POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering



Master's Degree Thesis

Development of a Vision-Guided Robotic Positioning for Aerospace Component Vibration Inspection Using Concentric Circle Markers Detection

Supervisors

- Prof. Christian Maria FIRRONE
- Prof. Giuseppe BATTIATO
- Eng. Umidjon USMANOV

Candidate

- Abduazim ABDULAZIZOV

July 2026

Abstract

This thesis presents the development of a vision-guided robotic positioning system designed for the autonomous inspection of aerospace components using vibration-based non-destructive testing. The system addresses a fundamental challenge in robotic inspection automation: enabling a robot arm to autonomously locate and approach specific test points on a component surface without manual probe positioning or pre-programmed target coordinates. The proposed solution uses coded concentric circular markers — similar to photogrammetric reference targets — placed on the component surface to define the vibration test points. A robot-mounted camera detects these markers autonomously, estimates their six-degree-of-freedom (6D) pose, and positions a solenoid actuator perpendicularly at each marked location for consistent mechanical excitation.

The work follows a progressive three-stage development methodology. In the first stage, a five-degree-of-freedom (5-DOF) serial robot arm was designed, constructed, and programmed using Denavit-Hartenberg kinematic modelling, numerical inverse kinematics based on the Levenberg-Marquardt algorithm, and an embedded vision pipeline running on a Raspberry Pi 5 with a calibrated camera. ArUco marker detection was used in this stage to validate the full coordinate transformation chain from camera frame to robot base frame.

In the second stage, an industrial Universal Robots UR3e collaborative robot was simulated in the RoboDK environment. A concentric circle (annulus) detection pipeline was developed, capable of estimating the 6D pose of circular marker targets from a single RGB camera using ellipse fitting and geometric tilt analysis. A two-stage motion strategy was designed and validated: an initial coarse joint-space approach followed by a close-range visual re-detection and precision linear refinement approach. In the third stage, the ROS 2 software architecture and Intel RealSense D405 depth camera integration are presented as the planned pathway for full hardware deployment.

Real-world vision tests on a physical aerospace component confirm that the detection pipeline operates correctly under laboratory lighting conditions. These results validate the core design objectives and provide a credible foundation for future deployment on the real UR3e hardware platform.

Acknowledgements

I am grateful to Prof. Christian Maria Firrone and Prof. Giuseppe Battiato for supervising this thesis. Their comments during progress meetings — often pointing to something I had overlooked — changed how I approached several key problems in the kinematic model and the detection pipeline.

I also want to thank Umidjon Usmanov and the DIMEAS laboratory staff for their practical help during the experimental phase. Getting access to the component and the equipment when I needed it made a real difference.

The open-source robotics community deserves a mention. The Robotics Toolbox for Python, OpenCV, RoboDK, and ROS 2 are not just free software — they are actively documented and maintained, which saved a considerable amount of implementation time.

Finally, I want to thank my family. They have been patient and supportive through all of this, and I could not have finished without them.

Table of Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Research Objectives and Contributions 3

1.2.1 Thesis Organization 4

1.3 Related Work 4

1.3.1 Robotic Inspection in Aerospace Manufacturing 5

1.3.2 Fiducial Marker Systems for Robot Vision 5

1.3.3 Square Coded Markers 6

1.3.4 Concentric Circle Targets 6

1.4 Camera Calibration and Hand-Eye Registration 7

1.5 Pose Estimation from a Single Camera 8

1.6 Robot Kinematics and Numerical Inverse Kinematics 8

1.7 Vision-Guided Robot Motion 9

1.8 Software Frameworks: ROS 2, MoveIt2, and RoboDK 9

1.9 Summary and Research Gap 10

2 System Architecture and Hardware **12**

2.1 System overview 12

2.1.1 Stage 1 - Five-DOF Prototype Arm with Raspberry Pi 5 . . 12

2.1.2 Stage 2 - UR3e Simulation in RoboDK 14

2.1.3 Stage 3 - ROS 2 Implementation with Gazebo and MoveIt2 . 15

2.2 Test Component and Marker Placement 17

3 Vision System and Pose Estimation **19**

3.1 Camera Model and Projection Geometry 19

3.1.1 The Perspective-n-Point Problem 20

3.2 Camera Calibration 21

3.2.1 Calibration Procedure 21

3.2.2 Calibration Results 22

3.2.3 Distance Validation Against Ruler Ground Truth 22

3.3 ArUco Marker Detection (Stage 1) 24

3.3.1 Detection Pipeline 24

3.4 Concentric Circle Marker Detection (Stages 2 and 3) 24

3.4.1 Motivation for Concentric Circle Markers 24

3.4.2 Marker Geometry 25
3.4.3 Detection Algorithm 25
3.4.4 Orientation Estimation from Ellipse Geometry 26
3.5 Vision System Characterisation 27
3.5.1 Detection Accuracy versus Distance (E2) 27
3.5.2 Orientation Accuracy versus Tilt (E3) 28
3.6 Coordinate Frame Definitions 29

4 Kinematic Modelling and Coordinate Transformations 31
4.1 Denavit-Hartenberg Parameterization 31
4.1.1 DH Convention 31
4.1.2 DH Parameters for the 5-DOF Arm 31
4.1.3 Joint Mechanical Convention 33
4.2 Forward Kinematics 34
4.3 Inverse Kinematics 35
4.3.1 Problem Formulation and the Five-DOF Orientation Deficit . 35
4.3.2 Levenberg-Marquardt Numerical Solution 35
4.4 Coordinate Transformation Chain 36
4.4.1 From Camera Frame to Robot Base Frame 36
4.4.2 Perpendicular Approach Target Computation 37

5 Simulation Implementation and Experimental Results 38
5.1 RoboDK Simulation Environment 38
5.2 Marker Detection in Simulation 39
5.3 Multi-Marker Target Identification 39
5.4 Two-Stage Motion Strategy 40
5.5 RoboDK Experiment Results 42
5.5.1 Experimental Setup 42
5.5.2 Positioning Accuracy and Strategy Comparison 42
5.6 ROS 2 Application: Vision-Guided Striking in Gazebo 43
5.6.1 Workspace and Launch Architecture 43
5.6.2 Simulation Environment and Perception 44
5.6.3 Striking Sequence and Motion Control 47
5.6.4 Operator Interface and Safety Behavior 49
5.6.5 Qualitative Validation 49

6 Real-World Vision Tests on an Aerospace Component 51
6.1 Test Setup 51
6.2 Detection Results 52
6.3 Comparison with Simulation Conditions 52

7 Conclusions and Future Work 55
7.1 Interpretation of Results 55
7.2 System Limitations 56

7.2.1 Single-Camera Orientation Ambiguity 56
7.2.2 Hand-Eye Calibration Approximation 56
7.2.3 Simulation-to-Real Transfer Gap 56
7.2.4 Excitation Not Instrumented 56
7.3 Summary of Achieved Objectives 57
7.4 Conclusions 58

A Software Module Overview **60**

B Key Algorithm Listings **61**
B.1 Concentric Circle Detection Core 61
B.2 Orientation from Ellipse Geometry 62
B.3 Masked Inverse Kinematics with Servo-Space Validation 62
B.4 Multi-Seed Inverse Kinematics in the ROS 2 Controller 63
B.5 Striking Sequence with Stop Supervision 64

Bibliography **65**

List of Figures

1.1 Overall system pipeline from image acquisition to perpendicular approach at the detected marker. Coordinate frames and the corresponding transformations are indicated at each stage. 2

2.1 System overview: RoboDK simulation environment with the UR3e robot, camera tool, test plate, and five concentric circle marker targets. 13
2.2 The five-DOF prototype arm used in Stage 1, with the five servo-driven joints, the end-effector camera, the Raspberry Pi 5, and the PCA9685 servo controller. 13
2.3 RoboDK simulation scene: UR3e robot, camera tool, test plate with five annulus marker targets, and simulated camera view inset. 15
2.4 ROS 2 architecture of the Stage 3 application: the detection node processes the simulated D405 streams and publishes marker poses, which the motion controller converts into MoveIt2 planning goals executed by the joint trajectory controller in Gazebo. 16
2.5 The aerospace test component on the laboratory bench. Concentric circle marker targets are placed at the inspection points. 18

3.1 First processing stage on recorded E3 debug frames: blurred greyscale input and binarised output for the marker at 0° and 30° tilt. Binarisation preserves the nested ring structure; the tilt appears as ellipse eccentricity in the contours. 21
3.2 Calibration quality assessment: (a) per-image RMS reprojection error across the 22 calibration views; (b) comparison of a raw and undistorted image showing the effect of the estimated distortion model. 23
3.3 Annulus marker detection result in RoboDK simulation: three markers simultaneously detected with ID labels and 6D coordinate axis overlays. 26
3.4 E2 results: (a) position error versus working distance; (b) estimated depth versus true distance, showing the linear relationship ($R^2 > 0.999$) with the systematic scale factor discussed in the text. 28
3.5 E3 results: measured tilt versus true tilt. The estimator tracks the ground truth closely above 5°; the deviation at 0° reflects the axis-ratio method's reduced sensitivity near head-on viewing. 29

3.6 Coordinate frames used in the system and the homogeneous transformations connecting them: $\boldsymbol{T}_{EE}^{B}$ from forward kinematics, $\boldsymbol{T}_{C}^{EE}$ from the camera mounting, and $\boldsymbol{T}_{M}^{C}$ from pose estimation. 30

4.1 Denavit-Hartenberg frame assignment for the five-DOF prototype arm, showing the joint axes, the assigned frames {0} to {5}, and the principal link dimensions. 32
4.2 Prototype arm at the home configuration 34
4.3 Perpendicular approach geometry: the tool z-axis is aligned antiparallel to the marker surface normal, and the far and near standoff targets are offset along the normal direction. 37

5.1 RoboDK simulation environment: UR3e robot, camera tool, test plate with five annulus marker targets. 39
5.2 Three concentric circle markers simultaneously detected in the RoboDK camera view, with ID labels and 6D coordinate axis overlays. 40
5.3 Two-stage motion strategy: far detection and joint-space approach, close-range re-detection, and linear perpendicular refinement, repeated for each marker in the identified sequence. 41
5.4 Simulated camera view of the same marker from the far observation pose (left) and from the 150 mm standoff after the first approach motion (right). The close view provides substantially more contour pixels for the ellipse fit. 41
5.5 UR3e end-effector at the final perpendicular approach position for one marker target. 41
5.6 E5 results: final positioning error for the single-stage and two-stage motion strategies across all completed trials. 42
5.7 TF tree of the running application. The robot link chain is maintained from joint states; the camera and hammer frames are static; the annulus frames are broadcast by the detection node and connect the perception output to the planning frame through a single consistent tree. 45
5.8 Binarized detector input at the two working distances of the motion sequence (E5 debug frames). The nested ring structure survives thresholding in both regimes; the contour pixel count available to the ellipse fit differs by roughly an order of magnitude. 46
5.9 One complete striking cycle in the Gazebo simulation: (a) home configuration with all five markers detected, (b) joint-space approach and perpendicular alignment at the 15 cm standoff, (c) strike contact, (d) linear retract, (e) return to home before the next target in other view. 46
5.10 Stage 3 perception: the Gazebo simulation (left) and the corresponding RViz view (right) showing the five detected markers as TF frames with axis overlays, together with the debug image of the detector. . . 47

5.11 The four-phase striking sequence executed for each target, with the stop condition evaluated between all phases. 48
5.12 The hammer tool at the perpendicular standoff pose (left) and at strike contact with the marker surface (right) in the Gazebo simulation. 49
5.13 Operator panel of the striking application, showing the sequence controls and the live status display. 50

6.1 Experimental setup: the aerospace component on the laboratory bench, with the camera on an articulated mount and the detection software running on the laptop. 52
6.2 Detection running live on the real component: the software reports 54 tracked circular features at 9.1 frames per second, each with an identifier and an estimated 6D pose axes. 53
6.3 Close view of the detection output: coloured identifier labels and coordinate axis overlays on the component's hole pattern, following the local surface orientation of the curved vane. 53

List of Tables

1.1 Comparison of fiducial marker families for the perpendicular-approach positioning task. 7

2.1 Stage 1 hardware specifications. 14
2.2 ROS 2 node architecture of the Stage 3 application. 17

3.1 Calibrated camera intrinsic parameters. 22
3.2 End-to-end distance validation of the calibrated detection model. . . 23
3.3 E2 — detection rate and depth estimate statistics versus working distance (30 frames per distance). 28
3.4 E3 — orientation estimation accuracy versus surface tilt (20 frames per angle, 100% detection). 29

4.1 DH parameters for the 5-DOF prototype arm. 33
4.2 Forward kinematics validation against physical measurement. 35

5.1 E5 — positioning error by motion strategy and per marker (5 trials per marker per condition). 42

7.1 Remaining steps toward real UR3e deployment. 57

A.1 Software modules developed in this thesis, grouped by development stage. 60

Acronyms

DH	Denavit-Hartenberg kinematic parameterisation.
DOF	Degrees of freedom.
FK	Forward kinematics.
IK	Inverse kinematics.
NDT	Non-destructive testing.
TCP	Tool Centre Point.
EE	End-effector.
CCM	Coded Concentric circular Marker.
PnP	Perspective-n-Point.
RPY	Roll-Pitch-Yaw.
RMS	Root Mean Square.
LM	Levenberg-Marquardt.
FOV	Field of view.
EMA	Exponential moving average.
ROS	Robot Operating System.
API	Application Programming Interface.

Chapter 1

Introduction

This chapter introduces the industrial motivation for autonomous robotic inspection of aerospace components, states the research objectives, identifies the scientific contributions of this work, and reviews the relevant literature. The chapter concludes with an overview of the thesis structure.

1.1 Motivation

The inspection of safety-critical aerospace components represents one of the most demanding challenges in modern manufacturing and maintenance operations. Components such as turbine blades, compressor shrouds, and structural frame sections are subjected to repeated thermal and mechanical stress during service, making systematic surface inspection a regulatory requirement in both civil aviation and defence applications. Among the available non-destructive testing methods, vibration-based testing — in which a mechanical excitation is applied to a specific surface location and the structural response is measured — provides valuable information about material integrity, crack initiation, and fatigue state without altering the component [1].

Traditional vibration testing relies on manual positioning. In this approach, a trained technician identifies each inspection point on the component surface, manually places the excitation probe or solenoid actuator, and visually verifies contact alignment. This method is constrained by human fatigue, operator variability, and the challenge of consistently accessing all surface locations on complex three-dimensional components. Repeatability, defined as the ability to return to the exact same surface point across multiple test sessions, is especially important for fatigue monitoring applications, where consistent excitation location is necessary for valid trend comparison [1].

Robotic systems offer a compelling alternative, providing sub-millimetre positioning repeatability, programmable coverage sequences, and the ability to operate without continuous operator intervention. However, the adoption of robotic systems in

aerospace inspection has been constrained by a fundamental challenge: the robot must be informed of the spatial location of every inspection point before motion planning can begin. Conventional approaches address this through offline programming, in which inspection trajectories are pre-computed from a CAD model of the part. This strategy introduces practical limitations: the programme must be adjusted whenever the component design changes, cannot accommodate part-to-part geometric variation arising from manufacturing tolerances, and requires significant engineering effort for each new component type [2].

This thesis investigates an autonomous alternative in which coded concentric circular markers — standard reference targets used in metrology and three-dimensional scanning workflows — are placed on the component surface to define the inspection points. The robot autonomously detects these markers using an on-board camera, estimates the six-degree-of-freedom pose of each marker via ellipse fitting and the Perspective-n-Point method, and positions the excitation tool perpendicular to each target to ensure consistent mechanical excitation. This method eliminates the need for a CAD model, offline trajectory programming, or manual probe positioning. The entire pipeline is validated in two complementary environments: an industrial offline-programming simulation in RoboDK and a full ROS 2 implementation. In the latter, a UR3e manipulator, guided by MoveIt2 motion planning and a simulated depth camera, autonomously approaches each detected marker and executes a perpendicular striking motion under operator supervision. The complete processing pipeline, from image acquisition to robot motion, is illustrated in Figure 1.1.

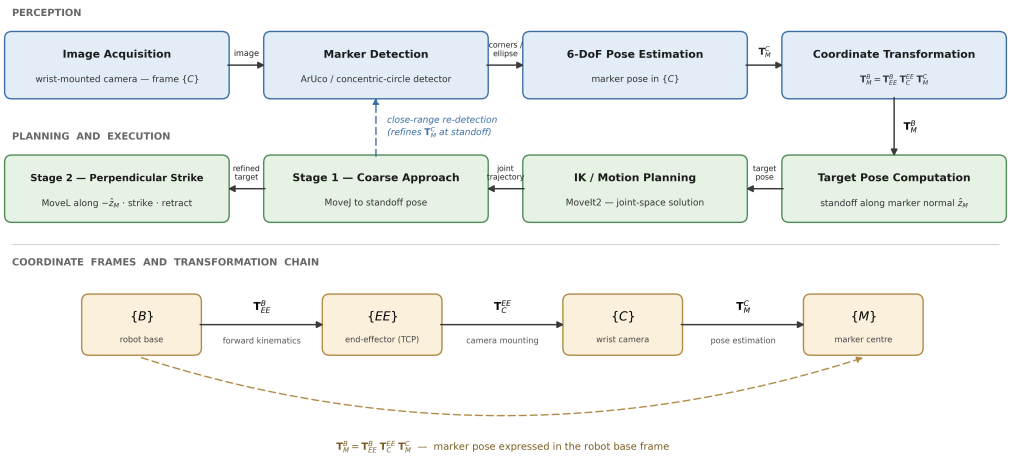


Figure 1.1: Overall system pipeline from image acquisition to perpendicular approach at the detected marker. Coordinate frames and the corresponding transformations are indicated at each stage.

1.2 Research Objectives and Contributions

The main goal of this work is to build and test a robotic positioning system that finds coded concentric circle markers on an aerospace component surface and approaches each one perpendicularly — without pre-programmed coordinates or manual adjustment. The research was structured around six specific objectives:

1. To design and implement a forward and inverse kinematic model for a five-degree-of-freedom (DOF) serial robot arm using the Denavit-Hartenberg convention, and to validate the model against physical measurements.

2. To develop a comprehensive coordinate transformation pipeline that accurately maps detected marker poses from the camera frame to the robot base frame.

3. To design and implement a concentric circle marker detection pipeline capable of identifying coded circular targets and estimating their six-degree-of-freedom (6D) pose from a single RGB camera, utilizing ellipse fitting and geometric tilt analysis.

4. To design and validate a two-stage motion strategy that employs coarse joint-space motion followed by a close-range visual refinement step to achieve high-precision perpendicular approaches to each marker target.

5. To implement and validate the complete system in a high-fidelity simulation environment using RoboDK and the Universal Robots UR3e platform, demonstrating sequential multi-target inspection.

6. To port the complete perception, planning, and execution pipeline to the ROS 2 framework and validate it in Gazebo simulation. This includes reliable detection of five concentric circle markers from a simulated depth camera, MoveIt2-based motion planning for the UR3e, a perpendicular approach, strike, and retract sequence at each target, and an operator interface providing start, stop, and homing controls with immediate motion cancellation for safety.

Research Contributions

The principal contributions of this thesis to the field of vision-guided robotic inspection are summarized below.

Three primary contributions are highlighted. The first is a detection pipeline for coded concentric circle markers that recovers the full six-degree-of-freedom (6D) pose from a single RGB image using contour hierarchy analysis, ellipse fitting, and Perspective-n-Point (PnP) geometry, without requiring any depth sensor.

The second contribution is a two-stage motion strategy in which the robot first approaches a standoff distance, then runs the detector again at close range to refine the initial estimate, and finally executes the final approach. This approach addresses

the degradation of single-camera depth estimates with increasing distance; the second detection step compensates for this limitation before the tool makes contact.

The third contribution is a comprehensive Robot Operating System 2 (ROS 2) application that integrates perception and motion components into a unified supervised autonomous system. The detection node processes the simulated depth-camera stream, publishes the pose of each visible marker, and broadcasts the corresponding transform frames for visualization in RViz. The motion controller node plans and executes the approach, strike, and retract sequence for each target using MoveIt2, ensuring the tool remains perpendicular to the marker surface and retracts to a safe standoff distance of approximately 15 cm before and after each strike.

In addition, a validated five-degree-of-freedom kinematic model is derived from Denavit-Hartenberg parameters, including identification and correction of a joint mechanical inversion specific to the prototype servo arm. A complete software implementation is presented spanning embedded real-time detection on Raspberry Pi 5, kinematic computation using the Robotics Toolbox for Python [3], industrial simulation in RoboDK, and the ROS 2 application described above.

1.2.1 Thesis Organization

The remainder of this thesis is organized as follows. Section 1.3 places the work in context by reviewing aerospace inspection robotics, fiducial marker systems, camera calibration, and robot kinematics. Chapter 2 introduces the three-stage development structure and the hardware and software used in each stage. Chapter 3 covers the calibration procedure and both detection pipelines — ArUco for Stage 1 and the concentric circle algorithm for Stages 2 and 3. Chapter 4 works through the kinematic model: DH parameters, forward kinematics, the numerical IK solver, and the coordinate transformation chain. Chapter 4 covers the kinematic model, including Denavit-Hartenberg (DH) parameters, forward kinematics, the numerical inverse kinematics (IK) solver, and the coordinate transformation chain. Chapter 5 explains the two simulation setups: the RoboDK environment with its two-stage motion strategy, and the ROS 2 application in Gazebo using MoveIt2 planning and the supervised striking sequence, as well as the quantitative experimental results. Chapter 6 describes vision tests on the actual aerospace component. Chapter 7 reviews the results, points out remaining gaps, and suggests next steps for deploying on real UR3e hardware.

1.3 Related Work

This chapter reviews the state of the art in each technical area that underpins the system developed in this thesis: robotic inspection of aerospace components, fiducial marker systems, camera calibration and hand-eye registration, pose estimation from a single camera, robot kinematics and numerical inverse kinematics, vision-guided

motion strategies, and the software frameworks used for implementation. For each area, existing methods are described, their limitations in the context of the present application are identified, and the approach selected for this work is justified. The chapter closes with the research gap that defines the scope of the thesis.

1.3.1 Robotic Inspection in Aerospace Manufacturing

The application of robotic systems to aerospace component inspection has been an active research area for over two decades, driven by the combined need for higher throughput, greater repeatability, and improved traceability in quality control processes [2]. Early robotic inspection systems were predominantly employed for ultrasonic testing of large structures such as fuselage panels and wing sections, where the robot's role was primarily to maintain consistent probe contact force and scan velocity across flat or mildly curved surfaces [4].

Vibration-based non-destructive testing represents a complementary inspection modality that is particularly sensitive to subsurface defects, delaminations, and fatigue cracks in metallic and composite aerospace structures [1]. In vibration-based testing, a mechanical excitation is applied at a specific surface point and the structural response — typically the frequency spectrum or damping characteristics — is compared against a baseline. The comparison is only meaningful if the excitation lands in the same place each time: a lateral shift of a few millimetres changes the local stiffness sensed by the actuator, which can move resonance peaks by amounts comparable to what a genuine structural defect would produce. This sensitivity to excitation location makes robotic positioning particularly valuable for vibration inspection.

More recent research has addressed the inspection of geometrically complex components — bladed disc assemblies, compressor stages, and turbine vane clusters — where the spatial variation in surface orientation demands adaptive positioning strategies [5]. Several authors have proposed model-based systems in which three-dimensional CAD representations of the component are used to pre-compute inspection paths, and the robot follows these paths using high-accuracy positional control [2]. While effective, these systems require precise part registration to align the physical component with the CAD model, a non-trivial task for freeform surfaces. The marker-based approach adopted in the present work avoids this requirement entirely: the operator defines the inspection points by placing markers, and the robot localizes them autonomously.

1.3.2 Fiducial Marker Systems for Robot Vision

Fiducial markers provide a controlled and computationally efficient reference for visual pose estimation in robotics applications. The idea predates modern robotics libraries: ARToolKit introduced square black-and-white markers for augmented reality tracking in the late 1990s [**Kato1999**], and ARTag later improved their robustness

with digital coding techniques and error correction [**Fiala2005**]. The families most widely adopted in robotics today are ArUco [6], AprilTag [**Wang2016**, 7], and photogrammetric concentric circle targets [8]. A recent experimental comparison across these systems is given by Kalaitzakis et al. [**Kalaitzakis2021**].

1.3.3 Square Coded Markers

ArUco markers have found widespread application in robotic grasping, augmented reality, and camera-to-robot calibration due to their computational efficiency and robustness to partial occlusion and lighting variation [6]; later work reduced their detection cost further for real-time applications [**RomeroRamirez2018**]. In the OpenCV `aruco` module, pose estimation is handled by `estimatePoseSingleMarkers`, which runs a Perspective-n-Point solve on the four detected corner points to recover the camera-to-marker transform. AprilTag follows the same square-marker principle with a different coding scheme and decoder, and is often preferred where identification robustness matters more than detection speed [**Wang2016**].

The square-marker approach is fast and reliable, but two properties limit it for the present application. First, its pose accuracy is bounded by how precisely the four corners can be located in the image — the pose rests on only four correspondences, so corner-level noise propagates directly into the estimate. Second, the printed pattern must remain flat and undamaged on the workpiece, which is a real constraint on curved, handled, or thermally cycled aerospace surfaces.

1.3.4 Concentric Circle Targets

Concentric circle markers — known in photogrammetry as ring targets or coded circular targets [8], and in the robotics literature as concentric contrasting circles [**Gatrell1991**] — have a geometric property that is directly useful here: a circle viewed at an angle projects to an ellipse, and the ratio of the ellipse axes encodes the out-of-plane tilt [9]. A single image is therefore enough to recover the full 6D pose without a depth sensor or stereo rig. In addition, the pose rests on fitted ellipses rather than on isolated corner points: the ellipse fit aggregates hundreds of contour pixels, which averages out pixel-level noise in a way a four-corner estimate cannot. These markers are already in routine use as reference stickers in 3D scanning workflows — GOM, Creaform, and similar systems rely on them to register measurement data to a workpiece [8].

Table 1.1 summarises the trade-offs between the marker families in the terms that matter for this application. The decisive rows are orientation encoding and noise behaviour: the concentric circle target is the only option whose geometry directly encodes the surface tilt that the perpendicular approach requires. Their use as robotic positioning targets for vibration NDT, where the robot must approach each

point perpendicularly for consistent probe contact, does not appear to have been explored in the literature before this work.

Table 1.1: Comparison of fiducial marker families for the perpendicular-approach positioning task.

Property	ArUco / AprilTag	Chessboard target	Concentric circles
Pose from single image	Yes (4 corners)	Yes (many corners)	Yes (fitted ellipses)
Points per estimate	4	20–80	Hundreds (contours)
Tilt encoding	Via PnP only	Via PnP only	Direct (axis ratio)
Identity coding	Binary payload	None	Code ring / sequence
Detection cost	Very low	Moderate	Low
Industrial availability	Printed	Printed	Standard metrology stickers
Analogue on real parts	None	None	Circular holes, eyelets

The last row of the table deserves emphasis. Square coded patterns have no natural counterpart on aerospace hardware, but circular features do: drilled holes, rivet lines, and rimmed eyelets present exactly the nested-contour geometry that a concentric circle detector looks for. The detector built for ring targets therefore generalizes, without algorithmic modification, to marker-free detection of existing structural features — a property demonstrated experimentally in Chapter 6.

1.4 Camera Calibration and Hand-Eye Registration

Camera calibration is the process of determining the intrinsic parameters of a camera model — focal lengths f_x and f_y, principal point (c_x, c_y), and lens distortion coefficients — from images of a known calibration target [10]. The Zhang flexible calibration method, implemented as `calibrateCamera` in OpenCV, uses a planar chessboard pattern photographed from multiple viewpoints. The quality of calibration is quantified by the root mean square reprojection error, which measures the average pixel-space distance between detected calibration corner positions and the positions predicted by the optimised camera model. A root mean square error below 0.5 pixels is considered excellent for vision-guided robotic applications [10].

Calibration of the camera alone is not sufficient for robot guidance: the rigid transform between the camera and the robot must also be known. When the camera is mounted on the arm, this is the hand-eye calibration problem, classically formulated as the matrix equation $\boldsymbol{AX} = \boldsymbol{XB}$, where $\boldsymbol{X}$ is the unknown camera-to-flange transform and $\boldsymbol{A}$, $\boldsymbol{B}$ are relative motions of the robot and of the observed target [11]. Closed-form solutions on the Euclidean group were later given by Park and Martin [**Park1994**]. Formal hand-eye calibration requires moving the robot through several configurations while observing a fixed target; where this is impractical, the transform can be measured physically, at the cost of an uncorrected systematic

error. The present work takes the physical-measurement route in Stage 1 and quantifies its consequences (Chapter 4), with formal Tsai-Lenz calibration identified as a prerequisite for hardware deployment.

1.5 Pose Estimation from a Single Camera

Six-degree-of-freedom pose estimation from a single calibrated RGB camera is a fundamental problem in computer vision [**Hartley2004**]. The Perspective-n-Point problem seeks the rigid-body transformation between a known three-dimensional object model and the camera reference frame, given N two-dimensional image projections of N known object points [12]. The `solvePnP` function in OpenCV implements several solution algorithms, including the Efficient PnP method [12] and the iterative Gauss-Newton refinement; robust variants reject outlier correspondences using the RANSAC hypothesis-and-verify scheme [**Fischler1981**]. A broad survey of pose estimation methods in the closely related augmented-reality setting is given by Marchand et al. [**Marchand2016**].

For planar markers — including both ArUco squares and concentric circle targets — the object model points lie in a single plane, which reduces the Perspective-n-Point problem to a homography estimation. The primary limitation of single-camera pose estimation is the depth uncertainty: the distance along the optical axis is estimated less accurately than lateral position because small angular perturbations at distance produce geometrically small image displacements [12]. This characteristic directly motivates the two-stage motion strategy developed in this thesis, in which a second detection cycle at close range is used to refine the initial long-range pose estimate.

1.6 Robot Kinematics and Numerical Inverse Kinematics

The Denavit-Hartenberg parameterisation provides a standardised and compact representation of serial robot kinematics [13]; its modern treatment, including frame assignment procedures and the properties of the resulting transformation chain, is given in the standard robotics literature [**Siciliano2009**]. In the DH convention, each link is characterised by four parameters: the link length a_i, the link twist α_i, the link offset d_i, and the joint angle θ_i. Forward kinematics is computed as the sequential product of homogeneous transformation matrices, one per link, yielding the end-effector pose as a function of the joint configuration vector.

Inverse kinematics admits closed-form solutions only for particular kinematic structures, such as arms with a spherical wrist; general or non-standard architectures are solved numerically. Damped least-squares formulations, introduced to robotics independently by Wampler [**Wampler1986**] and by Nakamura and Hana-

fusa [**Nakamura1986**], address the instability of the plain Jacobian pseudo-inverse near singularities by trading a small solution bias for bounded joint velocities. The Levenberg-Marquardt algorithm [14, 15] applies the same damping idea to the underlying non-linear least-squares problem, transitioning smoothly between Gauss-Newton and gradient-descent behaviour; it is the method implemented by `ikine_LM` in the Robotics Toolbox for Python [3], used throughout this work.

For manipulators with fewer than six joints, an additional structural issue arises: the arm cannot independently satisfy all six pose constraints, so the task specification itself must release the degrees of freedom the mechanism does not have. In the Robotics Toolbox this is expressed through a per-component task mask. The consequences of getting this specification wrong — and the reasoning behind the mask adopted here — are examined in detail in Chapter 4, since they proved to be one of the decisive implementation issues of the project.

1.7 Vision-Guided Robot Motion

Strategies for coupling vision to robot motion are classically divided into visual servoing, in which image measurements are fed back continuously into the motion controller [**Hutchinson1996**, **Chaumette2006**], and look-then-move schemes, in which the vision system estimates a target pose once and the robot then executes a conventional planned motion toward it. Continuous servoing offers the highest ultimate accuracy, since errors are corrected during motion, but it demands high-rate image processing, tight latency bounds, and careful stability design. Look-then-move schemes are simpler and integrate naturally with industrial motion planners, at the cost of full exposure to the initial pose estimation error.

The two-stage strategy developed in this thesis sits deliberately between these extremes: it is a look-then-move scheme executed twice, with the second look taken close to the target where single-camera estimation is most favourable. This retains the simplicity and planner compatibility of look-then-move — each motion is an ordinary planned trajectory — while recovering part of the error-correction benefit of feedback, specifically the part that addresses systematic long-range depth bias. The eye-in-hand camera configuration (camera mounted on the moving arm) is what makes the close-range second look possible without additional hardware.

1.8 Software Frameworks: ROS 2, MoveIt2, and RoboDK

The Robot Operating System, introduced as a research middleware in 2009 [**Quigley2009**], established the node-topic architecture that now dominates robot software integration. Its successor ROS 2 re-implements this architecture on the Data Distribution Service publish-subscribe standard, addressing the reliability and real-time limitations of

the original design [16]. ROS 2 Humble Hawksbill provides hardware driver support for the Universal Robots series through the UR ROS2 Driver package, and motion planning capabilities through MoveIt2 [17].

MoveIt2 provides a collision-aware motion planning framework integrating path planners with kinematic solvers and execution monitoring. For vision-guided manipulation, MoveIt2 supports constraint-based planning including orientation constraints that enforce approach directionality — the capability required for perpendicular surface approach in inspection tasks. In the present work, MoveIt2 is the planning and execution backend of the final ROS 2 application described in Chapter 5.

RoboDK occupies a different niche: it is a commercial robot simulation and offline programming platform supporting over 500 robot models with a Python scripting interface through the Robolink application programming interface [18]. For a UR3e collaborative robot, RoboDK accurately models the tool frame, joint limits, and singularity avoidance behaviour of the Universal Robots controller stack, providing a high-fidelity simulation environment for offline validation before any middleware integration. Using RoboDK for the quantitative experiments and ROS 2 for the integrated application — rather than choosing one platform — allowed the detection and motion logic to be validated in a controlled environment before being exposed to the full complexity of the middleware stack.

Simulation-to-real transfer in manipulation tasks requires systematic quantification of discrepancies before hardware deployment. The principal sources of discrepancy in the present context are camera parameter mismatch between the simulated and real camera models, TCP offset errors arising from imprecise camera mounting position, and lighting condition differences that affect detection performance. These are examined where they arise in the experimental chapters.

1.9 Summary and Research Gap

The reviewed literature shows that the individual building blocks of the proposed system — fiducial marker detection, single-camera pose estimation, numerical inverse kinematics, and constraint-based motion planning — are individually well established. What appears to be missing is their integration into a positioning system for vibration inspection in which coded concentric circle targets, already standard consumables in metrology workflows, serve as autonomous navigation references for a robot that must approach each point perpendicularly. Marker surveys [**Kalaitzakis2021**] evaluate detection accuracy in isolation rather than as part of a closed positioning task; robotic NDT systems [2] rely on CAD-based path programming rather than in-situ visual targeting; and the photogrammetric literature on circular targets [8, 9] treats them as passive measurement references, not as goals for robot motion. This gap — a

complete, validated pipeline from concentric circle detection to perpendicular robotic contact — defines the scope of the present thesis. The next chapter introduces the overall system architecture and the hardware platforms on which the work was developed.

Chapter 2

System Architecture and Hardware

This chapter describes the overall system architecture common to all three development stages and presents the hardware specifications for each platform. The common core across all stages is the vision-to-motion pipeline: coded concentric circular markers are placed on the component surface; the robot camera detects these markers and estimates their six-degree-of-freedom pose; a coordinate transformation maps the poses from the camera frame to the robot base frame; and the motion controller commands the robot to approach each marker target perpendicularly so that the excitation tool contacts the surface at the defined point.

2.1 System overview

Three development stages were followed, each validating a more capable hardware platform. The same underlying pipeline runs in all three: the camera detects concentric circle markers on the component surface, estimates their 6D pose, maps those poses into the robot base frame via a coordinate transformation, and commands the robot to approach each marker perpendicularly so the excitation tool makes contact at the correct location. Figure 2.1 illustrates the overall architecture of the system implemented in the Stage 2 simulation.

2.1.1 Stage 1 - Five-DOF Prototype Arm with Raspberry Pi 5

The first development stage uses a custom-built five-degree-of-freedom serial robot arm. The arm is constructed from aluminium profile and three-dimensional printed joint housings, with five rotary joints driven by servo motors controlled via an Adafruit PCA9685 16-channel PWM controller connected to a Raspberry Pi 5. The servo actuation uses pulse-width modulation with a range of 500 to 2500 µs, providing approximately 180° of rotation per joint. The physical arm and its main components are shown in Figure 2.2.

The physical dimensions of the arm determine the Denavit-Hartenberg kine-

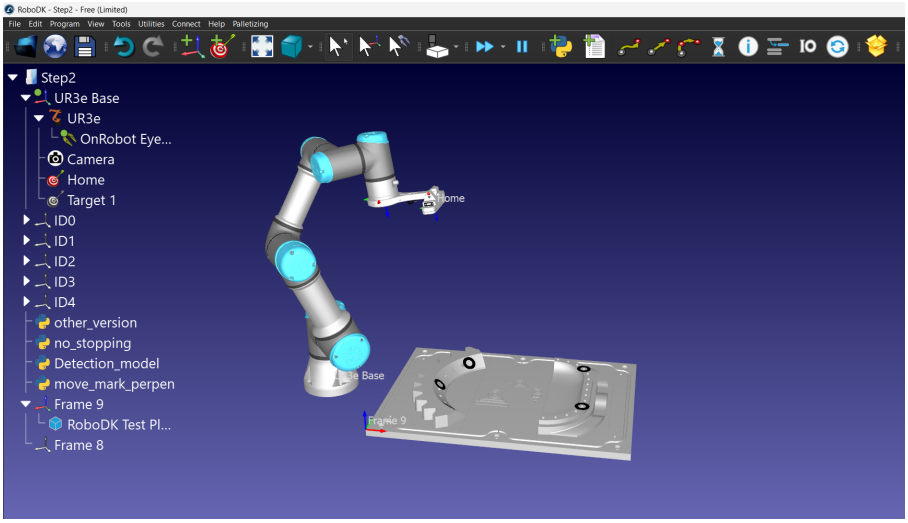


Figure 2.1: System overview: RoboDK simulation environment with the UR3e robot, camera tool, test plate, and five concentric circle marker targets.

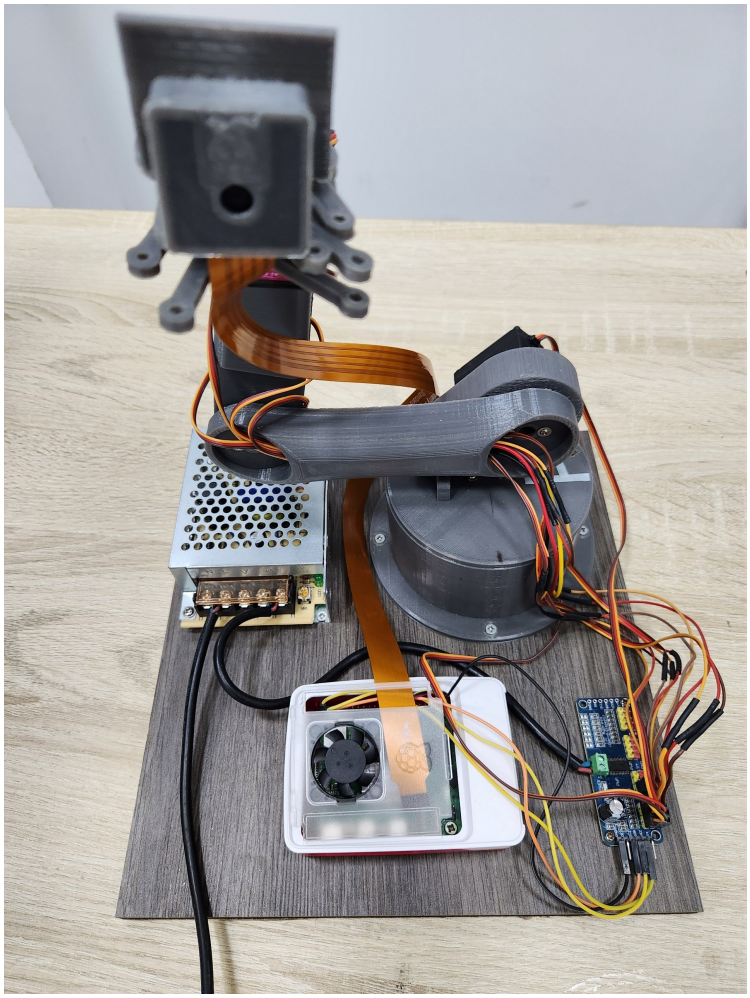


Figure 2.2: The five-DOF prototype arm used in Stage 1, with the five servo-driven joints, the end-effector camera, the Raspberry Pi 5, and the PCA9685 servo controller.

Table 2.1: Stage 1 hardware specifications.

Component	Model	Specification	Range	Interface
Microcontroller	Raspberry Pi 5	ARM Cortex-A76, 8 GB	—	GPIO, I^2C, USB
PWM controller	Adafruit PCA9685	16-ch, 12-bit	500–2500 μs	I^2C (0x40)
Camera	Picamera2 (IMX219)	8 MP	60° FOV	CSI
Joints 1–5	Metal gear servo	Steel shaft	0–180°	PWM

matic parameters, which are derived and validated in Chapter 4. The home configuration places the arm in an upright extended position corresponding to joint angles of $q = [90, 90, 110, 90, 0]°$. At this configuration, the end-effector is located at approximately $[8.0, 0.0, 35.5]$ cm in the robot base frame, as verified by direct measurement and confirmed by the forward kinematics model.

A calibrated camera is rigidly mounted on the end-effector, providing the visual input for marker detection. In Stage 1, ArUco markers were used as positioning targets to validate the pipeline before the concentric circle detector was developed. The camera-to-end-effector transform encodes the fixed spatial relationship between the camera optical centre and the end-effector frame, determined by physical measurement: translation $[22.5, 0, 21.5]$ mm, with the rotation matrix $R = \left[\begin{smallmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{smallmatrix}\right]$.

2.1.2 Stage 2 - UR3e Simulation in RoboDK

In the next phase, the system is prepared for use with real industrial robots by first testing everything in a virtual simulation. This approach reduces risk and helps identify any problems before working with actual hardware. The chosen robot, the UR3e, is a collaborative robot arm that can move in six different directions (or axes), allowing it to reach various positions and angles. It has a maximum reach of 500 millimeters (about half a meter), can lift objects weighing up to 3 kilograms, and is designed to repeat the same movement with a high degree of accuracy (within 0.03 millimeters). The UR3e was selected for this system for two key reasons: first, RoboDK, the simulation software, provides a reliable and ready-to-use digital model for this robot, making setup easier and more accurate; second, the `ur_ros2_driver` software package allows direct communication between the simulation and the real robot hardware. This means the same instructions used in the simulation can control the physical robot, making it much simpler to move from testing in the virtual environment to using the real robot in practice.

To accurately test the system before working with a real robot, a detailed simulation environment was built in RoboDK using the official UR3e model supplied by the manufacturer. In this virtual scene, several important components are included:

the UR3e robotic arm itself; an OnRobot Eye camera, which is attached at the end of the robot arm where tools are typically mounted (known as the Tool Center Point, or TCP); a flat plate that serves as a stand-in for the actual surface that will be inspected during real operations; and five circular marker targets, called annulus markers, which are used to help the system recognize positions during testing. The simulation was gradually developed in three stages, saved as separate RoboDK project files (named `Step1.rdk` through `Step3.rdk`). Each file represents a new step in the process, where additional features were added: starting with basic object detection, then adding the ability to calculate and transform coordinates, and finally incorporating advanced motion planning so the robot can move accurately and safely within the virtual environment.

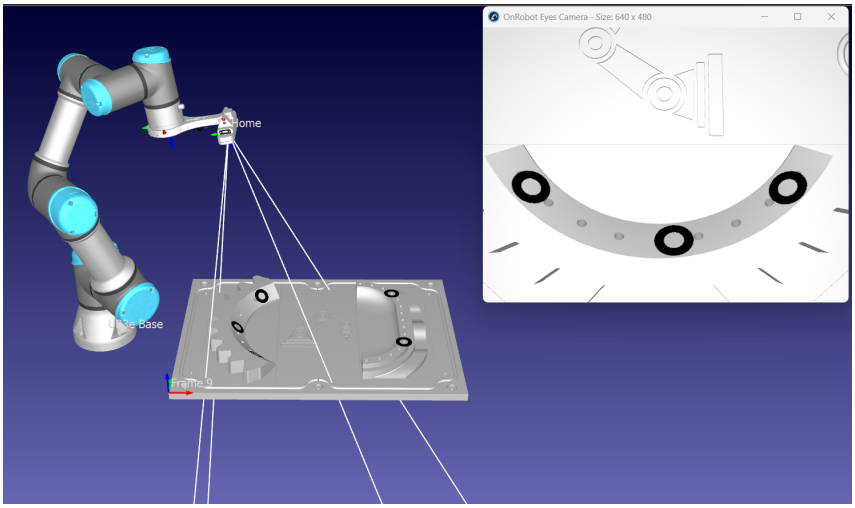


Figure 2.3: RoboDK simulation scene: UR3e robot, camera tool, test plate with five annulus marker targets, and simulated camera view inset.

2.1.3 Stage 3 - ROS 2 Implementation with Gazebo and MoveIt2

In the third stage of development, the entire robotic inspection system is set up to run within the ROS 2 software framework. This means that instead of using RoboDK's built-in scripting tools, the project now relies on ROS 2, which is an open-source robotics platform widely used for real-world robot applications. The software runs on ROS 2 Humble, a specific version of ROS, using Ubuntu 22.04 as the operating system. For simulations, the UR3e robot is modeled in Ignition Gazebo, which lets researchers test the setup in a realistic 3D environment. The robot's digital description is built using a custom XACRO file (`ur3e_camera` package). This is a type of script that defines what parts the robot has and how they are connected. In this case, the model is expanded to include two extra pieces: a virtual Intel RealSense D405 camera, which is attached securely to the end of the robot arm, and a digital hammer tool that acts as the device for tapping or exciting the component during inspection. The hammer's design creates a new reference point, called the hammer tip, which is 216 mm in front of the robot's main mounting point (the flange). All of

the robot's movement targets are calculated based on this hammer tip, so the system always plans around the spot that will actually touch the part, rather than the base of the tool itself. This approach improves accuracy when simulating or performing real inspection tasks.

The runtime architecture consists of standard infrastructure nodes —`robot_state_publisher`, ROS `_gz_bridge` connecting Gazebo topics to ROS 2, the `controller_manager` with joint state broadcaster and joint trajectory controller, the MoveIt2 `move_group` planning node, and RViz2 for visualization — together with two custom application nodes summarized in Table 2.2. The detection node processes the synchronized color and depth streams from the simulated D405, detects the five concentric circle markers, and publishes their 6D poses both as a `PoseArray` message and as TF frames (`annulus_1` to `annulus_5`) for direct inspection in RViz. The motion controller node subscribes to the detected poses, transforms them into the robot base frame through TF2, and executes the striking sequence for each target through the MoveIt2 planning interface. A graphical operator panel provides *Go Home*, *Start Sequence*, *Confirm*, and *STOP* controls; the stop command sets a flag that is checked between all motion steps and additionally cancels the trajectory currently in execution, bringing the robot to a halt immediately. The behaviour and quantitative validation of this stage are presented in Chapter 5. Figure 2.4 shows the node graph and the main communication topics of this architecture.

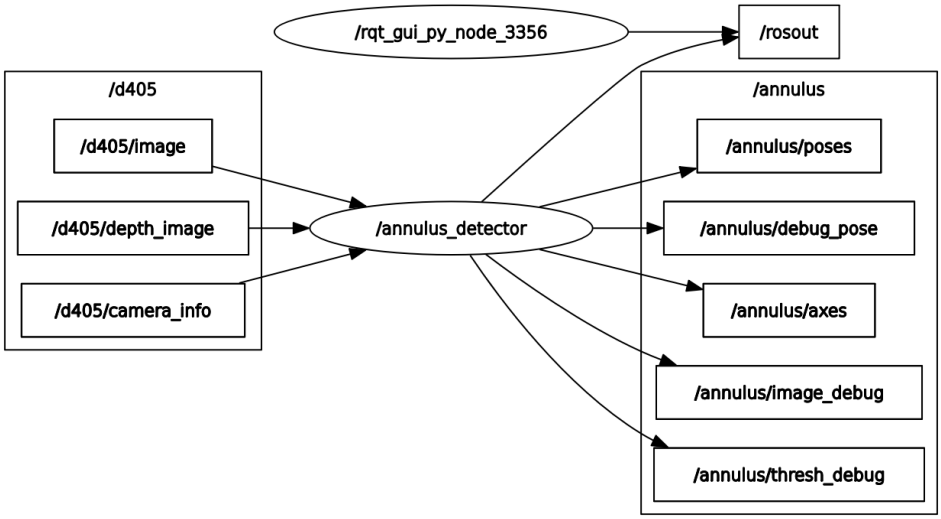


Figure 2.4: ROS 2 architecture of the Stage 3 application: the detection node processes the simulated D405 streams and publishes marker poses, which the motion controller converts into MoveIt2 planning goals executed by the joint trajectory controller in Gazebo.

Table 2.2: ROS 2 node architecture of the Stage 3 application.

Node	Package	Function
`annulus_detector`	`my_robsim` (custom)	Detects concentric circle markers in the D405 colour/depth streams; publishes `/annulus/poses` (`PoseArray`), RViz axis markers on `/annulus/axes`, a debug image, and TF frames `annulus_1`–`annulus_5`
`hammer_controller`	`my_robsim` (custom)	Transforms marker poses to the base frame via TF2; computes perpendicular strike targets; plans and executes approach–strike–retract through MoveIt2; hosts the operator GUI (start/stop/home)
`move_group`	`moveit_ros_move_group`	Motion planning, inverse kinematics service, and trajectory execution monitoring for the `ur_manipulator` group
`ros_gz_bridge`	`ros_gz_bridge`	Bridges camera, clock, and joint topics between Ignition Gazebo and ROS 2

2.2 Test Component and Marker Placement

Every experiment described in this thesis focuses on inspecting a real aerospace part. Specifically, the target is a curved blade made from aluminum alloy, which has a complicated 3D surface shape. To help the robot identify where to check for vibrations, special circular markers are attached to the blade. These markers are white rings on a black sticker, with an outer diameter of 32.22 mm and an inner diameter of 15.62 mm. There are five markers in total, and they are placed in positions that would make sense for a real-world inspection task. Importantly, the robot is not given the marker locations in advance. Instead, when the process begins, the robot looks at the component from a starting position, finds all the visible markers using its camera, and then creates a list of targets to inspect based on what it actually sees. This approach ensures the robot can adapt to the real placement of the markers, as illustrated in the accompanying Figure 2.5.

This chapter has described the three-stage development architecture and the hardware and software platforms used throughout the thesis. Chapter 3 presents the vision system, covering the camera calibration procedure and both marker detection pipelines.

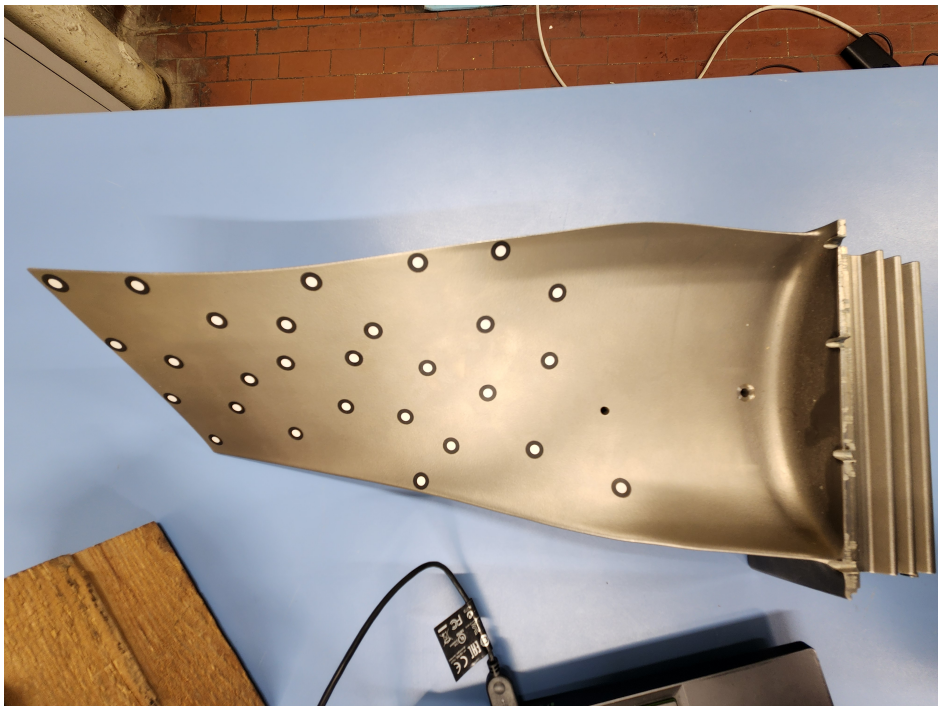


Figure 2.5: The aerospace test component on the laboratory bench. Concentric circle marker targets are placed at the inspection points.

Chapter 3

Vision System and Pose Estimation

This chapter presents the camera calibration procedure and the two marker detection pipelines developed in this thesis: the ArUco square marker detector used in Stage 1, and the concentric circle detector used in Stages 2 and 3. For each pipeline, the algorithm is described, the pose estimation method is explained, and the coordinate frame conventions are defined.

3.1 Camera Model and Projection Geometry

Every quantity the vision system produces — marker distances, tilt angles, target coordinates — is ultimately derived from the geometric model that relates a point in the three-dimensional world to a pixel in the image. This section states that the model and the parts of it the later sections rely on. The treatment follows the standard formulation [**Hartley2004**]; the purpose here is to make explicit which parameters the calibration of Section 3.2 must estimate, and where each source of error enters.

In the pinhole model, a point $\boldsymbol{P}_C = [X, Y, Z]^\top$ expressed in the camera frame projects onto the image plane according to the perspective projection:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \qquad \boldsymbol{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

where (u, v) are pixel coordinates, $s = Z$ is the projective scale, and $\boldsymbol{K}$ is the intrinsic matrix. The physical meaning of the parameters is direct: f_x and f_y are the focal length expressed in pixel units along each image axis, and (c_x, c_y) is the principal point, the pixel where the optical axis crosses the image plane. Two consequences of equation (3.1) shape the entire system design. First, the projection divides by Z: an object twice as far away appears half as large, which is precisely why the apparent size of a marker of known physical dimensions encodes its distance. Second,

the same division makes the model scale-ambiguous for unknown objects — depth can only be recovered because the marker geometry is known in advance. It is also worth noting that f_x and f_y are expressed in pixels, so they change with the sensor resolution: intrinsics calibrated at one resolution and applied at another scale every depth estimate by the resolution ratio, an effect encountered experimentally in Section 3.5.1.

Real lenses deviate from the ideal pinhole through distortion, modelled in OpenCV by a polynomial correction applied in normalized image coordinates. With $r^2 = x^2 + y^2$ for a normalised point (x, y), the corrected coordinates are:

$$\begin{aligned} x' &= x\,(1 + k_1 r^2 + k_2 r^4) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y' &= y\,(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned} \tag{3.2}$$

where k_1, k_2 are radial distortion coefficients and p_1, p_2 tangential ones. Radial distortion grows with the distance from the image center, so its practical effect on this system is position-dependent: a marker detected near the image border is displaced more than one near the center. If distortion were left uncorrected, the ellipse fitted to a ring contour would be systematically deformed, corrupting both the center estimate and the axis ratio on which the tilt calculation rests. This is why every frame is undistorted with the calibrated coefficients before contour extraction.

3.1.1 The Perspective-n-Point Problem

Pose estimation inverts the projection: given N known points $\boldsymbol{P}_i$ on the marker, their detected images (u_i, v_i), and the calibrated $\boldsymbol{K}$, find the rotation $\boldsymbol{R}$ and translation $\boldsymbol{t}$ of the marker frame in the camera frame. This is the Perspective-n-Point (PnP) problem [12]. The iterative solver used in this work minimizes the reprojection error:

$$\min_{\boldsymbol{R},\,\boldsymbol{t}} \sum_{i=1}^{N} \left\| \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \pi(\boldsymbol{K}\,(\boldsymbol{R}\,\boldsymbol{P}_i + \boldsymbol{t})) \right\|^2 \tag{3.3}$$

where $\pi(\cdot)$ denotes the perspective division of equation (3.1). The formulation makes the role of each ingredient visible: the object model $\boldsymbol{P}_i$ carries the physical marker dimensions (so an error in the assumed marker size scales $\boldsymbol{t}$ proportionally), the intrinsics carry the calibration quality, and the detected points carry the image-processing accuracy. For the annulus marker, the $\boldsymbol{P}_i$ are sixteen points on the two ring circumferences, all lying in the marker plane $Z = 0$ — the planar case, for which the projection reduces to a homography, and the problem is well conditioned for position even at moderate N.

Internally, rotations are exchanged with OpenCV in axis-angle form: `solvePnP` returns a rotation vector $\boldsymbol{r}$ whose direction is the rotation axis and whose magnitude $\theta = \|\boldsymbol{r}\|$ is the rotation angle. The rotation matrix follows from the Rodrigues

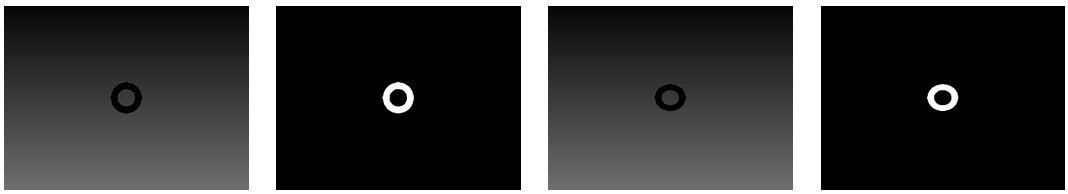
formula:

$$\boldsymbol{R} = \boldsymbol{I} + \sin\theta\,[\hat{\boldsymbol{r}}]_\times + (1 - \cos\theta)\,[\hat{\boldsymbol{r}}]_\times^2 \tag{3.4}$$

where $\hat{\boldsymbol{r}} = \boldsymbol{r}/\theta$ and $[\hat{\boldsymbol{r}}]_\times$ is the skew-symmetric cross-product matrix of $\hat{\boldsymbol{r}}$. The conversion matters practically because the transformation chain of Chapter 4 composes homogeneous matrices: every rotation vector from the detector passes through equation (3.4) before entering the chain, and one of the earlier implementation bugs of the project — applying an axis-reordering matrix to $\boldsymbol{r}$ instead of to $\boldsymbol{R}$ — produced non-orthogonal "rotation" matrices that broke the downstream pose computation. Reordering is only meaningful on the matrix, not on the axis-angle vector.

A final geometric fact underpins the marker design choice. Under the projection of equation (3.1), a circle in a tilted plane maps to an ellipse whose semi-minor-to-semi-major ratio equals the cosine of the tilt angle between the marker plane and the image plane [9]. The projection preserves the ellipse as a conic — it does not distort it into a general oval — which is why fitting an ellipse to the ring contour is a geometrically faithful operation and why the axis ratio can serve as a direct tilt measurement (Section 3.4.4).

The behavior of this stage is illustrated in Figure 3.1 with debug frames recorded during the E3 experiment: the blurred greyscale input and the corresponding binary image, for a marker viewed head-on and at a 30° tilt. The ring pair survives binarisation as two clean nested blobs in both cases; what changes with tilt is the eccentricity of the resulting contours, which is exactly the quantity the orientation estimator consumes.



(a) greyscale, 0° tilt **(b)** binary, 0° tilt **(c)** greyscale, 30° tilt **(d)** binary, 30° tilt

Figure 3.1: First processing stage on recorded E3 debug frames: blurred greyscale input and binarised output for the marker at 0° and 30° tilt. Binarisation preserves the nested ring structure; the tilt appears as ellipse eccentricity in the contours.

3.2 Camera Calibration

3.2.1 Calibration Procedure

Camera calibration was performed using the Zhang planar chessboard method [10], implemented through the OpenCV `calibrateCamera` function. The calibration target consisted of a printed chessboard pattern with 8×5 interior corner intersections and a square size of 25.0 mm. A set of 22 calibration images was captured from a range of viewing distances (300 to 800 mm) and viewing angles, ensuring that the calibration

grid occupied different positions and orientations within the image frame. Sub-pixel corner refinement (`cornerSubPix`) was applied with a search window of 11×11 pixels to improve corner localization accuracy. Images with root-mean-square reprojection error exceeding 1.0 pixel were excluded before computing the final calibration.

3.2.2 Calibration Results

The overall root-mean-square reprojection error was 0.63 pixels across the 22 calibration images, with per-image errors between approximately 0.38 and 0.42 pixels for the best views. This is slightly above the 0.5-pixel threshold often quoted for high-accuracy robotic vision work; the residual is attributed to the print quality of the paper chessboard target and to mild motion blur in a subset of hand-held views. As shown below, the resulting model is nevertheless sufficient for the intended working range, since the end-to-end distance validation remains within 3% of ruler ground truth. The calibrated intrinsic parameters are presented in Table 3.1, and Figure 3.2 shows the per-image reprojection error and the effect of distortion correction.

Table 3.1: Calibrated camera intrinsic parameters.

Parameter	Symbol	Value
Horizontal focal length	f_x (pixels)	615.90
Vertical focal length	f_y (pixels)	616.05
Principal point, horizontal	c_x (pixels)	321.45
Principal point, vertical	c_y (pixels)	218.81
Radial distortion	k_1, k_2	0.4483, −2.3292
Tangential distortion	p_1, p_2	−0.0182, 0.0018
RMS reprojection error	—	0.63 pixels

3.2.3 Distance Validation Against Ruler Ground Truth

Because the calibrated intrinsics directly scale every distance estimate produced by the pose estimator, the calibration was validated end-to-end: a marker was placed at eleven known distances between 143 mm and 650 mm, measured with a ruler, and the distance reported by the detection model was recorded at each position. Table 3.2 summarises the results. The absolute error grows roughly with distance, from 1.8 mm at 143 mm to 16.7 mm at 650 mm, but the relative error stays between 0.9% and 2.8% over the whole range. The error is not strictly monotonic — for example, the 600 mm measurement is more accurate than the 550 mm one — which suggests that a part of the residual comes from the manual ruler reference rather than from the vision system alone.

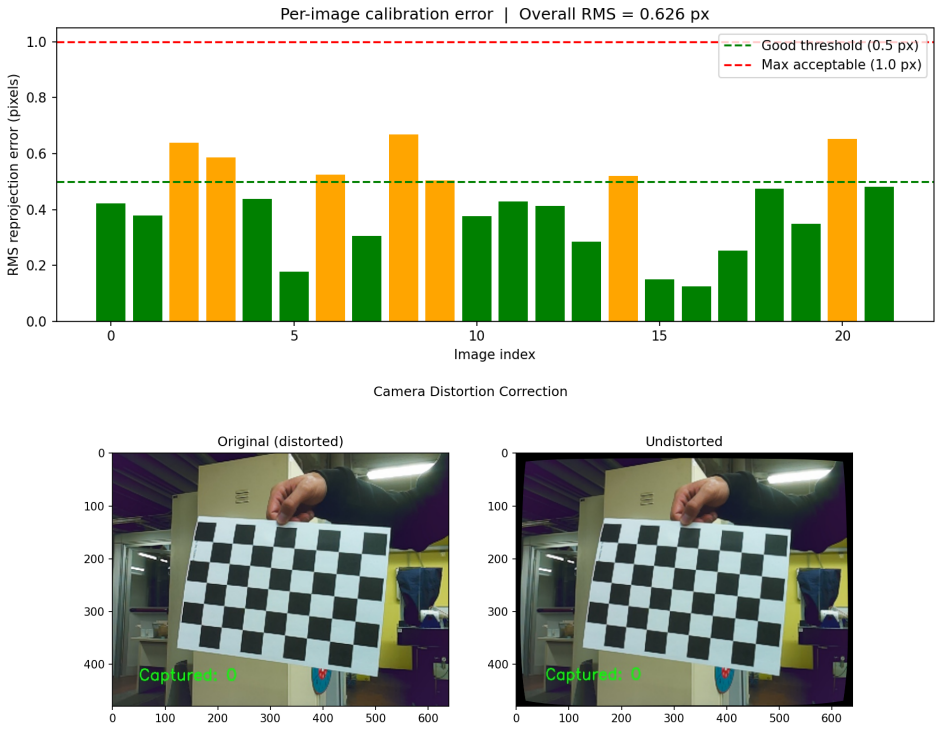


Figure 3.2: Calibration quality assessment: (a) per-image RMS reprojection error across the 22 calibration views; (b) comparison of a raw and undistorted image showing the effect of the estimated distortion model.

Table 3.2: End-to-end distance validation of the calibrated detection model.

Ruler distance (mm)	Estimated distance (mm)	Absolute error (mm)	Relative error (%)
143	144.8	1.8	1.2
218	220.0	2.0	0.9
250	247.2	2.8	1.1
300	297.0	3.0	1.0
350	354.5	4.5	1.3
400	408.8	8.8	2.2
450	460.0	10.0	2.2
500	514.4	14.4	2.8
550	562.2	12.2	2.2
600	610.1	10.1	1.7
650	666.7	16.7	2.5

3.3 ArUco Marker Detection (Stage 1)

3.3.1 Detection Pipeline

In Stage 1, ArUco markers from the DICT ARUCO ORIGINAL dictionary were used as positioning targets to validate the coordinate transformation pipeline. The detection pipeline operates on frames captured by the Picamera2 module at 1280×720 pixel resolution. Each captured frame is converted to grayscale and passed to the ArucoDetector. Sub-pixel corner refinement (CORNER REFINE SUBPIX) is applied to all detected corners, and the estimatePoseSingleMarkers function recovers the rotation vector r and translation vector t of each detected marker in the camera frame. The physical ArUco marker size used in Stage 1 experiments was 64.5 mm as measured by calipers; this value directly scales the translation estimate returned by the PnP solver, so it was verified against the printed marker before every experiment session. The fixed camera-to-end-effector transform that relates detected poses to the arm is given by equation (3.5).

$$\boldsymbol{T}_C^{EE} = \begin{bmatrix} 0 & 0 & 1 & 22.5 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 21.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

3.4 Concentric Circle Marker Detection (Stages 2 and 3)

3.4.1 Motivation for Concentric Circle Markers

Concentric circle markers were adopted for Stages 2 and 3 for two reasons. The first is geometric: a pair of concentric rings projects under perspective to a pair of nested ellipses, and the axis ratio of the outer ellipse directly encodes the surface tilt angle. This gives the full 6D pose — including out-of-plane orientation — from a single image, which matters here because the robot needs the surface normal to orient the solenoid, not just the marker's (x, y, z) position. The second reason is practical: these markers are sold as standard adhesive stickers for GOM-style scanning systems, so they were available without any custom printing.

The detection algorithm must reliably distinguish the coded concentric circle markers from other circular or elliptical shapes that may appear in the image — fastener heads, background patterns, or surface machining marks. This requirement motivates the hierarchical contour approach and the three-stage geometric consistency verification described below.

3.4.2 Marker Geometry

Each marker is a pair of concentric white rings on a black background. In the simulation environments, the outer ring has a diameter of 32.22 mm and the inner ring 15.62 mm, an inner-to-outer ratio of 0.485; the physical adhesive targets used in the real-camera experiments measure 39.5 mm and 19.0 mm respectively (ratio 0.481), as verified with callipers. The ratio enters the geometric consistency check described below, while the absolute dimensions define the 3D point model passed to `solvePnP` — an error in either value scales the estimated distance proportionally, which is why both were measured rather than taken from the vendor specification. The black annular region between the rings produces strong intensity contrast, which yields clean binary contours even under uneven lighting. Having two rings rather than one is also what makes the concentricity check possible — a single ring gives no way to distinguish a genuine marker from any other circular feature in the image.

3.4.3 Detection Algorithm

The annulus detection pipeline consists of four processing stages. In the first stage, the input frame is converted to greyscale, smoothed with a Gaussian blur (5×5 kernel), and binarised. Blurring before thresholding proved necessary in practice: applying Otsu thresholding directly to the raw sensor image shatters the ring boundary into speckle, whereas the blurred image produces closed, well-defined ring contours. The Stage 2 implementation uses global Otsu thresholding, which is adequate for the uniformly lit RoboDK scene; the Stage 3 ROS 2 node replaces it with adaptive thresholding, which proved more stable against the local illumination gradients produced by the Gazebo rendering and by specular metallic surfaces.

In the second stage, contours are extracted from the binary image using the RETR TREE retrieval mode, which preserves the parent-child hierarchy: a concentric circle marker produces two nested closed contours — the outer ring as a parent and the inner ring as a child with a non-negative child index. Contours with fewer than 60 points or with an enclosed area below a minimum threshold are discarded immediately, removing small noise blobs before any ellipse fitting is attempted.

In the third stage, for each surviving parent contour that has at least one child, `fitEllipse` is called on both the outer and inner contours to recover their centers, semi-axes (a, b), and orientations (ϕ). A candidate pair is accepted as a marker only if it passes three geometric consistency checks:

1. the distance between the two ellipse centers is below 10 pixels (concentricity);
2. the outer ellipse axis ratio b/a is above a minimum aspect bound, rejecting near-degenerate projections at extreme viewing angles;
3. The ratio of inner to outer ellipse size approximates the known physical ratio of the marker rings (geometry condition).

These checks are what allow the detector to reject fastener heads, drilled holes without the inner ring, and other circular features that would otherwise pass a single-ellipse test.

In the fourth stage, sixteen object-to-image point correspondences are constructed — eight cardinal points sampled on the outer ellipse and eight on the inner ellipse — and passed to `cv2.solvePnP` to recover the marker translation vector $\boldsymbol{t}$ in the camera frame. The rotation is not taken from the PnP solution directly: because the sampled points are nearly coplanar and concentric, the PnP orientation estimate is poorly conditioned, and the marker rotation matrix is instead constructed from the fitted ellipse geometry as described in Section 3.4.4. Figure 3.3 shows an example detection result from the RoboDK simulation.

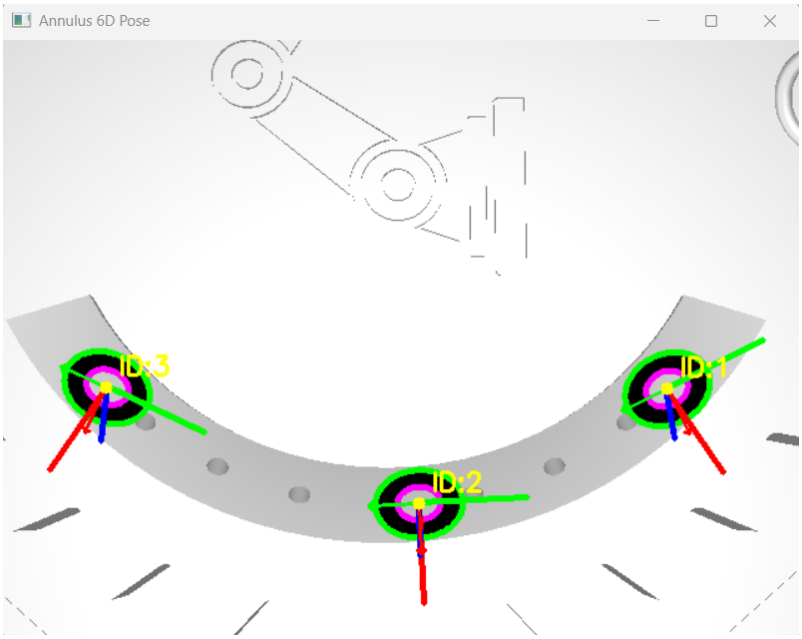


Figure 3.3: Annulus marker detection result in RoboDK simulation: three markers simultaneously detected with ID labels and 6D coordinate axis overlays.

3.4.4 Orientation Estimation from Ellipse Geometry

The out-of-plane tilt angle of the marker surface is directly encoded in the projected ellipse axis ratio. For a circle of radius r viewed at tilt angle theta from the camera optical axis, the projected ellipse has semi-major axis $a = r$ and semi-minor axis $b = r \cdot \cos(\theta)$. The tilt angle is therefore:

$$\theta = \arccos\left(\frac{b}{a}\right) \tag{3.6}$$

where b/a is clipped to $[-1, 1]$ before the arccosine to prevent numerical instability. The surface normal vector in the camera frame is then:

$$\hat{\boldsymbol{n}} = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ -\cos\theta \end{bmatrix} \tag{3.7}$$

where ϕ is the orientation angle of the ellipse major axis. The complete rotation matrix for the marker is constructed by defining the z-axis as this surface normal, selecting a consistent x-axis along the ellipse major axis, and computing the y-axis by cross product.

Equation (3.6) recovers only the magnitude of the tilt, not its sign: a circle tilted by $+\theta$ and by $-\theta$ projects to the same ellipse. The implementation resolves this ambiguity by convention, always constructing the normal so that the marker's z-axis points toward the camera. This assumption holds in every configuration used in this work because the camera necessarily views each marker from the approach side of the surface; it would need to be revisited for geometries in which markers could be observed from grazing or reverse angles.

3.5 Vision System Characterisation

Two experiments characterize the detector before it is used for robot guidance: detection accuracy as a function of working distance (E2), and orientation estimation accuracy as a function of surface tilt (E3). Together with the calibration validation of Section 3.2.3, they define the measurement performance that the motion strategy of the later chapters has to work with.

3.5.1 Detection Accuracy versus Distance (E2)

A physical annulus marker was placed at seven distances between 220 mm and 500 mm from the camera, and 30 frames were processed at each distance. Table 3.3 reports the detection rate and the estimated depth statistics. Two observations stand out. First, the detection rate is high across the range — 100% from 350 mm outward and no worse than 83% at the closest positions, where partial specular reflection on the glossy marker surface occasionally broke the inner contour. Second, the frame-to-frame repeatability of the depth estimate is excellent: the standard deviation remains below 0.34 mm across all distances, indicating that the ellipse-fitting is highly stable when detection succeeds.

The absolute depth values in this run, however, show a systematic scale offset: the estimated depth follows the true distance linearly (slope 0.421, intercept 12.2 mm, $R^2 > 0.999$) but underestimates it by a nearly constant factor. The cause was traced to a mismatch between the resolution at which the intrinsics were calibrated

and the sensor mode used during the run — a scaling that affects all depth estimates uniformly and leaves lateral accuracy and repeatability untouched. The result illustrates an important practical property of single-camera pose estimation: intrinsic calibration errors act as systematic, correctable biases rather than random noise. When calibration and acquisition resolution match, as in the validated configuration of Table 3.2, the absolute distance error remains below 3%.

Table 3.3: E2 — detection rate and depth estimate statistics versus working distance (30 frames per distance).

True distance (mm)	Estimated depth, mean ± std (mm)	Detection rate (%)
220	105.66 ± 0.06	96.7
260	121.64 ± 0.28	83.3
300	137.88 ± 0.11	96.7
350	158.88 ± 0.11	100.0
400	180.30 ± 0.26	100.0
450	201.52 ± 0.30	100.0
500	223.36 ± 0.33	100.0

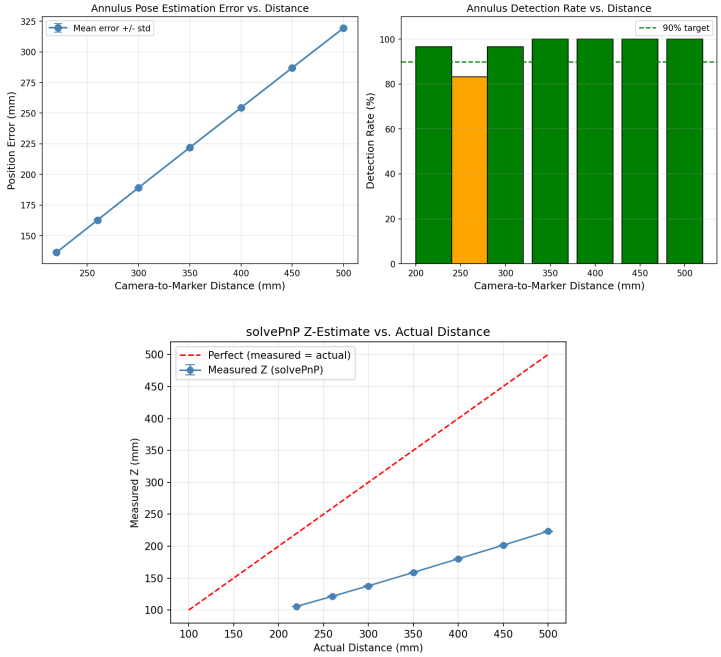


Figure 3.4: E2 results: (a) position error versus working distance; (b) estimated depth versus true distance, showing the linear relationship ($R^2 > 0.999$) with the systematic scale factor discussed in the text.

3.5.2 Orientation Accuracy versus Tilt (E3)

The orientation estimator of Section 3.4.4 was evaluated by rendering the marker at eight known tilt angles between 0° and 40°, with 20 frames per angle, and comparing the estimated tilt against the ground truth. The marker was detected in all 160 frames. Table 3.4 summarises the results: for tilts between 5° and 40° the absolute

angular error stays below 0.8°, and for most angles below 0.5°, comfortably inside the 5° perpendicularity budget adopted for the approach motion.

The only exception is the head-on case: at a true tilt of 0°, the estimator reports 3.7°. This is an expected structural weakness of the axis-ratio method rather than a tuning issue. Near zero tilt, $\cos\theta \approx 1 - \theta^2/2$, so the ellipse axis ratio becomes insensitive to the tilt angle and any residual pixel-level error in the fitted axes — including the slight ring erosion introduced by blurring and thresholding — maps to a disproportionately large angle. In practice, this error mode is benign for the present application: a marker viewed head-on is already nearly perpendicular to the camera axis, so a few degrees of overestimated tilt produce a negligible orientation correction at the target.

Table 3.4: E3 — orientation estimation accuracy versus surface tilt (20 frames per angle, 100% detection).

True tilt (°)	Measured tilt (°)	Absolute error (°)
0	3.68	3.68
5	5.29	0.29
10	9.24	0.76
15	14.65	0.35
20	20.64	0.64
25	25.06	0.06
30	30.50	0.50
40	40.33	0.33

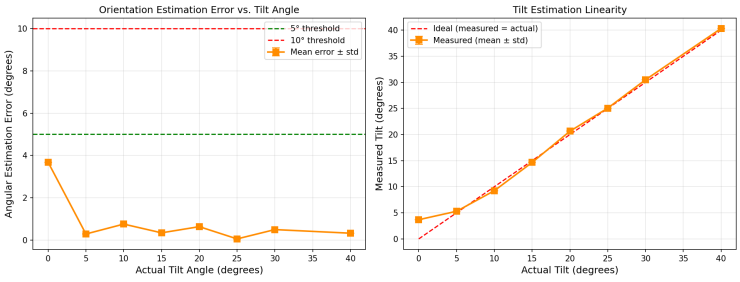


Figure 3.5: E3 results: measured tilt versus true tilt. The estimator tracks the ground truth closely above 5°; the deviation at 0° reflects the axis-ratio method's reduced sensitivity near head-on viewing.

3.6 Coordinate Frame Definitions

Four coordinate frames are used throughout the system. The camera frame $\{C\}$ has its origin at the camera optical center, with the z-axis pointing along the optical axis, the x-axis pointing to the right, and the y-axis pointing downward. The end-effector frame $\{EE\}$ has its origin at the TCP, with the x-axis pointing forward along the tool axis. The robot base frame $\{B\}$ has its origin at the base mounting plane, with the z-axis pointing upward. The marker frame $\{M\}$ has its origin at the

annulus marker center, with the z-axis pointing along the surface normal toward the camera, and the x and y axes in the marker plane. Figure 3.6 illustrates the relationships between these frames and the transformations that connect them.

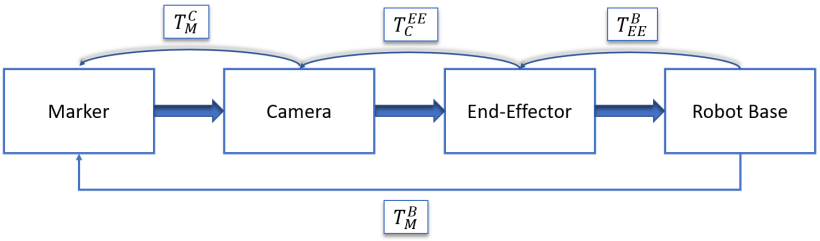


Figure 3.6: Coordinate frames used in the system and the homogeneous transformations connecting them: $\boldsymbol{T}_{EE}^{B}$ from forward kinematics, $\boldsymbol{T}_{C}^{EE}$ from the camera mounting, and $\boldsymbol{T}_{M}^{C}$ from pose estimation.

This chapter has presented the camera calibration procedure, both detection pipelines, and the experimental characterization of the vision system. The following chapter derives the kinematic model and the coordinate transformation chain that links the detected marker pose to the robot base frame.

Chapter 4

Kinematic Modelling and Coordinate Transformations

This chapter derives the kinematic model for the five-degree-of-freedom prototype arm and the coordinate transformation chain used to map detected marker poses into robot base frame coordinates. The DH parameterisation, forward kinematics, and numerical inverse kinematics solution are presented and validated. The transformation chain from camera frame to robot base frame is then derived and the perpendicular approach target computation is described.

4.1 Denavit-Hartenberg Parameterization

4.1.1 DH Convention

The kinematic model of the five-degree-of-freedom prototype arm is derived using the standard Denavit-Hartenberg convention [13]. The transformation from frame $\{i-1\}$ to frame $\{i\}$ is expressed as the product of four elementary transformations:

$$\boldsymbol{T}_i^{i-1} = \boldsymbol{R}_z(\theta_i)\,\boldsymbol{T}_z(d_i)\,\boldsymbol{T}_x(a_i)\,\boldsymbol{R}_x(\alpha_i) \tag{4.1}$$

where θ_i is the joint angle, d_i the link offset, a_i the link length, and α_i the link twist. Each transformation matrix $\boldsymbol{T}_i^{i-1}$ is a 4×4 homogeneous matrix belonging to the Special Euclidean group $SE(3)$. The physical meaning of the four parameters is worth stating explicitly, since assigning them correctly to a physical arm is where most modelling errors originate: d_i and a_i locate the joint axes relative to each other (along and perpendicular to the previous joint axis respectively), while α_i and θ_i orient them. Figure 4.1 shows the frame assignment used for the prototype arm.

4.1.2 DH Parameters for the 5-DOF Arm

To accurately represent the robot's geometry, the Denavit-Hartenberg (DH) parameters were obtained by measuring the actual assembled robotic arm. However, getting the correct parameters wasn't straightforward and took three rounds of

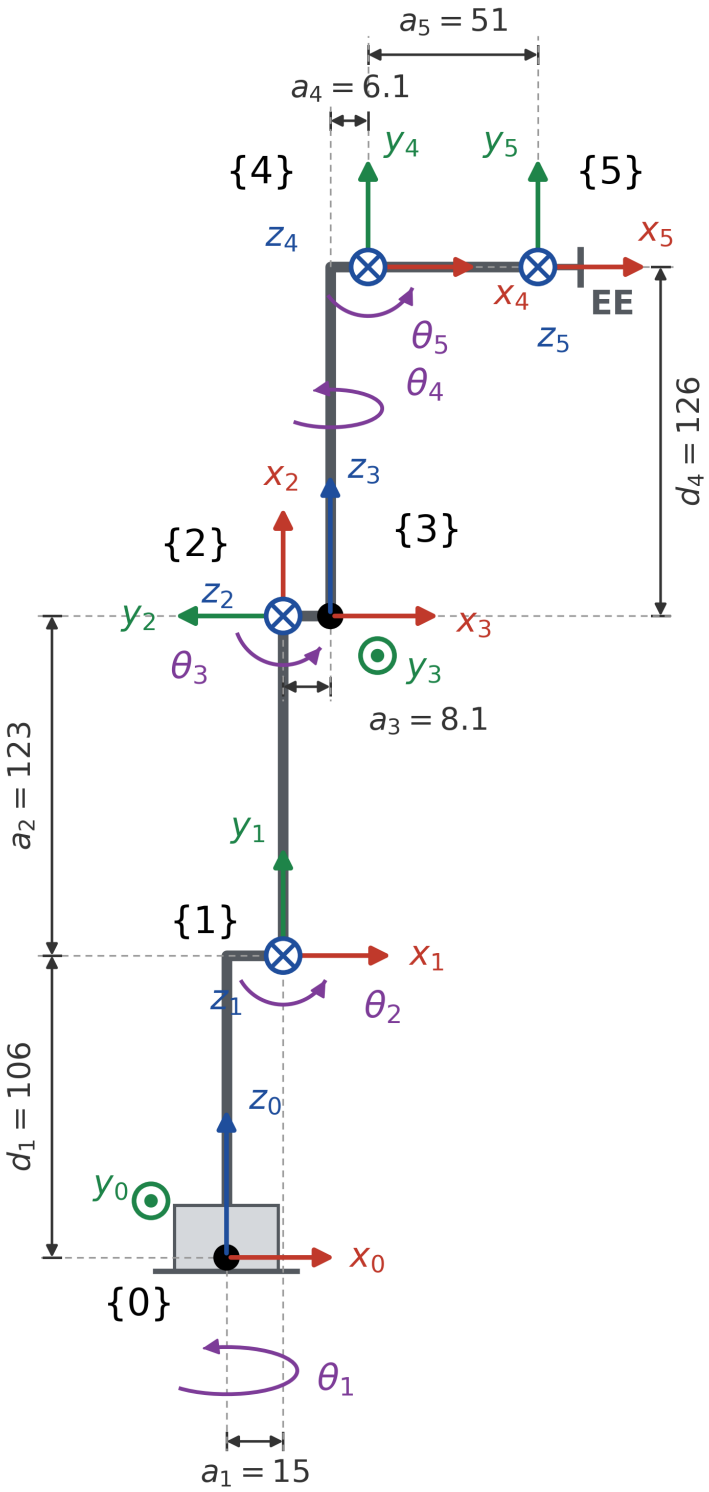


Figure 4.1: Denavit-Hartenberg frame assignment for the five-DOF prototype arm, showing the joint axes, the assigned frames {0} to {5}, and the principal link dimensions.

adjustments. Each round involved updating the measurements and testing whether the calculated position of the robot's end-effector (the working tip of the arm) matched where it actually ended up in real life. During this process, two notable mistakes were discovered. First, in the initial attempt, the offsets for the joint angles were entered using degrees, even though the model required them in radians. This mix-up caused the calculated position of the end-effector to be off by about 7 centimeters when the robot was in its home position—a big enough error to be noticed right away, though at first it was wrongly blamed on incorrect measurements of the robot's segments. In the next round, the team refined the values for some of the robot's link offsets, specifically d_3, and the small side-to-side offsets a_3 and a_4. These tweaks were made after comparing the calculated and actual positions of the end-effector at several different arm positions. After these corrections, the final set of DH parameters matched the real robot's movements and are listed in Table 4.1.

Table 4.1: DH parameters for the 5-DOF prototype arm.

Joint i	a_i (mm)	α_i (°)	d_i (mm)	θ_{offset} (°)	Type
1	15	90	106	−90	Revolute
2	123	0	0	0	Revolute
3	8.1	−90	−9.2	200	Revolute
4	6.1	90	126	−90	Revolute
5	51	0	10	0	Revolute

4.1.3 Joint Mechanical Convention

The third joint of the robot arm has a unique mechanical setup: when you increase the signal sent to its servo motor, the joint actually turns in the opposite direction to what the standard Denavit-Hartenberg (DH) convention defines as positive. This happens because of the way the servo horn—the piece that connects the motor to the joint—is attached inside the elbow. It's not an issue with the math or kinematic model, but a result of the physical assembly. If this isn't accounted for, the physical movement of the arm will be mirrored compared to the model, especially at the elbow. To address this, a specific conversion is used to translate the angle calculated by the model into the correct servo command for joint 3:

$$q_{\text{servo},3} = |q_{\text{model},3} - 180^\circ| \tag{4.2}$$

This conversion applies exclusively to joint 3. The home configuration in servo space is $\boldsymbol{q}_{\text{servo}} = [90, 90, 110, 90, 0]^\circ$, corresponding to $\boldsymbol{q}_{\text{model}} = [90, 90, 70, 90, 0]^\circ$.

4.2 Forward Kinematics

The forward kinematics maps the joint configuration $\boldsymbol{q} = [q_1, q_2, q_3, q_4, q_5]^\top$ to the end-effector pose $\boldsymbol{T}_{EE}^{B}$ as the sequential matrix product:

$$\boldsymbol{T}_{EE}^{B}(\boldsymbol{q}) = \boldsymbol{T}_{1}^{0}(q_1)\,\boldsymbol{T}_{2}^{1}(q_2)\,\boldsymbol{T}_{3}^{2}(q_3)\,\boldsymbol{T}_{4}^{3}(q_4)\,\boldsymbol{T}_{5}^{4}(q_5) \tag{4.3}$$

The model was implemented twice, deliberately: once with the Robotics Toolbox for Python [3] (`DHRobot` with `RevoluteDH` links), and once as an independent NumPy implementation that builds each matrix of equation (4.1) explicitly. Agreement between the two implementations was used as a first correctness check before any physical validation, which isolates coding errors from parameter errors.

To make sure the mathematical model of the robot arm accurately reflected the real hardware, its predicted positions were compared directly with measurements taken from the actual robot. These results are summarized in Table 4.2. When the robot was set to its standard 'home' position, the model's calculated tip location of $[8.02, -0.08, 35.50]$ cm was almost identical to the measured position of $[8.0, 0.0, 35.5]$ cm—differing by only about 1 millimeter. For a second test, the arm was moved to a less symmetrical position, and the difference between the model and real measurements grew to about 3 to 5 millimeters on each axis. This larger discrepancy is likely due to small errors from the servo motors and slight bending in the robot's links, which the mathematical model does not account for. However, these small differences are considered acceptable for this project, especially because the vision-based calibration step will correct any remaining bias when the robot is actually performing its inspection tasks.

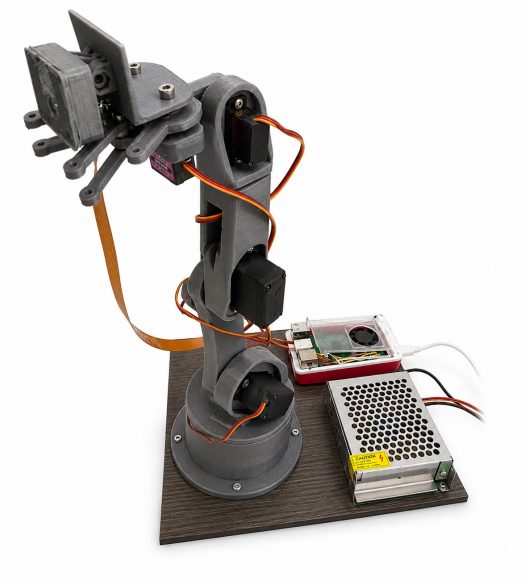


Figure 4.2: Prototype arm at the home configuration

Table 4.2: Forward kinematics validation against physical measurement.

Servo configuration $\boldsymbol{q}$ (°)	FK position (cm)	Measured position (cm)
$[90, 90, 110, 90, 0]$ (home)	$[8.02, -0.08, 35.50]$	$[8.0, 0.0, 35.5]$
$[90, 90, 100, 90, 20]$	$[10.11, -0.08, 35.95]$	$[9.8, 0.0, 35.5]$

4.3 Inverse Kinematics

4.3.1 Problem Formulation and the Five-DOF Orientation Deficit

A robot arm with five joints (five degrees of freedom) can only precisely control five aspects of its position and orientation in space, even though a full pose in 3D requires six (three for position, three for orientation). This means the system must let go of control over one orientation axis. Choosing which axis to ignore is important: for tasks where the tool needs to approach a marker perpendicularly, it's the position (x, y, z) and the direction of approach (pitch and yaw) that matter most. The roll axis, which is the tool spinning around its own length, doesn't affect the contact point or angle during these operations. So, it's logical to leave roll unconstrained. In the Levenberg-Marquardt solver, this is set up using a mask vector $[1, 1, 1, 0, 1, 1]$—the zero means roll is left free, while the others are enforced.

This lesson about which axis to leave uncontrolled was actually discovered through hands-on testing before it was fully understood in theory. In the beginning, when the solver tried to make the robot match all six pose requirements, it often picked joint angles that placed the tool in the wrong spot—sometimes off by over 14 centimeters—because it tried to satisfy an orientation the robot simply couldn't achieve. Once the roll constraint was removed, these big errors disappeared, and the robot's position became extremely accurate. This experience shows how important it is not to ask a reduced-DOF robot to do more than it's physically able to do. For anyone using robots with fewer than six degrees of freedom, carefully choosing which constraints to enforce can make or break the system's performance.

4.3.2 Levenberg-Marquardt Numerical Solution

The production solver is `ikine_LM` from the Robotics Toolbox for Python [3], a damped least-squares (Levenberg-Marquardt [14, 15]) iteration with the DOF mask described above and joint limits of $[0, \pi]$ on every joint, matching the physical servo range. The solver is initialized with the current joint configuration $\boldsymbol{q}_{\text{current}}$, which biases convergence toward the nearest reachable solution and avoids large reconfiguration motions between successive targets. An independent implementation using `scipy.optimize.least_squares` with the LM method was written to cross-validate the toolbox results.

Two mask variants are used in deployment. The desktop implementation constrains position and approach direction ($[1, 1, 1, 0, 1, 1]$) as derived above. The

Raspberry Pi deployment uses a position-only mask ($[1, 1, 1, 0, 0, 0]$): at the short working distances of the prototype arm, the orientation reached by the position-only solution proved sufficiently close to perpendicular, while the reduced constraint set converges more robustly across the workspace and rejects fewer targets near the joint limits. Solutions are additionally validated after conversion to servo space — any joint outside the physical $[0ř, 180ř]$ range causes the target to be rejected before motion is commanded.

Solver performance was verified end-to-end against physical measurement: for a target at $[26.9, 0.0, 2.0]$ cm in the base frame, the converged solution (final cost 9.7×10^{-7} after 84 function evaluations) placed the end-effector within $[1.1, -0.2, -0.8]$ mm of the ruler-measured position, with an orientation residual of 2.6×10^{-4} rad. Within the model itself — excluding servo and measurement error — the position residual of the converged solution is on the order of 10^{-10} m, confirming that the physical error budget is dominated by the mechanics and the measurement chain rather than by the solver.

4.4 Coordinate Transformation Chain

4.4.1 From Camera Frame to Robot Base Frame

The full transformation chain from detected marker pose to robot base frame is:

$$\boldsymbol{T}_M^B = \boldsymbol{T}_{EE}^B \cdot \boldsymbol{T}_C^{EE} \cdot \boldsymbol{T}_M^C \tag{4.4}$$

where $\boldsymbol{T}_{EE}^B$ is the current end-effector pose from forward kinematics or the robot controller, $\boldsymbol{T}_C^{EE}$ is the fixed camera extrinsic transform defined in equation (3.5), and $\boldsymbol{T}_M^C$ is the marker pose from `solvePnP`. The `SE3` class from the Robotics Toolbox for Python [3] provides numerically stable composition and inversion. The chain is illustrated graphically in Figure 3.6.

It's worth highlighting that the way the camera's position relative to the robot's end-effector was measured introduces some uncertainty into the system. Instead of using a formal hand-eye calibration method—like the well-known Tsai-Lenz approach—the team simply measured the camera's mounting location by hand with a ruler. This practical shortcut means any small mistake in measurement (for example, being off by 2 millimeters) will shift every calculated target by exactly that amount. While the robot's two-stage approach (where it checks its position up close before final contact) helps to reduce the impact of this kind of error, the issue isn't fully eliminated. As a result, this method is acknowledged as a limitation in the project, and performing a more precise calibration is suggested as an important next step for future improvements.

4.4.2 Perpendicular Approach Target Computation

For vibration experiments, it's crucial that the robot's tool touches the marker exactly straight on—meaning it needs to approach along a line that's perpendicular to the surface. To achieve this, the robot's orientation is set using the marker's own reference frame: the tool's z-axis is pointed directly opposite the marker's normal (the line sticking straight out from the surface). Next, an x-axis is chosen within the plane of the marker, and a third axis is worked out using the cross product. To keep everything mathematically precise and avoid tiny calculation errors from piling up, the Gram-Schmidt process is used to keep all the axes perfectly at right angles. The initial and final approach points for the tool are then calculated by moving out along the surface normal, ensuring contact happens at the right spot and angle.

$$\boldsymbol{T}_{\text{far}} = \boldsymbol{T}_{M}^{B} \cdot \boldsymbol{T}_{z}(-150\,\text{mm}) \tag{4.5}$$

$$\boldsymbol{T}_{\text{near}} = \boldsymbol{T}_{M,\text{refined}}^{B} \cdot \boldsymbol{T}_{z}(-50\,\text{mm}) \tag{4.6}$$

where $\boldsymbol{T}_{M,\text{refined}}^{B}$ denotes the marker pose re-estimated at close range after the joint-space motion to $\boldsymbol{T}_{\text{far}}$.

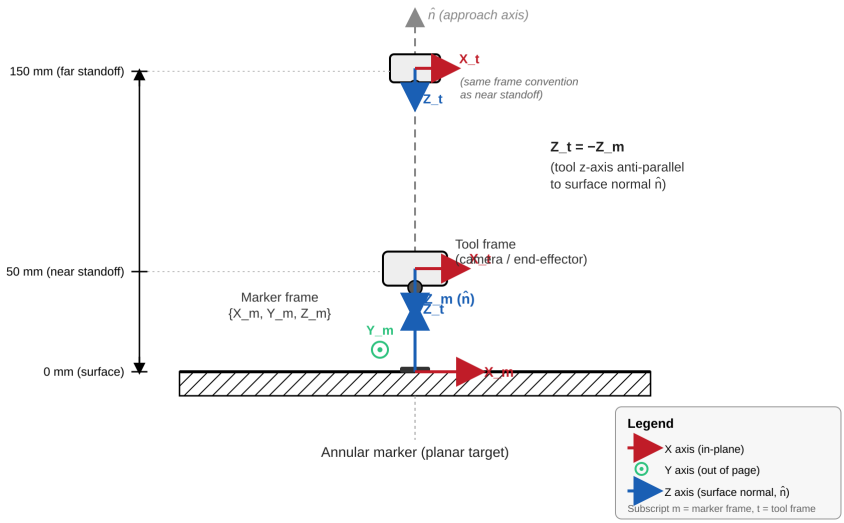


Figure 4.3: Perpendicular approach geometry: the tool z-axis is aligned anti-parallel to the marker surface normal, and the far and near standoff targets are offset along the normal direction.

This chapter has derived the complete kinematic model and coordinate transformation chain, validating both against physical measurements. The following chapter presents the implementation and experimental results obtained in the two simulation environments — RoboDK and the ROS 2 application in Gazebo.

Chapter 5

Simulation Implementation and Experimental Results

In this chapter, both simulation setups and their experimental results are introduced in a clear, step-by-step way. First, the chapter walks through the RoboDK environment—explaining how the virtual scene was built to mirror the real experimental setup, how the camera parameters in simulation were made to closely match the physical camera, how the system assigns unique IDs to markers when several are seen at once, and how the two-step movement method is evaluated for accuracy at each marker site. The next part focuses on the ROS 2 application, where the same perception and detection techniques are used to control a UR3e robot in the Gazebo simulator, with MoveIt2 handling robot movements so the robot can carry out a carefully supervised, perpendicular striking motion for every marker it detects.

5.1 RoboDK Simulation Environment

The RoboDK scene was set up to match the physical arrangement: a UR3e on a fixed base, a camera on the TCP end-effector, and a flat test plate with five concentric circle markers. The simulated camera was given the same intrinsic parameters as the real one — $f_x = f_y = 895.91$ pixels with a principal point at $[320, 240]$ for 640×480 resolution.

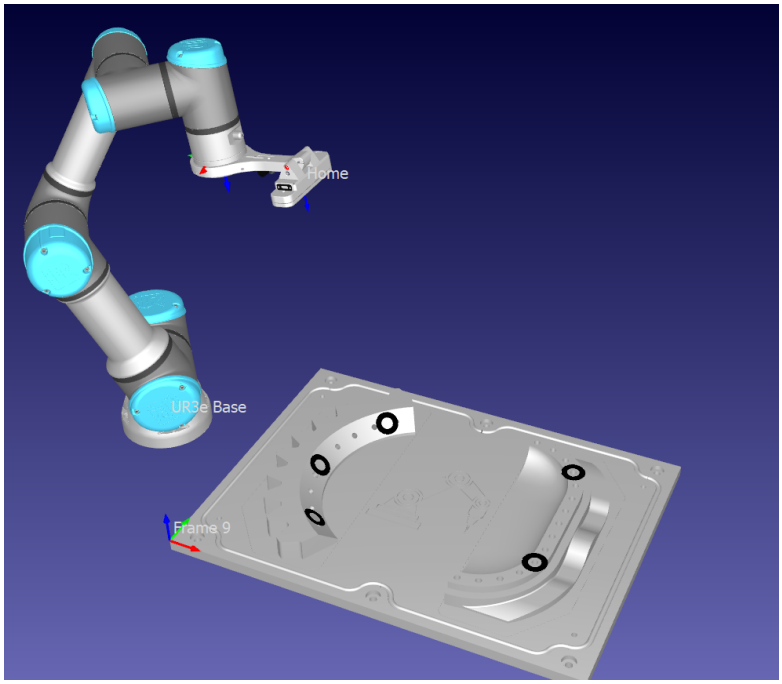


Figure 5.1: RoboDK simulation environment: UR3e robot, camera tool, test plate with five annulus marker targets.

5.2 Marker Detection in Simulation

The detection pipeline was integrated into the RoboDK scripting environment through the Robolink Python application programming interface. At each detection call, the robot TCP pose provides T_{EE}^{B}, the base-frame transformation for the detected marker poses. The detection function executes the full annulus detection pipeline — frame capture, edge detection, contour hierarchy analysis, ellipse fitting, Perspective-n-Point estimation — and returns all detected marker poses in the RoboDK XYZ-RPY base-frame representation, as illustrated in Figure 5.2.

5.3 Multi-Marker Target Identification

When several markers show up in the robot's field of view at the same time, the system uses the `assign_ids()` function to give each marker a unique number. It does this by sorting the markers based on their y-coordinates in the robot's base frame: the marker that's lowest along the y-axis gets a 0, and the one that's highest gets the largest number. Since the system uses the markers' real-world positions (not just where they appear in the camera image), the assigned numbers stay consistent, even if the markers shift around in the camera's view due to changes in perspective.

The stability of this assignment was verified experimentally (experiment E6): with three markers simultaneously visible, the assignment was recorded over 20 consecutive detection frames without moving the robot. The identifiers remained correct and stable in all 20 frames (100%, against a target of 95%). This result

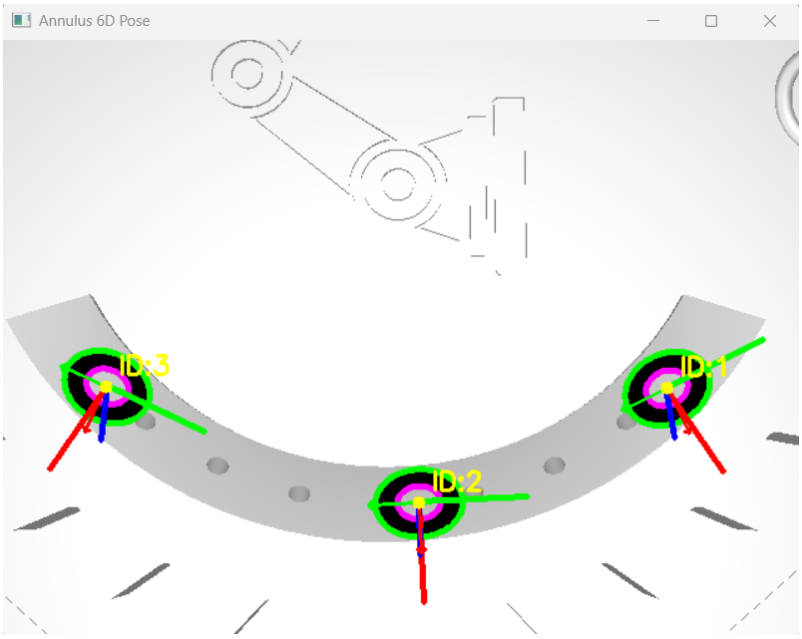


Figure 5.2: Three concentric circle markers simultaneously detected in the RoboDK camera view, with ID labels and 6D coordinate axis overlays.

is expected to be near-perfect in simulation, where the base-frame y-coordinates carry no sensor noise; its value is confirmatory — it establishes that the sorting logic itself introduces no instability, so any ID swaps observed on real hardware could be attributed to pose estimation noise rather than to the assignment strategy.

5.4 Two-Stage Motion Strategy

For each marker, the robot follows a three-step process, which is shown in Figure 5.3. First, it moves to a position 150 mm above the marker (called $\boldsymbol{T}_{\text{far}}$) using joint-space motion. This type of movement is chosen because it helps avoid tricky positions, or singularities, that could arise if the robot tried to move in a straight line from its starting point. After reaching this higher position, the robot runs the detection routine again—now that it's closer, the marker is larger and clearer in the camera image, reducing distortion and allowing the system to get a more accurate fix on the marker's location. With this updated marker position ($\boldsymbol{T}^{B}_{M,\text{refined}}$), the robot then moves in a straight line to a point just 50 mm above the marker, positioning itself perpendicularly to prepare for contact. After finishing the task at that marker, the robot pulls back to its earlier standoff position before moving on to the next marker in the sequence.

Figure 5.4 shows what the re-detection step gains in image terms: at the far observation pose, the marker occupies a small region of the frame, while from the 150 mm standoff it fills a substantial portion of the image, giving the ellipse fitter an order of magnitude more contour pixels to work with. Figure 5.5 shows the end-effector at the final perpendicular approach position.

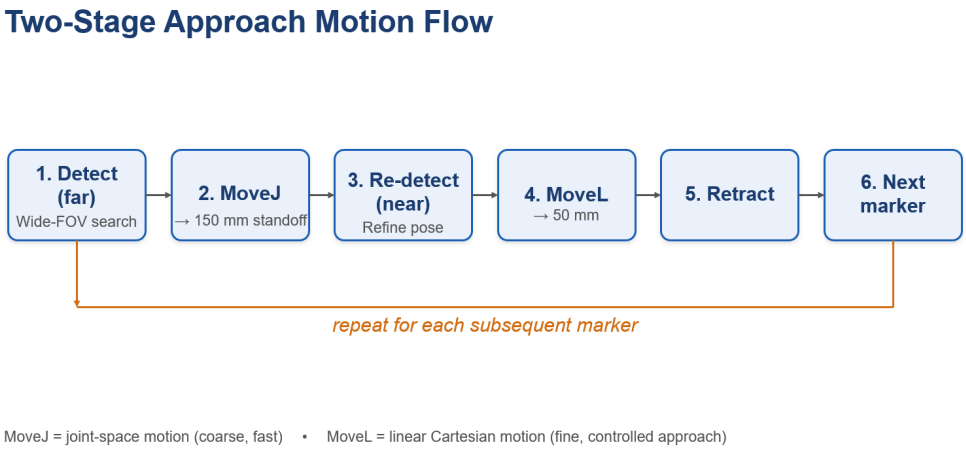


Figure 5.3: Two-stage motion strategy: far detection and joint-space approach, close-range re-detection, and linear perpendicular refinement, repeated for each marker in the identified sequence.

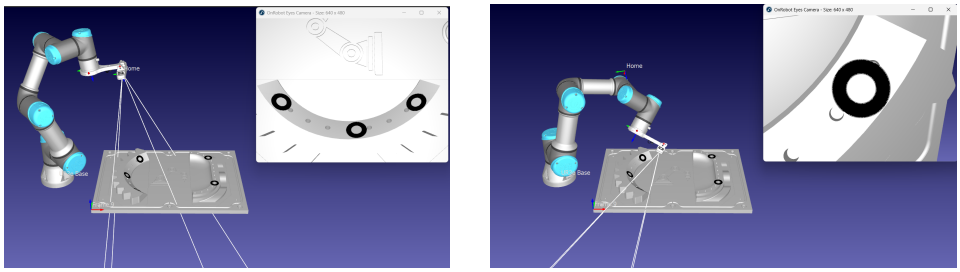


Figure 5.4: Simulated camera view of the same marker from the far observation pose (left) and from the 150 mm standoff after the first approach motion (right). The close view provides substantially more contour pixels for the ellipse fit.

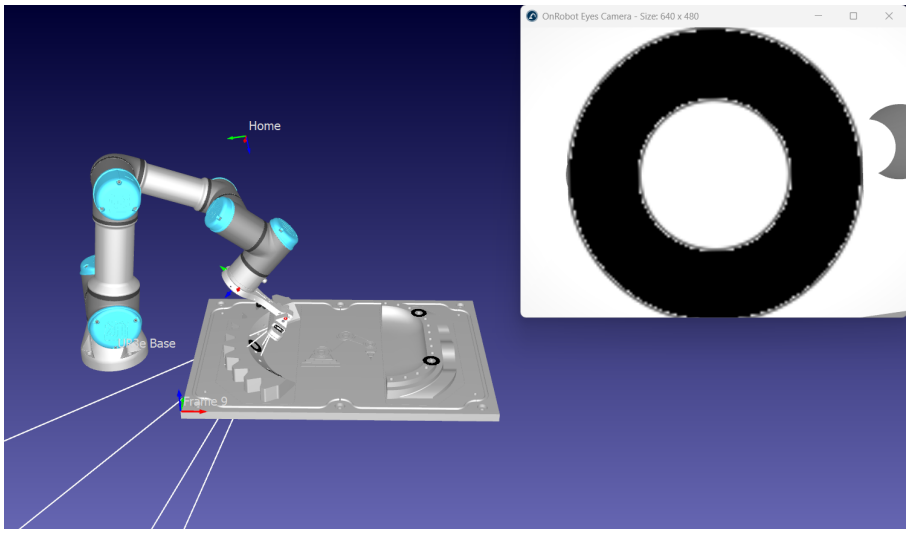


Figure 5.5: UR3e end-effector at the final perpendicular approach position for one marker target.

5.5 RoboDK Experiment Results

5.5.1 Experimental Setup

The positioning experiment (E5) used three markers with known ground-truth positions in the RoboDK scene. For each marker, the full approach was executed under two conditions: a single-stage condition, in which the robot moved directly to the target computed from the far detection, and a two-stage condition as in Section 5.4. Each condition was repeated 5 times per marker. After each completed approach, the final TCP position reported by RoboDK was compared with the ground-truth target, yielding the Euclidean positioning error. The final data set contains 11 completed single-stage approaches and 13 completed two-stage approaches out of 15 attempts each; the missing cases are trials in which far detection failed for that marker, thereby terminating the approach under both conditions.

5.5.2 Positioning Accuracy and Strategy Comparison

Table 5.1 gives a summary of the results, and Figure 5.6 illustrates how the errors are distributed. No matter which motion strategy was used, the robot's tool tip ended up within about 2 millimeters of the target in every successful test. This is comfortably better than the 3 mm accuracy needed for the task, which makes sense since the simulation assumes the robot's movements are perfect—so any error comes only from the vision system's detection process.

Table 5.1: E5 — positioning error by motion strategy and per marker (5 trials per marker per condition).

Condition	Completed	Mean error (mm)	Std. dev. (mm)	Max error (mm)
Single-stage	11/15	1.03	0.44	1.75
Two-stage	13/15	1.75	0.33	2.02
Two-stage condition, per marker:				
Marker 0	5/5	1.84	0.03	1.87
Marker 1	4/5	1.96	0.05	2.02
Marker 2	4/5	1.42	0.42	1.86

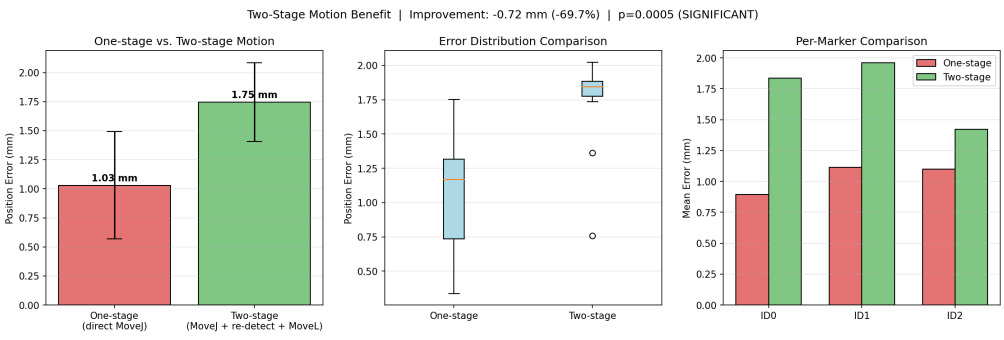


Figure 5.6: E5 results: final positioning error for the single-stage and two-stage motion strategies across all completed trials.

Comparing the two movement strategies gave a surprising result: in this simulated setup, the single-stage approach actually performed a bit better (with a mean error of 1.03 mm, compared to 1.75 mm for the two-stage approach; Welch's t-test, $p = 0.0005$). Also, the close-up re-detections were less accurate on average than the detections made from farther away (1.87 mm versus 1.19 mm). This makes sense when you consider what the re-detection step is really for. In the real world, using a single camera often introduces a consistent depth error due to imperfect calibration, and the two-stage strategy is meant to correct for this by checking again at close range. But in the simulation, the camera is modeled perfectly with no such bias, so the re-detection just adds another random bit of detection error. In fact, at closer distances, the marker occupies more of the camera's view, and its contours are more affected by blurring and thresholding, which could explain the slightly higher error.

There are two main takeaways from these results. First, the two-stage method made the system more reliable, with more completed approaches (13 out of 15) than the single-stage strategy (11 out of 15), thanks to its ability to recover from weak detections as the robot gets closer. Second, the extra accuracy from re-detection only matters if there is a consistent, long-range error to begin with—which happens in real-world situations but doesn't show up in the perfect simulation. Instead of showing that the two-stage approach is unnecessary, these findings help clarify its true value: it's designed to correct for bias, not just to reduce random errors. The benefit of this method will be more apparent when it's tested with the real robot and camera, and that is highlighted as a goal for future experiments.

5.6 ROS 2 Application: Vision-Guided Striking in Gazebo

The final implementation stage ports the pipeline to ROS 2, replacing RoboDK's proprietary scripting with the middleware stack used on real UR deployments. Beyond the change in platform, this stage extends the system's functionality: the robot no longer only approaches each marker but also executes a complete supervised striking cycle — approach, perpendicular contact, retract, and return — under operator control. The application's architecture was described in Chapter 2; this section describes its behavior and the design decisions that emerged during integration.

5.6.1 Workspace and Launch Architecture

The application lives in a single colcon workspace containing two custom packages alongside the standard Universal Robots description and simulation packages. The `ur3e_camera` package contains the robot model. It uses an XACRO description based on the official UR3e definition and attaches two components to the tool flange: the simulated Intel RealSense D405 and the hammer tool. The hammer tool's geometry ends at the virtual `hammer_tip` frame, which serves as the planning end-effector. The `my_robsim` package includes everything needed to run the system, such as the

Gazebo world with the five-marker target plate, the two application nodes, the RViz configuration, and the launch files.

A single launch file starts the whole system in a specific order, which is important in practice. The controller spawners must wait until Gazebo physics and the `robot_state_publisher` are running, or else the joint controllers will try to configure a robot that does not exist yet. The launch process begins by starting Ignition Gazebo with the world file, then spawning the robot from the XACRO description. Next, it starts the `ros_gz_bridge` to map the camera, clock, and joint topics into ROS 2. After that, it activates the `joint_state_broadcaster` and `joint_trajectory_controller` through the controller manager. Finally, it starts the MoveIt2 `move_group` node and RViz. The detection node and the motion controller are launched last, after the topics they need are available. All nodes use simulation time so that TF lookups, trajectory time-stamping, and the planner all use the Gazebo clock.

The TF tree connects all parts of the system. The static chain includes the world—to—robot base, the robot link chain managed by `robot_state_publisher` from the joint states, and the fixed transforms from the flange to the camera and from the flange to `hammer_tip` defined in the XACRO. At runtime, the detection node adds dynamic frames, `annulus_1` to `annulus_5`, under `camera_optical_frame`. Since all frames are part of one consistent tree, the motion controller can use TF2 to get any marker pose directly in the planning frame. The same lookup system machinery that MoveIt2 uses ensures that perception and planning agree on the geometry. Figure 5.7 shows the resulting tree.

Figure 5.8 illustrates what the detector works with at the two ends of the striking sequence, using binarised debug frames recorded during the E5 experiment: from the observation pose the five markers appear as small nested blobs, while from the standoff pose a single marker dominates the frame. The same adaptive-threshold front end handles both regimes; only the geometric filters see different contour sizes.

The complete cycle for one target is shown in Figure 5.9 as it appears in the simulation.

The smoothness of the traces — no discontinuities or oscillations at the phase boundaries — reflects the time-parameterized trajectories produced by the planner and confirms that the phase transitions hand over cleanly.

5.6.2 Simulation Environment and Perception

The UR3e is simulated in Ignition Gazebo together with a table-mounted target plate carrying five concentric circle markers. The simulated Intel RealSense D405 publishes synchronized color and depth images, which the detection node processes with the adaptive-thresholding variant of the annulus pipeline from Chapter 3. Detected marker poses are published continuously and broadcast as TF frames, so the entire perception state is visible in RViz at all times — each marker appears as a

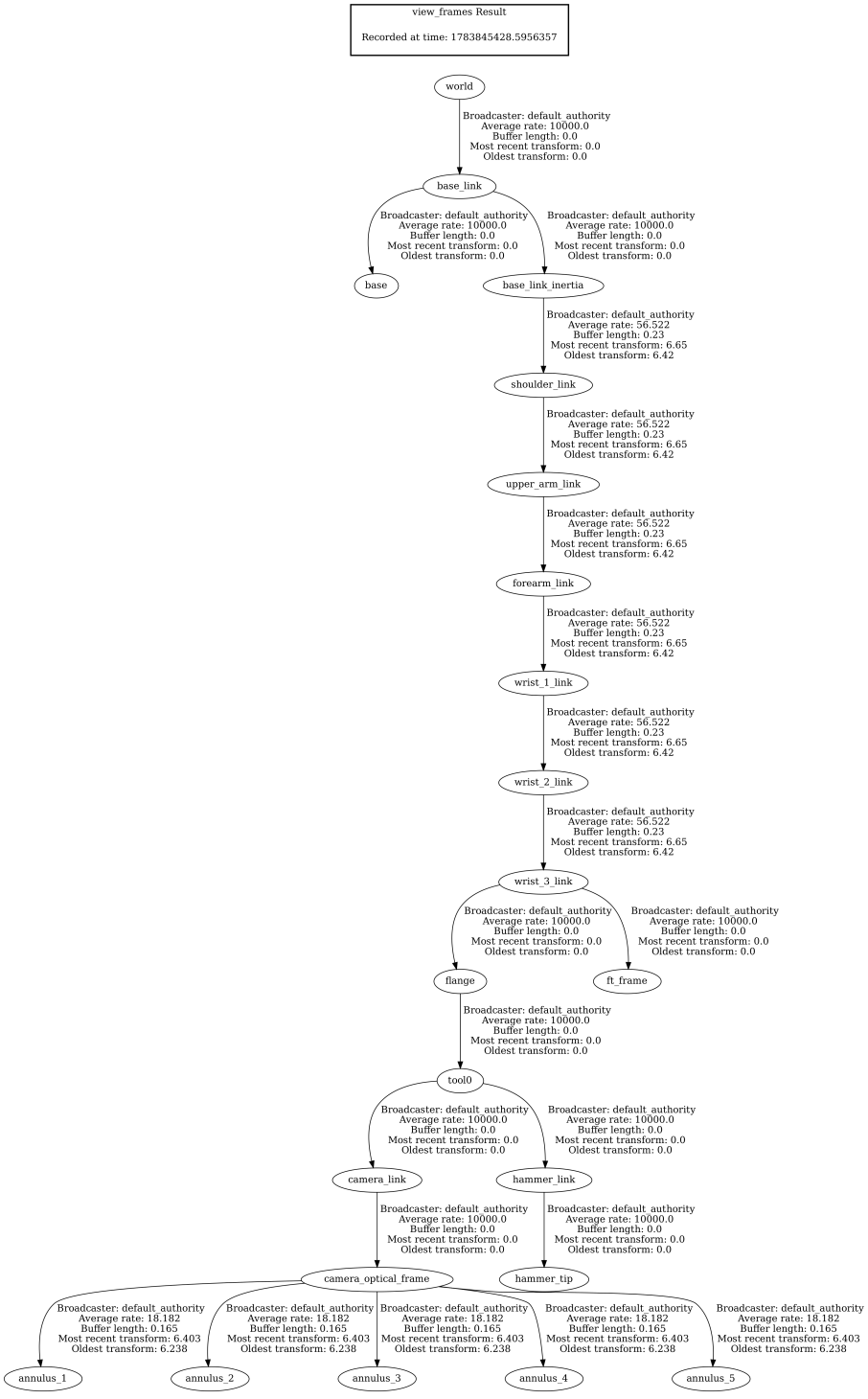
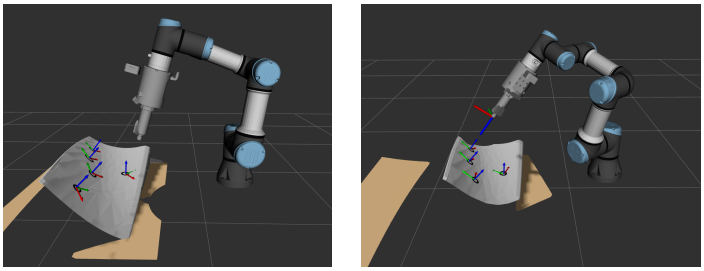


Figure 5.7: TF tree of the running application. The robot link chain is maintained from joint states; the camera and hammer frames are static; the annulus frames are broadcast by the detection node and connect the perception output to the planning frame through a single consistent tree.

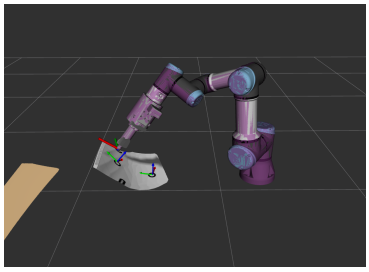


(a) binarised, observation pose **(b)** binarised, 150 mm standoff

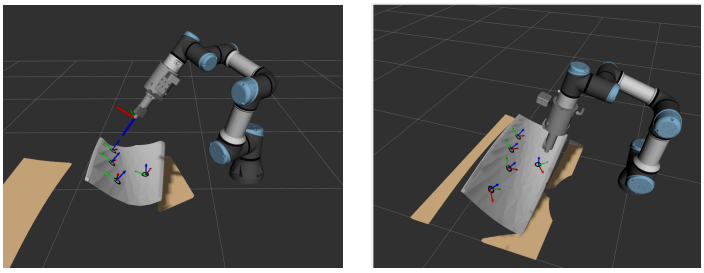
Figure 5.8: Binarized detector input at the two working distances of the motion sequence (E5 debug frames). The nested ring structure survives thresholding in both regimes; the contour pixel count available to the ellipse fit differs by roughly an order of magnitude.



(a) home, targets detected **(b)** perpendicular standoff



(c) strike contact



(d) linear retract **(e)** return to home

Figure 5.9: One complete striking cycle in the Gazebo simulation: (a) home configuration with all five markers detected, (b) joint-space approach and perpendicular alignment at the 15 cm standoff, (c) strike contact, (d) linear retract, (e) return to home before the next target in other view.

labeled frame (`annulus_1` to `annulus_5`) with RGB axis markers at its estimated 6D pose.

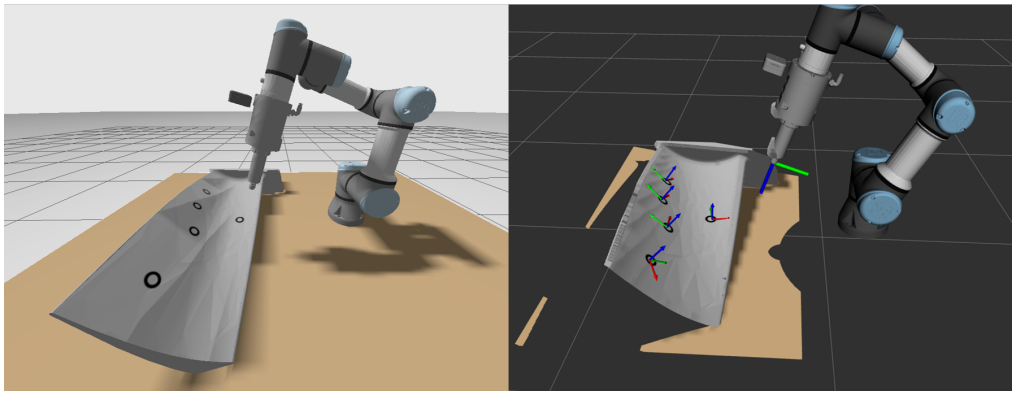


Figure 5.10: Stage 3 perception: the Gazebo simulation (left) and the corresponding RViz view (right) showing the five detected markers as TF frames with axis overlays, together with the debug image of the detector.

Reaching reliable detection of all five markers required tuning that is worth recording. With the initial contour area threshold, the marker closest to the image periphery was intermittently rejected — as visible in the debug logging, it appeared in every frame but failed the area test. Lowering the minimum contour area (from 1700 to 900 square pixels) admitted it, and a compensating tightening of the geometric ratio check prevented the lower threshold from admitting background clutter. This tuning cycle was driven entirely by the per-candidate rejection logging built into the node, which records why each rejected contour failed; without that instrumentation, diagnosing an intermittently missing marker among five would have been substantially harder.

5.6.3 Striking Sequence and Motion Control

For each confirmed target, the motion controller executes a four-phase sequence, shown in Figure 5.11. First, it computes the perpendicular approach pose at a 15 cm standoff along the marker normal and moves there with a joint-space motion. The standoff value is a workspace compromise: the hammer tip sits 216 mm ahead of the wrist flange, so at a 15 cm standoff the wrist is already about 37 cm behind the marker surface — close to the boundary of the UR3e's roughly 50 cm reach. Larger standoffs, tried first, were consistently unreachable. Second, the controller advances linearly along the normal to a distance of 5 mm from the surface, thereby executing the strike contact. Third, it retracts linearly to the standoff distance, and fourth, it returns to the home configuration before the next target is selected.

Two implementation decisions proved essential for reliability. The first concerns inverse kinematics. Sending Cartesian pose goals directly to the planner failed intermittently with no-solution errors because the underlying KDL solver is a local method whose success strongly depends on the seed configuration. The controller therefore solves IK explicitly before planning, trying a sequence of seeds — the current

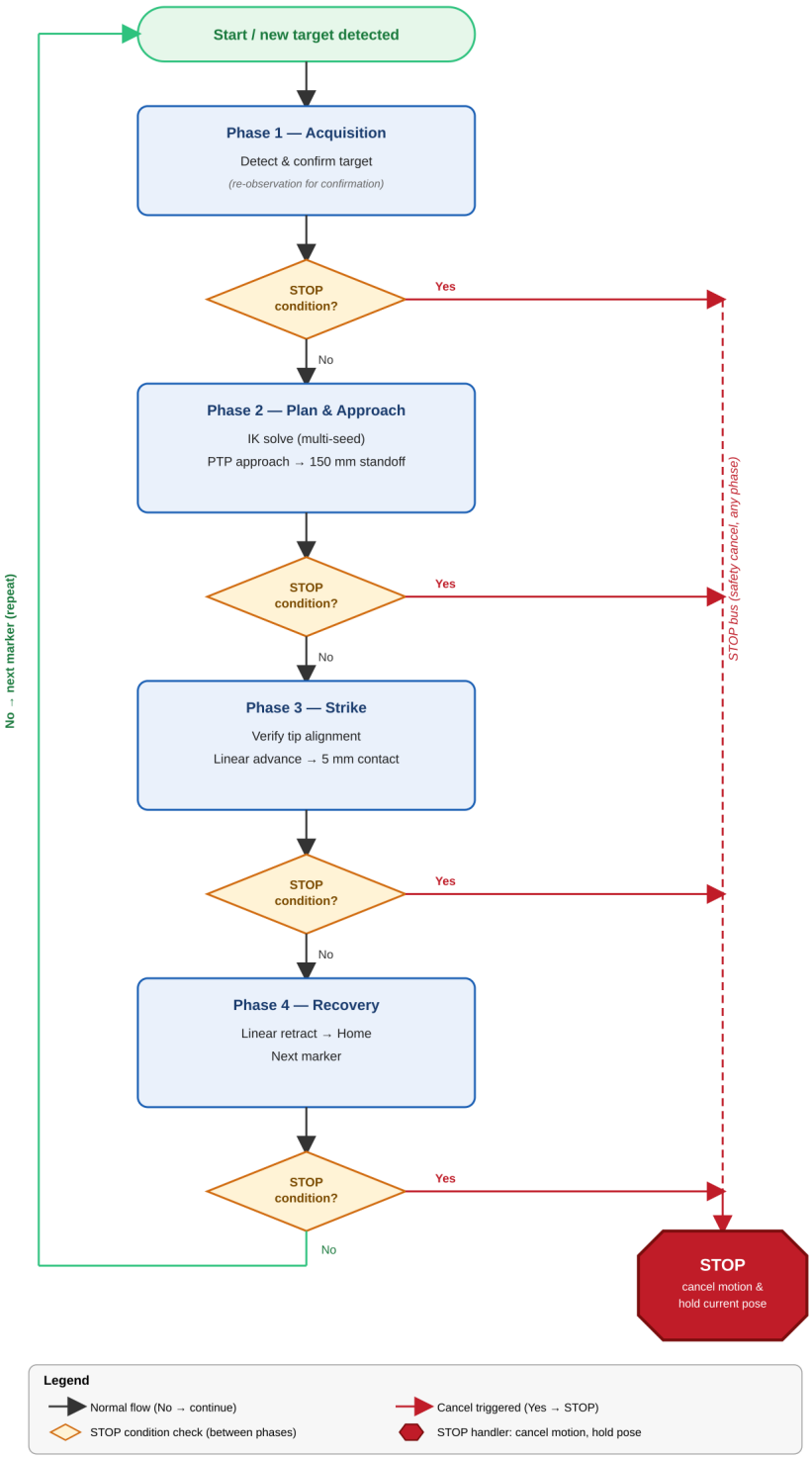


Figure 5.11: The four-phase striking sequence executed for each target, with the stop condition evaluated between all phases.

configuration, the home configuration, and six randomized perturbations around home drawn from a fixed-seed generator for repeatability — and then sends the resulting joint configuration as a joint-space goal, which cannot fail for reachability reasons at the planning stage. The second concerns orientation correctness: after the approach motion completes, the controller explicitly verifies tip alignment by computing the scalar product between the commanded tool z-axis and the marker z-axis, which must equal -1 for a perpendicular approach. This runtime check converts a silent orientation error — the kind produced by a frame convention mistake — into an immediately visible diagnostic.

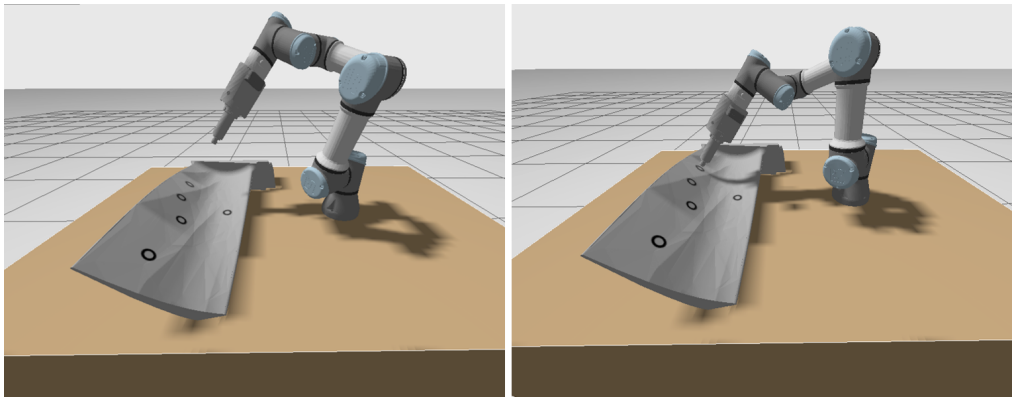


Figure 5.12: The hammer tool at the perpendicular standoff pose (left) and at strike contact with the marker surface (right) in the Gazebo simulation.

5.6.4 Operator Interface and Safety Behavior

The application is supervised rather than fully autonomous by design: a graphical panel presents *Go Home*, *Start Sequence*, *Confirm*, and *STOP* controls, and the operator confirms the detected target set before any motion begins. The STOP command acts at two levels — it sets a flag that is checked between phases of the striking sequence, and it also cancels the trajectory currently executing via the MoveIt2 interface, so the robot halts mid-motion rather than completing the current phase. The Go Home command is available independently of the sequence state, always through a planned, collision-checked motion. This two-level stop design reflects a lesson from testing: a flag checked only between phases leaves the robot committed to a potentially long motion, which is unacceptable when the tool is centimeters from the workpiece.

5.6.5 Qualitative Validation

The complete application was validated in repeated end-to-end runs: all five markers detected simultaneously and continuously, the operator confirming the target set, and the robot executing the full approach–strike–retract–home cycle for each marker in sequence, with the commanded tip alignment verified at every standoff pose ($\hat{\boldsymbol{z}}_{\text{tip}} \cdot \hat{\boldsymbol{z}}_{\text{marker}} = -1.000$ by construction, confirmed by the runtime check after

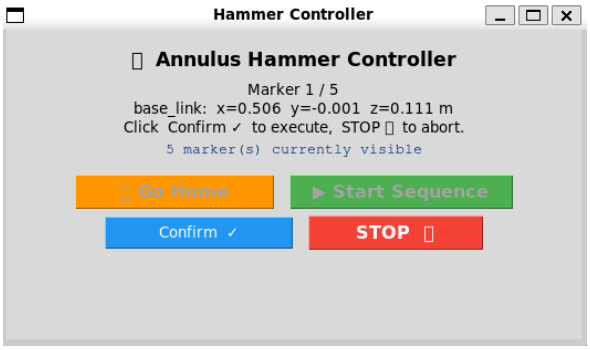


Figure 5.13: Operator panel of the striking application, showing the sequence controls and the live status display.

each approach). The STOP behavior was exercised deliberately during motion in all phases. In contrast to the RoboDK experiments, this validation is qualitative — systematic quantitative logging of final tip positions in Gazebo was not performed within the scope of this work and is identified as a straightforward extension. The demonstration nevertheless establishes the central integration claim of this stage: the same perception principle validated quantitatively in RoboDK drives a standard industrial middleware stack all the way through to physical-contact behavior, with no proprietary components in the loop.

This chapter has presented the two simulation implementations and their experimental results. The following chapter turns to the real world, reporting detection tests performed on the physical aerospace component with the real camera.

Chapter 6

Real-World Vision Tests on an Aerospace Component

This chapter reports the experimental validation of the concentric circle detection pipeline on a physical aerospace component under real laboratory conditions. The test setup is described, the detection results are presented, and the key differences between the simulation and the real-world conditions are analyzed. The chapter closes with the implications of these tests for hardware deployment.

6.1 Test Setup

To validate the detection pipeline under real optical conditions — surface reflectance variation, depth of field, and ambient laboratory lighting — the concentric circle detector was tested on the physical aerospace component described in Section 2.2: a curved metallic vane segment carrying a regular grid of circular through-holes, each surrounded by a bright-rimmed eyelet. A standard RGB camera mounted on an articulated arm observed the component from a fixed position, with the optical axis oriented toward the hole pattern. The detection software ran live on a laptop, displaying the camera stream with the detected features, their identifiers, and the estimated coordinate axes overlaid in real time.

An aspect of this test deserves emphasis. No adhesive markers were placed on the component: the detection targets were the component's *own* circular features. Each rimmed hole presents exactly the geometry the detector was designed for — a bright ring around a dark center produces a nested pair of elliptical contours under the hierarchy analysis of Chapter 3, in the same way a printed concentric circle target does. The test therefore probes a stronger capability than the sticker-based workflow characterized in the earlier experiments: fully marker-free targeting based on existing structural features, which requires no modification of the workpiece at all.

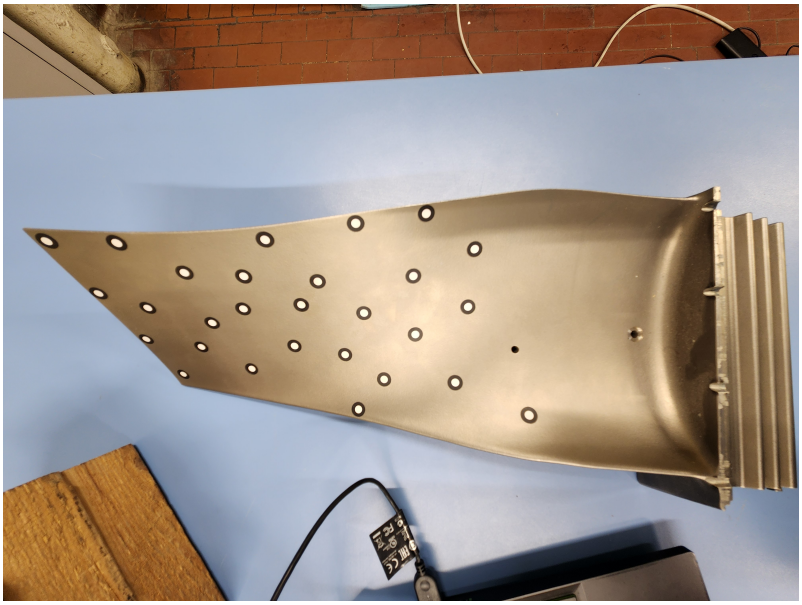


Figure 6.1: Experimental setup: the aerospace component on the laboratory bench, with the camera on an articulated mount and the detection software running on the laptop.

6.2 Detection Results

Figures 6.2 and 6.3 show the detection system operating on the real component. The detector identified the circular features across the visible hole grid simultaneously — the software reported 54 tracked marker detections in the frame shown — with each detection carrying a persistent identifier and a 6D coordinate axis overlay. The processing rate was approximately 9 frames per second on the laptop CPU, sufficient for the detect–move–redetect workflow of the two-stage strategy, which requires detection only at discrete waypoints rather than continuous high-rate tracking.

The axis overlays show estimated surface normals pointing consistently out of the component surface across the curved hole field, and the estimated orientations follow the visible curvature of the vane: holes near the strongly curved region show visibly tilted axes relative to those on the flatter central region. This qualitative behavior is exactly what the ellipse-tilt estimator of Chapter 3 should produce, and it could not have been verified in the flat-plate simulation scenes. A quantitative comparison against externally measured hole positions was not performed — the laboratory session did not include a metrology reference — so these results validate detection capability and the plausibility of orientation rather than absolute accuracy.

6.3 Comparison with Simulation Conditions

Moving from simulation to the real component introduced several practical complications. The laboratory lighting is directional rather than uniform, which creates specular highlights on the polished eyelet rims and the metallic body; these

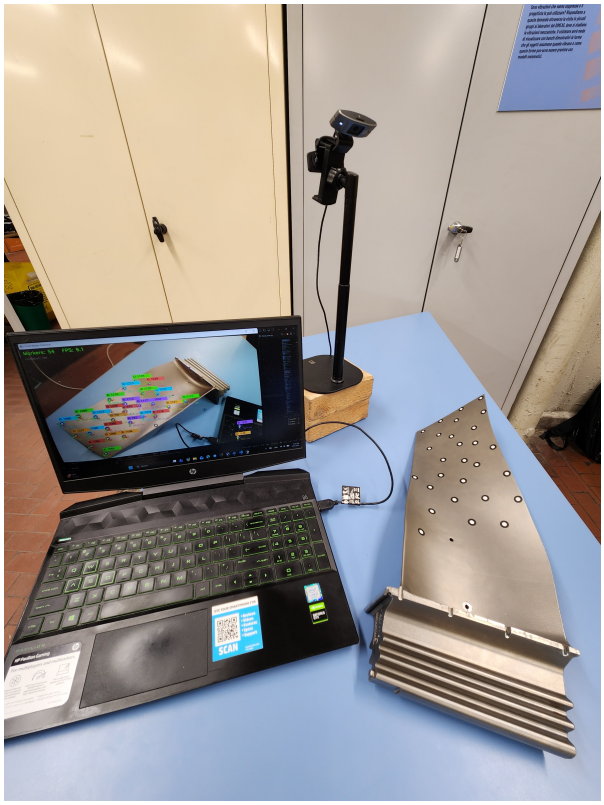


Figure 6.2: Detection running live on the real component: the software reports 54 tracked circular features at 9.1 frames per second, each with an identifier and an estimated 6D pose axes.

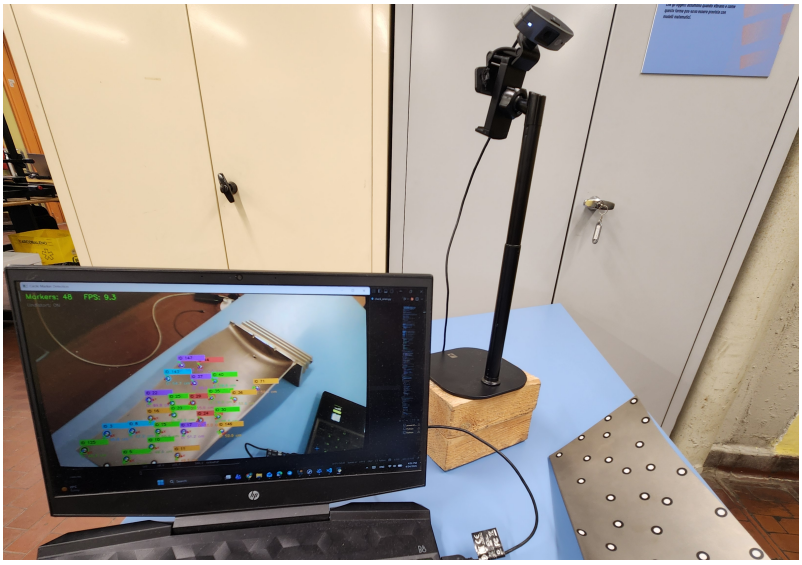


Figure 6.3: Close view of the detection output: coloured identifier labels and coordinate axis overlays on the component's hole pattern, following the local surface orientation of the curved vane.

appear as intensity discontinuities that fragment contours in the binarised image, and the threshold settings required more tuning than in the simulated scenes. The real features also differ geometrically from the printed targets: the hole rims are three-dimensional (raised eyelets rather than flat printed rings), so their projected contour pair deviates slightly from the ideal concentric ellipse model, and the hole diameter is smaller than that of the sticker targets, reducing the number of contour pixels available to the ellipse fit at a given distance. Machining marks and surface scratches on the component body generate additional candidate contours; in practice, the concentricity condition and the diameter ratio filter remove these false candidates, and no spurious detections are visible in the recorded frames.

Two conclusions follow for hardware deployment. First, the detection principle transfers: the pipeline developed against printed targets detects real structural features without algorithmic modification, at a rate compatible with the two-stage motion workflow. Second, the accuracy of pose estimation on real features remains to be quantified — the physical hole dimensions must be measured and entered as the PnP object model, and a metrology-grade reference (or the robot itself, once deployed) is needed to establish ground truth. This quantification is the natural first experiment of the hardware deployment phase discussed in Chapter 7.

This chapter has confirmed that the detection pipeline operates under real optical conditions on a real aerospace component, using the component's own circular features as targets. The final chapter summarises the thesis results against its objectives, discusses the system limitations, and outlines the path to deployment on real robot hardware.

Chapter 7

Conclusions and Future Work

This chapter interprets the experimental results obtained across all development stages, identifies the principal system limitations, presents the remaining path to deployment on real UR3e hardware, and summarises the outcomes against each research objective before stating the conclusions of the thesis.

7.1 Interpretation of Results

The experimental results of Chapters 3, 5, and 6 together demonstrate that the vision-guided positioning system achieves its core design objectives within the conditions tested. On the vision side, the calibrated single-camera pipeline keeps distance errors below 3% across the 143-650 mm working range, and the ellipse-tilt orientation estimator remains within 0.8° of the ground truth for surface tilts between 5° and 40° — comfortably within the 5° perpendicularity budget adopted for the excitation task. On the motion side, the RoboDK experiments place the end-effector within about 2 mm of the target in every completed approach, against a standoff tolerance of ±2-5 mm, which is typically acceptable for repeatable contact excitation. The ROS 2 application then demonstrates that the same perception principle drives a full industrial middleware stack — Gazebo, MoveIt2, TF2 — through complete supervised striking cycles on five simultaneously tracked markers.

The comparison between the single-stage and two-stage motion strategies produced the most instructive result of the experimental campaign. In the ideal simulation, the refinement step did not improve accuracy — it slightly degraded it — while improving completion reliability (13/15 approaches versus 11/15). As discussed in Section 5.5, this is not a defect of the strategy but a consequence of what re-detection can correct: it cancels systematic bias, and the ideal simulated camera has none, whereas the real camera exhibits exactly the systematic depth bias (Section 3.5.1) the strategy was designed against. The result sharpens the design rationale for the two-stage approach and defines the experiment to be run on real hardware: the same comparison, in which the bias-canceling benefit is expected to appear.

7.2 System Limitations

7.2.1 Single-Camera Orientation Ambiguity

The ellipse-tilt method for orientation estimation (Section 3.4.4) has an inherent sign ambiguity: a marker tilted at $+\theta$ and at $-\theta$ produces an identical ellipse projection. The system resolves this by always selecting the sign convention consistent with a camera approaching from the front of the component. For deployment in environments where the component could be approached from multiple directions, this limitation would require either supplementation with a depth camera or a multi-view disambiguation step. The related weakness at near-zero tilt — where the axis-ratio method loses sensitivity and reports a 3.7° tilt for a head-on marker in experiment E3 — is benign for the present task but should be kept in mind if tighter perpendicularity budgets are ever required.

7.2.2 Hand-Eye Calibration Approximation

The camera extrinsic transform $\boldsymbol{T}_C^{EE}$ was determined by direct physical measurement with an estimated uncertainty of ±2 mm. A systematic error in this transform propagates rigidly as an offset in all computed marker positions in the base frame. Formal hand-eye calibration using the Tsai-Lenz method [11] should be performed before deploying the UR3e in the field to eliminate this source of systematic error.

7.2.3 Simulation-to-Real Transfer Gap

The quantitative motion results were obtained in RoboDK, where the robot kinematics are exact and no mechanical imperfections exist, and the ROS 2 striking application was validated qualitatively in Gazebo rather than through systematic position logging. Real hardware will introduce joint-level errors, link flexure, controller tracking behavior, and real camera optics simultaneously. The real-component vision tests of Chapter 6 close part of this gap — they establish that detection works under real optical conditions — but they do not measure the end-to-end positioning accuracy of the physical system. That measurement is the defining experiment of the next project phase.

7.2.4 Excitation Not Instrumented

The system positions and drives the excitation tool to surface contact, but no force sensing or structural response measurement was included in the scope of this work. The striking behavior is therefore validated geometrically (correct location, perpendicular orientation, controlled contact depth) rather than functionally as a vibration test. Completing the measurement chain is a natural extension rather than a redesign, since the positioning layer is independent of the instrumentation.

With the ROS 2 application complete and validated in simulation, the remaining work toward physical deployment is focused and concrete rather than a

migration — the architecture, nodes, and planning stack are the ones used on real UR robots, and no proprietary components remain in the loop. Table 7.1 summarises the remaining steps. The essential ones are connecting the `ur_ros2_driver` to the physical robot in place of the Gazebo simulation, replacing the simulated D405 topics with the RealSense driver, performing formal hand-eye calibration of the physically mounted camera, and repeating the E5 strategy comparison with quantitative logging — first in Gazebo, then on hardware.

Table 7.1: Remaining steps toward real UR3e deployment.

Step	Task	Technical requirement
1	Quantitative logging in Gazebo	Record final tip pose vs. marker ground truth over repeated striking runs; establish the simulation baseline
2	Connect UR3e hardware driver	`ur_ros2_driver` with correct IP and robot description; verify home motion and controller activation
3	Integrate physical RealSense D405	`realsense2_camera` node replacing simulated topics; verify depth output over the 70–500 mm range
4	Hand-eye calibration	Tsai-Lenz procedure with the physically mounted camera; replace the measured $\boldsymbol{T}_C^{EE}$
5	End-to-end test on one target	Full detect $\rightarrow$ transform $\rightarrow$ plan $\rightarrow$ strike cycle on hardware; measure error against the simulation baseline
6	Multi-target sequential test	Full 5-marker striking sequence; repeat the two-stage vs. single-stage comparison where the bias-cancelling benefit is expected

7.3 Summary of Achieved Objectives

This thesis has presented the development, implementation, and validation of a vision-guided robotic positioning system for the autonomous vibration inspection of aerospace components. Against each of the six research objectives stated in Section 1.2, the following results were achieved.

Objective 1 was met. DH parameters for the five-DOF arm were derived and physically validated, including the identification of joint-3 mechanical inversion that required a correction to the servo command mapping. The FK model matches direct measurements within about 1 mm at the home position. The masked Levenberg-Marquardt solver also achieves sub-millimeter accuracy for reachable targets.

Objective 2 was achieved. The transformation chain $\boldsymbol{T}_M^B = \boldsymbol{T}_{EE}^B \cdot \boldsymbol{T}_C^{EE} \cdot \boldsymbol{T}_M^C$ was set up and tested in three stages: first with the Robotics Toolbox, then with the RoboDK API, and finally with the ROS 2 TF2 framework. The geometry stayed consistent in all three cases.

Objective 3 was achieved. The concentric circle pipeline finds coded targets

and determines their 6D pose using ellipse fitting, `solvePnP`, and the ellipse-tilt orientation estimator from equation (3.6). After calibration, the system showed less than 3

Objective 4 was achieved, and the results helped refine the design rationale. The two-stage strategy was tested against a single-stage baseline in RoboDK. It improved the reliability of approach completion. However, its accuracy advantage depended on a long-range bias seen with real cameras but not in the ideal simulation (Section 5.5).

Objective 5 was achieved. The full five-marker inspection sequence ran in RoboDK, with stable left-to-right target identification (100

Objective 6 was achieved in simulation. The full perception, planning, and execution pipeline runs in ROS 2. Five markers are detected continuously from the simulated depth camera. MoveIt2 plans perpendicular striking sequences using multi-seed IK, with tip-alignment checked during runtime. There is also a supervised operator interface that allows immediate motion cancellation. The next priority is to deploy on the physical UR3e, following the roadmap in Section 5.6.2.

7.4 Conclusions

This thesis has presented the development, implementation, and validation of a vision-guided robotic system for autonomous vibration inspection of aerospace components, carried out through three platforms: a self-built five-DOF arm with embedded detection on a Raspberry Pi 5, an industrial UR3e simulation in RoboDK with quantitative accuracy experiments, and a complete ROS 2 application executing supervised perpendicular striking in Gazebo through MoveIt2. The system detects coded concentric circle markers — and, as the real-component tests showed, unmodified structural features of the workpiece itself — estimates their six-degree-of-freedom pose from a single calibrated camera, and positions the excitation tool perpendicularly at each target without pre-programmed coordinates, manual positioning, or a CAD model.

What this work shows is that the perception requirements of vibration inspection can be met with deliberately modest means: one RGB camera, printed or existing circular features, and careful attention to calibration, frame consistency, and the conditioning of the inverse kinematics. The quantitative results — sub-3% distance accuracy, sub-degree orientation estimation, approximately 2 mm positioning error in simulation — are within the tolerances of the intended task, and the honest characterization of where the two-stage strategy helps and where it does not gives the hardware deployment phase a clear, testable starting point. The complete ROS 2 implementation, built entirely from standard components, provides a credible

foundation for that final step.

Appendix A

Software Module Overview

Table A.1 lists all software modules developed in the course of this thesis, grouped by project stage, together with their primary function.

Table A.1: Software modules developed in this thesis, grouped by development stage.

Module	Function
Stage 1 — Prototype arm (Raspberry Pi 5)	
`kinematics_dh.py`	FK/IK from scratch: `dh_transform()`, `forward_kinematics()`, `solve_ik_numeric()`
`robot_model.py`	Robotics Toolbox `DHRobot` model, `ikine_LM` wrapper with DOF mask
`aruco_to_target.py`	SE3 transformation chain $\boldsymbol{T}_M^B$ from ArUco detection
`marker_detection_system.py`	ArUco detection on Raspberry Pi 5, Picamera2 capture
`main_control.py`	Stage 1 control loop: detect → IK → servo motion
`servo_controller.py`	PCA9685 servo controller with trapezoidal motion profile
Stage 2 — Detection development and RoboDK simulation	
`annulus_detection.py`	Concentric circle detection, basic pipeline
`annulus_detection_far.py`	Robust annulus detection: concentricity filter, EMA tracking
`Detection_model.py`	RoboDK annulus detection only (no motion)
`move_mark_perpen.py`	Full RoboDK pipeline: detection → transform → two-stage motion
Stage 3 — ROS 2 application (`my_robsim`, `ur3e_camera`)	
`annulus_detection_node.py`	ROS 2 detection node: adaptive-threshold annulus pipeline on simulated D405 streams; publishes poses, TF frames, RViz markers
`hammer_controller_node.py`	ROS 2 motion controller: multi-seed IK, MoveIt2 striking sequence, tip-alignment verification, operator GUI
`ur3e_with_camera.xacro`	UR3e description extended with D405 sensor and hammer tool (`hammer_tip` frame)
`sim.launch.py`	Launch pipeline: Gazebo, bridge, controllers, MoveIt2 `move_group`, RViz
Experiments (`thesis_experiment/scripts`)	
`run_E1`–`run_E6` scripts	Experiment automation and data logging for E1–E3, E5, E6
`annulus_core.py`	Shared detection core and calibration handling for the experiment scripts

Appendix B

Key Algorithm Listings

This appendix collects short, commented excerpts of the algorithms that carry the main technical decisions of the thesis. The listings are simplified from the project source code — logging, error handling, and visualisation calls are removed — but the algorithmic content is unchanged. Appendix A maps each listing to its source module.

B.1 Concentric Circle Detection Core

Listing B.1 shows the contour-hierarchy detection loop used in the RoboDK pipeline (Chapter 3). The parent-child relationship of the contour tree encodes the nested ring structure, and the concentricity check rejects circular features that are not genuine ring pairs.

```
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
_, th = cv.threshold(gray, 0, 255,
                     cv.THRESH_BINARY + cv.THRESH_OTSU)
contours, h = cv.findContours(th, cv.RETR_TREE,
                              cv.CHAIN_APPROX_NONE)
for i, c in enumerate(contours):
    if len(c) < 60 or cv.contourArea(c) < 2000:
        continue                        # too small / too noisy
    outer = cv.fitEllipse(c)
    child = h[0][i][2]                  # index of nested contour
    if child == -1:
        continue                        # no inner ring -> not a marker
    inner = cv.fitEllipse(contours[child])
    (x1, y1), _, _ = outer
    (x2, y2), _, _ = inner
    if np.hypot(x1 - x2, y1 - y2) > 10:
        continue                        # centres not concentric
    # 16 sampled points (8 outer + 8 inner) -> PnP
    img_pts = np.array(sample_ellipse(outer)
                       + sample_ellipse(inner), np.float32)
    ok, rvec, tvec = cv.solvePnP(model_points, img_pts,
                                 camera_matrix, dist_coeffs)
```

Listing B.1: Annulus detection: hierarchy analysis, ellipse fitting, and concentricity check.

B.2 Orientation from Ellipse Geometry

Listing B.2 constructs the marker rotation matrix from the fitted outer ellipse, implementing the tilt relation of equation (3.6). The `np.clip` guard prevents a numerically degenerate axis ratio (b/a marginally above 1) from producing a NaN in the arccosine — a failure mode that was found the hard way, as a runtime crash during a robot approach.

```
def get_annulus_rotation(ellipse):
    (cx, cy), (MA, ma), angle = ellipse
    a = max(MA, ma) / 2.0            # semi-major axis
    b = min(MA, ma) / 2.0            # semi-minor axis
    theta = np.deg2rad(angle)        # major-axis direction

    # tilt of the marker plane; clip guards against b/a > 1
    tilt = np.arccos(np.clip(b / a, -1.0, 1.0))

    # surface normal (sign fixed: z points toward the camera)
    z = np.array([-np.sin(tilt) * np.cos(theta),
                  -np.sin(tilt) * np.sin(theta),
                  -np.cos(tilt)])
    z /= np.linalg.norm(z)

    # in-plane x along the ellipse major axis, then Gram-Schmidt
    x = np.array([np.cos(theta), np.sin(theta), 0.0])
    y = np.cross(z, x); y /= np.linalg.norm(y)
    x = np.cross(y, z)               # re-orthogonalised
    return np.column_stack((x, y, z))
```

Listing B.2: Marker rotation matrix from the fitted ellipse (tilt-axis method).

B.3 Masked Inverse Kinematics with Servo-Space Validation

Listing B.3 is the Stage-1 target-approach function. The DOF mask releases the orientation components the five-DOF arm cannot control (Chapter 4), and the solution is validated in servo space — including the mechanical joint conversions — before any motion is commanded.

```
def move_to_marker(rvec, tvec):
    global q_current
    T_base_ee = robot.fkine(q_current)          # current FK
    T_target = aruco_to_target_SE3(rvec, tvec,
                                   T_base_ee, T_ee_cam)
```

```
6
7       mask = [1, 1, 1, 0, 0, 0]         # position-only (Pi variant)
8       sol = robot.ikine_LM(T_target, q0=q_current, mask=mask)
9       if not sol.success:
10          return None
11
12      # model -> servo space, including mechanical conversions
13      q_real_deg = model_to_real(sol.q)
14      if np.any(q_real_deg < 0) or np.any(q_real_deg > 180):
15          return None                   # outside physical range
16
17      q_current = sol.q                 # commit state only if valid
18      return q_real_deg
19
20  def model_to_real(q_model):
21      q_real = q_model.copy()
22      q_real[2] = np.radians(170) - q_model[2]    # joint-3 inversion
23      q_real[4] = q_model[4] + np.radians(18.5)   # joint-5 zero shift
24      return np.degrees(q_real)
```

Listing B.3: Masked LM inverse kinematics and joint validation (Raspberry Pi deployment).

B.4 Multi-Seed Inverse Kinematics in the ROS 2 Controller

Listing B.4 shows the seeding strategy that made pose goals reliable in the ROS 2 application (Chapter 5). The KDL solver behind the MoveIt2 IK service is local, so its success depends on the seed; trying the current state, the home configuration, and a small set of repeatable random perturbations in sequence removed the intermittent no-solution failures.

```
1  def _ik_with_seeds(self, ps):
2      # 1) seed with the current configuration
3      sol = self._ik_once(ps, seed=self._current_positions())
4      if sol is not None:
5          return sol
6      # 2) seed with the home configuration
7      sol = self._ik_once(ps, seed=list(HOME_JOINTS))
8      if sol is not None:
9          return sol
10     # 3) repeatable random seeds around home (fixed RNG seed)
11     for _ in range(IK_SEED_TRIES):
12         seed = [h + d for h, d in
13                 zip(HOME_JOINTS,
14                     self._ik_rng.uniform(-1.2, 1.2, 6))]
15         sol = self._ik_once(ps, seed=seed)
16         if sol is not None:
17             return sol
18     return None
```

Listing B.4: Multi-seed IK strategy in the ROS 2 motion controller.

B.5 Striking Sequence with Stop Supervision

Listing B.5 condenses the four-phase striking sequence of the ROS 2 application. The user-stop flag is evaluated between every phase, and the commanded orientation is checked against the marker normal after the approach.

```
def execute_hammer_sequence(self, ps):
    contact  = self.make_approach_pose(ps, CONTACT_DIST_M)  # 5 mm
    approach = self.make_approach_pose(ps, APPROACH_DIST_M) # 15 cm

    if self._user_stop: return False
    sol = self._ik_with_seeds(approach)      # explicit IK first
    if sol is None: return False              # target unreachable
    if not self.go_to_joints(sol):            # [1/4] PTP approach
        return False
    self.verify_tip_alignment(ps)             # tip_Z . marker_Z = -1?

    if self._user_stop: return False
    if not self.go_linear(contact):           # [2/4] linear strike
        return False
    if self._user_stop: return False
    if not self.go_linear(approach):          # [3/4] linear retract
        return False
    if self._user_stop: return False
    return self.go_home()                     # [4/4] planned home
```

Listing B.5: Four-phase striking sequence with stop checks (ROS 2 controller).

Bibliography

[1] R. D. Adams and P. Cawley. "A review of defect types and nondestructive testing techniques for composites and bonded joints". In: *NDT&E Int.* 21.4 (1988), pp. 208–222 (cit. on pp. 1, 5).

[2] C. Mineo, D. MacLeod, D. Beveridge, S. G. Pierce, P. Nicholson, and C. Morozov. "Robotic path planning for non-destructive testing of complex shaped surfaces". In: *AIP Conf. Proc.* Vol. 2102. 2019 (cit. on pp. 2, 5, 10).

[3] P. I. Corke. *Robotics, Vision and Control: Fundamental Algorithms in Python.* Cham: Springer, 2021 (cit. on pp. 4, 9, 34–36).

[4] S. Mahaut et al. "New advances in ultrasonic testing of aeronautical components". In: *Proc. 7th Int. Conf. on NDE in Relation to Structural Integrity for Nuclear and Pressurised Components.* 2009 (cit. on p. 5).

[5] C. Mineo, S. G. Pierce, B. Wright, I. Cooper, and P. Nicholson. "Novel algorithms for 3D surface point cloud boundary detection and edge reconstruction". In: *J. Comput. Des. Eng.* 6.1 (2016), pp. 81–91 (cit. on p. 5).

[6] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292 (cit. on p. 6).

[7] E. Olson. "AprilTag: A robust and flexible visual fiducial system". In: *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA).* 2011, pp. 3400–3407 (cit. on p. 6).

[8] M. R. Shortis, T. A. Clarke, and T. Short. "A comparison of some techniques for the subpixel location of discrete target images". In: *Videometrics IV, SPIE.* Vol. 2598. 1995, pp. 239–251 (cit. on pp. 6, 10).

[9] S. J. Ahn, W. Rauh, and H.-J. Warnecke. "Least-squares orthogonal distances fitting of circle, sphere, ellipse, hyperbola, and parabola". In: *Pattern Recognition* 34.12 (2001), pp. 2283–2303 (cit. on pp. 6, 10, 21).

[10] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22.11 (2000), pp. 1330–1334 (cit. on pp. 7, 21).

[11] R. Y. Tsai and R. K. Lenz. "A new technique for fully autonomous and efficient 3D robotics hand/eye calibration". In: *IEEE Trans. Robot. Autom.* 5.3 (1989), pp. 345–358 (cit. on pp. 7, 56).

[12] V. Lepetit, F. Moreno-Noguer, and P. Fua. "EPnP: An accurate $O(n)$ solution to the PnP problem". In: *Int. J. Comput. Vis.* 81.2 (2009), pp. 155–166 (cit. on pp. 8, 20).

[13] J. Denavit and R. S. Hartenberg. "A kinematic notation for lower-pair mechanisms based on matrices". In: *J. Appl. Mech.* 22 (1955), pp. 215–221 (cit. on pp. 8, 31).

[14] K. Levenberg. "A method for the solution of certain non-linear problems in least squares". In: *Q. Appl. Math.* 2.2 (1944), pp. 164–168 (cit. on pp. 9, 35).

[15] D. W. Marquardt. "An algorithm for least-squares estimation of nonlinear parameters". In: *J. Soc. Ind. Appl. Math.* 11.2 (1963), pp. 431–441 (cit. on pp. 9, 35).

[16] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022) (cit. on p. 10).

[17] D. Coleman, I. Sucan, S. Chitta, and N. Correll. "Reducing the barrier to entry of complex robotic software: a MoveIt! case study". In: *J. of Software Engineering for Robotics* 5.1 (2014), pp. 3–16 (cit. on p. 10).

[18] RoboDK Inc. *RoboDK Documentation*. [Online]. Available: `https://robodk.com/doc/`. 2024 (cit. on p. 10).