell as a Discrete Event System (DES) (Cassandras; Lafortune, 2008), where the asynchronous arrival of a specific visual identifier acts as the state-transition trigger that initiates a pre-defined deterministic sequence.

Figure 31: Separation of Concerns: static Python engine decoupled from YAML task definitions.



Source: Created by the Author.

This data-driven approach fundamentally leverages the core software engineering principles of Separation of Concerns (SoC) (Dijkstra, 1982) and Information Hiding (Parnas, 1972), which remain foundational methodologies for structuring modern, maintainable software architectures (Sommerville, 2011). By decoupling the static execution engine (the Python parsing and ROS communication logic) from the dynamic task definitions (the sequence waypoints), the system achieves a high degree of modularity.

In the context of Flexible Manufacturing Systems (FMS), this architectural choice is highly advantageous. It allows shop-floor operators to alter task parameters, introduce new workpieces, or fine-tune trajectories directly through human-readable text files without necessitating source code recompilation or disrupting the underlying programming

framework. By significantly reducing setup times and minimizing reprogramming downtime during process adjustments, this methodology directly fulfills the core flexibility requirements of modern FMS (Groover, 2015).

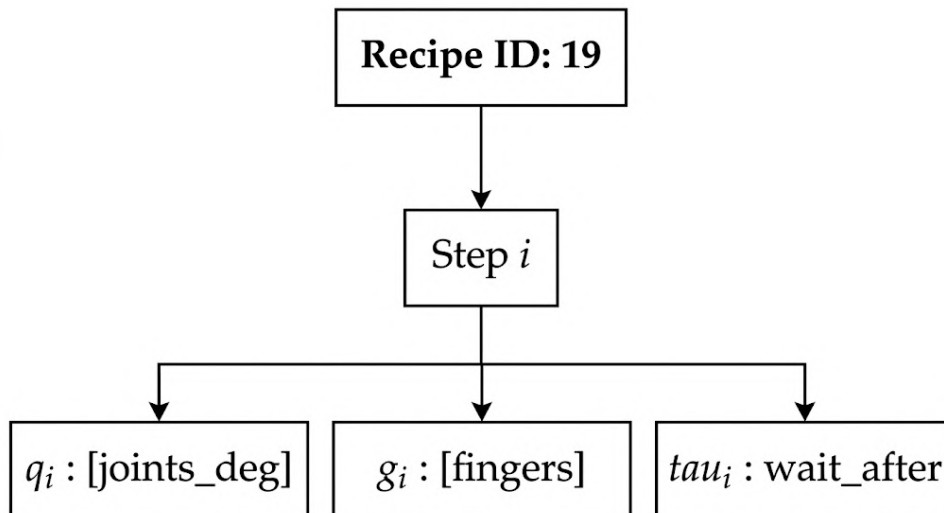
Mathematically and structurally, each YAML recipe defines a finite, ordered sequence of operations required to complete a specific machine-tending task. A selected recipe can be formalized as an ordered set of waypoints $W = \{w_1, w_2, \dots, w_n\}$. Each waypoint w_i is explicitly represented by a functional tuple, as visualized in Figure 32:

$$w_i = (q_i, g_i, \tau_i) \quad (14)$$

Where:

- $q_i \in \mathbb{R}^6$ represents the exact joint-space target configuration (`joints_deg`), ensuring the manipulator achieves the desired pose unambiguously, circumventing the computational overhead and unpredictability of runtime inverse kinematics.
- g_i denotes the specific end-effector actuation state (`fingers`), commanding the position or force of the gripper proportional to the workpiece geometry.
- $\tau_i \in \mathbb{R}^+$ dictates a localized kinetic dissipation period (`wait_after`). This temporal dwell is strategically parameterized after large joint-space interpolations to allow residual mechanical vibrations within the manipulator's flexible joints and links to settle (Spong; Hutchinson; Vidyasagar, 2006). Consequently, this parameter physically stabilizes the system prior to executing sensitive Cartesian grasping operations.

Figure 32: Waypoint tuple $\rightarrow$ YAML parameter mapping per operational step.



Source: Created by the Author.

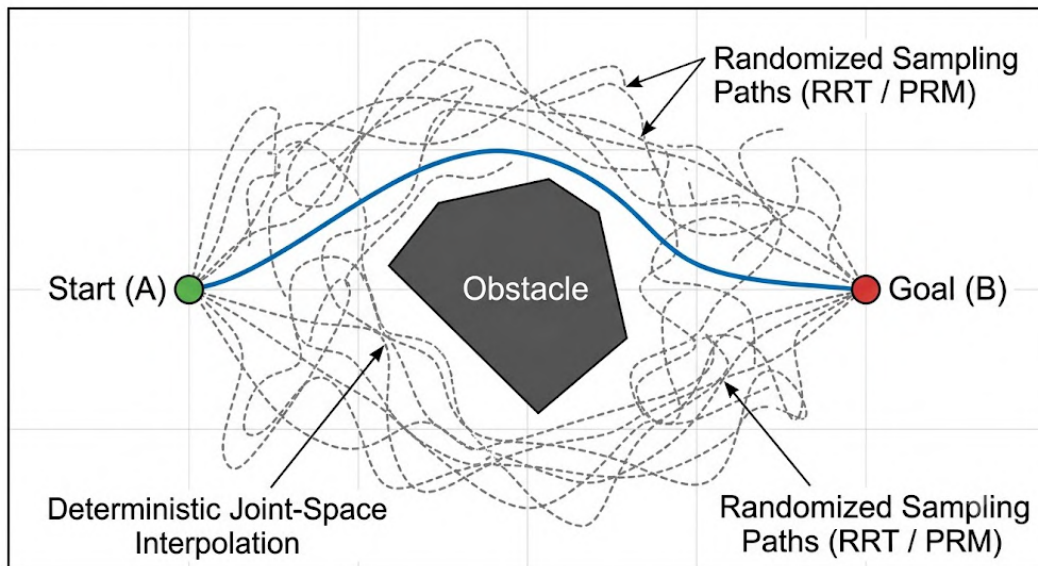
By serializing these critical process parameters in a structured format, the execution node simply acts as a deterministic state machine parser, advancing sequentially

through the set W and relaying precise commands directly to the low-level Kinova hardware drivers.

5.1.3 Deterministic Kinematic Execution

Runtime motion planners, such as MoveIt and the Open Motion Planning Library (OMPL) (Chitta; Sucan; Cousins, 2012), utilize sampling-based algorithms like Rapidly-exploring Random Trees (RRT) or Probabilistic Roadmaps (PRM). While these randomized algorithms guarantee probabilistic completeness, they inherently introduce latency, non-determinism, and path unpredictability. Consequently, they may generate wildly different Cartesian trajectories for the exact same start and goal configurations during successive execution cycles, as visualized in Figure 33.

Figure 33: Probabilistic paths (RRT/PRM) vs. deterministic joint-space interpolation.



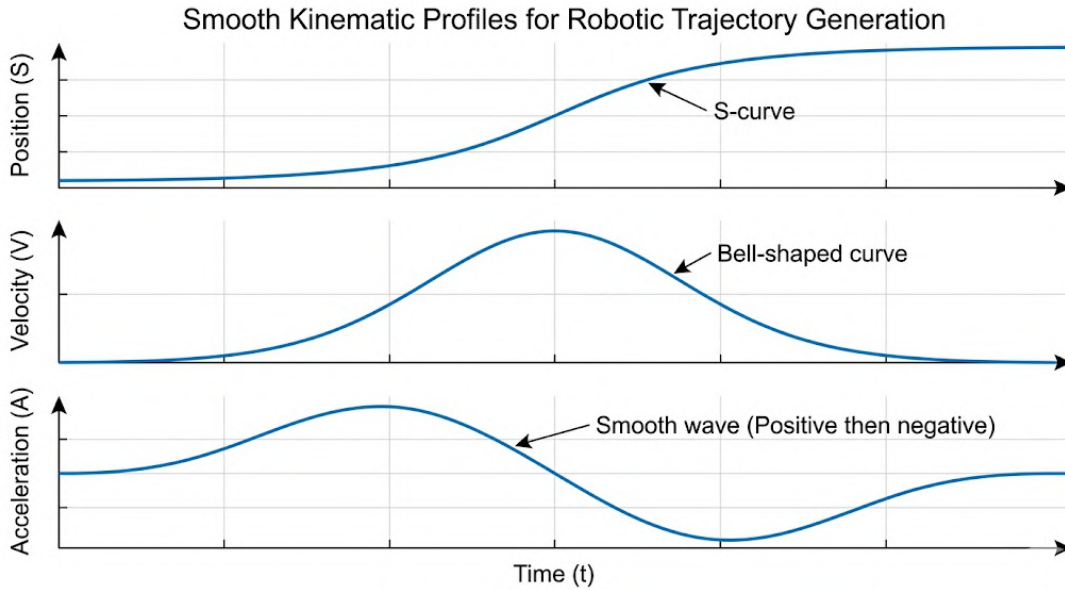
Source: Created by the Author.

To bypass these operational issues and ensure strict path consistency for the industrial task, the execution node routes commands directly to the `j2n6s300_driver`. While MoveIt and RViz were critically utilized during the offline teach-programming phase for manual waypoint authoring, collision checking, and static state visualization, the online execution strictly avoids dynamic trajectory generation to ensure a highly repeatable industrial cycle.

By publishing direct joint-space targets (q_{target}), the system exploits the Kinova manipulator's internal low-level controller. This control strategy ensures that the transition between authored waypoints (e.g., approach, pre-grasp, and retreat) follows a mathematically deterministic trajectory profile. Rather than calculating step-by-step Cartesian poses, the driver utilizes polynomial interpolation (typically cubic or quintic

splines) to compute the intermediate joint states. According to classic trajectory generation theory (Craig, 2005), this mathematical approach ensures $\mathcal{C}^1$ and $\mathcal{C}^2$ continuity. As illustrated in Figure 34, the velocity and acceleration profiles remain smooth and bounded, thereby preventing mechanical jerk that could damage the harmonic drive gearboxes or disrupt the payload.

Figure 34: Polynomial joint-space interpolation: $\mathcal{C}^1$ and $\mathcal{C}^2$ continuity ensures smooth actuator transitions.



Source: Created by the Author.

Operating entirely within the joint-space configuration ($\mathbb{R}^6$) rather than Cartesian space ($\mathbb{R}^3 \times SO(3)$) mathematically guarantees the avoidance of kinematic singularities during operational cycles. In Cartesian control, the relationship between end-effector velocity $\dot{x}$ and joint velocity $\dot{q}$ is governed by the inverse Jacobian matrix, $\dot{q} = J^{-1}(q)\dot{x}$. According to kinematic theory (Siciliano *et al.*, 2010), as the manipulator approaches a singularity, the Jacobian matrix $J(q)$ becomes rank-deficient and its determinant approaches zero ($\det(J) \rightarrow 0$), which mathematically results in unbounded commands for joint velocities. In an applied industrial context, attempting to execute these theoretically infinite velocities immediately triggers controller safety faults. By interpolating purely between joint vectors, the need to compute the inverse Jacobian online is bypassed, rendering the path completely immune to singularity-induced halts.

Furthermore, as extensively established in Section 4.3.2, this deterministic joint interpolation is explicitly synchronized with the temporal `wait_after` dwell periods. This ensures that the inherently underdamped response of the manipulator's links reaches a steady state prior to actuation, guaranteeing optimal static friction (μ_s) conditions and preventing dynamic part slippage during high-precision CNC grasping tasks.

5.2 TROUBLESHOOTING AND ENCOUNTERED ISSUES

The deployment phase required significant troubleshooting to achieve system stability, particularly concerning the ROS environment and the perception pipeline.

5.2.1 Software and Configuration Fixes

Developing an integrated robotic cell requires orchestrating multiple asynchronous software nodes, bridging high-level perception algorithms with low-level kinematic controllers. Several configuration hurdles and system anomalies were encountered and systematically resolved to establish a deterministic execution environment.

Timestamp Synchronization and TF Warnings: A recurring issue involved stale coordinate transforms (TF) and dropped joint feedback messages, often triggering warnings such as *"Didn't receive robot state"*. In a distributed robotic framework like ROS, temporal synchronization is critical; spatial transformations are inherently bound to specific timestamps to maintain kinematic coherence (Foote, 2013). The root cause was traced to inconsistent clock settings across nodes, specifically the `/use_sim_time` parameter being improperly set during real hardware execution. Disabling simulation time and ensuring all active nodes strictly utilized the system's hardware clock resolved the temporal mismatch, preventing the TF tree from discarding data deemed outdated.

Visualization and State Feedback Disconnects: During the manual waypoint authoring phase, instances occurred where MoveIt and RViz failed to reflect the expected robot state, or the UI simply did not initialize. In the ROS publish-subscribe architecture, visualization tools are entirely dependent on a continuous and correctly mapped stream of joint feedback. Troubleshooting revealed that this symptom was caused by incorrect namespace routing, delayed driver initialization, or missing `/joint_states` messages. The solution required strictly enforcing the launch order (ensuring the Kinova driver was fully initialized before visualization nodes) and verifying topic names to restore the data flow between the hardware interface and the GUI.

Sensor Hardware Interface and Instability: The perception pipeline experienced intermittent dropouts, missing topics, and inconsistent ArUco marker detection. While integrating high-throughput RGB-D sensors, such as the Intel RealSense L515, into a Linux-based ROS environment is standard practice, it frequently introduces unmapped hardware-software interface bottlenecks. In this system architecture, the dropouts were diagnosed and primarily attributed to USB bandwidth limitations and missing Linux `udev` rule configurations, which govern system-level device permissions. By securing the physical connections, documenting the specific USB port assignments, establishing the correct `udev` rules, and tuning the stream parameters to reduce bandwidth overhead, the camera's publish-subscribe stability was fully restored.

Scripting Environment and Execution Errors: At the application level, execution was occasionally halted by syntax strictness, such as Python `TabError` exceptions indicating inconsistent use of tabs and spaces within custom execution scripts like `marker_to_jaco.py`. Because the ROS architecture heavily relies on Python for high-level node orchestration, resolving these interpreter exceptions requires strict adherence to the language's official formatting conventions. According to the Python Style Guide (PEP 8) (Rossum; Warsaw; Coghlan, 2001), mixing tabs and spaces is strictly disallowed, and a uniform 4-space indentation is the mandated standard. Consequently, converting all custom scripts to comply with this 4-space rule successfully resolved the build-time parsing errors.

Environment Dependencies and Physical Interruptions: Workspace conflicts arose from mixed ROS1 and ROS2 dependencies, causing `rosdep` resolution failures during the build process. A clean workspace rebuild, strictly aligned with the ROS1 Noetic branches and ensuring all system dependencies were installed, was necessary to restore stability. Beyond software, environmental factors directly impacted experimental continuity. Power outages exposed vulnerabilities in the laboratory's security infrastructure. Although the laboratory door locks are equipped with backup batteries, their low autonomy proved insufficient during extended outages, leading to unintended door unlocking. This hardware limitation required manual intervention to secure the workspace and safeguard the robotic equipment, temporarily interrupting the continuous open-loop testing cycles.

5.2.2 Transition from Pose-Based to ID-Driven Workflow

Initially, the project aimed to implement continuous 6D pose-based grasping. In this paradigm, the workpiece pose is estimated online via a custom ArUco detection pipeline, utilizing the RGB-D data from the Intel RealSense camera. Based on classical Position-Based Visual Servoing (PBVS) theory (Corke, P., 2011), the robot's control architecture would then dynamically calculate kinematic paths to this spatial target in Cartesian space. However, empirical testing in the laboratory revealed that the end-to-end accuracy of this dynamic method was fundamentally insufficient for the tight millimeter-level tolerances demanded by the CNC loading and unloading workflow.

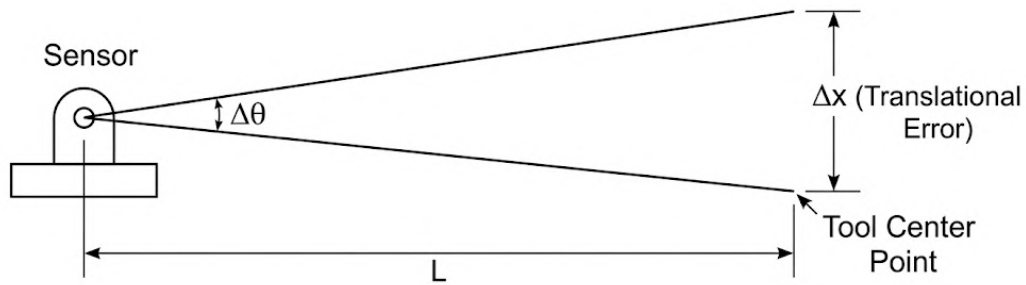
The failure of the dynamic pose-based approach is deeply rooted in the physical and mathematical propagation of spatial errors originating from the camera-to-robot calibration matrix. As established in the hand-eye calibration formulations discussed in previous sections, even minute inaccuracies resulting from calibration sensitivity act as multiplier coefficients for subsequent measurement errors (Shiu; Ahmad, 1989).

This phenomenon is mathematically governed by the lever-arm effect (or Abbe error), illustrated in Figure 35. If the vision system outputs a pose with a slight angular estimation error $\Delta\theta$, the resulting translational error Δx at the manipulator's Tool

Center Point (TCP) scales linearly with the distance L from the camera to the target, approximated by:

$$\Delta x \approx L \cdot \tan(\Delta\theta) \quad (15)$$

Figure 35: Abbe Error: angular inaccuracy ($\Delta\theta$) causing magnified TCP displacement (Δx).



Source: Created by the Author.

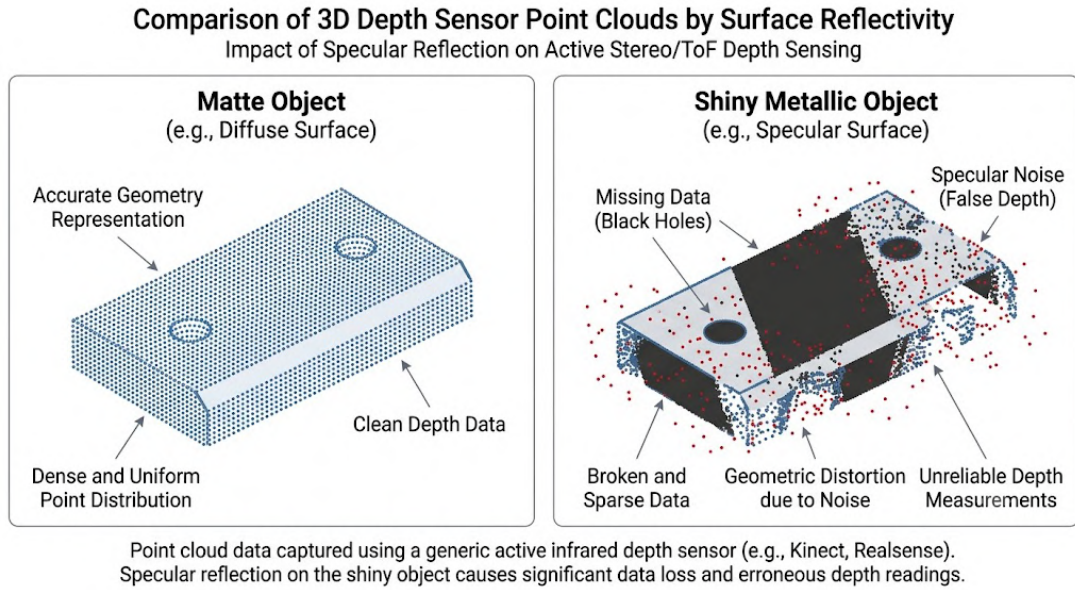
Consequently, a seemingly negligible rotational jitter of just 1 or 2 degrees at the sensor frame results in several millimeters of deviation at the remote gripper, leading to critical collisions or failed grasps against the rigid CNC fixtures.

Furthermore, the physical nature of the CNC machining environment actively exacerbates these perceptual inaccuracies. According to established error models for consumer-grade RGB-D sensors, depth noise inherently scales quadratically with distance (Khoshelham; Elberink, 2012). More critically within the context of this specific industrial application, machined metallic parts continuously exhibit high specular reflection.

Empirical observations demonstrated that this optical property scatters the sensor's infrared structured light projection. As visually compared in Figure 36, this scattering causes missing depth data, multipath interference, and severe high-frequency jitter in the ArUco marker's Z -axis (depth) estimation. Ultimately, when this detection jitter, optical noise, and transformation alignment offsets are combined, the cumulative spatial error renders continuous online path generation unviable.

To fulfill the strict industrial requirement for rigorous repeatability over continuous manufacturing cycles, the system was redesigned into the ID-driven topological workflow. In this new paradigm, the ArUco marker's role is strictly reduced to discrete

Figure 36: Specular reflection artifacts: infrared scattering causing depth "holes" and pose instability.



Source: Created by the Author.

identification, acting solely as a trigger. Once identified, execution relies entirely on the pre-recorded deterministic joint-space recipes. By executing purely deterministic joint-angle commands, the system completely bypasses error-prone dynamic Cartesian calculations and vision-induced spatial jitter, ultimately ensuring a safe and reliable physical interaction within the CNC machine workspace.

5.2.3 Execution Workflow

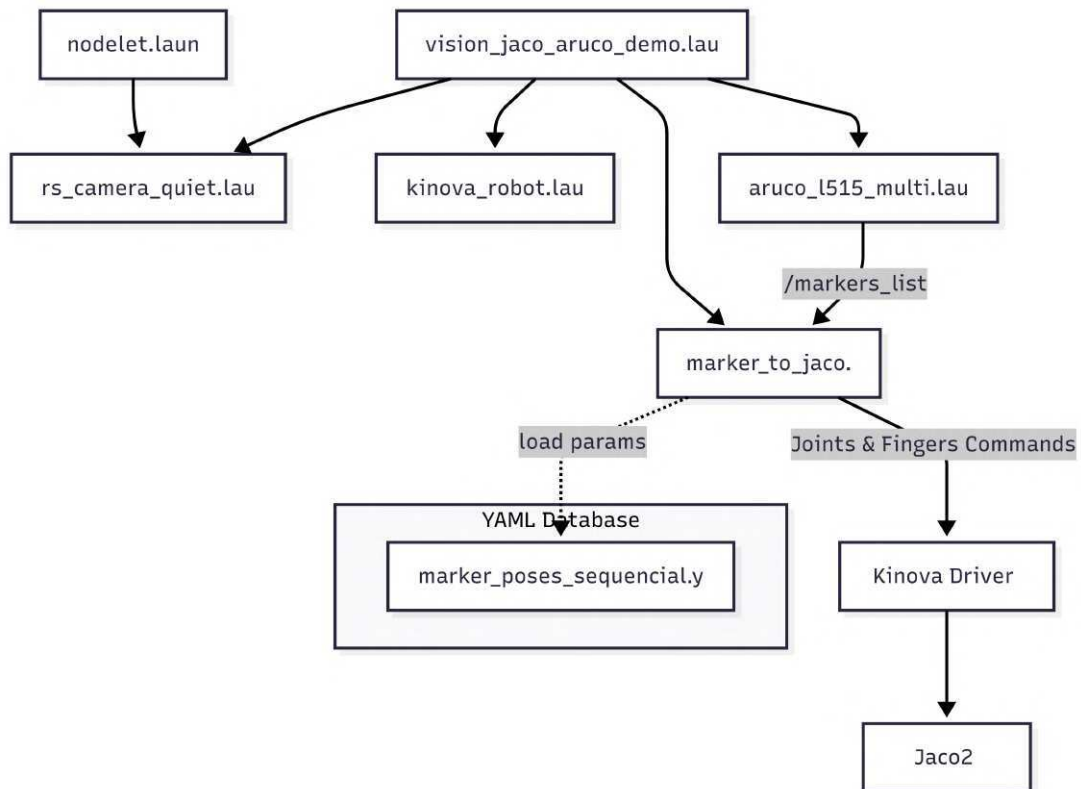
The software architecture is integrated into a unified execution pipeline, orchestrated by the `vision_jaco_aruco_demo.launch` file. This file acts as the system entry point, initializing the robotic hardware, the perception sensors, and the logic controllers in a synchronized start-up sequence.

As illustrated in Figure 37, the execution pipeline establishes a sequential data flow, starting from the system initialization and culminating in the deterministic joint-space control of the manipulator.

The workflow follows a deterministic structure:

1. **System Initialization:** The launch file invokes `kinova_robot.launch` to establish communication with the Jaco² driver and `rs_camera_quiet.launch` to initialize the RealSense L515 camera. The `rs_camera_quiet.launch` script is a modified configuration of the standard driver, designed to suppress non-critical warnings and optimize the stream initialization for the operational workspace.

Figure 37: Data flow diagram of the ID-to-Recipe execution pipeline.



Source: Created by the Author.

2. **Perception Pipeline:** The `aruco_l515_multi.launch` file configures the `aruco_ros` node. This node processes the RGB camera stream and publishes detected marker identifiers to the `/aruco_marker_publisher/markers_list` topic.
3. **Logic and Decision Making:** The core control node, `marker_to_jaco.py`, serves as the execution orchestrator. Upon start-up, it loads the task definitions from `marker_poses_sequential.yaml` as a ROS parameter server dictionary. The script implements a stability counter; it only triggers a movement if a consistent marker ID is detected for a specified number of consecutive frames (defined by the `stable_count` parameter).
4. **Deterministic Execution:** Once a stable ID is validated, `marker_to_jaco.py` parses the corresponding sequence from the loaded YAML configuration. The script iterates through the sequence, sending direct joint-space commands to the Kinova driver via action servers. Each step integrates joint targets and gripper states, while the `wait_after` parameter enforces a programmable dwell time, ensuring that residual mechanical oscillations are dissipated before subsequent actions.

This modular approach ensures that the high-level decision logic remains decoupled from the low-level kinematic drivers, providing both operational safety and reconfigurability for different CNC machine-tending tasks.

6 RESULTS AND EXPERIMENTAL VALIDATION

The system's operational metrics were evaluated based on its capacity to sustain continuous loading and unloading cycles in the CNC environment. The primary focus of this analysis is the trade-off between deterministic execution reliability and the physical constraints of open-loop manipulation.

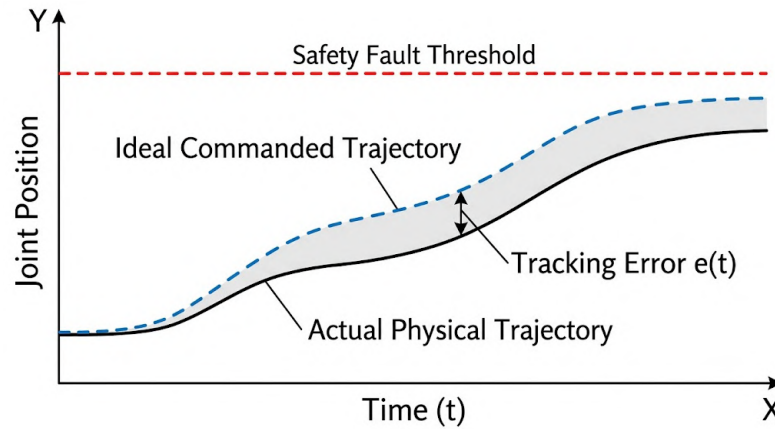
6.1 EVALUATED SCENARIOS AND EXPERIMENTAL SETUP

To properly evaluate the reliability, repeatability, and robustness of the developed ArUco-triggered pick-and-place system, a series of continuous operational experiments were conducted. The testing scenario involved the Kinova Jaco2 manipulator performing sequential loops between two primary workstations: extracting the workpiece from the CNC machine and placing it on the table (Trigger ID20), and picking the workpiece from the table to insert it back into the CNC machine (Trigger ID21).

The validation was designed to test the system in an open-loop continuous execution scenario. The robot was commanded to perform the loops without human interference or intermediate pose recalibration. In total, 21 physical experiments (task executions) were recorded and analyzed across multiple continuous sessions to evaluate both temporal stability and spatial error accumulation over time.

Initially, the project scope included an intended comparative analysis using a multi-arm configuration. However, critical hardware degradation in the secondary Kinova Jaco manipulator rendered it unable to safely lift the specified payload. This cumulative physical wear fundamentally altered the manipulator's nominal parameters, leading to increased gear backlash and reduced joint stiffness. When the degraded joint attempted to move a payload, the physical position lagged significantly, triggering protective halts, as shown in Figure 38. Consequently, the physical limitations of the legacy hardware strictly constrained the experimental validation to the single-arm setup described in this study.

Figure 38: Tracking error accumulation ($e(t)$): protective halt triggered upon exceeding safety thresholds.



Source: Created by the Author.

6.2 QUANTITATIVE PERFORMANCE AND REPEATABILITY

The quantitative results of the experimental validation are detailed in Table 6 and visually summarized in Figure 39. The performance metrics focus on the task success rate, the mean cycle time (μ), and the standard deviation (σ) to assess the system's temporal repeatability.

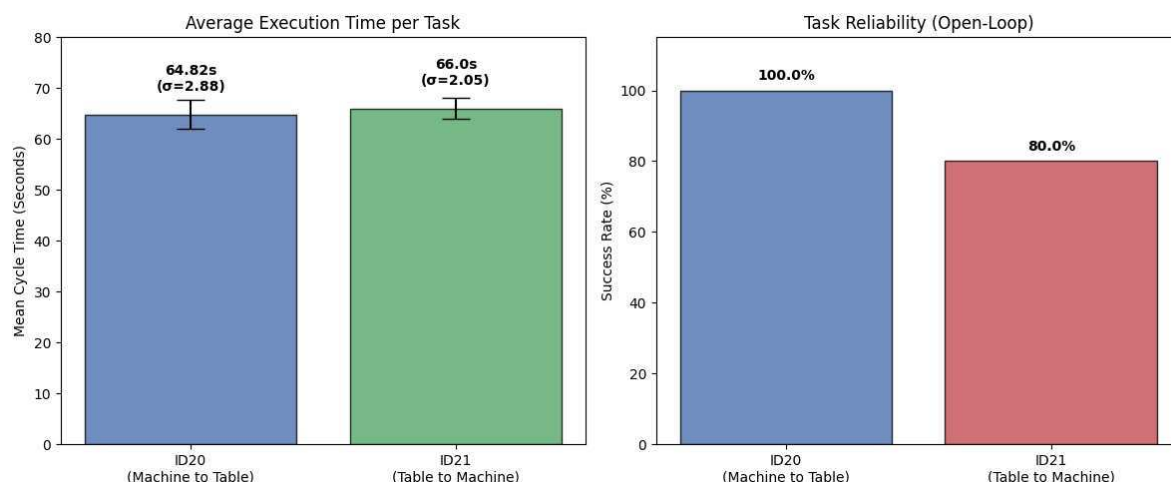
Table 6 – Task performance: high repeatability in cycle time and success rate.

Task	Attempts	Successes	Success Rate (%)	Mean Time (s)	Std. Dev. (s)
ID20 (Machine to Table)	11	11	100.0	64.82	2.88
ID21 (Table to Machine)	10	8	80.0	66.00	2.05

As observed in the data, the system demonstrated high temporal determinism. The average cycle time for the ID20 task was 64.82 seconds (approximately 1 minute and 4 seconds), with a very low standard deviation of 2.88s. Similarly, the ID21 task averaged 66.00 seconds (approximately 1 minute and 6 seconds, $\sigma = 2.05$ s). The tight clustering of these execution times indicates that the joint-space recipes mapped to the ArUco IDs are executed with high consistency by the robot's controller.

Furthermore, an analysis of the sub-task durations revealed that the specific transition time between grasping and releasing the workpiece consistently took approximately 20 seconds across all successful trials, regardless of the active trigger. Industrial machine tending requires this strict adherence to cycle times to optimize Overall Equipment Effectiveness (OEE) (Muchiri; Pintelon, 2008). Dynamic trajectory generation introduces latency and cycle time variance (jitter) due to the varying computational load of search algorithms. Conversely, parsing a predefined sequence from a

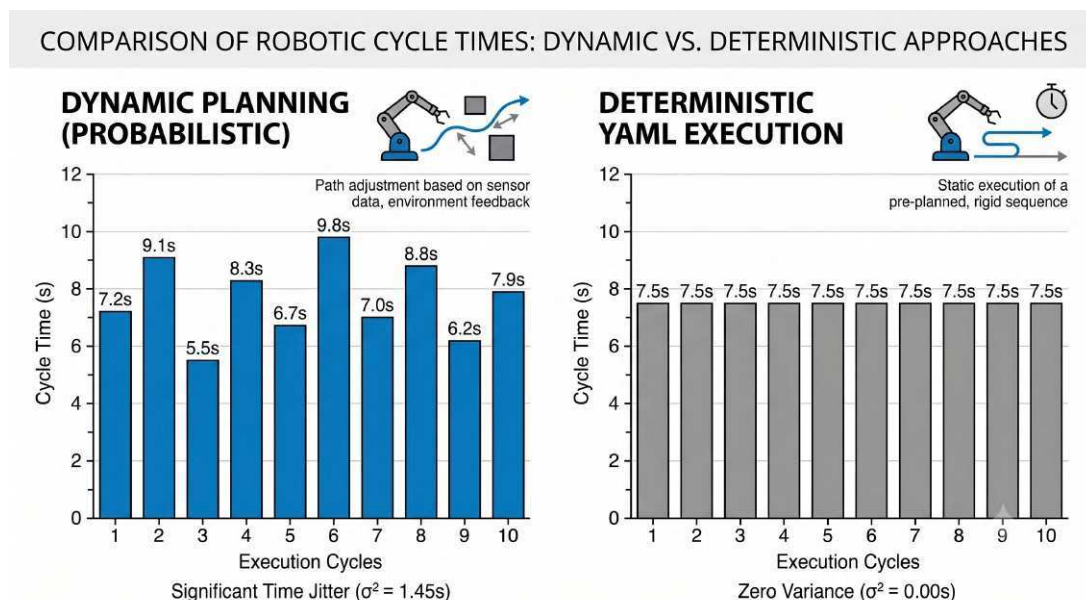
Figure 39: Average cycle time and success rate for pick-and-place tasks, illustrating high temporal repeatability.



Source: Created by the Author.

YAML file is an $O(1)$ complexity operation. As illustrated in Figure 40, this guarantees that the manipulator's motion profile is temporally constant across iterations.

Figure 40: Deterministic YAML execution ($O(1)$): eliminating latency jitter to ensure consistent industrial OEE.

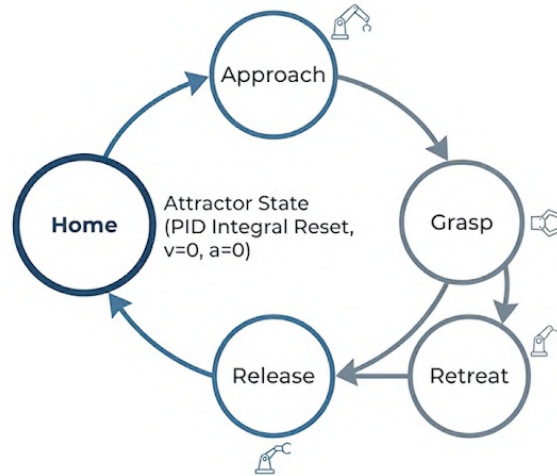


Source: Created by the Author.

Moreover, the system strictly enforces that all operational sequences begin and terminate at a singular “Home” configuration. As modeled in the Finite State Machine (FSM) in Figure 41, this architectural reset neutralizes any accumulated kinematic errors from the previous cycle within the internal PID controllers.

Figure 41: FSM logic: 'Home' configuration as a zero-velocity boundary attractor.

ROBOTIC CYCLE FSM DIAGRAM



Source: Created by the Author.

6.3 OPEN-LOOP DRIFT AND FAILURE ANALYSIS

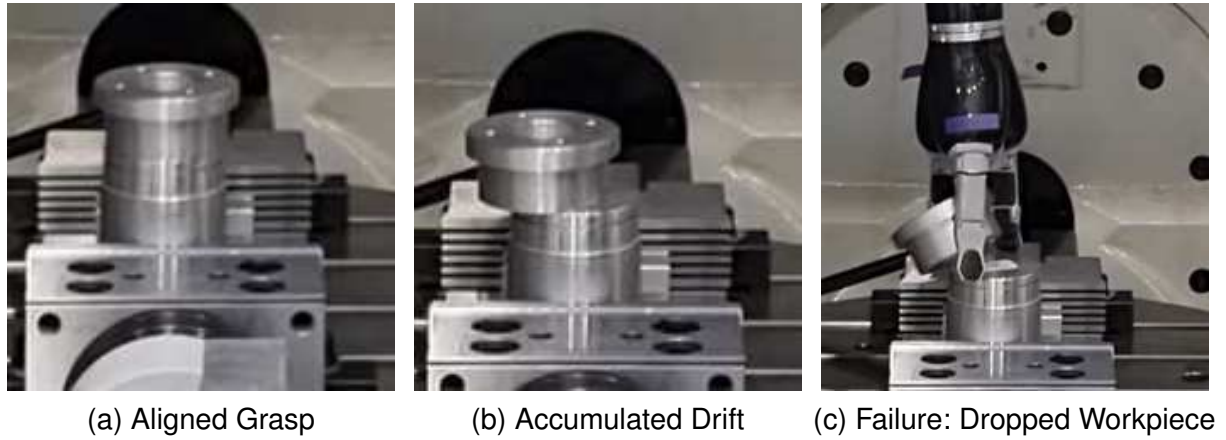
While the temporal execution of the system is highly stable, the experimental scenarios revealed limitations regarding spatial robustness over prolonged continuous cycles. The success rate metrics show that while the ID20 task achieved a 100% success rate, the ID21 task experienced a 20% failure rate (2 drops) over the evaluated cycles.

Because the current architecture relies on predefined joint-space configurations triggered by ArUco marker identification, without a continuous closed-loop visual feedback mechanism to correct the end-effector pose relative to the workpiece during the grasp, small placement offsets accumulate after each release phase.

Visual inspection of the continuous loops confirms this open-loop spatial drift. Over successive cycles, the workpiece gradually shifted laterally from its ideal center position within the two-finger gripper. As illustrated in the chronological progression in Figure 42, the robot initially establishes a perfectly centered grasp during the first cycle (Figure 42a). However, as the continuous execution progresses without intermediate sensory recalibration, a clear lateral offset accumulates by the fourth cycle (Figure 42b). By the time this accumulated drift exceeds the physical tolerance of the CNC fixture and the gripper's grasp stability, the hold becomes critically compromised, causing the workpiece to fall during the transport or insertion phase (Figure 42c).

This phenomenon is mathematically driven by release dynamics and grasping micro-slips. According to the foundational mechanics of robotic manipulation (Mason, 2001), the sudden release of static friction introduces inherent physical indeterminacy

Figure 42: Consecutive grasp cycles: progression of lateral drift (a $\rightarrow$ b) leading to failure (c).

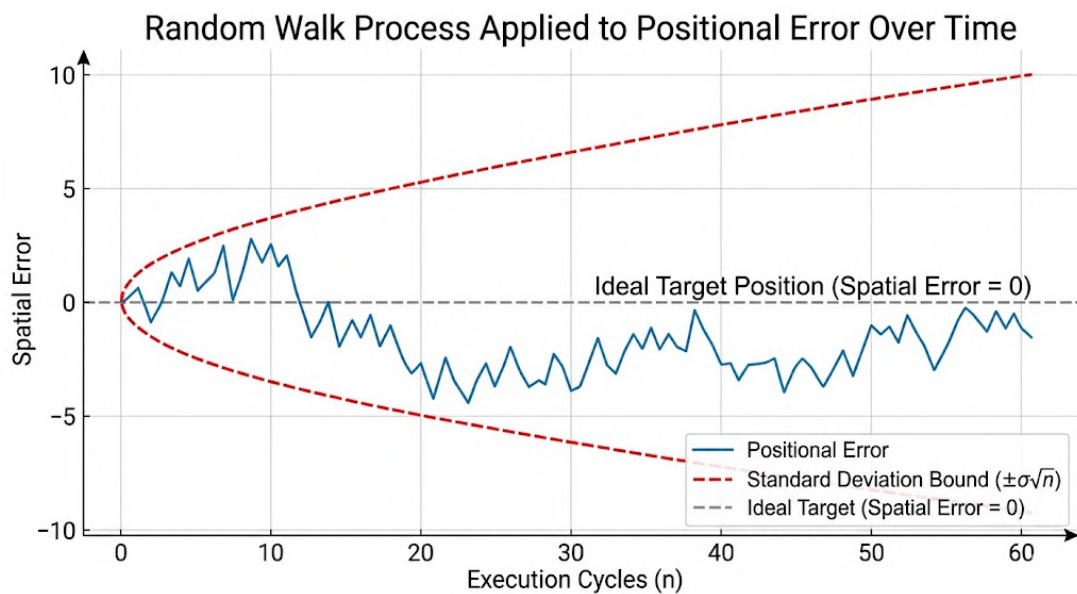


Source: Created by the Author.

into how an object settles. Because the manipulator blindly returns to the exact mathematical joint-space configuration dictated by the YAML recipe in the subsequent cycle, these minute displacements physically accumulate.

This open-loop error accumulation can be mathematically modeled as a stochastic random walk process. Assuming each release event introduces an independent, zero-mean displacement error with variance σ^2 , the standard deviation of the workpiece's physical position diverges proportionally to $\sqrt{n}$, as visually conceptualized in Figure 43.

Figure 43: Stochastic drift accumulation: variance expansion ($\sqrt{n}$) violating gripper tolerances.



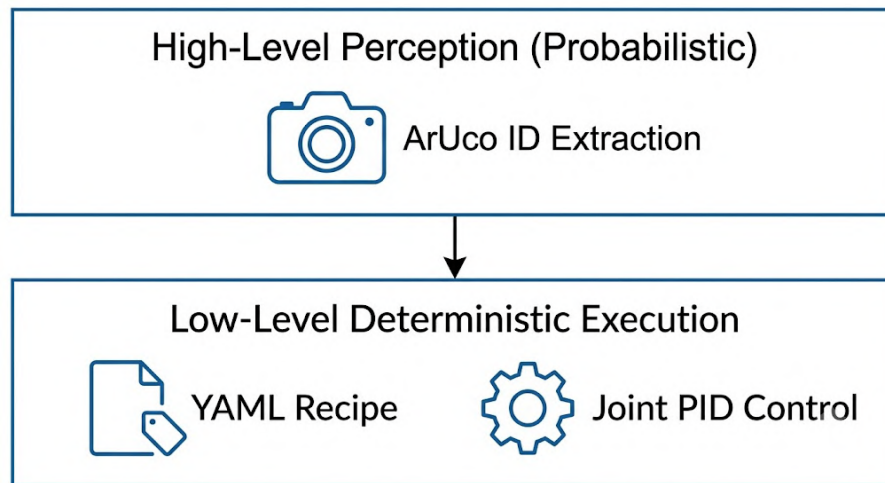
Source: Created by the Author.

This validation confirms that while the ID-to-sequence approach provides excellent short-term repeatability, its long-term robustness is inherently limited by open-loop error propagation, requiring mechanical fixturing compliance to force the workpiece back to a nominal zero-state.

6.4 CURRENT VALIDATED CAPABILITIES

Currently, the system reliably performs ID-to-recipe execution and demonstrates highly effective trajectory shaping. By physically implementing hierarchical decoupling, as illustrated in Figure 44, the custom system prevents perceptual variance from directly corrupting the manipulator's motion profile.

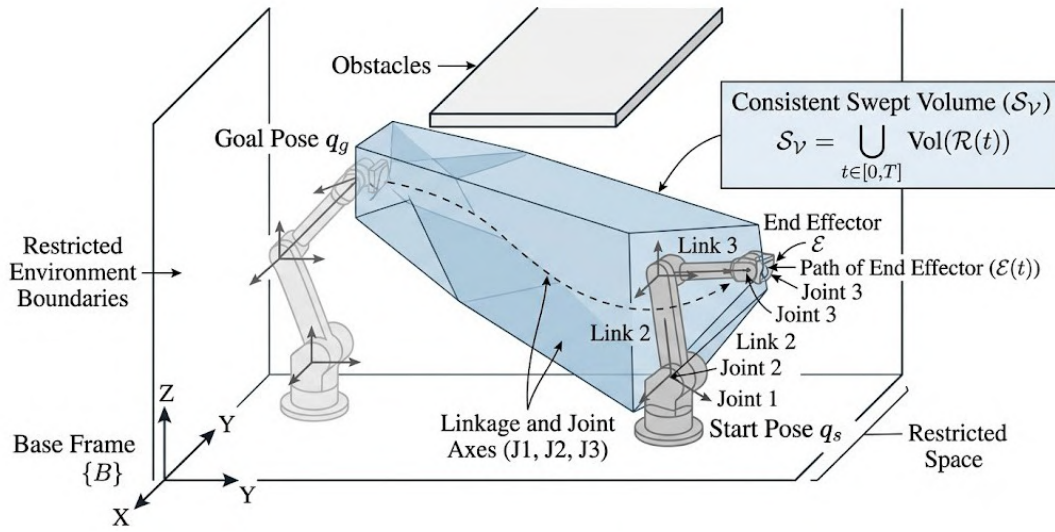
Figure 44: Hierarchical control: decoupled probabilistic perception from deterministic joint-space execution.



Source: Created by the Author.

Furthermore, the trajectory shaping achieved through explicit intermediate waypoints provides a rigorous safety guarantee. The physical space occupied by the manipulator's links during this motion remains perfectly constant across all cycles, carving a mathematically guaranteed, collision-free geometric corridor through the workspace, visualized in Figure 45.

Figure 45: Collision-free swept volume: deterministic joint-space routing ensures workspace safety.



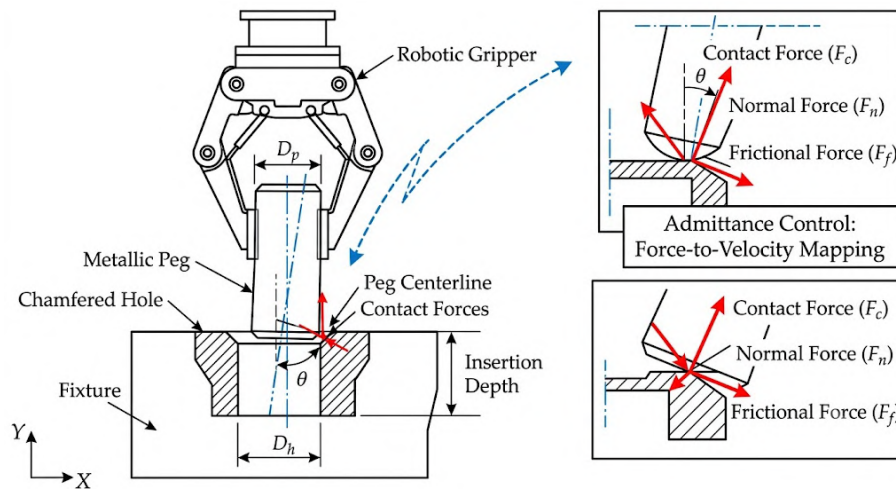
Source: Created by the Author.

6.5 PROPOSED SOLUTIONS FOR FUTURE WORK

To transition this prototype into a fully autonomous, industrial-grade solution capable of long-term machine tending, future research must mitigate the open-loop drift limitation. To achieve this without sacrificing the global stability of the deterministic YAML-based trajectory, a hybrid macro/micro manipulation strategy is recommended. The YAML sequences will serve as the global (macro) routing framework, while local (micro) closed-loop controllers will handle the final millimeters of interaction. Suggested continuations include:

- **Exteroceptive Force Feedback:** Integrating a 6-axis Force/Torque (F/T) sensor at the manipulator's wrist would provide the necessary perceptual hardware to handle dynamic physical interactions. According to the foundational theory of manipulation mechanics (Hogan, 1985), this real-time force feedback allows the control architecture to modulate the robot's dynamic behavior, enabling the implementation of active compliance through formally defined admittance or impedance control strategies. In CNC machine tending, placing a workpiece into a fixture is kinematically analogous to the classic "peg-in-hole" assembly problem (Whitney, 2004). As illustrated in Figure 46, rather than moving blindly to a rigid geometric coordinate, admittance control allows the robot to act mathematically like a spring-mass-damper system. By mapping sensed contact forces to velocity modifications, the manipulator can compliantly "feel" its way into the CNC fixture, dynamically resolving spatial misalignments and jamming forces during the placement phase without triggering safety faults.

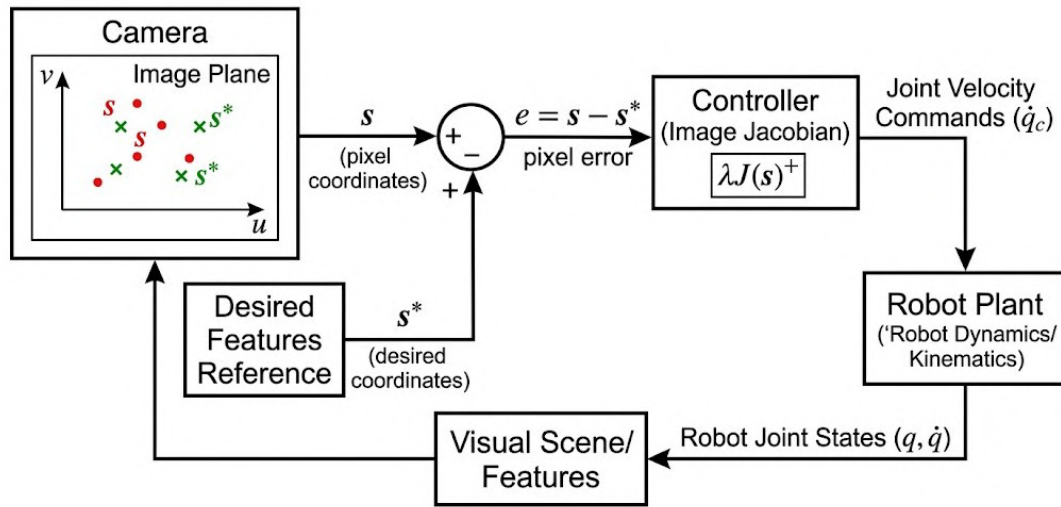
Figure 46: Admittance control in peg-in-hole insertion: compliant resolution of contact forces.



Source: Created by the Author.

- Image-Based Visual Servoing (IBVS):** Instead of relying on a single static pose estimation, which classical computer vision theory shows to be highly sensitive to Hand-Eye calibration errors and depth noise (Chaumette; Hutchinson, 2006), the custom system could implement Image-Based Visual Servoing (IBVS). By utilizing the RealSense camera mounted directly on the end-effector (eye-in-hand setup), the robot can approach the general pre-grasp area using the deterministic YAML recipe, and then smoothly switch to a continuous visual control loop. According to the foundational taxonomy of visual servoing (Hutchinson; Hager; Corke, P. I., 1996), unlike Position-Based Visual Servoing (PBVS), which attempts to reconstruct a 3D pose and is highly sensitive to estimation errors, IBVS computes the control law directly within the 2D image plane. Applying this principle in Figure 47, the custom architecture calculates the pixel error vector between the current and target ArUco corner coordinates. By mapping this specific feature error directly to joint velocities via the classical Image Jacobian, the IBVS controller dynamically guides the gripper to a perfectly aligned pose, effectively rendering the practical approach robust to the underlying 3D calibration inaccuracies.

Figure 47: Proposed IBVS loop: robust 3D pose estimation via 2D feature space minimization.



Source: Created by the Author.

7 CONCLUSION

7.1 SUMMARY OF WORK

This project addressed the critical engineering challenge of integrating legacy assistive robotic hardware into high-precision Reconfigurable Manufacturing Systems (RMS), a manufacturing paradigm specifically designed for adaptable Computer Numerical Control (CNC) machine tending. Modern manufacturing cells require strict millimeter-level insertion tolerances. Initial approaches utilizing continuous 6D pose estimation in ($SE(3)$) and dynamic trajectory planning via MoveIt proved fundamentally unreliable for this specific industrial context. As observed empirically, this instability was primarily driven by the mathematical propagation of spatial errors inherent to the classical $AX = XB$ hand-eye calibration formulation, where minute angular offsets at the camera mount manifest as severe Tool Center Point (TCP) deviations due to the lever-arm effect. Furthermore, empirical testing revealed that the highly reflective metallic surfaces of CNC workpieces severely exacerbated the Intel RealSense L515 sensor's depth noise. This optical interference inherently introduced unacceptable high-frequency Cartesian jitter into the system's online motion planners, the core algorithms responsible for continuous trajectory generation, ultimately rendering the open-loop approach unviable for precision machine tending.

To circumvent these perceptual vulnerabilities, this work engineered a paradigm shift from dynamic Cartesian tracking to a deterministic "ID-to-Recipe" software architecture. The visual perception pipeline, leveraging the foundational ROS middleware, was strictly constrained to perform discrete topological identification. Rather than computing continuous geometric coordinates, the system utilized robust binary decoding of ArUco markers to extract a stable, integer-based identifier.

This discrete ID functioned as a deterministic state-transition trigger, querying a modular YAML database for a pre-recorded, collision-free joint-space sequence. By operating entirely in the joint configuration space ($\mathbb{R}^6$) rather than the Cartesian task space, the custom execution node routed targeted positional vectors directly to the low-level Kinova motor drivers. This architectural choice actively bypassed online sampling-based planners, which are inherently probabilistic, thereby ensuring strict $\mathcal{O}(1)$ computational determinism. Furthermore, by avoiding operational space control entirely, the system completely eliminated the risk of encountering mathematical singularities inherent to inverse Jacobian matrix calculations, a well-documented vulnerability in classical manipulator kinematics.

Finally, to guarantee physical safety and repeatability within the dense CNC workcell, the trajectory was rigidly shaped using manual waypoint authoring. The custom system explicitly managed the manipulator's inherent mechanical compliance, characteristic of its harmonic drive reducers, by enforcing parameterized dwell times (`wait_after`).

According to manipulator dynamics theory, this critical temporal buffer allows structural kinetic oscillations to naturally dampen before end-effector actuation. Consequently, this state of rest ensures optimal static friction conditions during part grasping, effectively mitigating dynamic slippage as dictated by classical manipulation mechanics.

7.2 RESEARCH OBJECTIVES AND LIMITATIONS

The main research objective of this work, deploying and validating an automation workflow based on the foundational Robot Operating System (ROS) to retrofit legacy hardware for CNC machine tending, was successfully accomplished. The custom system effectively bridged the gap between discrete vision-based perception and deterministic robot execution, integrating a 2017 legacy Kinova Jaco2 manipulator with an Intel RealSense L515 sensor within a unified ROS Noetic environment.

The transition to an ArUco-ID-driven recipe system decisively proved its effectiveness over continuous online planners for short-term industrial operations. By decoupling the high-frequency kinematic servoing from the noisy perception pipeline, the custom architecture successfully neutralized the empirical depth noise scattering that caused the dynamic 6D approach to fail. In doing so, it effectively bypassed the geometric vulnerabilities of continuous $SE(3)$ pose tracking and the severe lever-arm calibration effects mathematically predicted by classical hand-eye theory. Furthermore, according to fundamental manipulator kinematics, continuous inverse kinematics (IK) solvers are inherently prone to unpredictable structural reconfigurations and mathematical singularities (Jacobian rank deficiencies). By executing pre-validated deterministic joint-space paths, the proposed system completely avoided these algorithmic risks. Ultimately, this validated the core hypothesis of the project: localized, joint-space recipes significantly improve operational simplicity and short-term execution repeatability within the demanding context of precision manufacturing cells.

However, the transition from a laboratory prototype to an industrial-grade autonomous system revealed significant physical and algorithmic limitations during experimental validation:

- **Hardware Degradation:** Experimental validation was originally intended to include a comparative multi-arm configuration to evaluate collaborative workspace sharing and cycle optimization. However, this objective was abandoned due to critical hardware fatigue discovered in the secondary Kinova Jaco2 manipulator. Actuator wear, which literature confirms severely alters joint friction profiles over extended operation, and specifically structural fatigue within the harmonic drive's flexspline, resulted in reduced joint stiffness and insufficient torque generation in a mid-link actuator. When subjected to the dynamic load of metallic payloads, the weakened actuator suffered from severe positional lag as the internal PID con-

troller saturated. Consequently, this excessive tracking error caused the system logic to trigger protective safety halts. This mechanical limitation restricted the experimental scope to a single-arm setup, empirically highlighting the physical vulnerabilities of repurposing legacy assistive hardware for high-stress industrial tasks.

- **Open-Loop Drift:** The most severe algorithmic limitation empirically encountered during continuous autonomous loops was open-loop drift. Because the custom YAML-based execution relies strictly on proprioceptive dead-reckoning without exteroceptive closed-loop correction, the system is inherently susceptible to unmeasured disturbances, a fundamental vulnerability of open-loop control architectures. During the physical release of the workpiece, transitional contact forces and residual kinetic vibrations introduce microscopic spatial displacements, governed by the mechanics of robotic manipulation. Modeled probabilistically as a stochastic random walk process, these uncorrected micro-displacements inevitably accumulate over successive cycles. Ultimately, the variance of this spatial error grows proportionally to the number of executions. As observed in the workcell, this eventually causes the workpiece to drift beyond the strict sub-millimeter tolerances universally required for CNC fixturing and precision manufacturing, thereby compromising long-term reliability.

7.3 TECHNICAL LESSONS LEARNED

The development and deployment of this automation cell provided several foundational engineering insights regarding the integration of legacy hardware into modern software ecosystems:

- **Strict Decoupling and Information Hiding:** According to foundational manipulator control architectures, isolating high-level, noise-sensitive perception tasks from low-level, high-frequency kinematic servoing is a vital requirement for ensuring physical safety and determinism. In the context of handling rigid metallic parts, the custom architecture was engineered as a Discrete Event System (DES) where the vision pipeline strictly broadcasts integer IDs rather than continuous spatial coordinates. By adopting this specific decoupling strategy, the system successfully adhered to the fundamental software engineering principle of Separation of Concerns (SoC). Ultimately, this structural isolation prevented perception jitter from corrupting the physical trajectory.
- **Kinematic Buffer Management and Friction:** The explicit programming of mechanical dwell times (`wait_after`) proved essential for reliable manipulation. Actuators utilizing harmonic drive reducers inherently introduce structural elasticity

into the kinematic chain. According to classical manipulator dynamics, sudden point-to-point halts cause transient underdamped responses, introducing micro-vibrations at the end-effector. Allowing these kinetic oscillations to naturally dissipate ensures that the tool-to-part interaction occurs under optimal static friction (μ_s) conditions. As dictated by Coulomb friction models, this effectively prevents dynamic slippage during grasping.

- **Environmental Security and Failsafe Halts:** Practical deployments are strictly bound by the physical infrastructure supporting them. Although the laboratory security system possesses backup batteries, their low autonomy causes doors to unlock during extended power outages. This hardware reality necessitated the implementation of strict software-level failsafe halting procedures. In robotic cells operating without continuous human supervision, the control loop must be capable of instantly freezing all joint commands to secure the perimeter when facility-level power interruptions occur.
- **Time Synchronization in Distributed Systems:** Operating a decoupled, node-based middleware like ROS relies heavily on precise temporal coherence. As defined by the foundational architecture of the Coordinate Transform (`tf`) library, all spatial transformations are strictly bound to chronological stamps and finite history buffers; queries falling outside this synchronized cache inevitably trigger system exceptions. Consequently, early deployment phases of this project revealed that asynchronous clock settings across distributed nodes caused the `tf` tree to drop critical messages, evaluating them as stale data. To definitively resolve this architectural vulnerability, ensuring strict hardware clock synchronization (and explicitly disabling simulated time) across the entire industrial PC environment was established as a mandatory pre-condition for stable robotic perception.
- **Data-Driven Reconfigurability:** Hardcoding trajectories directly into compiled nodes creates a monolithic architecture that hinders operational flexibility. By structuring the deterministic routes into human-readable YAML configurations, the custom system drastically reduced setup times. This architectural approach allows shop-floor operators to adjust machine-tending parameters without source-code recompilation, directly fulfilling the agility and rapid ramp-up requirements of Reconfigurable Manufacturing Systems (RMS).
- **Passive Mechanical Compliance:** According to fundamental manufacturing principles, precision industrial CNC workcells must heavily rely on rigorous mechanical fixturing to ensure exact positioning and repeatability. To sustainably arrest the empirically observed open-loop drift in the purely deterministic robotic setup without requiring real-time spatial updates, this project leveraged the concept of

passive mechanical compliance. By enhancing the physical workcell with specialized custom fixturing, such as chamfered guides and pneumatic self-centering jaws, the system physically forces the workpiece back into its nominal zero-state position at each placement. Ultimately, this physical approach effectively breaks the stochastic error accumulation chain and restores long-term repeatability without adding any computational overhead to the control architecture.

7.4 IMPACTS OF THE RESEARCH

This project generated substantial impacts across multiple domains for the Werkzeugmaschinenlabor (WZL) and the broader automation field:

- **Technological and Organizational (Agility for SMEs):** This work provides an open-source, vendor-independent blueprint for Reconfigurable Manufacturing Systems (RMS). Traditional industrial robots often operate on proprietary, closed-source controllers, creating a technological lock-in that is rigid and cost-prohibitive. This barrier is particularly critical for Small and Medium-sized Enterprises (SMEs) transitioning to Industry 4.0 paradigms through equipment retrofitting. By utilizing ROS as a foundational abstraction layer and structuring the execution through human-readable YAML configurations, shop-floor operators can rapidly adjust trajectories to accommodate High-Mix, Low-Volume (HMLV) batch production. Ultimately, this custom architecture completely bypasses the need for complex source code recompilation, significantly reducing reprogramming downtime and empowering organizational agility.
- **Financial and Ecological (Sustainable Retrofitting):** According to recent literature on cyber-physical systems, technological retrofitting provides a highly sustainable engineering framework that effectively bypasses the massive capital expenditure typically required for complete infrastructure replacement. Aligned with this paradigm, this project empirically validates the concept by rehabilitating a 2017 legacy Kinova Jaco2 arm, hardware originally designed for low-torque assistive medical tasks, into a viable platform for advanced manufacturing research. Ultimately, this sustainable approach actively reduces electronic waste and demonstrates how institutions can maximize their existing hardware asset lifecycles.
- **Academic and Institutional (WZL and IoP):** Academically, this implementation directly supports the “Internet of Production” (IoP) excellence cluster at the WZL. By advancing the deployment of flexible, digitized automation cells, formally recognized in the literature as Cyber-Physical Production Systems (CPPS), the project successfully aligns legacy hardware with modern Industry 4.0 paradigms. Further-

more, the detailed documentation of the system's physical boundaries, specifically the degradation of cycles due to open-loop drift, provides foundational empirical data. Ultimately, the developed ID-to-Recipe architecture serves as a stable baseline benchmark for the institute's future research initiatives, particularly for the development of hybrid control architectures where this low-overhead open-loop routine can be augmented with discrete closed-loop validation steps.

7.5 FUTURE WORK

To transition this prototype into a fully autonomous, industrial-grade solution capable of long-term machine tending, future research must mitigate the open-loop drift limitation. As detailed in Section 6.5, a hybrid macro/micro manipulation strategy is recommended. The YAML sequences will serve as the global (macro) routing framework, while local (micro) closed-loop controllers will handle the final millimeters of interaction.

Specifically, the integration of a 6-axis Force/Torque (F/T) sensor to enable active compliance (admittance control), and the implementation of Image-Based Visual Servoing (IBVS) using an eye-in-hand setup, are the primary pathways to bridge the gap between deterministic motion and dynamic adaptability, evolving the robotic cell into a fully autonomous solution.

REFERENCES

ARIF, Zubair; FU, Yili. Mix Frame Visual Servo Control Framework for Autonomous Assistive Robotic Arms. **Sensors**, v. 22, Jan. 2022. DOI: 10.3390/s22020642.

BI, Z. M.; ZHANG, W. J. Flexible fixture design and automation: Review, issues and future directions. **International Journal of Production Research**, Taylor & Francis, v. 39, n. 13, 2001. DOI: 10.1080/00207540110054579.

BIAGIOTTI, Luigi; MELCHIORRI, Claudio. **Trajectory Planning for Automatic Machines and Robots**. Berlin, Heidelberg: Springer-Verlag, 2008. ISBN 978-3-540-85628-3.

BIRGLEN, Lionel; LALIBERTÉ, Thierry; GOSSELIN, Clément M. **Underactuated Robotic Hands**. Berlin, Heidelberg: Springer, 2008. ISBN 978-3-540-77458-7.

BITTENCOURT, André Carvalho. **Modeling and Diagnosis of Friction and Wear in Industrial Robots**. 2014. PhD thesis – Linköping University Electronic Press, Sweden.

BOOTHROYD, Geoffrey; DEWHURST, Peter; KNIGHT, Winston A. **Product Design for Manufacture and Assembly**. 3rd. Boca Raton, FL: CRC Press, 2010.

CAMPEAU-LECOURS, Alexandre; BÉLANGER, Pierre-Luc; BÉRUBÉ, Marc-Antoine; FERLAND, François; LECOURS, Alexandre; LÉTOURNEAU, Dominic; GOSSELIN, Clément. Kinova Modular Robot Arms for Service Robotics Applications. *In: SERVICE Robotics within the Digital Transformation*. [S. l.: s. n.], 2019.

CAMPEAU-LECOURS, Alexandre *et al.* Kinova assistive robot for people with upper-limb disabilities: Freeing and making the impossible possible. *In: IEEE. 2017 International Conference on Rehabilitation Robotics (ICORR)*. [S. l.: s. n.], 2017.

CASSANDRAS, Christos G.; LAFORTUNE, Stéphane. **Introduction to Discrete Event Systems**. 2nd. New York, NY: Springer, 2008. ISBN 978-0387333328.

CHAUMETTE, François; HUTCHINSON, Seth. Visual servo control. I. Basic approaches. **IEEE Robotics & Automation Magazine**, IEEE, v. 13, n. 4, 2006. Classic paper explaining why open-loop fails and closed-loop visual feedback is needed.

CHITTA, Sachin; SUCAN, Ioan; COUSINS, Steve. MoveIt! [ROS Topics]. **IEEE Robotics & Automation Magazine**, 2012. DOI: 10.1109/MRA.2011.2181749.

CORKE, Peter. **Robotics, Vision and Control: Fundamental Algorithms in MATLAB**. [S. l.]: Springer, 2011. v. 73. (Springer Tracts in Advanced Robotics). ISBN 978-3-642-20143-1.

CRAIG, John J. **Introduction to Robotics: Mechanics and Control**. 3. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2005. ISBN 0-13-123629-6.

CURTO, Eva; ARAUJO, Helder. An Experimental Assessment of Depth Estimation in Transparent and Translucent Scenes for Intel RealSense D415, SR305 and L515. **Sensors**, MDPI, v. 22, n. 19, 2022. DOI: 10.3390/s22197378.

DIJKSTRA, Edsger W. On the Role of Scientific Thought. *In*: SELECTED Writings on Computing: A Personal Perspective. [S. l.]: Springer, 1982.

FOOTE, Tully. tf: The transform library. *In*: 2013 IEEE International Conference on Technologies for Practical Robot Applications (TePRA). [S. l.: s. n.], 2013. DOI: 10.1109/TePRA.2013.6556373.

GARRIDO-JURADO, Sergio; MUÑOZ-SALINAS, Rafael; MADRID-CUEVAS, Francisco José; MARÍN-JIMÉNEZ, Manuel Jesús. Automatic generation and detection of highly reliable fiducial markers under occlusion. **Pattern Recognition**, Elsevier, v. 47, n. 6, 2014.

GOLNABI, H.; ASADPOUR, A. Design and application of industrial machine vision systems. **Robotics and Computer-Integrated Manufacturing**, Elsevier, v. 23, n. 6, 2007. Classic overview of the environmental challenges (lighting, hazards) for vision systems in manufacturing.

GROOVER, Mikell P. **Automation, Production Systems, and Computer-Integrated Manufacturing**. 4. ed. Boston, MA: Pearson, 2015. ISBN 978-0-13-349961-2.

HARTLEY, Richard; ZISSERMAN, Andrew. **Multiple View Geometry in Computer Vision**. 2nd. [S. l.]: Cambridge University Press, 2004.

HOGAN, Neville. Impedance control: An approach to manipulation. *In*: IEEE. PROCEEDINGS of the 1985 American Control Conference. [S. l.: s. n.], 1985.

HUTCHINSON, Seth; HAGER, Gregory D.; CORKE, Peter I. A tutorial on visual servo control. **IEEE Transactions on Robotics and Automation**, IEEE, v. 12, n. 5, p. 651–670, 1996. DOI: 10.1109/70.538972.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Manipulating industrial robots — Performance criteria and related test methods**. Geneva, Switzerland, Apr. 1998.

KAVRAKI, Lydia E.; ŠVESTKA, Petr; LATOMBE, Jean-Claude; OVERMARS, Mark H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. **IEEE Transactions on Robotics and Automation**, IEEE, v. 12, n. 4, 1996. DOI: 10.1109/70.508439.

KHOSHELHAM, Kourosh; ELBERINK, Sander Oude. Accuracy and resolution of Kinect depth data for indoor mapping applications. **Sensors**, Molecular Diversity Preservation International, v. 12, n. 2, 2012.

KINOVA ROBOTICS. **Jaco Assistive Robotic Arm**. [S. l.: s. n.], 2026.
<https://assistive.kinovarobotics.com/product/jaco-robotic-arm>.

KOREN, Y.; HEISEL, U.; JOVANE, F.; MORIWAKI, T.; PRITSCHOW, G.; ULSOY, G.; VAN BRUSSEL, H. Reconfigurable Manufacturing Systems. **Annals of the CIRP**, v. 48, n. 2, 1999.

KRAGIC, Danica; CHRISTENSEN, Henrik I. Survey on visual servoing for manipulation. **Computational Vision and Active Perception Laboratory, Royal Institute of Technology, Stockholm, Sweden**, 2002.

LAVALLE, Steven M. **Planning Algorithms**. [S. l.]: Cambridge University Press, 2006. The standard text for robotic path planning, waypoints, and avoiding kinematic singularities.

LEPETIT, Vincent; MORENO-NOGUER, Francesc; FUA, Pascal. EPnP: An accurate $O(n)$ solution to the PnP problem. **International Journal of Computer Vision**, Springer, v. 81, n. 2, 2009.

LINS, Theo; OLIVEIRA, Ricardo Augusto Rabelo. Cyber-physical production systems retrofitting in context of industry 4.0. **Computers & Industrial Engineering**, Elsevier, v. 139, 2020. DOI: 10.1016/j.cie.2019.106193.

LYNCH, Kevin M.; PARK, Frank C. **Modern Robotics: Mechanics, Planning, and Control**. Cambridge, UK: Cambridge University Press, 2017. ISBN 978-1-107-15630-2.

MASON, Matthew T. **Mechanics of Robotic Manipulation**. Cambridge, MA: MIT Press, 2001. ISBN 978-0-262-13396-8.

MUCHIRI, Peter; PINTELON, Liliane. Performance measurement using overall equipment effectiveness (OEE): literature review and practical application discussion. **International Journal of Production Research**, Taylor & Francis, v. 46, n. 13, 2008. DOI: 10.1080/00207540601142645.

NTCHIP. **Difference Between Open Loop and Closed Loop**. [S. l.: s. n.], 2026.
<https://www.ntchip.com/electronics-news/difference-between-open-loop-and-closed-loop>.

OGATA, Katsuhiko. **Modern Control Engineering**. 5. ed. Upper Saddle River, NJ: Prentice Hall, 2010. ISBN 978-0-13-615673-4.

PARNAS, David L. On the Criteria To Be Used in Decomposing Systems into Modules. **Communications of the ACM**, ACM, v. 15, n. 12, 1972.

PERZYLO, Alexander *et al.* SMERobotics: Smart Robots for Flexible Manufacturing. **IEEE Robotics & Automation Magazine**, Mar. 2019. DOI: 10.1109/MRA.2018.2879747.

QUIGLEY, Morgan; GERKEY, Brian; CONLEY, Ken; FAUST, Josh; FOOTE, Tully; LEIBS, Jeremy; BERGER, Eric; WHEELER, Rob; NG, Andrew. ROS: an open-source Robot Operating System. *In*: ICRA Workshop on Open Source Software. [S. l.: s. n.], 2009.

REALSENSE, Intel. **realsense-ros: ROS wrapper for Intel RealSense devices**. [S. l.: s. n.], 2026. Disponível em: <https://github.com/IntelRealSense/realsense-ros>. Available from: <https://github.com/IntelRealSense/realsense-ros>.

ROBOTICS, Kinova. **kinova-ros: ROS stack for Kinova robots**. [S. l.: s. n.], 2026. Disponível em: <https://github.com/Kinovarobotics/kinova-ros>. Available from: <https://github.com/Kinovarobotics/kinova-ros>.

ROBOTICS, PAL. **aruco_ros: ROS wrapper for ArUco marker detection**. [S. l.: s. n.], 2026. Disponível em: https://github.com/pal-robotics/aruco_ros. Available from: https://github.com/pal-robotics/aruco_ros.

ROMERO, Francisco; MUNOZ, Rafael; MEDINA, Rafael. Speeded up detection of squared fiducial markers. **Image and Vision Computing**, v. 76, Aug. 2018. DOI: 10.1016/j.imavis.2018.05.004. Available from: <https://doi.org/10.1016/j.imavis.2018.05.004>.

ROSSUM, Guido van; WARSAW, Barry; COGHLAN, Nick. **PEP 8 – Style Guide for Python Code**. [S. l.: s. n.], 2001. Python Enhancement Proposals. Created: 05-Jul-2001. <https://peps.python.org/pep-0008/>.

SHIU, Yin-Cheng; AHMAD, Shaheen. Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form $AX=XB$. **IEEE Transactions on Robotics and Automation**, IEEE, v. 5, n. 1, 1989. DOI: 10.1109/70.88014.

SICILIANO, Bruno; SCIAVICCO, Lorenzo; VILLANI, Luigi; ORIOLO, Giuseppe. **Robotics: Modelling, Planning and Control**. London: Springer, 2010. ISBN 978-1-84628-641-4.

SMYTH, Daniel. **Trabalhador operando morsa em máquina CNC**. [S. l.: s. n.], 2026. <https://www.pexels.com/pt-br/procurar/?q=metal%20workpiece%20clamped%20in%20cnc>. Imagem obtida via banco de imagens Pexels.

SOMMERVILLE, Ian. **Software Engineering**. 9th. Boston, MA: Addison-Wesley, 2011. ISBN 978-0-13-703515-1.

SPONG, Mark W.; HUTCHINSON, Seth; VIDYASAGAR, M. **Robot Modeling and Control**. New York: Wiley, 2006. ISBN 0471765791.

STOCK, Tim; SELIGER, Günther. Opportunities of Sustainable Manufacturing in Industry 4.0. **Procedia CIRP**, Elsevier, v. 40, 2016.

TENTONE. **ArUco: A minimal C++ library for Augmented Reality applications based on OpenCV**. [S. l.]: GitHub, 2021. <https://github.com/tentone/aruco>.

TSAI, Roger Y; LENZ, Reimar K. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. **IEEE Transactions on robotics and automation**, IEEE, v. 5, n. 3, 1989.

WHITNEY, Daniel E. **Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development**. New York, NY: Oxford University Press, 2004. ISBN 978-0195157826.

APPENDIX A – SUPPLEMENTARY MULTIMEDIA CONTENT

This appendix provides multimedia evidence documenting the development, testing, and diagnostic analysis of the proposed machine-tending architecture. These resources serve to validate system performance under various operational conditions.

Ref. ID	Description	QR Code/Link
Video 1	Manual Trajectory Authoring: Demonstration of the waypoint recording process for the YAML recipe database.	LINK
Video 2	Telemetry Visualization: Continuous operation monitoring using ROS logs and real-time terminal feedback.	LINK
Video 3	Kinematic Performance Analysis: Comparative test showing the nominal response of Arm 1 versus the degraded performance of Arm 2.	LINK
Video 4	Adaptability Validation: System performance during the manipulation of smaller payload geometries.	LINK
Video 5	Repetitive Cycle Testing: Continuous pick-and-place execution validating system repeatability.	LINK

Table 7 – Index of supplementary multimedia materials.

A.1 DIAGNOSTIC NOTES ON KINEMATIC LIMITATIONS

Regarding Video 3 (Kinematic Performance Analysis), it should be noted that the performance degradation observed in Arm 2 was essential for diagnosing structural gear fatigue in the legacy Kinova hardware. This failure was instrumental in defining the system's operational boundaries and emphasizing the necessity of maintenance protocols for repurposed industrial manipulators.

APPENDIX B – SOFTWARE IMPLEMENTATION AND CONFIGURATION

Note: The source code presented in this appendix represents original work developed specifically for this research project, including the state machine logic and the YAML-based trajectory sequencer.

This appendix provides the full implementation details of the custom software developed for the machine-tending robotic cell.

B.1 CONTROL ORCHESTRATOR: MARKER_TO_JACO.PY

This script is the core execution engine of the automation cell. It implements the state machine that processes the ArUco marker IDs and executes the corresponding joint-space trajectories.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5  import rospy
6  import actionlib
7  import time
8
9  from std_msgs.msg import UInt32MultiArray
10 from kinova_msgs.msg import (
11         ArmJointAnglesAction,
12         ArmJointAnglesGoal,
13         JointAngles,
14         SetFingersPositionAction,
15         SetFingersPositionGoal,
16         FingerPosition,
17 )
18
19
20 class MarkerToJaco(object):
21     def __init__(self):
22         rospy.init_node('marker_to_jaco')
23
24         self.robot_name = rospy.get_param('~robot_name', 'j2n6s300')
25         self.stable_count_required = rospy.get_param('~stable_count', 5)
26         self.markers_topic = rospy.get_param(
27                 '~markers_list_topic',
28                 '/aruco_marker_publisher/markers_list'
29         )
30
31     # finger_max: maximum raw value (Jaco2 ~ 6800)

```

```

32     self.finger_max = rospy.get_param('~finger_max', 6800.0)
33     # timeout and tolerance to wait for fingers to reach the target
34     self.finger_timeout = rospy.get_param('~finger_timeout', 5.0)
35     # seconds
36     self.finger_tol_ratio = rospy.get_param('~finger_tol_ratio',
37     0.08)    # 8% of the range
38
39     # Markers configuration (ID -> command)
40     self.markers_cfg = rospy.get_param('~markers', {})
41     if not self.markers_cfg:
42         rospy.logwarn("No configuration found in ~markers.")
43
44     # State for stable detection
45     self.last_marker_id = None
46     self.current_count = 0
47     self.triggered_for_current = False # has the command been
48     # executed for the current marker?
49
50     # Fingers and joints state (read from topics)
51     self.current_finger_pos = None
52     self.current_joint_angles = None # [j1, j2, j3, j4, j5, j6]
53
54     # Action clients
55     joint_action_name = '{}/_driver/joints_action/joint_angles'.
56     format(self.robot_name)
57     finger_action_name = '{}/_driver/fingers_action/finger_positions
58     '.format(self.robot_name)
59
60     rospy.loginfo("Connecting to joints action: %s",
61     joint_action_name)
62     self.joint_client = actionlib.SimpleActionClient(
63     joint_action_name,
64     ArmJointAnglesAction
65     )
66     self.joint_client.wait_for_server()
67     rospy.loginfo("Connected to joints action.")
68
69     rospy.loginfo("Connecting to fingers action: %s",
70     finger_action_name)
71     self.finger_client = actionlib.SimpleActionClient(
72     finger_action_name,
73     SetFingersPositionAction
74     )
75     self.finger_client.wait_for_server()
76     rospy.loginfo("Connected to fingers action.")
77
78     # ArUco IDs subscriber

```

```

72         self.sub = rospy.Subscriber(
73             self.markers_topic,
74             UInt32MultiArray,
75             self.markers_callback,
76             queue_size=1
77         )
78
79         # Fingers current position subscriber
80         finger_state_topic = '{}/_driver/out/finger_position'.format(
self.robot_name)
81         rospy.Subscriber(
82             finger_state_topic,
83             FingerPosition,
84             self.finger_state_cb,
85             queue_size=1
86         )
87
88         # --- NEW: Joint angles current position subscriber ---
89         joint_state_topic = '{}/_driver/out/joint_angles'.format(self.
robot_name)
90         rospy.Subscriber(
91             joint_state_topic,
92             JointAngles,
93             self.joint_state_cb,
94             queue_size=1
95         )
96
97         rospy.loginfo("Listening for states on: %s and %s",
finger_state_topic, joint_state_topic)
98         rospy.loginfo("marker_to_jaco started. Listening to %s", self.
markers_topic)
99
100         # -----
101         # Callbacks / State
102         # -----
103
104         def finger_state_cb(self, msg):
105             """Callback to store current finger position."""
106             self.current_finger_pos = msg
107
108         def joint_state_cb(self, msg):
109             """
110             Callback to store current joint positions.
111             Converts the JointAngles message to a list of floats for easier
handling.
112             """
113             self.current_joint_angles = [

```

```

114         msg.joint1, msg.joint2, msg.joint3,
115         msg.joint4, msg.joint5, msg.joint6
116     ]
117
118     # -----
119     # Command helper functions
120     # -----
121
122     def send_single_joint_command(self, joints_deg):
123         """
124         Sends a single absolute joint command (degrees) to the JACO2.
125         Used internally by the sequential function.
126         """
127         if not joints_deg or len(joints_deg) < 6:
128             return False
129
130         angles = JointAngles()
131         angles.joint1 = joints_deg[0]
132         angles.joint2 = joints_deg[1]
133         angles.joint3 = joints_deg[2]
134         angles.joint4 = joints_deg[3]
135         angles.joint5 = joints_deg[4]
136         angles.joint6 = joints_deg[5]
137
138         goal = ArmJointAnglesGoal()
139         goal.angles = angles
140
141         # Send and wait
142         self.joint_client.send_goal(goal)
143         self.joint_client.wait_for_result()
144         return True
145
146     def send_joint_goal_sequential(self, target_deg):
147         """
148         Moves the robot to 'target_deg' moving ONE joint at a time.
149         Order: J1 -> J2 -> ... -> J6.
150         """
151         if not target_deg or len(target_deg) < 6:
152             rospy.logwarn("Invalid joints_deg: %s", str(target_deg))
153             return False
154
155         # Verify if we have current reading to start
156         if self.current_joint_angles is None:
157             rospy.logwarn("No current joint reading available (topic /
out/joint_angles). Aborting.")
158             return False
159

```

```

160     rospy.loginfo("Starting SEQUENTIAL movement to: %s", str(
161         target_deg))
162
163     # We need a local copy of the current state to modify
164     # We start where the robot IS NOW.
165     command_vector = list(self.current_joint_angles)
166
167     # Loop through the 6 joints
168     for i in range(6):
169         # If the target is very close to current (< 1 degree), skip
170         # to save time
171         diff = abs(target_deg[i] - command_vector[i])
172         if diff < 1.0:
173             continue
174
175         rospy.loginfo("Moving Joint %d: from %.1f to %.1f", i+1,
176             command_vector[i], target_deg[i])
177
178         # Update only joint 'i' in the command vector
179         command_vector[i] = target_deg[i]
180
181         # Send command
182         success = self.send_single_joint_command(command_vector)
183         if not success:
184             rospy.logerr("Failure moving joint %d", i+1)
185             return False
186
187         # Small pause for stabilization (optional, helps avoid jerks
188         # between joints)
189         rospy.sleep(0.1)
190
191         rospy.loginfo("Sequential movement completed.")
192         return True
193
194     def wait_for_fingers_reached(self, targets_raw):
195         """
196         Wait until the fingers are close to the target (in raw values),
197         or until timeout occurs.
198         """
199         tol = self.finger_tol_ratio * self.finger_max
200         deadline = rospy.Time.now() + rospy.Duration(self.finger_timeout)
201
202         while not rospy.is_shutdown() and rospy.Time.now() < deadline:
203             if self.current_finger_pos is None:
204                 rospy.sleep(0.05)
205                 continue

```

```

202
203         e1 = abs(self.current_finger_pos.finger1 - targets_raw[0])
204         e2 = abs(self.current_finger_pos.finger2 - targets_raw[1])
205         e3 = abs(self.current_finger_pos.finger3 - targets_raw[2])
206
207         if e1 < tol and e2 < tol and e3 < tol:
208             rospy.loginfo(
209                 "Fingers reached target (error [%.1f, %.1f, %.1f] <
210                 tol=%.1f)",
211                 e1, e2, e3, tol
212             )
213             return True
214
215         rospy.sleep(0.05)
216
217         if self.current_finger_pos is None:
218             rospy.logwarn("Could not read finger position within timeout
219             .")
220         else:
221             e1 = abs(self.current_finger_pos.finger1 - targets_raw[0])
222             e2 = abs(self.current_finger_pos.finger2 - targets_raw[1])
223             e3 = abs(self.current_finger_pos.finger3 - targets_raw[2])
224             rospy.logwarn(
225                 "Fingers did NOT reach target (timeout). Final error
226                 [%.1f, %.1f, %.1f]", e1, e2, e3
227             )
228             return False
229
230     def send_fingers_goal(self, fingers_norm):
231         """
232         Sends finger command with retries.
233         fingers_norm: list [f1, f2, f3] between 0.0 (open) and 1.0 (
234         closed).
235         """
236         if not fingers_norm or len(fingers_norm) < 3:
237             rospy.logwarn("Invalid fingers: %s", str(fingers_norm))
238             return False
239
240         # Normalize 0..1
241         f1 = max(0.0, min(1.0, float(fingers_norm[0])))
242         f2 = max(0.0, min(1.0, float(fingers_norm[1])))
243         f3 = max(0.0, min(1.0, float(fingers_norm[2])))
244
245         # Convert to raw target value
246         t1 = f1 * self.finger_max
247         t2 = f2 * self.finger_max
248         t3 = f3 * self.finger_max

```

```

245     targets_raw = [t1, t2, t3]
246
247     retries = int(rospy.get_param('~finger_retries', 3))
248
249     for attempt in range(1, retries + 1):
250         finger_pos = FingerPosition()
251         finger_pos.finger1 = t1
252         finger_pos.finger2 = t2
253         finger_pos.finger3 = t3
254
255         goal = SetFingersPositionGoal()
256         goal.fingers = finger_pos
257
258         rospy.loginfo(
259             "Attempt %d/%d - Fingers (norm): [%.2f, %.2f, %.2f]",
260             attempt, retries, f1, f2, f3
261         )
262
263         self.finger_client.send_goal(goal)
264         self.finger_client.wait_for_result()
265
266         if self.wait_for_fingers_reached(targets_raw):
267             return True
268
269         rospy.logwarn("Attempt %d failed. Retrying...", attempt)
270
271         rospy.logwarn("Failure to reach finger target after retries.")
272         return False
273
274     # -----
275     # Command execution based on config
276     # -----
277
278     def execute_marker_command(self, marker_id):
279         key = str(marker_id)
280         if key not in self.markers_cfg:
281             rospy.logwarn("Marker id %s detected, but not configured.",
key)
282             return
283
284         cfg = self.markers_cfg[key]
285         cmd_type = cfg.get('type', 'joints')
286
287         if cmd_type == 'joints':
288             joints_deg = cfg.get('joints_deg', None)
289             fingers = cfg.get('fingers', None)
290

```

```

291         # CHANGED: Calls sequential function instead of direct
292         if joints_deg and self.send_joint_goal_sequential(joints_deg
293     ):
294         if fingers is not None:
295             self.send_fingers_goal(fingers)
296
297         elif cmd_type == 'sequence':
298             seq = cfg.get('sequence', [])
299             if not seq:
300                 return
301
302             rospy.loginfo("Executing sequence for marker %d (%d steps)."
303 , marker_id, len(seq))
304
305             for i, step in enumerate(seq):
306                 name = step.get('name', 'step_%d' % i)
307                 joints_deg = step.get('joints_deg', None)
308                 fingers = step.get('fingers', None)
309                 wait_after = float(step.get('wait_after', 0.0))
310
311                 rospy.loginfo("--- Step %d: %s ---", i, name)
312
313                 # CHANGED: Calls sequential function
314                 if joints_deg is not None:
315                     success = self.send_joint_goal_sequential(joints_deg
316 )
317
318                     if not success:
319                         rospy.logerr("Aborting sequence due to movement
320 failure.")
321
322                         break
323
324                     if fingers is not None:
325                         self.send_fingers_goal(fingers)
326
327                     if wait_after > 0.0:
328                         rospy.loginfo("Waiting %.2f s...", wait_after)
329                         rospy.sleep(wait_after)
330
331                 else:
332                     rospy.logwarn("Unknown type: %s", cmd_type)
333
334             # -----
335             # ArUco IDs Callback
336             # -----
337
338             def markers_callback(self, msg):
339                 if not msg.data:
340                     self.last_marker_id = None

```

```

334         self.current_count = 0
335         self.triggered_for_current = False
336         return
337
338     marker_id = int(msg.data[0])
339
340     if marker_id == self.last_marker_id:
341         self.current_count += 1
342     else:
343         self.last_marker_id = marker_id
344         self.current_count = 1
345         self.triggered_for_current = False
346
347     if self.current_count < self.stable_count_required:
348         return
349
350     if self.triggered_for_current:
351         return
352
353     rospy.loginfo("Stable marker: %d. Executing...", marker_id)
354     self.triggered_for_current = True
355     self.execute_marker_command(marker_id)
356
357
358 if __name__ == '__main__':
359     try:
360         node = MarkerToJaco()
361         rospy.spin()
362     except rospy.ROSInterruptException:
363         pass

```

Listing B.1 – Main execution script for marker-to-joint mapping

B.2 TASK RECIPES: MARKER_POSES_SEQUENTIAL.YAML

This configuration file serves as the knowledge base for the robotic cell. It contains the deterministic joint-space waypoints and gripper configurations for each recognized workpiece.

```

1 markers:
2   # --- Composite Marker ---
3   "19":
4     type: sequence
5     description: "Optimized Sequence Robot2 - PICK Small round part"
6     sequence:
7       # --- START (OUTBOUND) ---
8       - name: "1 - Home"

```

```

9     joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
10     fingers:    [0.0, 0.0, 0.0]
11     wait_after: 1.0
12
13     # INTERMEDIATE: Smoothens J2 and J3 descent
14     - name: "1.5 - Approach"
15       joints_deg: [286.0, 190.0, 71.0, 240.0, 81.0, 77.0]
16       fingers:    [0.0, 0.0, 0.0]
17       wait_after: 0.1
18
19     - name: "2 - Pre-position"
20       joints_deg: [298.0, 215.0, 86.0, 240.0, 79.0, 78.0]
21       wait_after: 0.5
22
23     # INTERMEDIATE: Helps J4 rotate from 240 to 348 without jerking
24     - name: "2.5 - J4 Rotation"
25       joints_deg: [292.0, 225.0, 103.0, 294.0, 75.0, 78.0]
26       wait_after: 0.1
27
28     - name: "3 - Fine Alignment"
29       joints_deg: [287.0, 234.0, 121.0, 348.0, 70.0, 79.0]
30       wait_after: 0.5
31
32     - name: "4 - Final Descent"
33       joints_deg: [283.0, 244.0, 123.0, 348.0, 70.0, 79.0]
34       wait_after: 1.0
35
36     # --- PICK PART ---
37     - name: "5 - Pick (Position)"
38       joints_deg: [287.0, 249.0, 127.0, 347.0, 72.0, 79.0]
39       wait_after: 0.5
40
41     - name: "5.5 - CLOSE Gripper"
42       joints_deg: [287.0, 249.0, 127.0, 347.0, 72.0, 79.0]
43       fingers:    [0.87, 0.87, 0.87] # Fingers almost fully closed (
small part)
44       wait_after: 1.5
45
46     - name: "6 - Small Retraction"
47       joints_deg: [283.0, 237.0, 125.0, 348.0, 69.0, 79.0]
48       wait_after: 0.5
49
50     # INTERMEDIATE: Smooth base transition
51     - name: "6.5 - Base Transition"
52       joints_deg: [305.0, 220.0, 95.0, 348.0, 69.0, 79.0]
53       wait_after: 0.1
54

```

```

55     - name: "7 - High Point"
56       joints_deg: [323.0, 207.0, 71.0, 348.0, 69.0, 79.0]
57       wait_after: 0.5
58
59     - name: "8 - Pre-Destination"
60       joints_deg: [354.0, 197.0, 55.0, 348.0, 69.0, 79.0]
61       wait_after: 1.0
62
63     # J4 goes to 365 (360+5). This prevents the wrist from rotating in
64     reverse. Correct.
65
66     - name: "9 - Orientation Fitting"
67       joints_deg: [354.0, 219.0, 51.0, 365.0, 29.0, 80.0]
68       wait_after: 1.0
69
70     - name: "10 - Final Fitting"
71       joints_deg: [355.0, 222.0, 40.0, 364.0, -1.0, 81.0]
72       wait_after: 1.0
73
74     # --- DROP PART ---
75     - name: "11 - DROP"
76       joints_deg: [355.0, 222.0, 40.0, 364.0, -1.0, 81.0]
77       fingers:    [0.0, 0.0, 0.0]
78       wait_after: 1.0
79
80     # --- RETURN HOME (Smoothed) ---
81
82     - name: "12 - Departure from Target"
83       joints_deg: [339.0, 197.0, 40.0, 364.0, -1.0, 81.0]
84       wait_after: 1.0
85
86     # INTERMEDIATE: Unwind J4 (364->300) and J5 (-1->50)
87     # This prevents the robot from making a jerky movement when
88     returning Home
89
90     - name: "12.5 - Unwind Wrist"
91       joints_deg: [310.0, 190.0, 50.0, 300.0, 50.0, 80.0]
92       wait_after: 0.1
93
94     # INTERMEDIATE: Preparing J1 and J2 for Home positions
95     - name: "13.5 - Pre-Home"
96       joints_deg: [285.0, 180.0, 55.0, 260.0, 70.0, 78.0]
97       wait_after: 0.1
98
99     - name: "14 - Final Home"
100       joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
101       fingers:    [0.0, 0.0, 0.0]
102       wait_after: 1.5

```

```

100
101 # --- Marker with sequence of movements ---
102 "20":
103     type: sequence
104     description: "Optimized Sequence Robot2 - PICK Large round part"
105     sequence:
106         # --- START ---
107         - name: "1 - Home"
108           joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
109           fingers:    [0.0, 0.0, 0.0]
110           wait_after: 1.0
111
112         # INTERMEDIATE: Smoothens J2 (168->215) and J3 descent
113         - name: "1.5 - Approach"
114           joints_deg: [286.0, 190.0, 71.0, 240.0, 81.0, 77.0]
115           fingers:    [0.0, 0.0, 0.0]
116           wait_after: 0.1
117
118         - name: "2 - Pre-position"
119           joints_deg: [298.0, 215.0, 86.0, 240.0, 79.0, 78.0]
120           wait_after: 0.5
121
122         # INTERMEDIATE: J4 was jumping from 240 to 348 (+108 degrees) here
123         .
124         # I added a midpoint (294) to avoid whipping the cable.
125         - name: "2.5 - J4 Rotation"
126           joints_deg: [292.0, 225.0, 103.0, 294.0, 75.0, 78.0]
127           wait_after: 0.1
128
129         - name: "3 - Fine Alignment"
130           joints_deg: [287.0, 234.0, 121.0, 348.0, 70.0, 79.0]
131           wait_after: 0.5
132
133         - name: "4 - Final Descent"
134           joints_deg: [284.0, 244.0, 123.0, 348.0, 70.0, 79.0]
135           wait_after: 1.0
136
137         - name: "5 - Pick"
138           joints_deg: [284.0, 250.0, 125.0, 348.0, 70.0, 79.0]
139           wait_after: 0.5
140
141         # --- CLOSE GRIPPER ---
142         - name: "5 - CLOSE Gripper"
143           joints_deg: [284.0, 250.0, 125.0, 348.0, 69.0, 79.0]
144           fingers:    [0.69, 0.69, 0.69]
145           wait_after: 1.5

```

```
146     - name: "6 - Small Retraction"
147       joints_deg: [284.0, 237.0, 125.0, 348.0, 69.0, 79.0]
148       wait_after: 0.5
149
150     # INTERMEDIATE: Smoothens J1 transition (284->323) and J3
151     retraction (125->71)
152     # Prevents the arm from hitting the table while rotating the base.
153     - name: "6.5 - Base Transition"
154       joints_deg: [305.0, 220.0, 95.0, 348.0, 69.0, 79.0]
155       wait_after: 0.1
156
157     - name: "7 - High Point"
158       joints_deg: [323.0, 207.0, 71.0, 348.0, 69.0, 79.0]
159       wait_after: 0.5
160
161     - name: "8 - Pre-Destination"
162       joints_deg: [354.0, 197.0, 55.0, 348.0, 69.0, 79.0]
163       wait_after: 1.0
164
165     # HERE J4 goes from 348 to 365. This is great, maintains rotation
166     direction.
167     # J5 descends from 69 to 29. Complex movement, but direct.
168     - name: "9 - Orientation Fitting"
169       joints_deg: [354.0, 219.0, 51.0, 365.0, 29.0, 80.0]
170       wait_after: 1.0
171
172     - name: "10 - Final Fitting"
173       joints_deg: [355.0, 225.0, 51.0, 359.0, 20.0, 73.0]
174       wait_after: 1.0
175
176     # --- OPEN GRIPPER ---
177     - name: "10 - Release"
178       joints_deg: [355.0, 225.0, 51.0, 359.0, 20.0, 73.0]
179       fingers:    [0.0, 0.0, 0.0] # Open
180       wait_after: 1.5
181
182     - name: "11"
183       joints_deg: [355.0, 190.0, 51.0, 359.0, 20.0, 73.0]
184       wait_after: 1.0
185
186     # INTERMEDIATE: J1 will return from 355 to 244 (-111 degrees).
187     # It's a long base rotation. Let's split it.
188     - name: "11.5 - Base Return"
189       joints_deg: [300.0, 193.0, 41.0, 359.0, 20.0, 73.0]
190       wait_after: 0.1
191
192     - name: "12 - Departure Point"
```

```

191     joints_deg: [244.0, 195.0, 50.0, 359.0, 20.0, 73.0]
192     wait_after: 1.0
193
194     # INTERMEDIATE: Preparing for Home.
195     # J4 needs to return from 359 to 241 (-118 degrees). Let's break
    this down.
196     # J2 goes from 195 to 168.
197     - name: "13.5 - Return J4/Home"
198       joints_deg: [260.0, 180.0, 57.0, 300.0, 50.0, 74.0]
199       wait_after: 0.1
200
201     - name: "14 - Final Home"
202       joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
203       fingers:    [0.0, 0.0, 0.0]
204       wait_after: 1.5
205
206 "21":
207     type: sequence
208     description: "REVERSE Sequence - PICK at End and PLACE at Start"
209     sequence:
210         # --- DEPARTURE FROM HOME TO END (Where the part is now) ---
211
212         # Starts from Original Home
213         - name: "1 - Home Start"
214           joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
215           fingers:    [0.0, 0.0, 0.0] # Gripper OPEN to fetch
216           wait_after: 1.0
217
218         # Reverse path of departure (formerly 13.5)
219         - name: "2 - Reverse Approach"
220           joints_deg: [260.0, 180.0, 57.0, 300.0, 50.0, 74.0]
221           wait_after: 0.1
222
223         # Formerly 12
224         - name: "3 - Entry Point"
225           joints_deg: [244.0, 195.0, 57.0, 359.0, 20.0, 73.0]
226           wait_after: 1.0
227
228         # Formerly 11.5 (Base rotation)
229         - name: "3.5 - Base Outbound Rotation"
230           joints_deg: [300.0, 193.0, 55.0, 359.0, 20.0, 73.0]
231           wait_after: 0.1
232
233         # Formerly 11
234         - name: "4 - Pre-Pick"
235           joints_deg: [355.0, 190.0, 51.0, 359.0, 20.0, 73.0]
236           wait_after: 1.0

```

```
237
238     # Formerly 10 - Reaches target (with gripper open)
239     - name: "5 - Arrival at Target"
240       joints_deg: [355.0, 226.0, 49.0, 355.0, 12.0, 73.0]
241       fingers:    [0.0, 0.0, 0.0]
242       wait_after: 1.0
243
244     # --- GRAB THE PART (Location B) ---
245     - name: "5 - CLOSE AT TARGET"
246       joints_deg: [355.0, 226.0, 49.0, 355.0, 12.0, 73.0]
247       fingers:    [0.69, 0.69, 0.69] # Close
248       wait_after: 1.5
249
250     # --- TAKE BACK TO START (Location A) ---
251
252     # Formerly 9 (Lifts slightly and orients)
253     - name: "6 - Lift"
254       joints_deg: [354.0, 213.0, 51.0, 365.0, 29.0, 73.0]
255       wait_after: 1.0
256
257     # Formerly 8
258     - name: "7 - Return Path"
259       joints_deg: [354.0, 197.0, 55.0, 348.0, 69.0, 73.0]
260       wait_after: 1.0
261
262     # Formerly 7 (High point)
263     - name: "8 - High Point"
264       joints_deg: [323.0, 207.0, 71.0, 348.0, 69.0, 79.0]
265       wait_after: 1.0
266
267     # Formerly 6.5 (Base rotation returning)
268     - name: "8.5 - Base Return Rotation"
269       joints_deg: [305.0, 220.0, 95.0, 348.0, 69.0, 79.0]
270       wait_after: 0.1
271
272     # Formerly 6
273     - name: "9 - Deposit Approach"
274       joints_deg: [284.0, 237.0, 125.0, 348.0, 69.0, 79.0]
275       wait_after: 1.0
276
277     # Formerly 5 (Original part location)
278     - name: "10 - Deposit Position"
279       joints_deg: [284.0, 250.0, 125.0, 348.0, 69.0, 79.0]
280       wait_after: 1.0
281
282     # --- RELEASE THE PART (Location A) ---
283     - name: "10 - RELEASE AT START"
```

```

284     joints_deg: [284.0, 250.0, 125.0, 348.0, 70.0, 79.0]
285     fingers:    [0.0, 0.0, 0.0] # Open
286     wait_after: 1.5
287
288     # --- RETREAT HOME ---
289
290     # Formerly 4
291     - name: "11 - Retreat"
292       joints_deg: [284.0, 244.0, 123.0, 348.0, 70.0, 79.0]
293       wait_after: 1.0
294
295     # Formerly 3
296     - name: "12 - Clearance"
297       joints_deg: [287.0, 234.0, 121.0, 348.0, 70.0, 79.0]
298       wait_after: 0.5
299
300     # Formerly 2.5
301     - name: "12.5 - J4 Return Rotation"
302       joints_deg: [292.0, 225.0, 103.0, 294.0, 75.0, 78.0]
303       wait_after: 0.1
304
305     # Formerly 2
306     - name: "13 - Pre-Home"
307       joints_deg: [298.0, 215.0, 86.0, 240.0, 79.0, 78.0]
308       wait_after: 0.5
309
310     # Formerly 1.5
311     - name: "13.5 - Arriving Home"
312       joints_deg: [286.0, 190.0, 71.0, 240.0, 81.0, 77.0]
313       wait_after: 0.5
314
315     # Formerly 1 - Final
316     - name: "14 - Final Home"
317       joints_deg: [275.0, 168.0, 57.0, 241.0, 83.0, 76.0]
318       fingers:    [0.0, 0.0, 0.0]
319       wait_after: 1.5

```

Listing B.2 – YAML database for task sequences

B.3 EXECUTION ORCHESTRATION: VISION_JACO_ARUCO_DEMO.LAUNCH

This is the master launch file that initializes the hardware drivers and nodes simultaneously.

```

1 <launch>
2   <arg name="robot_type" default="j2n6s300" />
3   <arg name="markerSize" default="0.06" />
4

```

```

5  <!-- 1) Rob Kinova JACO -->
6  <!-- Ajuste os args conforme voc j usa no seu setup -->
7  <include file="$(find kinova_bringup)/launch/kinova_robot.launch">
8    <arg name="kinova_robotType" value="$(arg robot_type)" />
9  </include>
10
11 <!-- 2) C mera RealSense L515 -->
12 <include file="$(find jaco_arm_robot)/launch/rs_camera_quiet.launch" /
13 >
14 <!-- 3) ArUco multi (publica markers_list e markers) -->
15 <include file="$(find jaco_arm_robot)/launch/aruco_l515_multi.launch">
16   <arg name="markerSize" value="$(arg markerSize)" />
17 </include>
18
19 <!-- 4) Carrega configura o ID -> Pose no namespace do n -->
20 <rosparam file="$(find jaco_arm_robot)/config/marker_poses_sequential.
21   yaml"
22       command="load"
23       ns="marker_to_jaco" />
24
25 <!-- 5) N que converte marker ID em comando pro JACO -->
26 <node pkg="jaco_arm_robot"
27   type="marker_to_jaco.py"
28   name="marker_to_jaco"
29   output="screen">
30   <param name="robot_name" value="$(arg robot_type)" />
31   <param name="stable_count" value="5" />
32   <param name="markers_list_topic" value="/aruco_marker_publisher/
33   markers_list" />
34 </node>
</launch>

```

Listing B.3 – Master launch file

B.4 SUPPORTING CONFIGURATION FILES

For the sake of modularity, the following auxiliary files were created to configure the vision system and optimize the driver performance.

B.4.1 aruco_l515_multi.launch

```

1 <launch>
2   <!-- Tamanho do quadrado em metros (medir o lado preto da plaquinha)
3   -->
4   <arg name="markerSize" default="0.036" />   <!-- 5 cm = 0.05 -->

```

```

4
5 <node pkg="aruco_ros" type="marker_publisher" name="
  aruco_marker_publisher" output="screen">
6   <!-- Conecta na camera -->
7   <remap from="/camera_info" to="/camera/color/camera_info" />
8   <remap from="/image" to="/camera/color/image_raw" />
9
10  <!-- Par metros principais -->
11  <param name="marker_size" value="$(arg markerSize)"/>
12  <param name="image_is_rectified" value="True"/>
13  <param name="reference_frame" value="camera_color_optical_frame"/
14  >
15  <param name="camera_frame" value="camera_color_optical_frame"/
16  >
17 </node>
18 </launch>

```

Listing B.4 – ArUco detection node configuration

B.4.2 rs_camera_quiet.launch

```

1 <launch>
2   <arg name="serial_no" default="" />
3   <arg name="usb_port_id" default="" />
4   <arg name="device_type" default="" />
5   <arg name="json_file_path" default="" />
6   <arg name="camera" default="camera" />
7   <arg name="tf_prefix" default="$(arg camera)" />
8   <arg name="external_manager" default="false" />
9   <arg name="manager" default="realsense2_camera_manager" />
10  <arg name="output" default="log" />
11  <arg name="respawn" default="false" />
12
13  <arg name="fisheye_width" default="-1" />
14  <arg name="fisheye_height" default="-1" />
15  <arg name="enable_fisheye" default="false" />
16
17  <arg name="depth_width" default="-1" />
18  <arg name="depth_height" default="-1" />
19  <arg name="enable_depth" default="true" />
20
21  <arg name="confidence_width" default="-1" />
22  <arg name="confidence_height" default="-1" />
23  <arg name="enable_confidence" default="true" />
24  <arg name="confidence_fps" default="-1" />
25
26  <arg name="infra_width" default="848" />
27  <arg name="infra_height" default="480" />

```

```

28 <arg name="enable_infra"           default="false"/>
29 <arg name="enable_infra1"         default="false"/>
30 <arg name="enable_infra2"         default="false"/>
31 <arg name="infra_rgb"             default="false"/>
32
33 <arg name="color_width"           default="-1"/>
34 <arg name="color_height"          default="-1"/>
35 <arg name="enable_color"          default="true"/>
36
37 <arg name="fisheye_fps"           default="-1"/>
38 <arg name="depth_fps"             default="-1"/>
39 <arg name="infra_fps"             default="30"/>
40 <arg name="color_fps"             default="-1"/>
41 <arg name="gyro_fps"              default="-1"/>
42 <arg name="accel_fps"             default="-1"/>
43 <arg name="enable_gyro"           default="false"/>
44 <arg name="enable_accel"          default="false"/>
45
46 <arg name="enable_pointcloud"     default="false"/>
47 <arg name="pointcloud_texture_stream" default="RS2_STREAM_COLOR"/>
48 <arg name="pointcloud_texture_index" default="0"/>
49 <arg name="allow_no_texture_points" default="false"/>
50 <arg name="ordered_pc"            default="false"/>
51
52 <arg name="enable_sync"           default="false"/>
53 <arg name="align_depth"           default="false"/>
54
55 <arg name="publish_tf"            default="true"/>
56 <arg name="tf_publish_rate"       default="0"/>
57
58 <arg name="filters"               default=""/>
59 <arg name="clip_distance"          default="-2"/>
60 <arg name="linear_accel_cov"       default="0.01"/>
61 <arg name="initial_reset"          default="false"/>
62 <arg name="reconnect_timeout"      default="6.0"/>
63 <arg name="wait_for_device_timeout" default="-1.0"/>
64 <arg name="unite_imu_method"        default=""/>
65 <arg name="topic_odom_in"          default="odom_in"/>
66 <arg name="calib_odom_file"        default=""/>
67 <arg name="publish_odom_tf"        default="true"/>
68
69 <arg name="stereo_module/exposure/1" default="7500"/>
70 <arg name="stereo_module/gain/1"    default="16"/>
71 <arg name="stereo_module/exposure/2" default="1"/>
72 <arg name="stereo_module/gain/2"    default="16"/>
73
74

```

```

75
76 <group ns="$(arg camera)">
77   <include file="$(find realsense2_camera)/launch/includes/nodelet.
    launch.xml">
78     <arg name="tf_prefix"                value="$(arg tf_prefix)"/>
79     <arg name="external_manager"          value="$(arg external_manager)
    )"/>
80     <arg name="manager"                  value="$(arg manager)"/>
81     <arg name="output"                   value="$(arg output)"/>
82     <arg name="respawn"                  value="$(arg respawn)"/>
83     <arg name="serial_no"                 value="$(arg serial_no)"/>
84     <arg name="usb_port_id"              value="$(arg usb_port_id)"/>
85     <arg name="device_type"              value="$(arg device_type)"/>
86     <arg name="json_file_path"           value="$(arg json_file_path)"
    />
87
88     <arg name="enable_pointcloud"         value="$(arg
    enable_pointcloud)"/>
89     <arg name="pointcloud_texture_stream" value="$(arg
    pointcloud_texture_stream)"/>
90     <arg name="pointcloud_texture_index"  value="$(arg
    pointcloud_texture_index)"/>
91     <arg name="enable_sync"               value="$(arg enable_sync)"/>
92     <arg name="align_depth"              value="$(arg align_depth)"/>
93
94     <arg name="fisheye_width"            value="$(arg fisheye_width)"/
    >
95     <arg name="fisheye_height"           value="$(arg fisheye_height)"
    />
96     <arg name="enable_fisheye"           value="$(arg enable_fisheye)"
    />
97
98     <arg name="depth_width"              value="$(arg depth_width)"/>
99     <arg name="depth_height"             value="$(arg depth_height)"/>
100    <arg name="enable_depth"              value="$(arg enable_depth)"/>
101
102    <arg name="confidence_width"           value="$(arg confidence_width)
    )"/>
103    <arg name="confidence_height"          value="$(arg
    confidence_height)"/>
104    <arg name="enable_confidence"          value="$(arg
    enable_confidence)"/>
105    <arg name="confidence_fps"            value="$(arg confidence_fps)"
    />
106
107    <arg name="color_width"               value="$(arg color_width)"/>
108    <arg name="color_height"              value="$(arg color_height)"/>

```

```

109     <arg name="enable_color"                value="$(arg enable_color)"/>
110
111     <arg name="infra_width"                  value="$(arg infra_width)"/>
112     <arg name="infra_height"                 value="$(arg infra_height)"/>
113     <arg name="enable_infra"                 value="$(arg enable_infra)"/>
114     <arg name="enable_infra1"                value="$(arg enable_infra1)"/
115 >
116     <arg name="enable_infra2"                value="$(arg enable_infra2)"/
117 >
118     <arg name="infra_rgb"                    value="$(arg infra_rgb)"/>
119
120     <arg name="fisheye_fps"                  value="$(arg fisheye_fps)"/>
121     <arg name="depth_fps"                    value="$(arg depth_fps)"/>
122     <arg name="infra_fps"                    value="$(arg infra_fps)"/>
123     <arg name="color_fps"                    value="$(arg color_fps)"/>
124     <arg name="gyro_fps"                     value="$(arg gyro_fps)"/>
125     <arg name="accel_fps"                    value="$(arg accel_fps)"/>
126
127     <arg name="enable_gyro"                  value="$(arg enable_gyro)"/>
128     <arg name="enable_accel"                 value="$(arg enable_accel)"/>
129
130     <arg name="publish_tf"                   value="$(arg publish_tf)"/>
131     <arg name="tf_publish_rate"              value="$(arg tf_publish_rate)
132 "/>
133
134     <arg name="filters"                      value="$(arg filters)"/>
135     <arg name="clip_distance"                value="$(arg clip_distance)"/
136 >
137     <arg name="linear_accel_cov"              value="$(arg linear_accel_cov
138 )"/>
139     <arg name="initial_reset"                value="$(arg initial_reset)"/
140 >
141     <arg name="reconnect_timeout"             value="$(arg
142 reconnect_timeout)"/>
143     <arg name="wait_for_device_timeout"       value="$(arg
144 wait_for_device_timeout)"/>
145     <arg name="unite_imu_method"              value="$(arg unite_imu_method
146 )"/>
147     <arg name="topic_odom_in"                value="$(arg topic_odom_in)"/
148 >
149     <arg name="calib_odom_file"              value="$(arg calib_odom_file)
150 "/>
151     <arg name="publish_odom_tf"              value="$(arg publish_odom_tf)
152 "/>
153     <arg name="stereo_module/exposure/1"     value="$(arg stereo_module/
154 exposure/1)"/>
155     <arg name="stereo_module/gain/1"         value="$(arg stereo_module/
156 gain/1)"/>

```

```
142     <arg name="stereo_module/exposure/2" value="$(arg stereo_module/
    exposure/2)"/>
143     <arg name="stereo_module/gain/2" value="$(arg stereo_module/
    gain/2)"/>
144
145     <arg name="allow_no_texture_points" value="$(arg
    allow_no_texture_points)"/>
146     <arg name="ordered_pc" value="$(arg ordered_pc)"/>
147
148     </include>
149 </group>
150 </launch>
```

Listing B.5 – Optimized RealSense initialization

APPENDIX C – THIRD-PARTY DEPENDENCIES

Note: The source code presented in this appendix represents original work developed specifically for this research project, including the state machine logic and the YAML-based trajectory sequencer. This work utilizes standard industrial drivers provided by the respective hardware and middleware maintainers. The original repositories are referenced below, and no modifications were made to the core logic of these drivers; only the launch configurations were adapted for integration into the robotic workcell.

The implementation relies on open-source packages maintained by the robotics community. These drivers form the middleware layer, responsible for hardware abstraction and vision processing.

C.1 SUMMARY OF DEPENDENCIES

The following table details the primary third-party repositories integrated into this project.

Repository	Function	Source/Reference
aruco_ros	Fiducial marker detection	(Robotics, P., 2026)
kinova-ros	Jaco2 hardware drivers	(Robotics, K., 2026)
realsense-ros	L515 sensor interface	(RealSense, 2026)

Table 8 – Third-party dependencies utilized in this research.

C.2 INSTALLATION AND CONFIGURATION

All dependencies were installed via standard ROS package management tools. Adaptation for this project was limited to the creation of custom launch files and configuration parameters (specifically in the `.yaml` and `.launch` files described in Appendix B), ensuring the system remained modular and compatible with the original upstream drivers.