Technische Universität Berlin
Institut für Luft und Raumfahrt

TECHNISCHE UNIVERSITÄT BERLIN

Master Thesis

Autonomous Vision-Based Grasping for a Feedback-Free Low-Cost Robotic Arm in Planetary Exploration

Ivan Gilberto Martin Enciso

B.Sc. Digital Systems and Robotics

Supervisors: Prof. Dr. Enrico Stoll
Dr.-Ing. Armin Wedler
M. Sc. Maximilian von Unwerth

Place and Date: Berlin, 18.02.2026

To Natalia, Laura, and Gilberto — your support made this happen.

Declaration of Authorship

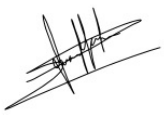
Hereby, I confirm that I have completed this work independently without third-party assistance, using only the sources and tools listed. I have clearly indicated all passages that are directly or indirectly derived from the sources and tools used.

If generative AI tools were employed, I have specified the product names, manufacturers, software versions used, and their purposes (e.g., linguistic review and improvement of texts, systematic research). I am fully responsible for the selection, use, and all results of the AI- generated output I utilized.

I have taken note of the Regulations for Ensuring Good Scientific Practice at TU Berlin, dated February 15, 2023, available at: `https://www.static.tu.berlin/fileadmin/www/10002457/K3-AMBl/Amtsblatt_2023/Amtliches_Mitteilungsblatt_Nr._16_vom_30.05.2023.pdf`.

I further declare that I have not submitted this work, in the same or similar form, to any other examination authority.

Berlin, 18.02.2026



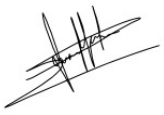
Place and Date

Signature of the student

Agreement on rights of utilization

The Technische Universität Berlin, represented by the Chair of Space Technology, may use the results of the thesis at hand in education and research. It receives simple (non-exclusive) rights of utilization as according to § 31 Abs. 2 Urheberrechtsgesetz (UrhG). This right of utilization is unlimited and involves content of any kind (e.g. documentation, presentations, animations, photos, videos, equipment, parts, procedures, designs, drawings, software including source code and similar). An eventual commercial use on part of the Technische Universität Berlin will only be carried out with approval of the author of the thesis at hand under appropriate share of earnings.

Berlin, 18.02.2026



Place and Date

Signature of the student

Abstract

Autonomous object grasping is essential for planetary exploration missions where rovers must collect geological samples without continuous ground operator control. This thesis develops and validates a complete vision-based autonomous grasping pipeline for the *Lunar Rover Mini (LRM)*, a small-scale planetary exploration platform. The system uses a low-cost monocular camera mounted on the robotic arm end-effector combined with a RealSense depth camera at the rover front, enabling object detection, pose estimation, and grasp execution without force sensors or specialized tactile feedback.

The main technical challenge of the *LRM* robotic arm is the lack of joint position feedback from the servo motors. Once a position command is sent, the onboard computer cannot directly verify the final arm configuration. Effects such as gravity, structural elasticity, and load variations introduce positioning errors that accumulate during manipulation and prevent reliable grasp execution. Visual odometry drift from the rover's 180° rotation after initial detection further compounds these errors. This work addresses these limitations through visual feedback, allowing the system to estimate the relative pose between the end-effector and the target object and iteratively correct positioning errors.

The system performs feature-based pose estimation between the RealSense depth image and the monocular end-effector camera image. *Scale-Invariant Feature Transform (SIFT)* feature extraction and matching establishes 3D-2D correspondences, which the system solves using *Perspective-n-Point (PNP)* with *Random Sample Consensus (RANSAC)*-based outlier rejection and Levenberg-Marquardt refinement. An object center reprojection error metric validates pose estimates before commanding arm motion. Image-based visual refinement compensates for residual positioning errors, where the system iteratively adjusts gripper position based on pixel error feedback until the object center aligns with the target pixel location. Grasp detection uses motor current measurements, including pre-grasp baseline calibration and threshold-based stability evaluation.

The complete pipeline uses *RMC advanced Flow Control (RAFCON)* hierarchical state machines with modular Python components. The *Robot Operating System (ROS) Transform (TF)* tree handles all coordinate transformations. Experimental validation at the *German Aerospace Center (DLR)* Planetary Exploration Lab evaluated both individual stages and integrated performance. Object detection achieved 100% success in the integrated runs. *PNP* pose estimation converged in 77.8% of runs (7/9), with accepted object center reprojection errors between 30.96 and 48.93 px. Visual refinement and grasp detection achieved 100% success in runs that reached these stages. End-to-end task completion reached 55.6% (5/9 runs), limited primarily by *PNP* convergence. Once pose estimation converged and the object remained within the reachable workspace and field of view, downstream stages completed without failure.

Acknowledgments

I would like to thank Armin Wedler for giving me the opportunity to participate in this project. I am also grateful to Maximilian von Unwerth for his continuous support throughout the thesis. I further thank Riccardo Giubilato for his assistance during debugging and for explaining key aspects of the internal system, which helped clarify the correct approach to achieving the project objectives. Finally, I would like to thank Ryo Sakagami and Julius Karsten for their support and assistance with specific tasks within the institute.

Contents

List of Figures xvii

List of Tables xx

Acronyms xxii

1 Introduction 1
1.1 Motivation 1
1.2 Problem Statement 2
1.3 Objectives and Scope 2

2 Background and Related Work 4
2.1 Related Techniques and Methods 4
2.2 Camera Theory 5
2.2.1 Camera Models 5
2.2.1.1 Pinhole Model 5
2.2.1.2 Radial-Tangential Distortion Model 6
2.2.1.3 Fisheye Model (Kannala-Brandt) 7
2.2.2 Intrinsics 9
2.2.3 Extrinsics 10
2.3 Object Detection and Segmentation 10
2.3.1 YOLOv7 Implementation 10
2.4 Feature Extraction 11
2.4.1 ORB 11
2.4.2 SIFT 11
2.5 Feature Matching 12
2.5.1 Brute-Force Matcher 13
2.5.2 FLANN-Based Matcher 13
2.5.2.1 Lowe's ratio 13
2.6 Camera Pose Estimation 14
2.6.1 The Perspective-n-Point Problem 14
2.7 Kinematics 15
2.7.1 Forward Kinematics 15
2.7.2 Inverse Kinematics 15
2.7.3 Elasto-Kinematic Model 16
2.8 Visual Servoing 16
2.8.1 Image Based Visual Servoing 16
2.8.2 Position Based Visual Servoing 17
2.9 Gripping Detection 17

3 System Design and Architecture 19
3.1 Overall System Architecture 19
3.1.1 Hardware 19
3.1.2 Software 21

4 System Setup and Calibration 25
4.1 Coordinate System Definition 26
4.1.1 TF Tree Architecture 26
4.2 Camera Calibration 29
4.2.1 Intrinsic Calibration 29
4.2.1.1 Monocular End Effector Camera 29
4.2.1.2 RealSense Depth Camera 30
4.2.2 Extrinsic Calibration 31
4.2.2.1 Monocular End Effector Camera 31
4.2.2.2 RealSense Depth Camera 32
4.2.3 Validation 32
4.3 Servo Configurations and Calibration 33
4.3.1 Hardware-Level Servo Configuration 33
4.3.2 PWM Based Servo Calibration 36
4.4 Mission Setup 37

5 System Integration 40
5.1 Coordinate System Integration 40
5.1.1 ROS and Links-and-Nodes Integration 40
5.1.1.1 Robotic Arm TF Publisher 40
5.1.1.2 End-Effector Camera Publisher 41
5.1.1.3 ROS to Links-and-Nodes Bridge 41
5.1.2 Object Detection Integration 41
5.2 Mission States 41

6 Object Localization and Visual Odometry Considerations 49
6.1 RealSense Depth-Based Localization 50
6.1.1 Publishing Object Coordinates 51
6.2 Localization Observations and Discussion 52

7 Visual Pose Estimation and Target Refinement 54
7.1 Feature Extraction and Matching 54
7.1.1 Reference Image Acquisition 54
7.1.2 Re-localization Using the End-Effector Camera 55
7.1.3 SIFT Feature Detection 56
7.1.4 Feature Matching and Correspondence 58
7.1.4.1 Homography-based Outlier Rejection 59
7.1.4.2 Visualization and Quality Assessment 61

7.2 Camera Pose Estimation 62
7.2.1 PnP Formulation with Known Depth 62
7.2.1.1 3D Point Reconstruction from Depth Image 62
7.2.2 RANSAC-Based Robust Estimation 63
7.2.3 Pose Refinement 63
7.2.4 Pose Quality Assessment 64
7.2.5 Object Center Reprojection Validation 64
7.3 Coordinate Transformation to Arm Workspace 65
7.4 Image-Based Position Refinement 67
7.4.1 Operating Constraints 67
7.4.2 Calibration and Target Pixels 68
7.4.3 Refinement Procedure 68
7.4.4 Performance and Limitations 70

8 Control and Manipulation 71
8.1 Reachability-Based Target Validation 71
8.1.1 Workspace Constraints 71
8.2 Path Planning 72
8.2.1 RMPL Integration 72
8.2.2 Path Interpolation 73
8.3 Autonomous Grasp Execution Using Motor Current Feedback 73
8.3.1 Observed Current Behavior 74
8.3.2 Initial Grasp Detection Logic 74
8.3.3 Pre-Grasp Current Calibration 74
8.3.4 Delta-Current Based Detection 75
8.3.5 Stable Current Detection and Post-Grasp Tightening 75
8.3.6 Failure Classification 75

9 Experimental Evaluation 77
9.1 Object Detection Results 78
9.1.1 RealSense Depth-Based Object Detection 78
9.1.1.1 Analysis 80
9.1.2 Monocular End-Effector Camera Object Detection for Centering ... 81
9.1.2.1 Analysis 83
9.2 PnP Pose Estimation Evaluation 83
9.2.1 Feature Matching and PnP Inlier Statistics 83
9.2.2 PnP Convergence Outcome Based on Object Center Reprojection Error 84
9.2.3 Retry Behavior Above Object Center Reprojection Error Threshold 85
9.2.4 PnP Accepted but Run Failed in Later Stages 89
9.2.5 Analysis 89
9.3 Visual Refinement Evaluation 90
9.3.1 Per-Run Summary 91
9.3.2 Object Re-acquisition Grid Search 92

9.3.3 Centering Convergence 93
9.3.4 Pixel Error at Convergence 95
9.3.5 Analysis 95
9.4 Grasp Detection Evaluation 96
9.4.1 Experimental Protocol 96
9.4.2 Baseline Current Calibration 97
9.4.3 Delta-Current Detection Performance 97
9.4.4 Grasp Detection Accuracy 98
9.4.5 Failure Mode Classification 98
9.4.6 Analysis 98
9.5 End-to-End Pipeline Performance 101
9.5.1 Overall Success Rate 101
9.5.2 Pipeline Stage Success Breakdown 102
9.5.3 Analysis 103
9.6 Summary 104

10 Conclusion and Future Work 105
10.1 Summary of Contributions 105
10.2 Limitations 106
10.3 Future Work 108
10.3.1 Object Detection Improvements 108
10.3.2 Multi-Object Scenarios and Object Selection 108
10.3.3 PNP Pose Estimation Improvements 109
10.3.4 Visual Refinement Improvements 109
10.3.5 Motion Planning and Grasp Pose Selection 110
10.3.6 Grasp Force Control and Tactile Feedback 110

Appendices 111

A User Interface Development 112
A.1 Low-Level GUI 112
A.1.1 Overview 112
A.1.2 Communication Setup 113
A.1.3 Enabling Arm Control 113
A.1.4 Joint Slider Interface 113
A.1.5 Predefined Position Management 113
A.1.6 Servo Status Monitoring 113
A.2 High-Level GUI 114
A.2.1 Overview 114
A.2.2 Permission Management 114
A.2.3 Robotic Arm Control Panel 115
Path Planning Parameters 115
Arm Control Methods 115
A.2.4 Path Planning and Execution 115

B Calibration Data 116
B.1 SQ11 Camera Specifications 116
B.2 D435i RealSense Camera Specifications 117
B.3 Servo Specification Data 118
B.3.1 A80BHMF Servo Datasheet 118
B.3.2 A35BHL Servo Datasheet 118
B.3.3 A35BHL Servo Configuration 119
B.3.4 A20CLS Servo Datasheet 119
B.3.5 A20CLS Servo Configuration 120

C ROS Topic and Message Definitions 121

D RAFCON State Machines and Tables 122
D.1 Master State Machine 122
D.2 Object Detection State Machine 123
D.3 Vision-Guided Position Refinement State Machine 123
D.4 Additional State Machines 124

E Results 126
E.1 All Object Detection Results 126
E.2 Image-Based Position Refinement Results 128

F Contact 132

References 132

List of Figures

Figure 1.1 Both Lunar Rovers Mini in the 2025 summer school experiments in Sweden. 1

Figure 2.1 Pinhole camera model showing the projection of a 3D point onto the image plane. The focal length f, optical axis, and principal point define the basic projection geometry. 6

Figure 2.2 Radial distortion effects on a checkerboard pattern. Left: no distortion (ideal pinhole model). Center: barrel distortion (negative radial distortion, lines bow outward). Right: pincushion distortion (positive radial distortion, lines bow inward). 7

Figure 2.3 Fisheye camera model projection geometry. (a) Different projection models showing the relationship between incident angle θ and image radius r: (i) stereographic, (ii) equidistance, (iii) equisolid angle, (iv) orthogonal, (v) perspective (pinhole). (b) Geometric representation of the fisheye projection where point P projects to point p based on the incident angle θ and focal length f 8

Figure 2.4 *SIFT* scale-space representation and keypoint detection. Left: Scale-space pyramid with multiple octaves and Gaussian blur levels. Right: *Difference-of-Gaussians (DOG)* used to identify keypoints by finding local extrema. 12

Figure 3.1 Overview of the LRM system components. 19

Figure 3.2 New camera mount and gripper linear mechanism. 21

Figure 3.3 SQ11 Full HD 1080P camera. 21

Figure 3.4 Overview of the system software architecture and interactions between its main components. Rectangular blocks represent software processes or modules, arrows show data flow and communication, and configuration files are shown separately and linked to the components that consume them. Hardware components are depicted at the bottom of the diagram. The Links and Nodes Manager acts as the central coordination layer between high-level software modules, simulation and visualization tools, and low-level hardware interfaces. RAP and VP refer to the Robotic Arm Package and Vision Package, respectively. 22

Figure 4.1 Relevant coordinate frames of the robotic arm and vision system. Top: side view showing the transformation chain from `base_link` and `boogie_base` through the arm links (`link_1`–`link_6`) to the end-effector, gripper tip, and camera optical frames. Bottom: rear view illustrating the `boogie_base` frame, which defines the working space of the robotic arm. All frames follow the right-handed convention used in the system *TF* tree. 27

Figure 4.2 New camera mount and gripper seen from side view with dimensions.. 31

Figure 4.3 Reprojection of checkerboard corners using the estimated Kannala–Brandt intrinsic parameters from two different viewpoints. 33
Figure 4.4 Representative hardware-level servo configuration results for the A80BHMF servos used in joints 1–3. 36
Figure 4.5 Side view of the rover with the rock setup and orange cube in sight at the start of the mission. 38
Figure 4.6 RViz visualization of the mission starting configuration, showing the RealSense camera view (top right), the end-effector camera view (bottom right), and the TF-based 3D scene representation (left). 38
Figure 5.1 TF tree overview and visualization in RViz. 40
Figure 5.2 State machine diagram of the autonomous grasping pipeline. Green arrows indicate successful happy path transitions, red arrows indicate error paths, black and dashed arrows show retry and fallback mechanisms. 42
Figure 6.1 Orange cube (object) detected with a confidence score by the RealSense camera. 50
Figure 6.2 RViz visualization of the object marker (orange cube) shown together with the current RealSense view before rover rotation. 51
Figure 6.3 RViz visualization of the object markers after a 180°rover rotation. 52
Figure 6.4 RViz visualization of the TF tree with two markers and the visual odometry output (seen tilted) after the rotation. 53
Figure 7.1 Viewpoint comparison between the RealSense scene (left) and the monocular end-effector camera scene (right). 55
Figure 7.2 *SIFT* keypoints detected in the RealSense reference image. 57
Figure 7.3 *SIFT* keypoints detected in the end-effector monocular camera image. 57
Figure 7.4 Feature matching and correspondence pipeline showing the complete flow from SIFT feature extraction through ratio test filtering, homography-based RANSAC verification, and minimum inlier check before passing correspondences to the PnP solver. Each step is detailed in the following subsections. 59
Figure 7.5 Matching results before *RANSAC* filtering, with 157 good matches between the RealSense reference image (left) and end-effector monocular camera image (right). 61
Figure 7.6 Matching results showing the same scene after geometric verification, where 49 inliers remain. Colored lines connect geometrically consistent *SIFT* keypoint pairs. 61
Figure 7.7 Reprojection error visualization after pose refinement. The green circle marks the detected object center (YOLO bounding box center), and the red circle shows the reprojected 3D point position based on the estimated camera pose. The pixel error of 30.96 pixels indicates good pose accuracy. 65
Figure 7.8 Calibration images at 50 mm (left) and 100 mm (right) showing target pixel locations (red crosshair) for centered object positioning. 68

Figure 7.9 Visual refinement iteration showing target pixel (green circle), detected object center (red circle), and correction vector (yellow arrow). On the top of the image an overlay with the image number (represents the iteration of the loop), height (100 mm), and the pixel error in both u (91px) and v (96px) axis. 69
Figure 9.1 Experimental setup at the DLR Planetary Exploration Lab showing the Lunar Rover Mini operating in a simulated lunar environment. 77
Figure 9.2 Green cube detection under lights-off (top) and lights-on (bottom) conditions. Detection fails when the light is on due to the cube's reflective surface. 79
Figure 9.3 Orange cube detection under lights-on with object shifted laterally in field of view. Detection confidence remains high (0.92) despite position offset, confirming robustness to moderate lateral shifts. The matte orange surface shows minimal reflection artifacts compared to the green and blue cubes. 80
Figure 9.4 Monocular detection at 70 mm (top) and detection failure at 50 mm (bottom) due to image blur. At 70 mm, the cube is in focus and detected with 90.12% confidence. At 50 mm, the image exceeds the camera's minimum focus distance, resulting in blur that prevents feature extraction. No bounding box is drawn when detection fails. 82
Figure 9.5 Inlier feature correspondences for the accepted attempt of run R2. The lines connect geometrically consistent SIFT keypoint matches between the reference RealSense image (left) and the monocular end-effector image (right). These 3D–2D correspondences are used by solvePnPRansac to estimate the object pose. 84
Figure 9.6 Object center reprojection error used for PnP acceptance. Left: accepted run example with smaller pixel offset between the detected object center (green) and the reprojected object center (red). Right: rejected attempt example with a larger pixel offset, which triggered additional retries. 85
Figure 9.7 Monocular reprojection with refined inlier correspondences for the accepted attempt of run R2 (46.03 px error). The visualization shows the quality of the pose estimate through the reprojected features and object center alignment. 86
Figure 9.8 Object center reprojection frames for the best observed attempt of rejected run NC1 (51.93 px). Top: detected object center (green) vs. reprojected center (red). Bottom: monocular reprojection with error visualization showing pose quality. 87
Figure 9.9 Object center reprojection frames for the final attempt of rejected run NC1 before manual termination (62.76 px). Top: detected object center (green) vs. reprojected center (red) with larger offset. Bottom: monocular reprojection showing degraded pose quality. 88

Figure 9.10 Convergence trajectory for Run 4 (28 iterations). The plotted value corresponds to the pixel error norm $\|e\| = \sqrt{du^2 + dv^2}$ at each iteration. Convergence occurs when both $|du|$ and $|dv|$ fall below 35 px..... 93
Figure 9.11 Convergence trajectory for Run 3 (2 iterations). Due to a smaller initial image error, only one corrective step was required before satisfying the convergence condition. 94
Figure 9.12 Current profile during a successful grasp (hard configuration). Light blue region (0–3 s): baseline calibration with blue trace and noise band (dashed). seeking region (5–11 s): grasp execution with orange trace oscillating during seeking, then rising at contact detection (blue vertical dotted line). Baseline I_{base} (teal dash-dot), lower threshold (yellow), and upper threshold (red) shown constant. 99
Figure A.1 Low-level *Graphical User Interface (GUI)*: Robotic arm panel. 112
Figure A.2 High-level *GUI*: Robotic arm control interface. 114
Figure B.1 Representative hardware-level servo configuration results for the A35BHL servos used in joints 4–6. 119
Figure B.2 Representative hardware-level servo configuration results for the A20CLS servo used in the gripper..... 120
Figure D.1 Master state machine showing the complete autonomous grasping pipeline structure with initialization, perception, motion, and error recovery states. 122
Figure D.2 Object detection state machine shared between RealSense and monocular perception modes, showing image acquisition, YOLO inference, and detection validation logic. 123
Figure D.3 Vision-guided position refinement state machine implementing multi-viewpoint PnP pose estimation with RANSAC outlier rejection and object center reprojection error validation. 124
Figure E.1 Post-inference results for the blue cube with centered object position under lights-off (left) and lights-on (right) conditions. 126
Figure E.2 Post-inference results for the blue cube with object positioned to the right under lights-off (left) and lights-on (right) conditions. 126
Figure E.3 Post-inference results for the green cube with object positioned to the right under lights-off (left) and lights-on (right) conditions. 127
Figure E.4 Post-inference results for the orange cube with centered object position under lights-off (left) and lights-on (right) conditions. 127
Figure E.5 Post-inference results showing the orange cube positioned to the right under lights-off conditions. 127
Figure E.6 Post-inference results for monocular camera detection at heights of 60 mm (left) and 80 mm (right)..... 128
Figure E.7 Post-inference results for monocular camera detection at heights of 90 mm (left) and 100 mm (right). 128
Figure E.8 Visual centering iterations 1-2. Left: Initial detection with large pixel offset. Right: After first correction step. 129

Figure E.9 Visual centering iterations 3-4. The pixel error decreases as the gripper position is adjusted based on visual feedback. 129
Figure E.10 Visual centering iterations 5-6. Continued convergence toward the target pixel location. 129
Figure E.11 Visual centering iterations 7-8. The correction vector shortens as the object center approaches the target. 130
Figure E.12 Visual centering iterations 9-10. Progressive error reduction continues. 130
Figure E.13 Visual centering iterations 11-12. Approaching convergence. 130
Figure E.14 Visual centering iteration 13 (final). The pixel error satisfied the stopping condition (du = -25.0 px, dv = -17.0 px), resulting in a final error norm of 30.23 px. 131

List of Tables

Table 4.1 Hardware-level servo parameters configured for the LRM robotic arm... 34
Table 4.2 Final hardware-level configuration values per servo type..... 35
Table 4.3 Overload protection configuration per servo type..... 35
Table 4.4 Results of the PWM based calibration for each joint..... 37
Table 5.1 State descriptions in the autonomous grasping state machine..... 45
Table 5.2 State transition logic in the autonomous grasping state machine (Part I) 46
Table 5.3 State transition logic in the autonomous grasping state machine (Part II) 47
Table 5.4 Approach modes and their effects on state transitions..... 48
Table 7.1 Target pixel coordinates for object centering at different gripper heights. 68
Table 9.1 RealSense object detection results: detection success and confidence scores under varying lighting and object position conditions. 78
Table 9.2 RealSense 3D position reconstruction for successful detections. Coordinates are shown in camera optical frame, base_link frame, and map frame. 79
Table 9.3 Monocular object detection results as a function of camera height during centering operation. 81
Table 9.4 Feature matching and PnP inlier statistics at the accepted attempt. The inlier ratio is computed as inliers divided by total correspondences. 84
Table 9.5 PnP outcome summary using the object center reprojection error (green detected center vs. red reprojected center). “Accepted” means the run reached the 50 px acceptance threshold. “Rejected” means the system terminated after extended retries without meeting the threshold. 85
Table 9.6 Example of retry behavior in an accepted run (R2). Each attempt is a new estimation cycle. Acceptance depends on the object center reprojection error..... 86
Table 9.7 Representative attempts of a rejected run (NC1): first attempt, best attempt (minimum error), and final attempt before manual termination. 87
Table 9.8 Runs where PnP converged (accepted) but the overall run did not complete successfully due to downstream causes..... 89
Table 9.9 Hover height and grid search behavior for all successful runs..... 91
Table 9.10 Centering convergence results. The reported final errors correspond to the last iteration before the stopping condition was satisfied..... 91
Table 9.11 Grid search statistics for object re-acquisition. 92
Table 9.12 Convergence statistics for the visual centering loop across all successful runs ($N = 5$). 93
Table 9.13 Final pixel error statistics at convergence ($N = 5$). 95
Table 9.14 Baseline current and computed delta-current detection thresholds for easy and hard arm configurations. 97
Table 9.15 Summary of grasp detection trials..... 98
Table 9.16 Confusion matrix for grasp detection across the evaluated trials ($N = 5$). 98

Table 9.17 Observed failure mode distribution for unsuccessful grasp attempts ($N = 3$). 98
Table 9.18 End-to-end pipeline outcomes across the nine evaluated autonomous runs ($N = 9$). 102
Table 9.19 Stage-wise success rates for the nine autonomous runs. Each stage is conditional on the previous stage succeeding. 102
Table B.1 Technical specifications of the SQ11 mini DV camera..... 116
Table B.2 Technical specifications of the Realsense D435i..... 117
Table B.3 Specifications of the AGFRC A80BHMF Servo 118
Table B.4 Specifications of the AGFRC A35BHL Servo 118
Table B.5 Specifications of the AGFRC A20CLS Servo 119
Table D.1 Recovery and retry mechanisms implemented in the grasping pipeline .. 125
Table D.2 Key configuration parameters used in the autonomous grasping pipeline 125

Acronyms

A

API - Application Programming Interface 72

B

BRIEF - Binary Robust Independent Elementary Features 11

C

CAD - Computer-Aided Design 31, 33

D

DH - Denavit-Hartenberg 15
DLR - German Aerospace Center ..... iii, 1, 2, 15, 22, 29, 37, 72, 77, 104, 107
DOF - Degrees of Freedom 1, 15, 19, 20, 73
DOG - Difference-of-Gaussians ..... xiii, 11, 12

E

ESA - European Space Agency 1

F

FAST - Features from Accelerated Segment Test 11
FLANN - Fast Library for Approximate Nearest Neighbors 4, 13, 58
FOV - Field of View 32, 43

G

GUI - Graphical User Interface ..... xvi, 23, 34, 72, 112–115

I

IBVS - Image-Based Visual Servoing 16, 17, 70, 96, 109
IMU - Inertial Measurement Unit 20
IO - Input / Output 73
ISS - International Space Station 1

L

LN - Links-and-Nodes 1, 22–24, 40, 41, 43, 115
LRM - Lunar Rover Mini ..... iii, 1, 2, 10, 15, 19, 20, 25, 33, 71, 105

M

MAD - Median Absolute Deviation 74
MER - Mars Exploration Rover 1

N

NAN - Not a Number 62
NUC - Next Unit of Computing 20, 113

O

ORB - Oriented FAST and rotated BRIEF 11, 12, 56

P

PBVS - Position-Based Visual Servoing 16, 17
PNP - Perspective-n-Point iii, 4, 14, 15, 17, 44, 54, 55, 60–68, 77, 83, 84, 87, 89–92, 95, 96, 101–107, 109, 123
PWM - Pulse-Width Modulation 34, 36, 113

R

RAFCON - RMC advanced Flow Control... iii, 1, 3, 23, 24, 34, 41, 43, 45, 72, 105, 114, 123
RANSAC - Random Sample Consensus iii, xiv, 14, 15, 44, 58, 60, 61, 63, 64, 83, 90, 105, 123
RC - Radio Control 1, 33
RGB - Red-Green-Blue color model 7, 30, 54, 56, 62
RMPL - Robotics Motion Planner Library 15, 23, 72, 73
ROS - Robot Operating System iii, 1, 7, 20, 22–24, 28, 30–33, 40, 41, 43, 61, 66, 77, 105, 121

S

SIFT - Scale-Invariant Feature Transform... iii, xiii, xiv, 4, 11–13, 55–58, 61, 67, 83, 89, 105, 107, 109

T

TCP - Tool Center Point 73
TF - Transform ..... iii, xiii, 23, 26, 27, 29, 33, 40, 54, 65, 66, 90, 105, 109

U

URDF - Unified Robotics Description Format 15, 72, 109

Y

YOLO - You Only Look Once 4, 10, 41, 64, 67, 77, 78, 96, 106, 108

1. Introduction

1.1. Motivation

Robotic systems play an increasingly important role in space exploration, but their development and deployment remain extremely expensive. For example, the *Mars Exploration Rover (MER)* mission had an overall cost of approximately 1.08 billion USD [1], while the Curiosity and Perseverance rovers reached costs of about 2.5 and 2.7 billion USD respectively [2, 3]. These figures already include the scientific instruments and robotic subsystems required for mission operation. Although precise cost data for the European robotic arms aboard the *International Space Station (ISS)* are not publicly available, they are expected to follow similar cost scales.

These cost levels strongly limit access to space robotics research, especially for educational institutions. This challenge motivated the development of the *LRM*. The *LRM* seen in Figure 1.1 was developed as a ground-based, low-cost rover platform inspired by the *European Space Agency (ESA)* ExoMars mission [4]. The complete system costs approximately €2,000 and includes a six *Degrees of Freedom (DOF)* robotic arm built from standard *Radio Control (RC)* digital servos with a total cost of roughly €300. Despite its low cost, the LRM integrates a sophisticated software ecosystem, including *Links-and-Nodes (LN)*, *ROS* [5], Simulink [6] support, and the *RAFCON* state machine framework [7]. The platform is actively developed at the Institute of Robotics and Mechatronics of *DLR* and serves as a testbed for education and experimental autonomy research.



Figure 1.1: Both Lunar Rovers Mini in the 2025 summer school experiments in Sweden.

The *LRM* project represents a cumulative effort by *DLR* students over several years. Beginning with mechanical and electronics design, the work has progressed to high-level autonomous navigation capabilities [8]. Prior to this thesis, the rover was capable of successfully mapping unknown environments while detecting objects of interest. The missing piece was what comes after detection: what should the rover do once objects are detected? This thesis addresses this gap by implementing autonomous grasping capabilities. By combining multiple techniques to compensate for the lack of joint position feedback, the system can now interact with detected objects, completing the perception-to-manipulation pipeline.

1.2. Problem Statement

One of the main limitations of the *LRM* robotic arm is the lack of feedback from the servo motors. Once a position command is sent to them, there is no way for the *LRM* computer to know precisely the final position of the robotic arm. This is especially problematic due to the influence of gravity, structural elasticity, and load variations, which can cause the final pose of the robotic arm to be different from the intended one. For high-precision tasks, such as grasping and manipulating objects, these errors can accumulate to a level where the robotic arm cannot successfully grasp the object.

An additional source of positioning error comes from the rover configuration itself. The robotic arm is mounted at the rear of the rover, while the RealSense depth camera used for initial object detection is mounted at the front. After detecting an object, the rover must rotate approximately 180° to position the arm toward the target. Visual odometry drift accumulated during this rotation further degrades the reliability of the initial object coordinates, compounding the servo positioning errors.

To address these limitations, a camera mounted on the end-effector provides visual feedback. This allows the system to estimate the pose of the arm relative to the target object and iteratively correct positioning errors before grasp execution.

1.3. Objectives and Scope

The main goal of this thesis is the development of a complete autonomous grasping pipeline for the *LRM* robotic arm using a low-cost camera mounted on the end-effector. Since forward and inverse kinematics as well as basic path planning were already available prior to this work, the focus is placed on the design and implementation of the autonomous control logic required to execute a grasp successfully.

This includes the integration of the kinematic transform chain into the system, calibration and incorporation of the newly added end-effector camera, development of the image

acquisition and processing pipeline, and the implementation of a quasi-visual servoing [1] approach to close the control loop during grasping. The high-level autonomous decision logic is realized using *RAFCON* state machines and Python scripts to ensure a modular and structured control architecture.

The scope of this work is intentionally limited in several aspects.

- Only single-object grasping scenarios are considered.
- Object detection is restricted to a supervised dataset containing cubic objects.
- The visual refinement approach assumes sufficient lighting conditions and continuous visibility of the target object.
- Collision handling is limited to joint limit enforcement and basic workspace boundaries, more advanced obstacle avoidance is not addressed.
- Grasp pose optimization and object selection strategies are outside the scope of this thesis.
- Error handling is restricted to common failure cases in perception, planning, and execution using simple retry mechanisms.

These points outline the main boundaries of the current work, while potential extensions are discussed in Section 10.3.

[1] The approach is based on visual servoing, but not in a real-time sense. Instead of continuous closed-loop control, the system performs a number of discrete image–update iterations to correct the joint states. For this reason, the term visual refinement will be used instead of visual servoing throughout the document.

2. Background and Related Work

This chapter presents the background theory and related techniques used in the autonomous grasping pipeline. The approach combines computer vision methods for object localization with kinematic models and feedback-based position correction. In particular, feature-based pose estimation using *PNP* is used to re-localize the target after the rover rotates and the arm cannot rely on servo position feedback alone.

2.1. Related Techniques and Methods

The main components of the system include:

- **Camera Calibration** – Intrinsic and extrinsic camera parameters are needed to relate image measurements to three-dimensional coordinates in the rover workspace. Discussed in Section 2.2.
- **Object Detection and Segmentation** – *You Only Look Once (YOLO)*v7 detects and segments objects in the camera image, providing bounding boxes used to compute target positions. Discussed in Section 2.3.
- **Feature Extraction and Matching** – *SIFT* extracts visual features from images captured by different cameras and establishes correspondences between viewpoints. *Fast Library for Approximate Nearest Neighbors (FLANN)*-based matching is used to find feature pairs efficiently. Discussed in Sections 2.4 and 2.5.
- **Camera Pose Estimation** – The *PNP* algorithm estimates the camera pose from matched features and known depth information, providing the transformation between camera frames. Discussed in Section 2.6.
- **Kinematics** – Forward and inverse kinematics models compute the relationship between joint angles and end-effector position, allowing the system to command target positions in Cartesian space. Discussed in Section 2.7.
- **Visual Servoing** – Image-based and position-based visual servoing strategies use visual feedback to control robot motion and compensate for calibration and modeling errors. Discussed in Section 2.8.
- **Gripping Detection** – State-of-the-art grasp verification methods include force, tactile, vision-based, and current-sensing approaches for confirming successful object acquisition. Discussed in Section 2.9.

2.2. Camera Theory

In this thesis, computer vision is used to obtain the position of objects from camera images in order to support autonomous grasping. When a camera observes a three-dimensional scene, it produces a two-dimensional image. While the scene is defined in three dimensions, the image represents this information only in the image plane, expressed in pixel coordinates.

In this setting, points in the scene are described using three-dimensional coordinates (x, y, z), while their corresponding image locations are expressed in pixel coordinates (u, v). In standard computer vision conventions, u denotes the image width and v the image height, with the origin typically located at the top-left corner of the image.

To use visual information for manipulation, it is necessary to understand how a point in three-dimensional space (for example, a ball located at a specific (x, y, z) position) maps to its corresponding pixel (u, v) in the two-dimensional image. Establishing the mathematical relationship between these two representations, which belong to different coordinate systems, is a fundamental problem in computer vision [9]. Camera models, intrinsics, and extrinsics define this relationship.

2.2.1. Camera Models

Before discussing camera intrinsics and extrinsics, it is important to understand how images are formed mathematically. Camera models describe the geometry of how a three-dimensional point in space projects onto a two-dimensional image. Different models account for different lens types and distortion characteristics. This section covers three models relevant to this work: the pinhole model, the radial-tangential distortion model, and the fisheye model.

2.2.1.1. Pinhole Model

The pinhole model is the simplest camera model and forms the basis for most computer vision algorithms. It assumes an idealized camera where light rays pass through a single point (the optical center) and project onto an image plane. Under this model, a 3D point $\mathbf{P} = [X, Y, Z]^T$ in the camera coordinate frame projects to a 2D image point $\mathbf{p} = [u, v]^T$ according to Hartley and Zisserman [10]:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.1}$$

where $s = Z$ is the depth of the point, and $\mathbf{K}$ is the camera intrinsic matrix defined in Section 2.2.2.

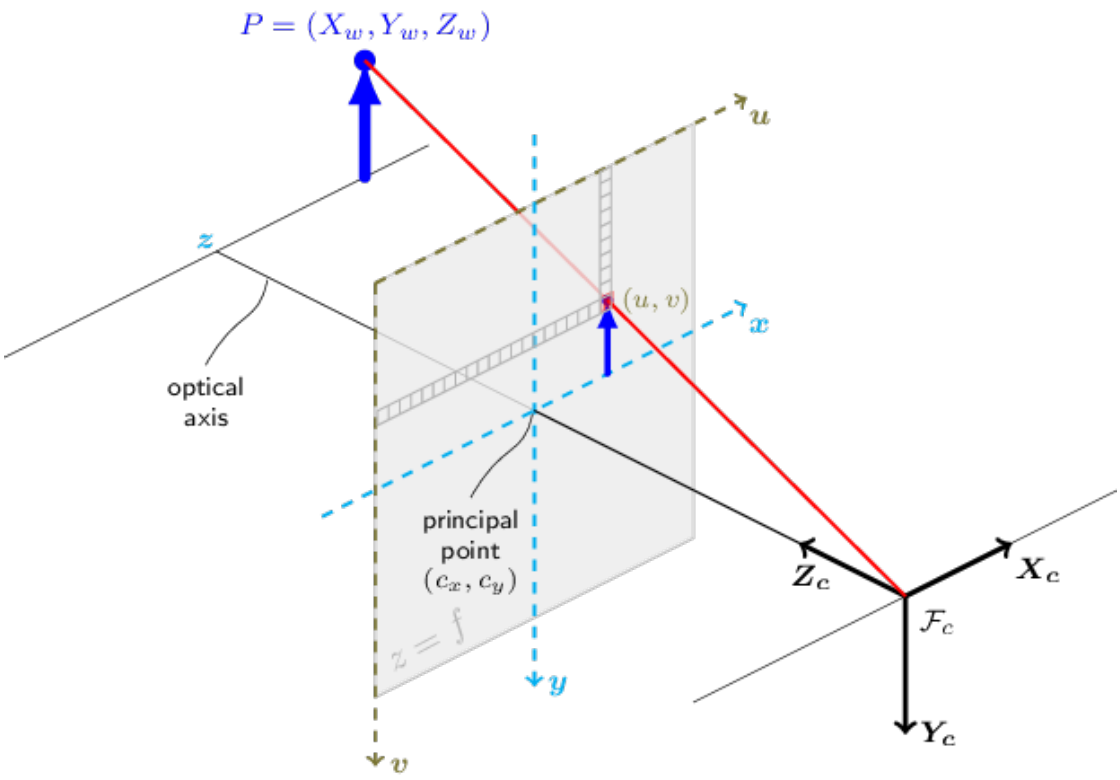


Figure 2.1: Pinhole camera model showing the projection of a 3D point onto the image plane. The focal length f, optical axis, and principal point define the basic projection geometry.

Figure 2.1 shows the projection of a 3D point onto the image plane, where the focal length f, optical axis, and principal point define the basic projection geometry [11]. This projection is purely perspective and does not account for lens distortion. Real cameras introduce distortions that need correction for accurate measurements.

2.2.1.2. Radial-Tangential Distortion Model

Real camera lenses introduce distortions that cause straight lines in the world to appear curved in the image. The most common distortion model extends the pinhole projection with radial and tangential distortion terms, also known as the Brown-Conrady model [12].

Radial distortion occurs because light rays are bent more at the edges of the lens than at the center, causing barrel distortion (where image points are displaced outward) or pincushion distortion (where points are displaced inward). Tangential distortion occurs when the lens is not perfectly aligned with the image sensor. Figure 2.2 illustrates these distortion effects on a checkerboard calibration pattern [9].

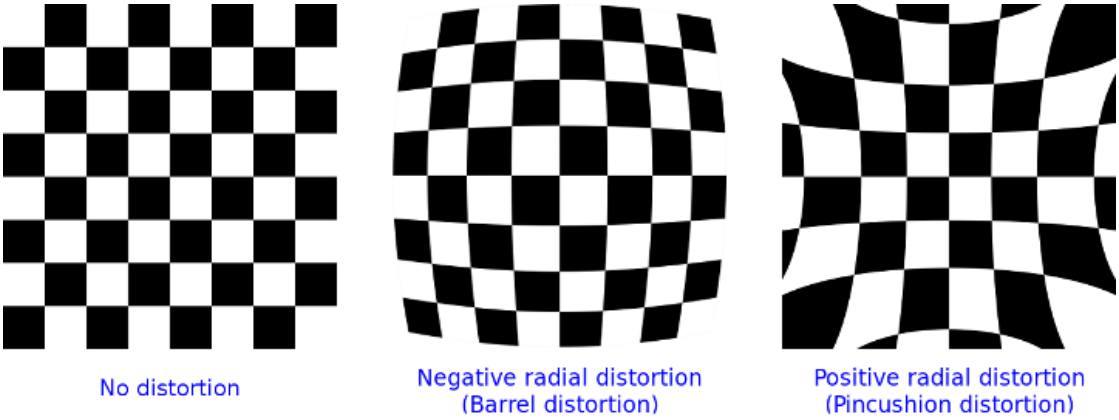


Figure 2.2: Radial distortion effects on a checkerboard pattern. Left: no distortion (ideal pinhole model). Center: barrel distortion (negative radial distortion, lines bow outward). Right: pincushion distortion (positive radial distortion, lines bow inward).

The distorted point coordinates $(\hat{u}, \hat{v})$ are computed from the normalized undistorted coordinates (x_n, y_n) as [12]:

$$x_d = x_n(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_n y_n + p_2(r^2 + 2x_n^2) \tag{2.2}$$

$$y_d = y_n(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y_n^2) + 2p_2 x_n y_n \tag{2.3}$$

where $r^2 = x_n^2 + y_n^2$, the coefficients k_1, k_2, k_3 are the radial distortion parameters, and p_1, p_2 are the tangential distortion parameters. OpenCV implements this model through the plumb_bob distortion convention [9].

The RealSense camera used in this work follows the radial–tangential distortion model (referred to as `plumb_bob` in *ROS* convention), but the *Red-Green-Blue color model (RGB)* stream provided by the camera driver is already rectified, meaning distortion has been corrected in the published images. As shown in Listing 4.2, all distortion coefficients are zero for the rectified stream, so the camera effectively behaves as a pinhole camera for the purposes of this work.

2.2.1.3. Fisheye Model (Kannala-Brandt)

Wide-angle cameras with fields of view exceeding approximately 120° cannot be adequately modeled using the standard pinhole and radial-tangential distortion framework. For these cameras, the fisheye model provides a more appropriate projection function.

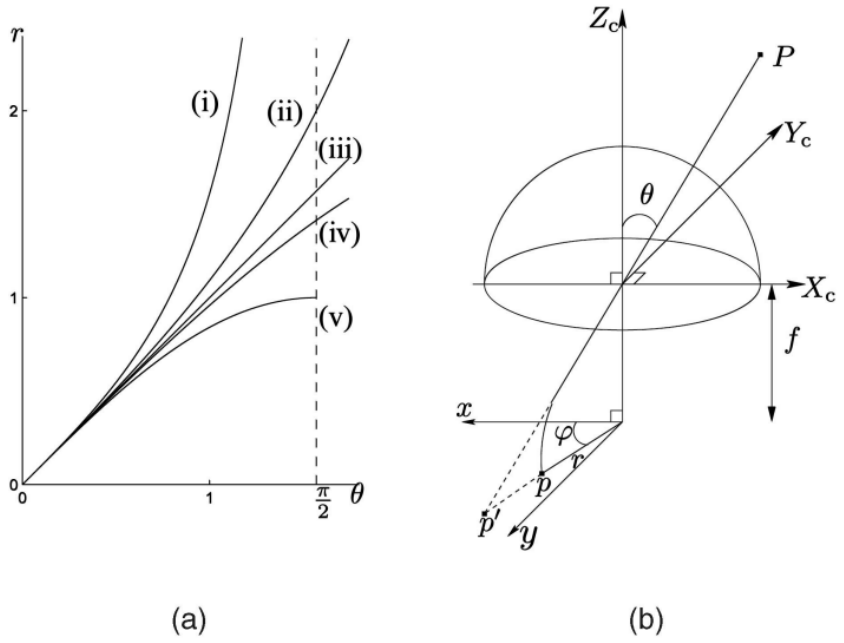


Figure 2.3: Fisheye camera model projection geometry. (a) Different projection models showing the relationship between incident angle θ and image radius r: (i) stereographic, (ii) equidistance, (iii) equisolid angle, (iv) orthogonal, (v) perspective (pinhole). (b) Geometric representation of the fisheye projection where point P projects to point p based on the incident angle θ and focal length f.

The Kannala-Brandt fisheye model [13] uses a different projection relationship. Figure 2.3 (adapted from [13]) shows the geometric projection model and different fisheye projection functions. Instead of the standard perspective projection, the model maps the angle θ between the 3D point and the optical axis to a radius r in the image plane. For a 3D point $\mathbf{P} = [X, Y, Z]^T$, the incident angle is computed as [13]:

$$\theta = \arctan\left(\frac{\sqrt{X^2 + Y^2}}{Z}\right) \tag{2.4}$$

The distorted radius is then modeled as a polynomial in θ:

$$r_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{2.5}$$

where k_1, k_2, k_3, k_4 are the fisheye distortion parameters. The final image coordinates are obtained by scaling and centering using the camera intrinsics [11]:

$$u = f_x \frac{x}{r} r_d + c_x \tag{2.6}$$

$$v = f_y \frac{y}{r} r_d + c_y \tag{2.7}$$

where $r = \sqrt{x^2 + y^2}$ and (x, y) are the normalized image plane coordinates corresponding to (X, Y, Z).

This model is used for the end-effector monocular camera in this work, which has a field of view of approximately 140° (see B.1). The calibration procedure to determine the fisheye distortion parameters is described in Section 4.2.

2.2.2. Intrinsics

The intrinsic parameters of a camera describe its internal geometric properties and define how points expressed in the camera coordinate frame are mapped onto the image plane. These parameters are independent of the camera pose in the environment and depend only on the camera optics and sensor configuration.

In computer vision, the intrinsic parameters are commonly represented by the intrinsic matrix $\mathbf{K}$, which relates normalized camera coordinates to pixel coordinates on the image plane. Using a pinhole camera abstraction, previously explained in Section 2.2.1.1, the intrinsic matrix is given by

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \text{[14]}, \tag{2.8}$$

where f_x and f_y are the focal lengths expressed in pixel units, c_x and c_y represent the principal point, and s is the skew coefficient between the image axes.

As illustrated in Figure 2.1, the focal lengths f_x and f_y encode the scaling between normalized camera coordinates and pixel coordinates along the horizontal and vertical image axes, respectively. The principal point (c_x, c_y) corresponds to the intersection of the optical axis with the image plane and typically lies close to the center of the image. The skew parameter s accounts for a non-orthogonality between the image axes and is usually zero for modern digital cameras.

Depending on the camera model, the intrinsic parameters may additionally include lens distortion terms to account for nonlinear projection effects introduced by the camera optics. The specific distortion model and corresponding parameters for the work on this thesis are addressed in the calibration procedure described in Section 4.2.

2.2.3. Extrinsics

While intrinsic parameters describe how points are projected onto the image plane, they do not include the pose of the camera with respect to the surrounding environment. In practice, a camera is not necessarily aligned with the reference frame in which objects are described, and its position and orientation must therefore be taken into account.

The extrinsic parameters describe the rigid transformation between the camera coordinate frame and a chosen reference frame. This transformation is defined by a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^{3\times1}$, which together encode the orientation and position of the camera relative to that reference frame. The transformation can be written in homogeneous form as

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{[14]}, \tag{2.9}$$

where $\mathbf{R}$ is an orthonormal rotation matrix and $\mathbf{t} = [\, t_x \; t_y \; t_z \,]^\top$ is the translation vector.

Using these extrinsic parameters, points observed in the camera frame can be transformed into other reference frames within the system. The definition and use of these transformations are further detailed in Sections 4.1 and 4.2.2.

2.3. Object Detection and Segmentation

Object detection is a fundamental component of the autonomous grasping pipeline. Convolutional neural network architectures, particularly the *YOLO* family, provide efficient solutions for real-time object detection by predicting bounding boxes and class probabilities from image data.

2.3.1. YOLOv7 Implementation

*YOLO*v7 [15] object detection was previously integrated into the *LRM* perception pipeline by Plácido de Castro [8]. In that work, the detector was combined with depth information from a RealSense camera to localize objects in three-dimensional space. The trained model and detection approach from that work are reused in this thesis without modifications to the model or training procedure.

This thesis takes that basis and builds on it by using the existing detection approach and adapting it into a modular, headless vision package. While the object detection model itself remains unchanged, its integration into the autonomous grasping pipeline was adapted to support both the RealSense camera and the new monocular end-effector camera.

2.4. Feature Extraction

Visual features are distinctive points or regions in an image that can be detected and matched across different views of the same scene. Feature extraction is a key step in many computer vision tasks, including image matching, object recognition, and camera pose estimation. A feature typically consists of a keypoint (a specific location in the image) and a descriptor (a numerical representation of the local image appearance around that keypoint). Reliable features must be repeatable across different viewpoints and distinctive enough to avoid ambiguous matches [16, 17].

Different feature extraction algorithms make different trade-offs between speed, robustness, and invariance to image transformations. This section describes two common feature detectors: *Oriented FAST and rotated BRIEF (ORB)* and *SIFT*. While both detect keypoints and compute descriptors, they differ in their computational cost and robustness to scale and rotation changes.

2.4.1. ORB

ORB is a feature detector and descriptor designed for real-time applications [18]. It combines the *Features from Accelerated Segment Test (FAST)* keypoint detector [19] with a modified version of the *Binary Robust Independent Elementary Features (BRIEF)* descriptor [20], adding orientation estimation to achieve rotation invariance.

ORB is computationally efficient and works well in scenarios where speed is critical. However, it is not inherently scale-invariant, meaning it may miss features when images are captured at different scales or distances. This makes it less suitable for applications where the camera viewpoint or zoom level changes significantly between images [11].

2.4.2. SIFT

Lowe developed the Scale-Invariant Feature Transform (*SIFT*) [16] algorithm to detect and describe local image features that remain stable across changes in scale and rotation. The algorithm creates a pyramid of progressively blurred and downsampled images, then finds keypoints by locating extrema across adjacent levels in the Difference-of-Gaussians (*DOG*) space. This approach makes *SIFT* robust to changes in image scale and rotation.

Figure 2.4 illustrates the core concepts of *SIFT* [21]. The left side shows the scale-space pyramid, where the image is repeatedly convolved with Gaussian filters and downsampled to create multiple octaves at different scales. The right side shows the *DOG* computation, where keypoints are detected by finding local extrema across adjacent *DOG* images in the scale space.

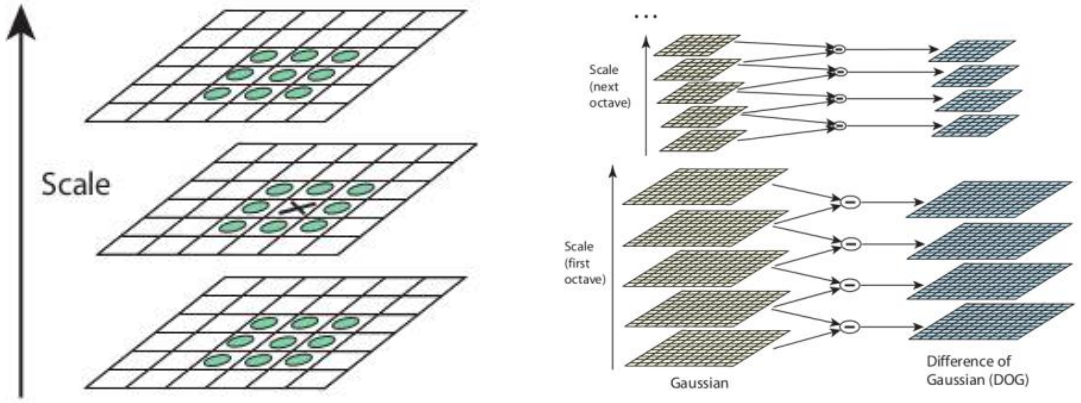


Figure 2.4: SIFT *scale-space representation and keypoint detection. Left: Scale-space pyramid with multiple octaves and Gaussian blur levels. Right:* DOG *used to identify keypoints by finding local extrema.*

For each detected keypoint, *SIFT* computes a 128-dimensional descriptor based on the gradient orientations in the local image region around the keypoint. The descriptor is designed to be invariant to scale, rotation, and partially invariant to changes in illumination and viewpoint [16]. *SIFT* handles viewpoint and scale changes more reliably than faster alternatives such as *ORB*, which was designed with computational efficiency as a primary objective [18].

The specific implementation and parameter choices for *SIFT* used in this work are described in Section 7.1.3.

2.5. Feature Matching

After extracting features from two images, the next step is to establish correspondences between them by finding which features in one image correspond to features in the other. This is known as feature matching. The goal is to match descriptors based on their similarity, typically measured using a distance metric such as Euclidean distance for *SIFT* descriptors.

Feature matching is challenging because descriptors from the same physical point may not be identical due to noise, viewpoint changes, or illumination differences [16]. Additionally, similar-looking image regions can produce similar descriptors, leading to ambiguous matches. To address these issues, matching algorithms typically find the nearest neighbors in descriptor space and apply additional tests to filter out unreliable correspondences [16].

2.5.1. Brute-Force Matcher

The Brute-Force matcher is the most straightforward approach to feature matching. For each descriptor in the first image, the matcher computes the Euclidean distance to every descriptor in the second image and returns the closest match (or the k closest matches for k-nearest neighbor search). This exhaustive search finds the globally optimal match rather than an approximation. The matcher is already implemented in the OpenCV library [11].

While Brute-Force matching is simple and accurate, it becomes computationally expensive for large numbers of features. For N features in each image, the algorithm requires $O(N^2)$ distance computations [11], making it impractical for real-time applications with thousands of features. However, it remains useful as a baseline for comparison and for cases where accuracy is more important than speed.

2.5.2. FLANN-Based Matcher

FLANN is a collection of algorithms optimized for fast nearest neighbor search in high-dimensional spaces [22]. Unlike Brute-Force matching, *FLANN* uses approximate search methods that trade a small amount of accuracy for significant speed improvements.

Because *SIFT* descriptors are 128-dimensional floating-point vectors, approximate nearest neighbor methods such as *FLANN* can be used to reduce matching time while maintaining acceptable accuracy [22]. *FLANN* builds multiple randomized KD-trees that split the descriptor space hierarchically and allows efficient approximate neighbor search without exhaustively comparing all descriptors.

The computational complexity of *FLANN* is approximately $O(N \log N)$ for typical cases [22], making it much faster than Brute-Force matching for large feature sets. The trade-off is that *FLANN* may occasionally miss the true nearest neighbor, but in practice, the approximate matches are usually sufficient for robust feature-based applications.

2.5.2.1. Lowe's ratio

A common technique to improve match reliability is Lowe's ratio test [16], which compares the distance to the nearest neighbor against the distance to the second-nearest neighbor. Matches where the best match is not significantly better than the second-best are rejected as ambiguous. This test works with both Brute-Force and *FLANN* matchers and is widely used to filter out unreliable correspondences.

The specific matching strategy and parameter choices used in this work, including the application of the ratio test and geometric verification, are described in Section 7.1.4.

2.6. Camera Pose Estimation

Camera pose estimation is the problem of determining the position and orientation of a camera relative to a scene or reference frame. This is typically expressed as a rigid transformation consisting of a rotation matrix $R \in \mathbb{R}^{3\times3}$ and a translation vector $t \in \mathbb{R}^3$. Robotic manipulation requires pose estimates accurate to within a few millimeters, while augmented reality applications can tolerate errors up to several centimeters [10].

Different pose estimation methods exist depending on the available information. When corresponding points are observed in two or more images without known 3D structure, methods such as the Essential matrix (for calibrated cameras) or Fundamental matrix (for uncalibrated cameras) can be used to recover relative camera motion [10]. However, when 3D coordinates of scene points are known, either from depth sensors or prior reconstruction, the *PNP* problem provides a more direct approach to pose estimation. In this work, 3D information is available from the RealSense depth camera, making *PNP* a good choice for pose estimation.

2.6.1. The Perspective-n-Point Problem

The *PNP* problem involves estimating camera pose given a set of 3D points in world coordinates and their corresponding 2D projections in the image [23]. This is particularly useful when depth information is available, as it allows direct computation of the 3D-2D correspondences, or matches, needed for pose estimation.

The minimal *PNP* problem requires at least four non-coplanar 3D-2D point correspondences to uniquely determine the camera pose [24]. With more correspondences, the pose can be estimated more reliably, and overdetermined solutions can be computed using least-squares optimization to minimize reprojection error [23]. Reprojection error is defined as the Euclidean distance between the observed image points and the projected 3D points obtained from the estimated camera pose.

One challenge in *PNP* estimation is handling outliers in the correspondence set. Mismatched features, inaccurate depth measurements, or ambiguous descriptors can introduce incorrect correspondences that degrade the pose estimate. To address this, *RANSAC* is commonly used [24]. *RANSAC* iteratively selects random minimal subsets of correspondences, estimates a pose hypothesis from each subset, and identifies the hypothesis that is consistent with the largest number of correspondences (inliers). This approach is robust to high outlier ratios and is widely used in practical pose estimation systems [10].

Once an initial pose is estimated, it can be refined using non-linear optimization methods such as Levenberg-Marquardt or bundle adjustment, which minimize the reprojection error across all inliers [10]. This refinement step typically improves pose accuracy by reducing residual errors in the projection model.

The specific *PNP* formulation, *RANSAC* parameters, and refinement strategy used in this work are described in detail in Section 7.2.

2.7. Kinematics

Another important component of the system is the robotic arm. At the stage of this work, the arm implementation was already complete with full functionality and mathematical description through a *Unified Robotics Description Format (URDF)* model, forward kinematics model, inverse kinematics provided by a *DLR* tool and elasto-kinematic compensation for arm elasticity during motion. Conenna [25] developed these models and calibration procedures. The following sections provide a brief overview of the forward and inverse kinematics implemented by Conenna's work. For this thesis, these models were used without modification to the kinematic model. The main contribution was integrating the robotic arm workspace into the rover and map coordinate frames, which is discussed in Sections 4.1, 5 and 8.

2.7.1. Forward Kinematics

Forward kinematics computes the position and orientation of the end-effector given the joint angles of the robotic arm [26]. For a serial manipulator, this involves chaining the transformations from the base to each successive joint until reaching the end-effector. The *LRM* robotic arm has six *DOF*, allowing it to reach positions and orientations within its workspace.

The forward kinematics model uses the *Denavit-Hartenberg (DH)* parameter convention to define the geometric relationship between consecutive joints [26]. Each joint contributes a transformation that depends on its current angle, and multiplying these transformations in sequence gives the end-effector pose relative to the arm base. Conenna [25] developed the complete *DH* table and forward kinematics implementation for the *LRM* arm, including the *URDF* model used throughout this work.

2.7.2. Inverse Kinematics

Inverse kinematics solves the opposite problem: given a desired end-effector position and orientation, it computes the joint angles needed to reach that pose [26]. This is essential for manipulation tasks, where the target position is known in Cartesian space but the arm must be commanded in joint space.

For the inverse kinematics, an in-house tool called the *Robotics Motion Planner Library (RMPL)*, developed at *DLR*, is used. This library requires only the *URDF* model as input and provides functions to move links named in the *URDF* to specific coordinates in Cartesian space. The library handles the inverse kinematics computation internally, removing the need to implement custom solvers for this specific arm configuration. During grasping operations, the system specifies the target gripper position in the `boogie_base`

frame, and the library computes the required joint angles to achieve that configuration.

2.7.3. Elasto-Kinematic Model

In addition to the standard kinematic models, Conenna [25] developed an elasto-kinematic model to compensate for mechanical deflections in the arm structure under load. The servos and links exhibit elastic deformation during motion, especially when the arm extends or carries objects. This model estimates the actual end-effector position accounting for these deflections, improving positioning accuracy compared to the rigid-body kinematic model alone.

The elasto-kinematic calibration procedure developed by Conenna was kept in this work and forms the basis for the servo-level calibration discussed in Section 4.3. The calibration approach was reused and adjusted to account for the specific hardware configurations of each servo type discussed in Section 4.3.1.

2.8. Visual Servoing

Visual servoing refers to the use of visual feedback to control the motion of a robot. Instead of relying only on joint encoders or external position sensors, the robot uses information extracted from camera images to guide its movements toward a desired configuration [27]. This closed-loop approach compensates for calibration errors, model uncertainties, and disturbances by continuously adjusting the robot's motion based on observed visual features.

Two main approaches exist: *Image-Based Visual Servoing (IBVS)* and *Position-Based Visual Servoing (PBVS)*. The choice between them depends on the application, available sensors, and computational constraints.

2.8.1. Image Based Visual Servoing

IBVS controls the robot directly in the image space by minimizing the error between current and desired image features [27]. Features such as point coordinates, line orientations, or region centroids are extracted from the camera image, and the control law computes robot velocities that drive these features toward their target values. This approach requires computing the image Jacobian, which relates changes in image feature positions to camera velocity. *IBVS* assumes continuous velocity-level control of the robot and real-time computation of the feature-to-velocity mapping.

The main advantage of *IBVS* is that it operates directly on sensor measurements, which can make it robust to calibration errors in the camera-to-robot transformation. However, it requires a well-conditioned Jacobian and smooth feature tracking throughout the motion.

2.8.2. Position Based Visual Servoing

PBVS estimates the three-dimensional pose of the target or end-effector from image measurements and then controls the robot in Cartesian space [27]. This approach reconstructs the spatial relationship between the camera and the target, computes the desired end-effector motion in three-dimensional coordinates, and commands the robot accordingly. *PBVS* requires accurate camera calibration and pose estimation methods such as *PNP*.

The advantage of *PBVS* is that the control task is formulated in Cartesian space, which simplifies trajectory planning and allows direct specification of desired positions and orientations. However, it is more sensitive to calibration errors compared to *IBVS*, as errors in pose estimation directly affect the commanded motion.

Classical visual servoing assumes velocity-level robot control and continuous image feedback. For servo-based systems that accept position commands rather than velocities, a discrete adaptation is needed. Section 7.4 presents a simplified image-based correction method developed for this work, which operates iteratively rather than continuously to accommodate the position-controlled servo interface of the robotic arm.

2.9. Gripping Detection

Autonomous manipulation requires confirming that an object has been grasped before attempting to lift or transport it. In most robotic systems, this confirmation comes from sensors mounted at the gripper.

Force and torque sensors integrated into the gripper fingers or wrist measure contact forces during closure [28, 29]. These sensors detect successful grasps and can estimate grip strength. Tactile sensor arrays provide more detailed information by measuring contact distribution across the gripper surface [30, 31]. Vision-based methods use cameras mounted on the gripper or in the workspace to observe the closing motion and infer grasp success from visual feedback [32].

When sensor integration is limited by cost or hardware constraints, motor current sensing offers an alternative. Kim and Song [33] demonstrated sensorless grasping force control using current measurement in robotic grippers. The motor experiences increased resistance when the gripper contacts an object, which appears as elevated current draw. This method has been used in industrial grippers and low-cost robotic systems [34].

For this work, the gripper uses a linear mechanism actuated by a single servo motor (explained further in paragraph 3.1.1). The Middleboard (Section 3.1.1) does not provide additional input channels for force or tactile sensors, which constrains the available sensing modalities for grasp confirmation. Although vision-based grasp verification is

possible in principle, its reliability depends on camera placement, field of view, and the ability of the detection model to operate under close-range contact configurations.

A current sensor on the Middleboard measures the total electrical current drawn by all seven arm motors, including the gripper actuator. While this does not isolate the gripper current directly, it provides sufficient information for grasp detection when the arm remains stationary during closing. The implementation choices and a comparative discussion of sensing alternatives are presented in Section 8.3, with further analysis in Section 9.4.6.

3. System Design and Architecture

In this section, the *LRM* system components will be described, splitting them into hardware and software. For the hardware section, only the main computer, the middle board computer, the robotic arm, the gripper, and the additional camera placed on the end-effector will be described in detail. Further mechanical and electrical descriptions can be found in [4].

3.1. Overall System Architecture

3.1.1. Hardware

As seen in Figure 3.1 (credit to Placido de Castro [8]), two configurations of the rover are shown for clarity: one with the robotic arm mounted and one without, with the chassis opened to expose internal components. Each rover consists of six wheels with drive and steering mechanisms arranged in three articulated modules located at the left, right, and rear of the platform, as well as a pan-and-tilt mechanism on which a RealSense depth camera is mounted. The configuration used in this work includes a 6 *DOF* robotic arm built using low-cost hobby-grade servo motors and a low-cost monocular camera mounted on top of the end-effector.

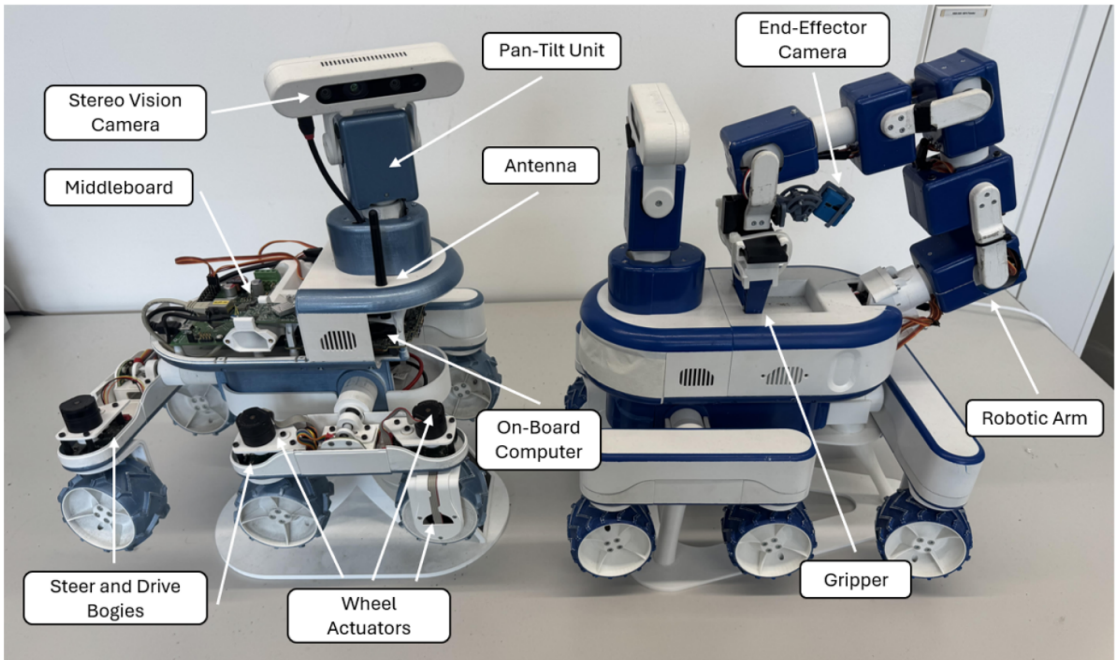


Figure 3.1: Overview of the LRM system components.

Computing Platform

The *LRM* has an Intel *Next Unit of Computing (NUC)* running openSUSE as its onboard computer. The Intel NUC´s compact size (10 cm × 10 cm× 2.5 cm) [35], relatively low power consumption (typically 15–28 W) [36], and computing power has proven sufficient for *ROS* and computer vision tasks. All processing algorithms, state machines, and control software run on this computer.

Power Distribution and Communication

The *NUC* is connected to a custom module called Middleboard, which includes an ATXMega32A4U microcontroller. The Middleboard serves two primary functions:

- **Power Management:** Distributes power from the onboard 14.7 V LiPo batteries to the rover subsystems, including the computing unit, communication hardware, sensors, and actuators. The LiPo batteries are connected directly to the system and are recharged manually.
- **Communication Interface:** Establishes and manages communication between the NUC and peripheral components

The Middleboard handles the distribution of data to the individual bogies, the robotic arm, the *Inertial Measurement Unit (IMU)*, and the current sensors, acting as a bridge between the high-level computing platform and the low-level hardware components.

Robotic Arm

The rover is equipped with a 6 *DOF* robotic arm actuated by hobby-grade servo motors. Each joint is position-controlled through commanded rotation angles, which are sent from the high-level software via the Middleboard to the hardware. The mechanical design and integration of the robotic arm were developed prior to this work and are not modified within the scope of this thesis. The forward kinematics model and joint configuration used for arm control are based on the formulation presented in the background (Section 2.7) and in [4].

Gripper Actuation Mechanism

The gripper is actuated using a single hobby-grade servo motor that, when rotated, drives a linear transmission mechanism. Although the gripper motion is linear, the control input is defined in terms of the servo rotation angle. The servo rotation results in a linear displacement of the gripper jaws through the transmission, causing the gripper to open or close. Gripper states, such as `home` or `fully_open`, correspond to predefined servo angles determined experimentally.

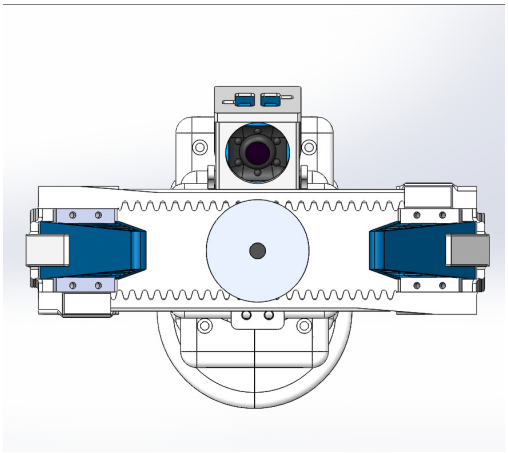


Figure 3.2: New camera mount and gripper linear mechanism.

End-Effector Camera

The chosen end-effector camera is the SQ11 Full HD 1080P from Ordinoscope. This camera was selected due to its low cost (approximately 25 €), low weight (15 g), and its support for Full HD video recording at resolutions up to 1920 × 1080 pixels [37]. In the context of this work, a resolution of 1280 × 720 pixels was used and proved sufficient. One of the challenges encountered during camera integration was the interface cable. The camera uses a Mini 8-pin USB interface with a relatively short cable. To pass the cable through the robotic arm joints without compromising the range of motion, a custom cable with an additional USB extension (male to female) was integrated to provide sufficient length.



Figure 3.3: SQ11 Full HD 1080P camera.

3.1.2. Software

Building on the original system design, Figure 3.4 provides an overview of the system software architecture and the interaction between its main components. The Simulink model, the arm planning server, the autonomous navigation stack, the communication module, and the Middleboard were developed by previous researchers [4]. In this work, the components developed or extended are described in detail.

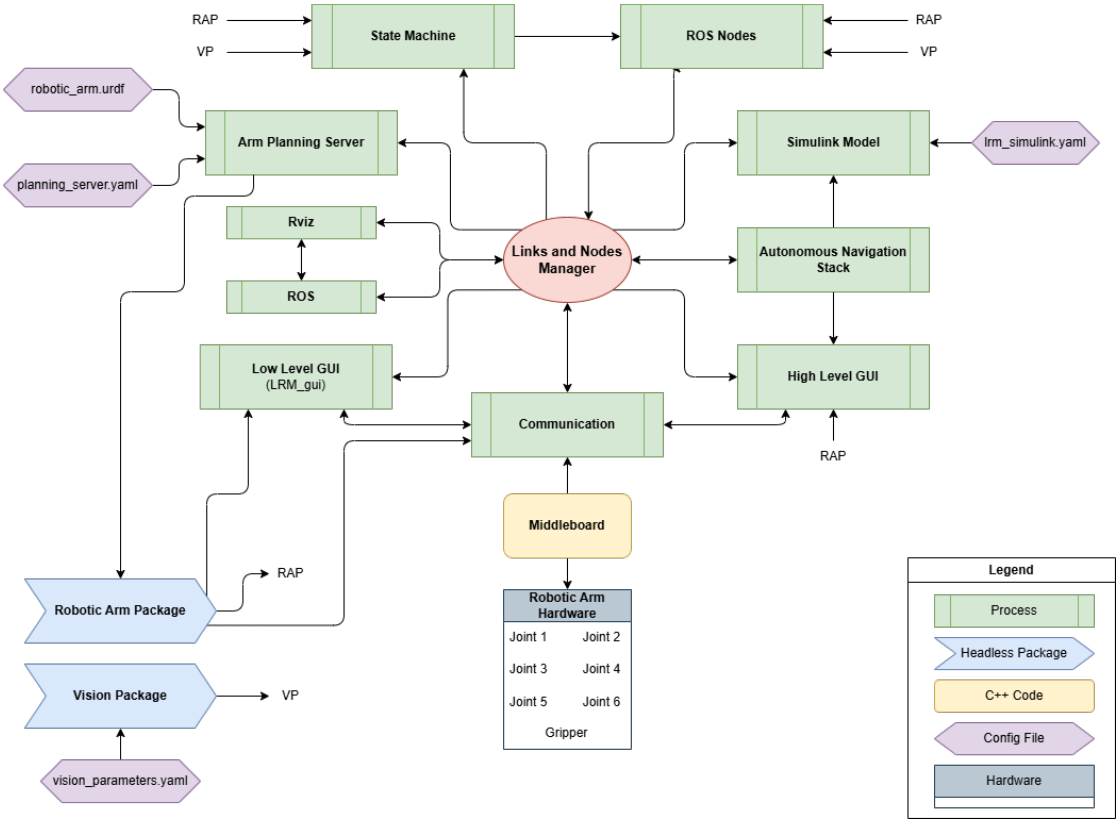


Figure 3.4: Overview of the system software architecture and interactions between its main components. Rectangular blocks represent software processes or modules, arrows show data flow and communication, and configuration files are shown separately and linked to the components that consume them. Hardware components are depicted at the bottom of the diagram. The Links and Nodes Manager acts as the central coordination layer between high-level software modules, simulation and visualization tools, and low-level hardware interfaces. RAP and VP refer to the Robotic Arm Package and Vision Package, respectively.

Links-and-Nodes Framework

The core of the software architecture is *LN*, a system deployment framework developed at *DLR*. It provides a centralized way to launch, monitor, and interconnect all major processes of the LRM system, including RViz, Simulink models, *ROS* nodes, and Python scripts. Similar to *ROS*, *LN* organizes the system into nodes (processes) and topics (messages between processes). In practice, LN is mainly used to manage startup order, ensure consistent data routing between components, and simplify debugging during integration.

Development Tools

Several software tools were developed and used throughout this thesis. *LN* serves as the main deployment environment, from which the low-level GUI, high-level GUI, ROS processes, and RAFCON state machines are launched. The low-level GUI (Section A.1)

and the high-level GUI (Section A.2) were created specifically for testing and development purposes. More details about these tools can be found in the Appendix A.

ROS Framework

In addition to the *LN* framework, *ROS* Melodic[1] was integrated into the project by previous students, specifically to handle autonomous navigation tasks. The integration of *ROS* was primarily motivated by the availability of mature navigation, visualization, and transformation tools, such as the navigation stack, RViz, and the *TF* framework. To achieve the integration, *ROS* nodes are executed as *LN* processes, which allows cross-communication between both frameworks. In this project, the focus is on the *ROS* node responsible for robotic arm control, which is further detailed in Section 4.1.

An important part of this integration is the development of custom *ROS* messages for arm-specific data exchange. The `GripperState` message contains the gripper's position name and a corresponding value expressed in degrees. This value corresponds to the commanded rotation angle of the gripper servo, which actuates the gripper through the linear transmission mechanism explained in Section 3.1.1. The `ArmControl` message contains two boolean flags: one for enabling arm control and another for enabling arm power. These flags are separated because the low-level middleware board and system architecture were designed such that boogie wheels and arm control are mutually exclusive operations, with the arm enable flag controlling this mode switch. These custom messages are used throughout the system, including within the *RAFCON* state machine framework. A third important message, `JointState` [38], is used for the joint states of the robotic arm. The specifics of the message definitions can be found in Appendix C.

Robotic Arm Package

The core logic for the robotic arm is implemented in a headless Python package designed for reusability across different control interfaces (low-level *GUI*, high-level *GUI*, *ROS* nodes and *RAFCON* state machines). The main classes follow a Singleton pattern [39], ensuring that instances are created only once and their properties persist across the autonomous cycle. This design prevents inconsistent joint states and duplicated command publishers when switching between manual and autonomous control modes.

The package contains three main components: The `MotionPlanning` class serves as a wrapper for the arm planning server (*RMPL*) and handles all kinematic computations. It exposes functions to plan self-collision-free paths from given initial joint states to either target joint configurations or Cartesian coordinates (x, y, z) with optional orientation constraints. This orientation constraint feature allows the planner to generate paths that maintain the end-effector pointing downward toward the ground throughout the motion, which is particularly useful for grasping operations. The `RoboticArm` class serves as the main entry point, managing joint state subscriptions, arm state, arm control commands,

[1] At the moment, ROS2 is also available for development, but its not the focus of the project to port from *ROS* to *ROS*2, hence all implementation was kept in this old framework.

and gripper operations. Moreover, the `ArmCommunicationManager` centralizes all ROS and Links-and-Nodes topic definitions and message publishing, providing a common place for *ROS* and *LN* publishers.

Vision Package

The vision package follows the same architectural principles as the robotic arm package and is dedicated to camera management and visual processing. Camera intrinsic and extrinsic parameters are defined using the Singleton pattern, ensuring calibration data remains consistent. The package includes OpenCV wrappers for SIFT-based feature extraction, Brute-Force and Flann based matching (properly explained in Section 7.1), as well as YOLOv7-based object detection capabilities. The object detection implementation builds upon work by Placido de Castro [8], adapted for both the pan-tilt and the end effector camera.

RAFCON Architecture

RAFCON serves as the main tool for autonomous control logic. This hierarchical state machine framework allows the implementation of reusable and hierarchically structured state machines, which simplifies the debugging and reconfiguration of individual behaviors. One of the main contributions of this work is a master state machine that handles the complete autonomous grasping pipeline. This master state machine manages a set of distinct operational states and executes transitions between them based on system inputs and execution outcomes. The general control flow will be discussed further in Section 5.2.

4. System Setup and Calibration

Before implementing any high-level logic, system setup and calibration were performed to establish the basic components needed to accomplish the full pipeline. This chapter describes the setup and calibration procedures that allow the *LRM* to mathematically determine where the arm joints are located in a given space, understand how the camera perceives the environment and translate this into usable data, and finally tune each servo to achieve improved precision during manipulation.

When a human tries to grab an apple, they first look at it, then instruct the brain to move the arm from its current position to where the apple is located, and finally use the hand to grasp it. This process relies on perception (seeing the apple) and localization (knowing where the apple is with respect to the arm). Translating this concept to the rover requires establishing a consistent spatial relationship between detected objects and the robotic arm workspace. Without proper calibration and knowledge of the relative pose between the object, the cameras, and the robotic arm, such a task would not be feasible.

To address this, three key elements are required: an inertial reference frame that relates all relevant coordinate frames (object, rover, arm joints, and cameras), as described in Section 4.1; proper camera calibration to correctly interpret visual data (Section 4.2); and servo calibration to achieve precise millimeter-level movements (Section 4.3).

The final section of this chapter, Section 4.4, discusses the mission setup and the connection between De Castro's [8] work and the present implementation. It outlines the sequence of steps that the autonomous pipeline follows and clarifies how previously developed components are integrated into the current system.

4.1. Coordinate System Definition

The robotic arm grasping pipeline relies on a consistent mathematical representation of the rover, the robotic arm, and the cameras. This representation is based on rigid body transformations that describe the relative poses between coordinate frames. In this work, the kinematic structure of the robotic arm and the rover is defined by existing models developed in prior work [25], which are not modified within the scope of this thesis.

The transformation relationships between frames are managed and exchanged using the *TF* framework. This allows object detections generated in the camera frames to be expressed in the global map frame and subsequently transformed into the robotic arm workspace.

The coordinate system definition combines static transforms that describe the fixed mechanical structure of the rover with dynamic transforms that reflect, in real time, the articulated state of the robotic arm. The following sections describe the overall *TF* tree structure as well as its integration within the system.

4.1.1. TF Tree Architecture

The *TF* tree defines how all coordinate frames in the system relate to each other. This ensures that positions and orientations can be consistently expressed across the rover and the robotic arm. At the top of the hierarchy are the `map` and `odom` frames, which provide global and local references for the rover's navigation. These frames are handled by the localization and mapping subsystem, which is outside the scope of this thesis and are mentioned here only for completeness.

The frames relevant to the arm start at `base_link`, which represents the main rover body[1]. From here, a fixed transform with translation and rotation connects to the `boogie_base` frame as follows:

$$
{}^{\texttt{base_link}}\mathbf{T}_{\texttt{boogie_base}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{t} = \begin{bmatrix} -0.18 \\ 0 \\ -0.15 \end{bmatrix} \text{ m}, \qquad \mathbf{r}_{\text{rpy}} = \begin{bmatrix} 0 \\ 0 \\ \pi \end{bmatrix} \text{ rad}, \tag{4.1}
$$

where the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ is obtained from the roll–pitch–yaw representation. [40]

[1] This frame moves with respect to the `map` frame whenever the rover moves. At startup, it is located at the center of the rover's body at coordinates $(0, 0, 0)$.

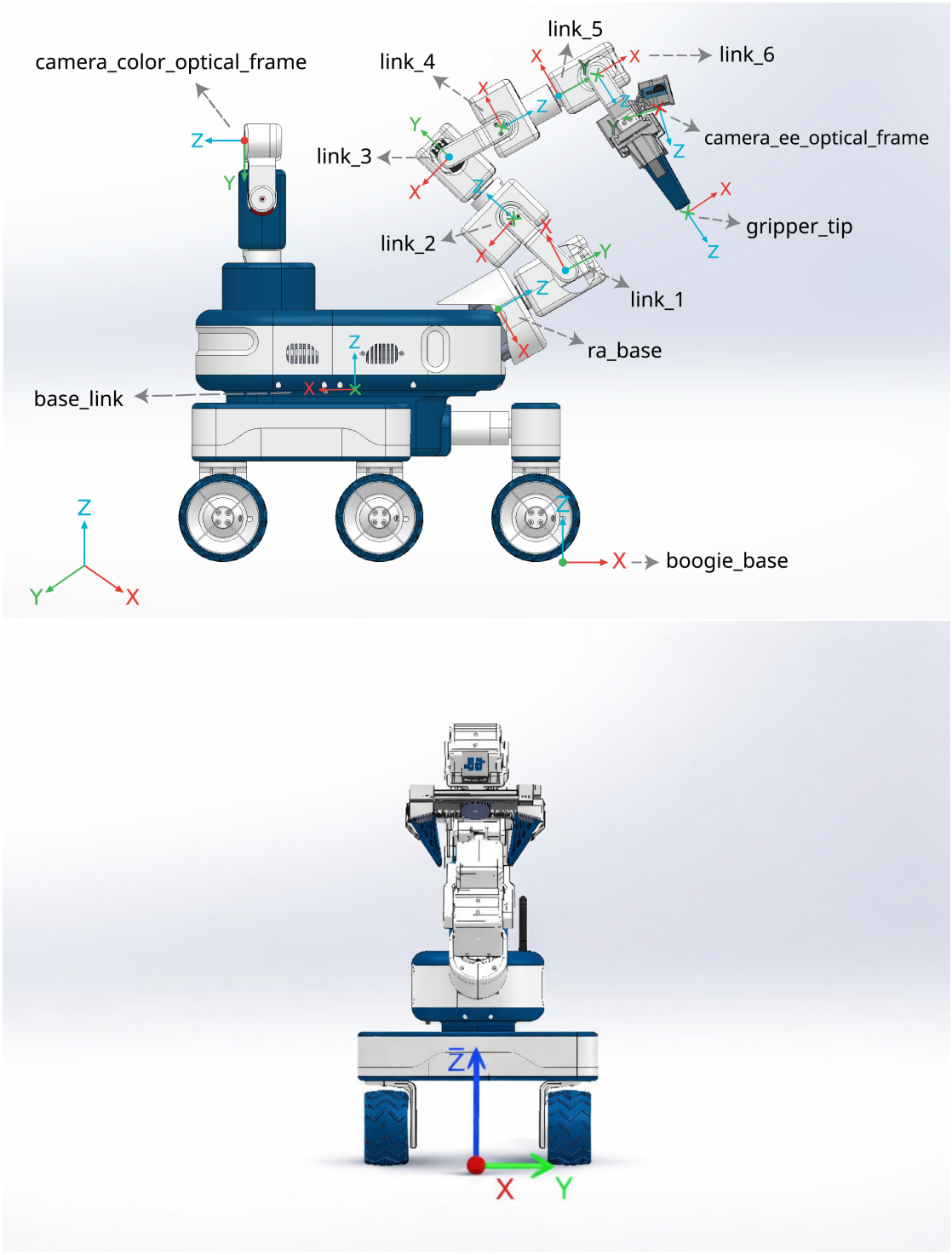


Figure 4.1: Relevant coordinate frames of the robotic arm and vision system. Top: side view showing the transformation chain from `base_link` *and* `boogie_base` *through the arm links (*`link_1`*–*`link_6`*) to the end-effector, gripper tip, and camera optical frames. Bottom: rear view illustrating the* `boogie_base` *frame, which defines the working space of the robotic arm. All frames follow the right-handed convention used in the system* TF *tree.*

At this point, `boogie_base` serves as the origin $(0, 0, 0)$ for the arm workspace. This frame is located at ground level, in the center of the rear boogie suspension[2], as shown in Figure 4.1. Another fixed transform then connects `boogie_base` to `ra_base`:

$$ {}^{\texttt{boogie_base}}\mathbf{T}_{\texttt{ra_base}}: \quad \mathbf{t} = \begin{bmatrix} -0.034975172 \\ 0 \\ 0.23608235 \end{bmatrix} \text{m}, \qquad \mathbf{r}_{\text{rpy}} = \begin{bmatrix} 0 \\ 0.8203047 \\ 0 \end{bmatrix} \text{rad.} \tag{4.2} $$

This frame represents the actual root of the robotic arm. From `ra_base`, the kinematic chain, explained in Section 2.7.1, continues through the individual arm links (`link_1` through `link_6`) and ends at the `gripper_tip`.

The camera mounted on the end-effector is represented by two fixed transforms. First, `camera_ee_rgb` is defined relative to the `gripper_tip` frame and represents the physical mounting location of the camera on the arm:

$$ {}^{\texttt{gripper_tip}}\mathbf{T}_{\texttt{camera_ee_rgb}}: \quad \mathbf{t} = \begin{bmatrix} 0.03049 \\ 0 \\ -0.09385 \end{bmatrix} \text{m}, \qquad \mathbf{r}_{\text{rpy}} = \begin{bmatrix} 0 \\ -0.3141593 \\ 0 \end{bmatrix} \text{rad.} \tag{4.3} $$

Second, `camera_ee_optical_frame` is defined relative to `camera_ee_rgb` and represents the camera's optical center, i.e., the point from which image rays are projected:

$$ {}^{\texttt{camera_ee_rgb}}\mathbf{T}_{\texttt{camera_ee_optical_frame}}: \quad \mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{m}, \qquad \mathbf{r}_{\text{rpy}} = \begin{bmatrix} 0 \\ 0 \\ 1.5707963 \end{bmatrix} \text{rad.} \tag{4.4} $$

This two-frame convention follows the standard *ROS* camera frame convention [41], where the `camera_ee_rgb` frame represents the physical camera body and the optical frame corresponds to the internal projection center used for image-based coordinate transformations. The optical frame orientation follows the OpenCV convention, with the x-axis pointing right in the image, the y-axis pointing downward, and the z-axis pointing forward along the viewing direction [9].

This frame structure allows visual observations from the end-effector camera to be transformed into the arm workspace. It also enables transformations between the RealSense camera optical frame, where objects are initially detected, and the `boogie_base` frame, where manipulation is performed.

[2]The result of the transformation yields a workspace orientation such that the positive x-axis points forward from the arm, the positive y-axis points to the left, and the positive z-axis points upward from the ground.

In addition to the robotic arm, a second kinematic branch describes the RealSense depth camera mounted on the rover. Prior to this work, a fixed transform connected `base_link` to the pan–tilt base frame `pt_base`. The pan–tilt joint itself was represented by the frame `pt_link_base`, whose orientation relative to `pt_base` was updated dynamically at runtime. A final fixed transform connected this joint frame to the camera frame, denoted as `camera_link`, followed by a fixed transform to the optical frame `camera_color_optical_frame`.

In the present system, this configuration was simplified by removing the pan–tilt mechanism from the *TF* tree and defining the RealSense camera as a fixed frame with a downward tilt of approximately 50°. The motivation for this design choice is discussed in Sections 2.3 and 6.2. This branch defines the full spatial relationship between the rover and the depth camera used for global object detection.

The complete integration of all frames and transforms into the system architecture is further detailed in Section 5.1.1.

4.2. Camera Calibration

In computer vision, camera calibration is required to correctly relate image measurements to three-dimensional geometry and to keep consistency across coordinate transformations in the grasping pipeline. As seen in Section 2.2, intrinsic parameters are used to describe the internal camera geometry and lens distortion, while extrinsic parameters define the rigid transformation between the camera and the end-effector reference frame (`gripper_tip`).

4.2.1. Intrinsic Calibration

4.2.1.1. Monocular End Effector Camera

The intrinsic parameters of the end-effector camera were estimated in-house using a checkerboard-based calibration procedure. A calibration tool from *DLR* institute was used to model the camera using the Kannala–Brandt projection model.

The calibration estimates the focal lengths (μ, ν), the principal point (u_0, v_0), and the radial distortion coefficients (k_2, k_3, k_4, k_5). The resulting intrinsic parameters used throughout the vision pipeline are summarized in Listing 4.1.

Listing 4.1: Estimated intrinsic parameters of the end-effector camera

```
model_type: KANNALA_BRANDT
image_width: 1280
image_height: 720
projection_parameters:
  mu: 1640.83
  mv: 1640.83
  u0: 664.69
  v0: 391.84
```

```
  k2: 0.467
  k3: -0.674
  k4: -0.279
  k5: 0.0
```

In this listing, the parameters μ and ν describe the focal lengths of the camera expressed in pixel units along the horizontal and vertical image axes. The parameters u_0 and v_0 define the location of the optical axis intersection with the image plane in pixel coordinates. Finally, the coefficients k_2 to k_5 correspond to the radial distortion terms of the Kannala–Brandt projection model, which account for nonlinear lens effects (distortion). The image width and height specify the resolution at which the intrinsic parameters were estimated.

4.2.1.2. RealSense Depth Camera

In addition to the end-effector camera, the Intel RealSense *RGB* camera [42] is used during the perception stage to get the reference *RGB*–Depth data. The intrinsic parameters of the RealSense camera are not estimated in this work and are instead taken directly from the official *ROS* RealSense driver. When running the RealSense *ROS* package, the intrinsic calibration parameters for each camera stream are published via the standard `camera_info` topic.

These parameters are factory-calibrated and specific to the individual camera and the selected image resolution. For completeness, the intrinsic parameters used in this work are listed in Listing 4.2.

Listing 4.2: Intrinsic parameters of the RealSense camera obtained from the *ROS* `camera_info` topic

```
frame_id: camera_color_optical_frame
image_width: 848
image_height: 480
distortion_model: plumb_bob
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K:
[608.6726, 0.0, 424.0317,
 0.0, 608.5322, 238.4163,
 0.0, 0.0, 1.0]
R:
[1.0, 0.0, 0.0,
 0.0, 1.0, 0.0,
 0.0, 0.0, 1.0]
P:
[608.6726, 0.0, 424.0317, 0.0,
 0.0, 608.5322, 238.4163, 0.0,
 0.0, 0.0, 1.0, 0.0]
```

Here, $\mathbf{K}$ denotes the intrinsic camera matrix containing the focal lengths and principal point coordinates, while $\mathbf{D}$ represents the distortion coefficients of the pinhole (`plumb_bob`) model discussed in Section 2.2.1.2. The rotation matrix $\mathbf{R}$ is the identity, and $\mathbf{P}$

corresponds to the projection matrix used for image rectification. Using the *ROS*-provided intrinsic parameters avoids performing any additional calibration step.

4.2.2. Extrinsic Calibration

4.2.2.1. Monocular End Effector Camera

The extrinsic parameters of the end-effector camera were defined using measurements from the *Computer-Aided Design (CAD)* model (see Figure 4.2). Since the camera is rigidly mounted on top of the end-effector, its position and orientation with respect to the robotic arm, the rover, and the global frame are determined by the pose of the end-effector itself.

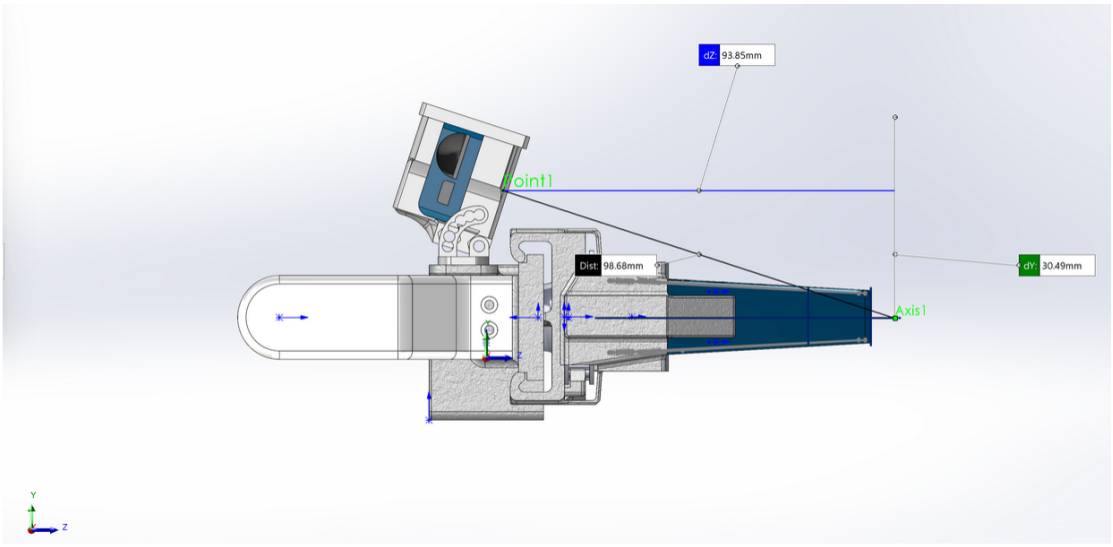


Figure 4.2: New camera mount and gripper seen from side view with dimensions.

As a result, defining a static transformation from the `gripper_tip` frame to the `camera_ee_rgb` frame is sufficient to fully describe the camera pose within the system.

This transformation, also stated in Equation 4.3, is represented using the standard homogeneous transformation matrix convention as:

$$ {}^{\texttt{gripper_tip}}\mathbf{T}_{\texttt{camera_ee_rgb}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \text{with} \quad \mathbf{t} = \begin{bmatrix} 0.03049 \\ 0 \\ -0.09385 \end{bmatrix} \text{m}, \tag{4.5} $$

where $\mathbf{R}$ is the rotation matrix corresponding to a roll–pitch–yaw rotation of

$$ \mathbf{r}_{\text{rpy}} = \begin{bmatrix} 0 \\ -0.3141593 \\ 0 \end{bmatrix} \text{rad}. \tag{4.6} $$

The pitch rotation of approximately -0.314 rad ($\approx -18°$) tilts the camera downward so that its *Field of View (FOV)* is centered on the gripper tips, ensuring that objects to be grasped remain within the camera image during manipulation.

4.2.2.2. RealSense Depth Camera

The extrinsic parameters of the RealSense depth camera are defined with respect to the rover base frame (`base_link`), which is located at $(0, 0, 0)$ by definition. For this work, the RealSense camera is rigidly mounted on the rover structure, hence the position is constant with respect to this frame.

The transformation is defined from the `base_link` frame to the `camera_link` frame. The `camera_link` frame serves as the parent frame for all camera-related frames published by the RealSense *ROS* driver, including the optical and stream-specific frames. As a result, defining this single static transformation is sufficient to fully describe the RealSense camera pose within the system.

Using the standard homogeneous transformation matrix notation, the transformation is written as:

$$ {}^{\texttt{base_link}}\mathbf{T}_{\texttt{camera_link}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \text{with} \quad \mathbf{t} = \begin{bmatrix} 0.09046 \\ 0.018 \\ 0.18149 \end{bmatrix} \text{ m}, \tag{4.7} $$

where the translation vector $\mathbf{t}$ corresponds to the measured position of the RealSense link frame relative to the rover base origin. The rotation matrix $\mathbf{R}$ corresponds to a roll–pitch–yaw rotation of

$$ \mathbf{r}_{\text{rpy}} = \begin{bmatrix} 0 \\ 0.87266 \\ 0 \end{bmatrix} \text{ rad}, \tag{4.8} $$

which represents a pitch rotation of approximately $50°$. This tilt aligns the RealSense camera downward toward the workspace in front of the rover, also discussed in Chapter 6.

4.2.3. Validation

The intrinsic calibration was validated using reprojection error. During calibration, checkerboard corner points are first detected in the image plane; these detected points are shown as green markers in Figure 4.3. Using the estimated intrinsic parameters, the corresponding three-dimensional checkerboard points are then reprojected back onto the image. The close alignment between the detected corner locations and their reprojected counterparts indicates that the camera model accurately maps between 3D geometry and image measurements. The average reprojection error was approximately 0.33 pixels, with a maximum error below 1 pixel, which confirms the intrinsic parameters work across different viewpoints.

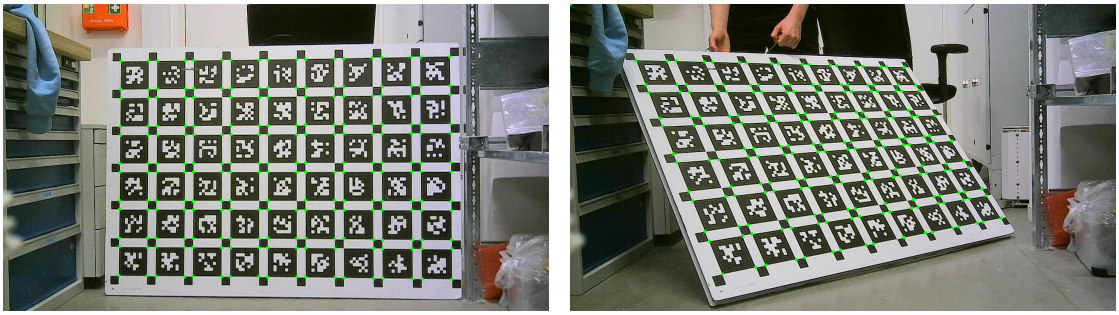


Figure 4.3: Reprojection of checkerboard corners using the estimated Kannala–Brandt intrinsic parameters from two different viewpoints.

It should be noted that the intrinsic parameters of the RealSense camera are not estimated within this work. These parameters are provided by the *ROS* RealSense driver via the `camera_info` topic. The same intrinsics were already used in previous work [8] on the LRM platform and are therefore considered reliable.

The extrinsic calibration was defined from the *CAD* model of the end-effector and camera mount. The resulting static transformation between the `gripper_tip` and `camera_ee_rgb` frames was validated using the *ROS TF* framework. In RViz, the relative orientation and position of the frames are visually consistent with the physical mounting of the camera on the end-effector (also shown in Figure 5.1). Additionally, querying the transformation using `tf_echo` returns translation and rotation values that match the parameters defined in the calibration. This confirms the correct integration of the extrinsic parameters within the system. For the RealSense camera, the extrinsic transformation was defined prior to this work and assumed to be correct, but was also verified by querying the corresponding *TF*.

4.3. Servo Configurations and Calibration

The *RC* servos (see specifications in Appendix B.3) used in the *LRM* robotic arm can be configured using the **AGFRC Servo Programmer** software, developed by the servo manufacturer [43]. This utility is used to configure several servo parameters via a USB interface. It allows each servo to be tuned according to its mechanical function in the arm, while also enabling protection features intended to reduce the risk of overload or mechanical damage.

4.3.1. Hardware-Level Servo Configuration

Each servo's internal electronics stores a set of adjustable parameters that affect its electrical, mechanical, and feedback behavior. Table 4.1 shows the parameters that were adjusted for the arm servos, taking into account that several other parameters were left

untouched.

Table 4.1: Hardware-level servo parameters configured for the LRM robotic arm

Parameter	**Description**	**Range**
Travel Range	Maximum allowed rotation of the servo.	1–255
Damping Factor	Reduces oscillations and jitter during motion.	50–600
PWM Power	Drive strength and available torque output.	0–100 %
Sensitivity	Input responsiveness; affects jitter and fine motion.	Low, Medium, High, or Ultra–High
Soft Start	Limits startup speed when powering the servo.	Levels 1–11
Overload Protection	Torque reduction under sustained load.	Levels 1–3, seconds and %
Lose PPM Protection	Behavior in case of signal loss.	Keep Position, Release, or Go Neutral

The robotic arm uses three servo types: A80BHMF (Appendix B.3.1) servos for joints 1, 2, and 3, A35BHL (Appendix B.3.2) servos for joints 4, 5, and 6, and an A20CLS (Appendix B.3.4) servo for the gripper. In contrast to the work presented by Conenna [25], multiple tests during arm operation revealed that achieving slow, steady motion without jitter (both in the high-level *GUI* and within the *RAFCON* state machine) could not be accomplished solely by adjusting the step size and update frequency of the joint commands sent to the *Pulse-Width Modulation (PWM)* output ports. Further investigation showed that several internal servo parameters also needed to be configured at the hardware level.

In practice, parameters such as motion limits, damping, power output, sensitivity, startup behavior, and overload protection had a noticeable influence on the smoothness and stability of the arm motion. Several iterations of tuning led to the final configuration described below. Soft start was enabled on all servos to reduce mechanical stress during power-up, and the overload protection thresholds were increased, in particular for the gripper servo and the larger arm servos. The final configuration values applied to each servo type are listed in Tables 4.2 and 4.3.

Table 4.2: Final hardware-level configuration values per servo type

Parameter	**A80BHM(F)**	**A35BHL**	**A20CLS**	**Remarks**
Travel range (L/R)	0–255	0–255	0–200	Full range used on arm joints; reduced on gripper to avoid the linear mechanism reaching its mechanical limits.
Damping factor	14	95	110	Increased for smaller servos to suppress vibration and jitter.
PWM power [%]	100	100	50	Gripper torque limited to avoid damaging mechanics; sufficient for the selected objects.
Sensitivity	Ultra High	Ultra High	High	Slightly reduced on gripper for smoother grasping.
Soft start	On	Level 3	Level 3	Limits startup shocks; same level sufficient in practice.
Lose PPM	Keep	Keep	Keep	Keeps the servo in position after signal loss.

Table 4.3: Overload protection configuration per servo type

Servo Type	**Overload Protection Levels (time, torque limit)**
A80BHM(F)	L1: 4.0 s @ 100%, L2: 8.0 s @ 100%, L3: 12.0 s @ 100%
A35BHL	L1: 5.1 s @ 100%, L2: 8.2 s @ 100%, L3: 10.2 s @ 31.4%
A20CLS	L1: 5.1 s @ 62.7%, L2: 8.2 s @ 50.2%, L3: 12.2 s @ 25.1%

The most influential parameters during testing were the damping factor and the PWM power. These settings had a direct impact on motion smoothness, stability at low speeds, and the suppression of oscillations under load. An example of the resulting tuned parameters for servos A80BHMF in shown in Figure 4.4.

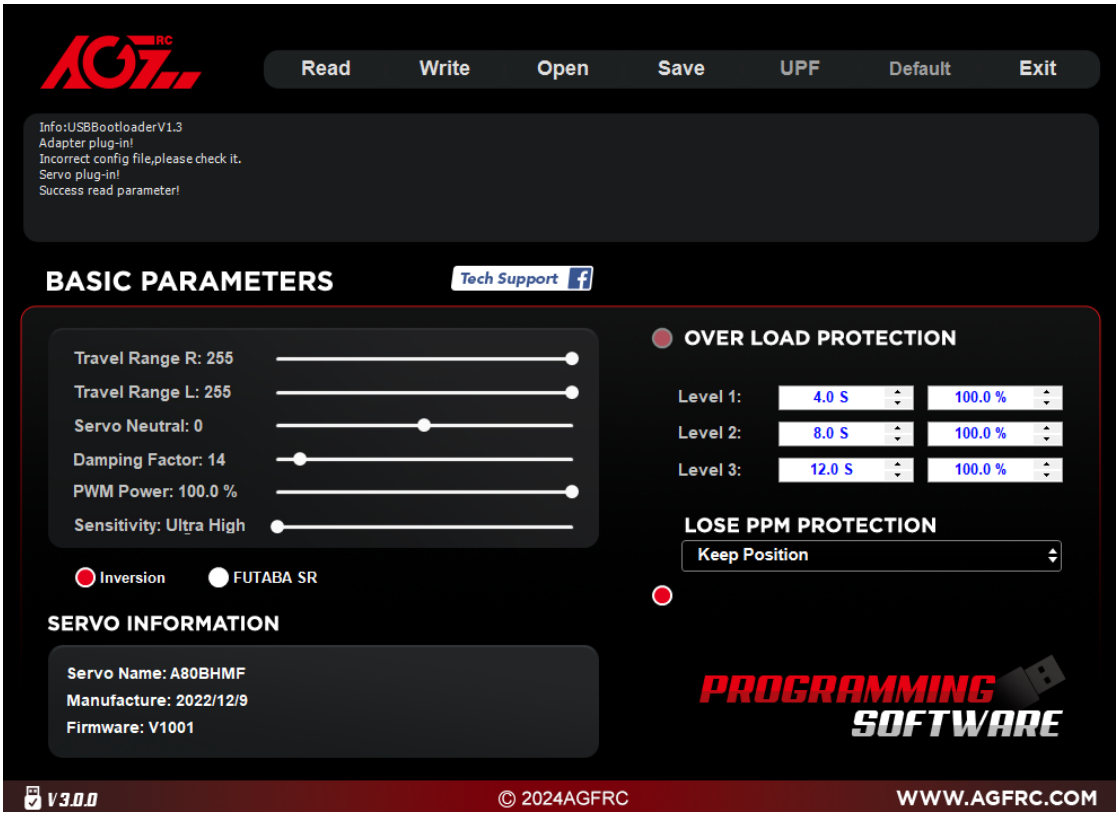


Figure 4.4: Representative hardware-level servo configuration results for the A80BHMF servos used in joints 1–3.

Configuration screenshots for the A35BHL and A20CLS servos are provided in Appendix B.3.

4.3.2. PWM Based Servo Calibration

A second *PWM* based calibration was also implemented, adapted from the calibration procedure originally introduced by Conenna. While the calibration concept and tooling follow this previous work, all measurements, calibration values, and integration into the control pipeline were performed specifically for the servos used in this robotic arm.

To conduct this calibration, Conenna's custom 3D-printed device was reprinted and reused. It features a circular protractor disk and a pointer fixed to the servo horn. This setup enables angular measurements with an accuracy of approximately 2°, allowing the identification of PWM signals corresponding to known servo angles [25].

The calibration aimed to identify the PWM values for three reference positions: $-100°$, $0°$, and $100°$. These reference points were chosen to span the full usable range of motion and to allow for linear interpolation during runtime.

Although servo manufacturers typically define the neutral position at 1520 µs (See servo specifications in Appendix B.3), real-world measurements show that individual servos deviate from this nominal value. The calibration revealed not only an offset at the 0° position, but also non-uniform endpoints across the range. As shown in Table 4.4, each servo requires distinct PWM pulse widths to reach −100°, 0°, and 100°. Once these reference points are recorded, the control software interpolates intermediate values. Different slope values are applied for the negative and positive ranges, resulting in more accurate angular positioning over the full motion span.

Table 4.4: Results of the PWM based calibration for each joint.

Angle	−100° [**µs**]	0° [**µs**]	100° [**µs**]
Joint 1	620	1540	2460
Joint 2	640	1590	2475
Joint 3	630	1555	2455
Joint 4	600	1505	2425
Joint 5	560	1490	2380
Joint 6	565	1495	2395
Gripper	580	1480	2380

4.4. Mission Setup

The mission setup defines the starting point of the grasping pipeline and places clear constraints on the scope of this thesis. The execution logic and complete pipeline are described later in Section 5.2. The pipeline assumes that the autonomous navigation stack from [8] has already explored the environment and detected objects of interest. All experiments were conducted at the *DLR* Planetary Exploration Lab in Oberpfaffenhofen, Germany, which simulates lunar surface conditions using basaltic grey sand and various rock formations. The test environment, shown in Figure 4.5, includes irregular rocks of different sizes scattered across the sandy terrain, representing the type of unstructured environment the rover would encounter during planetary exploration missions. The orange cube target object was placed among these rocks.



Figure 4.5: Side view of the rover with the rock setup and orange cube in sight at the start of the mission.

Once navigation is complete, the rover approaches a selected object (no logic for object selection is considered) and positions itself approximately 30 cm away from it, with the RealSense camera oriented toward the target. Figure 4.5 shows the real view of the rover at its start position of the pipeline, while figure 4.6 shows the RViz visualization at this initial configuration.

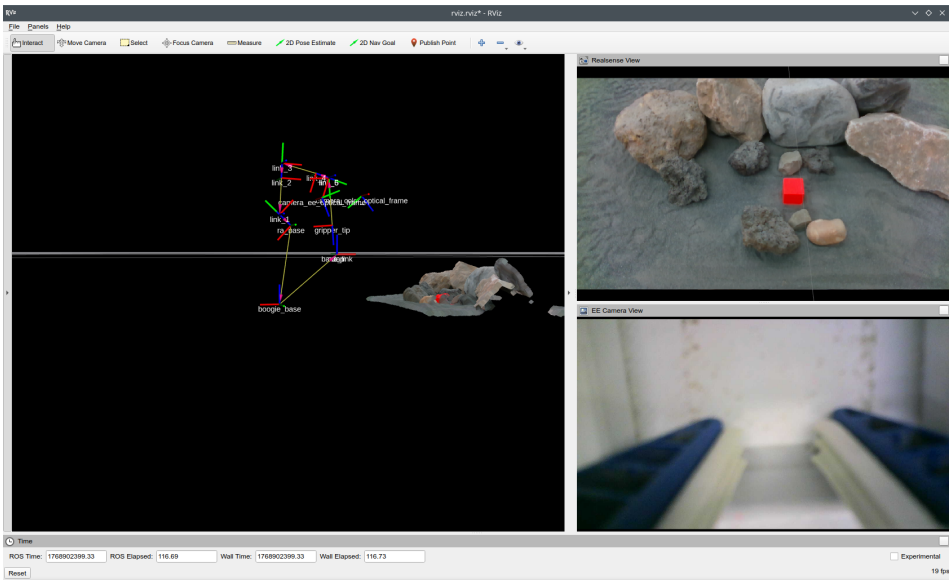


Figure 4.6: RViz visualization of the mission starting configuration, showing the RealSense camera view (top right), the end-effector camera view (bottom right), and the TF-based 3D scene representation (left).

The top-right panel displays the live view from the RealSense camera, showing the scene with surrounding rocks and the selected target object highlighted as an orange cube. The bottom-right panel shows the live image from the end-effector camera mounted on the robotic arm. The robotic arm is in its initial folded configuration, here referred to as a *home* pose, which is also visible in Figure 4.5. The left panel presents the RViz 3D visualization of the robotic arm, and the associated coordinate frames represented through the TF tree, as well as the projection of the RealSense camera data into the global frame. The current work assumes that everything up to this point functions correctly, and no development was carried out before this stage.

From here on, the autonomous grasping pipeline operates under the following assumptions:

- Only one object lies in front of the RealSense camera.
- The object is centered approximately 30 cm in front of the RealSense camera.
- The robotic arm is in its home position (default configuration for rover motion).
- All rover subsystems (communication, ROS nodes, TF processes) are running properly.
- Both cameras are powered on and operating correctly.

With these conditions satisfied, the pipeline can proceed to its first active step. The next chapter focuses on object detection and coarse localization using the RealSense depth camera, which provides the initial target estimates and the base depth image for the grasping process.

5. System Integration

This chapter describes how the individual components presented in previous chapters are integrated into a complete autonomous grasping pipeline. Section 5.1 details the coordinate transformation framework implementation using *ROS* and *LN*, connecting perception, localization, and manipulation through the *TF* tree. Section 5.2 presents the hierarchical state machine architecture that manages the grasping pipeline, including execution flow, error recovery, and state transitions.

5.1. Coordinate System Integration

5.1.1. ROS and Links-and-Nodes Integration

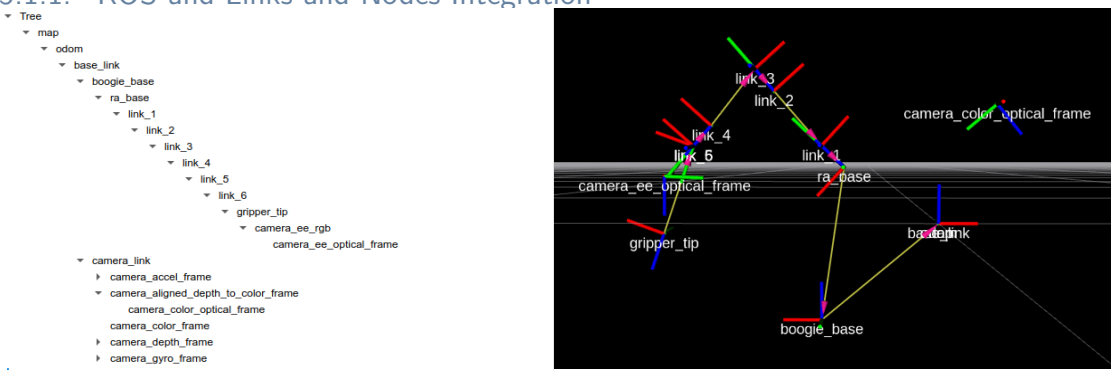


Figure 5.1: TF tree overview and visualization in RViz.

The *TF* architecture from Section 4.1 (Figure 5.1) runs inside a single *ROS* node managed by the *LN* framework. *LN* handles process management and inter-process communication, while *ROS* manages geometry and camera data streams.

The *TF* node contains three independent threads: the robotic arm publisher, pan-tilt publisher, and end-effector camera publisher. Each thread publishes transforms and sensor data within the same node. A shared static transform manager collects static transforms from all components during initialization and publishes the complete set once using a `StaticTransformBroadcaster`.

5.1.1.1. Robotic Arm TF Publisher

The robotic arm publisher subscribes to the canonical joint states topic from *LN* and computes the forward kinematic chain using the elasto-kinematic model from Section 2.7. Link transforms are published as dynamic *TF* frames. The end-effector pose is also published as a `PoseStamped` message in the `boogie_base` frame for monitoring and

visualization.

5.1.1.2. End-Effector Camera Publisher

The end-effector camera publisher provides the image stream, intrinsic calibration parameters, and the static extrinsic transform between `gripper_tip` and its child frame `camera_ee_optical_frame`. A parallel thread extracts and matches features from each frame against the RealSense reference image, republishing the results for debugging.

5.1.1.3. ROS to Links-and-Nodes Bridge

A bidirectional bridge between *ROS* and *LN* synchronizes joint states, arm commands, and gripper states. This keeps both frameworks aligned and ensures autonomous commands only reach the arm when execution permission is active.

5.1.2. Object Detection Integration

Object detection uses a *YOLO*v7 [15] network accelerated via OpenVINO [44]. The detector runs inside an `ObjectDetector` class that loads the trained network at startup and provides a single-image or batch inference interface. The output format follows the standard *YOLO* convention as

$$(x_1, y_1, x_2, y_2, s, c), \tag{5.1}$$

where (x_1, y_1) and (x_2, y_2) are the bounding box corners in pixel coordinates, s is the confidence score, and c is the class index.[15]

For depth camera detections, the system publishes 3D object positions as `PointStamped` messages on the `detected_object` topic after transformation to the `boogie_base` frame. RViz visualization uses the `objects_detected_rviz` topic. For monocular camera detections, only 2D bounding boxes are published for use in the visual centering stage.

Camera selection between the RealSense camera (initial 3D localization) and the end-effector camera (visual refinement) is controlled through a global variable in the *RAFCON* state machine manager. The state machine updates this variable based on the current pipeline phase, and the detection module instantiates the `ObjectDetector` with the corresponding camera configuration. State transition logic and retry mechanisms are detailed in Section 5.2.

5.2. Mission States

The autonomous grasping pipeline is implemented using a master state machine, shown in Figure 5.2. Each state corresponds to a specific step in the grasping process, while transitions define how the system progresses under nominal conditions and how failures are handled. A summary of all states is given in Table 5.1, and the corresponding transition logic is listed in Tables 5.2 and 5.3.

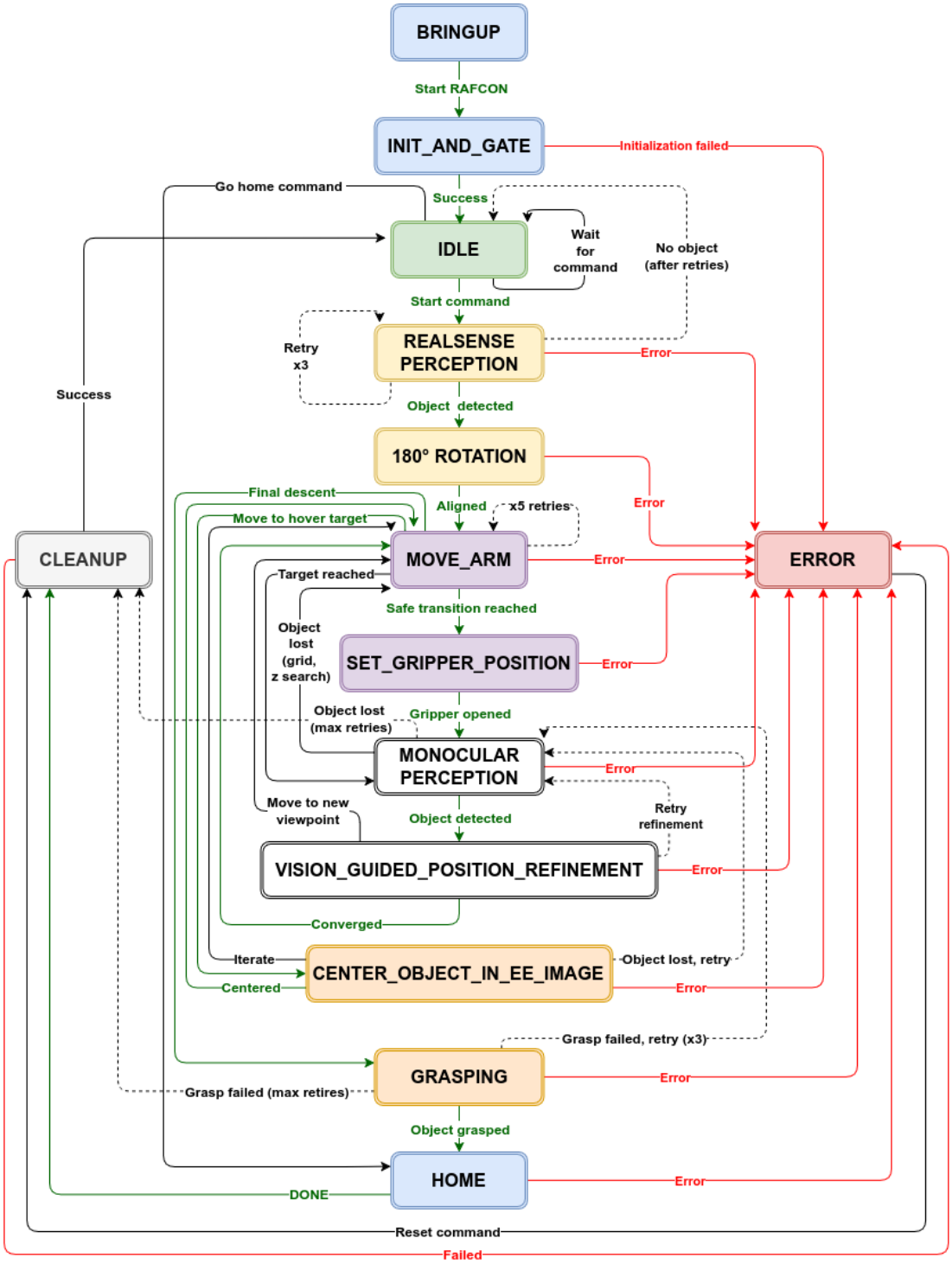


Figure 5.2: State machine diagram of the autonomous grasping pipeline. Green arrows indicate successful happy path transitions, red arrows indicate error paths, black and dashed arrows show retry and fallback mechanisms.

The pipeline begins in the `BRINGUP` state, which is entered after the *RAFCON* process is started. From there, the system transitions to `INIT_AND_GATE`, where all required components are initialized. During this phase, connections to the relevant *ROS* and *LN* topics are established, joint and gripper states are initialized, and valid image streams from both the RealSense and end-effector cameras are verified. If any of these checks fail, the system transitions to the `ERROR` state; otherwise, it proceeds to `IDLE`.

In the `IDLE` state, the system waits for external commands. A start command triggers the autonomous grasping sequence, while additional commands allow the operator to return the system to the home configuration or abort execution at any point. These commands are implemented as simple messages on a *ROS* topic and can be seen as a simulation for telemetry commands in a future deployment. As no autonomous logic is included to decide when to initiate a grasp, the pipeline is started, stopped, or aborted explicitly by a human operator.

Once a start command is received, the pipeline enters the `REALSENSE_PERCEPTION` state. In this state, the RealSense depth camera is used to detect the target object and compute an initial coarse three-dimensional position. If no object is detected, the perception step is retried up to a fixed number of attempts before returning to `IDLE`. A perception system failure results in a transition to the `ERROR` state.

After a successful detection, the pipeline proceeds to the `ROTATION` state. Since the robotic arm is mounted at the back of the rover, the rover base must first be rotated by approximately 180° so that the arm is facing the detected object. Only after this rotation can the arm reach the target. As a result of this rotation, the object will leave the *FOV* of the RealSense camera.

After the rover's rotation, the arm is moved from its home configuration to a predefined safe transition pose in the `MOVE_ARM` state. This pose is required mainly to avoid collisions with the RealSense camera during arm motion. Since self-collision avoidance between the arm and the rover is not implemented in this work, all arm movements between any position in the arm's workspace and the home state need to pass through this known safe position.

Next in the flow, the arm is moved through the `MOVE_ARM` state to a predefined viewpoint, referred to as the `pantilt_like_view` position. This pose places the end-effector camera in a perspective similar to the original RealSense view, ensuring that the object remains within the camera field of view for monocular perception.

Once the new position is reached, the gripper needs to be opened in the `SET_GRIPPER_POSITION` state so that the *FOV* of the end-effector camera is not blocked by the gripper tips. The pipeline then enters the `MONOCULAR_PERCEPTION` state. In this state, the end-effector monocular camera is used to re-detect the object and extract

features from a viewpoint similar to the original RealSense perspective.

After the object has been re-detected with the end-effector camera, the pipeline enters the `VISION_GUIDED_POSITION_REFINEMENT` state. In this state, visual features from the initial RealSense image and the current monocular image are matched, and a pose estimation is computed using a *PNP* approach with *RANSAC* (as detailed in Section 7). This step is used to correct errors introduced by odometry drift, imperfect rover rotation, mechanical tolerances, and arm elasticity. If the refinement does not converge, additional viewpoints are commanded using the `MOVE_ARM` state and the monocular perception step is repeated from a different view. Depending on the outcome, the pipeline either returns to `MONOCULAR_PERCEPTION` or proceeds to `CLEANUP`.

In practice, and as further discussed in Chapter 9, the refined object position can still be off by a few centimeters. For this reason, a final correction step is performed in the `CENTER_OBJECT_IN_EE_IMAGE` state. Here, visual feedback from the end-effector camera is used to iteratively center the object in the image frame before the final descent of the gripper.

Once the object is properly centered, the pipeline enters the `GRASPING` state, where the gripper is closed to attempt a grasp. If the grasp fails, the system can retry the centering and grasp sequence a limited number of times before aborting to the `CLEANUP` state. After a successful grasp, the arm is moved back through a safe transition pose and returned to the `HOME` state. The home configuration of the arm and gripper corresponds to the pose used to place the object into the basket, such that returning the arm home results in the object being dropped at the correct location.

The pipeline also includes dedicated states for recovery and failure handling. The `CLEANUP` state is used to reset internal variables and bring the system back to a known safe configuration after recoverable failures. The `ERROR` state is entered when a failure cannot be handled automatically and requires a manual reset command from the operator before execution can continue.

Table 5.1: State descriptions in the autonomous grasping state machine

State	Description	Phase
BRINGUP	Entry point after *RAFCON* start.	Initialization
INIT_AND_GATE	Initializes and checks required components.	Initialization
IDLE	Waits for start / abort / home commands.	Control
REALSENSE_PERCEPTION	Detects object and estimates coarse 3D position (retries limited).	Perception
ROTATION	Rotates rover base by $\sim 180^\circ$ to face the target.	Approach
MOVE_ARM	Moves arm to named poses or XYZ targets (incl. safe transition / viewpoints).	Motion
SET_GRIPPER_POSITION	Opens/closes gripper to a predefined position.	Manipulation
MONOCULAR_PERCEPTION	Re-detects object with end-effector camera; runs bounded search if lost.	Perception
VISION_GUIDED_POSITION_REFINEMENT	Refines object pose using feature matching and PnP (multi-view).	Refinement
CENTER_OBJECT_IN_EE_IMAGE	Centers object in the end-effector image before descent.	Refinement
GRASPING	Attempts grasp (bounded retries).	Manipulation
HOME	Returns arm to home (basket drop pose).	Motion
CLEANUP	Resets and returns to a safe configuration after failures.	Error Handling
ERROR	Requires operator reset command before continuing.	Error Handling

Table 5.2: State transition logic in the autonomous grasping state machine (Part I)

Source State	Target State	Transition Condition
	Initialization Phase	
BRINGUP	INIT_AND_GATE	System startup.
INIT_AND_GATE	IDLE	Initialization successful.
INIT_AND_GATE	ERROR	Initialization failed.
	Idle and Command Processing	
IDLE	IDLE	Waiting for command (self-loop).
IDLE	REALSENSE_PERCEPTION	Start pipeline command received.
IDLE	HOME	Go-home command received.
	Coarse Perception Phase	
REALSENSE_PERCEPTION	ROTATION	Object detected in image.
REALSENSE_PERCEPTION	REALSENSE_PERCEPTION	No object found, retry (up to 3 attempts).
REALSENSE_PERCEPTION	IDLE	No object after maximum retries.
REALSENSE_PERCEPTION	ERROR	Perception system error.
	Base Alignment Phase	
ROTATION	MOVE_ARM	Rover rotated by $\sim 180°$.
ROTATION	ERROR	Rotation system error.
	Arm Motion Phase	
MOVE_ARM	SET_GRIPPER_POSITION	Reached safe transition position.
MOVE_ARM	MOVE_ARM	Safe transition after grasp.
MOVE_ARM	MONOCULAR_PERCEPTION	Reached viewpoint or XYZ target.
MOVE_ARM	HOME	Reached home position.
MOVE_ARM	GRASPING	Reached final descend position.
MOVE_ARM	ERROR	Movement failed or unreachable.
	Gripper Control Phase	
SET_GRIPPER_POSITION	MONOCULAR_PERCEPTION	Gripper opened at pantilt-like view.
SET_GRIPPER_POSITION	IDLE	Gripper set to home position.
SET_GRIPPER_POSITION	ERROR	Gripper control error.
	Fine Perception Phase	
MONOCULAR_PERCEPTION	VGPF	Object detected (approach mode: refine_3d).
MONOCULAR_PERCEPTION	MOVE_ARM	Object detected (coarse / center_xy / final_descend).
MONOCULAR_PERCEPTION	MOVE_ARM	Object lost, Z-scan + grid search.
MONOCULAR_PERCEPTION	CLEANUP	Object not found after search exhaustion.
MONOCULAR_PERCEPTION	ERROR	Perception error.

VGPF stands for `VISION_GUIDED_POSITION_REFINEMENT` state and is used only for compactness within this table.

Table 5.3: State transition logic in the autonomous grasping state machine (Part II)

Source State	Target State	Transition Condition
3D Refinement Phase		
VGPF	MOVE_ARM	Position converged, move to hover target.
VGPF	MOVE_ARM	Additional viewpoint required.
VGPF	MONOCULAR_PERCEPTION	Further refinement needed.
VGPF	CLEANUP	No more viewpoints available.
VGPF	IDLE	Non-critical refinement error.
Visual Servoing Phase		
CENTER_OBJECT_IN_EE_IMAGE	MOVE_ARM	Object centered, proceed to final descend.
CENTER_OBJECT_IN_EE_IMAGE	MOVE_ARM	Iterative centering step.
CENTER_OBJECT_IN_EE_IMAGE	MONOCULAR_PERCEPTION	Object lost during centering.
CENTER_OBJECT_IN_EE_IMAGE	ERROR	Centering error.
Grasping Phase		
GRASPING	MOVE_ARM	Grasp successful, return via safe transition.
GRASPING	MONOCULAR_PERCEPTION	Grasp failed, retry (up to 3x).
GRASPING	CLEANUP	Grasp failed after max retries.
GRASPING	ERROR	Critical grasping error.
Return and Recovery		
HOME	IDLE	Successfully returned home.
HOME	ERROR	Failed to reach home.
CLEANUP	IDLE	Cleanup successful.
CLEANUP	ERROR	Cleanup failed.
ERROR	ERROR	Waiting for operator reset.
ERROR	CLEANUP	Operator reset received.

VGPF denotes the `VISION_GUIDED_POSITION_REFINEMENT` state and is used only for compactness within this table.

In the monocular perception stage, the same motion and perception states are reused for different phases of the approach. To distinguish between these phases without duplicating states, an internal *approach mode* variable is used. This variable controls how target positions are computed and which transitions are allowed, while the overall structure of the state machine remains unchanged. The available approach modes and their role within the grasping pipeline are summarized in Table 5.4.

Table 5.4: Approach modes and their effects on state transitions

Approach Mode	**Description**	**Used After**
refine_3d	3D position refinement with multiple viewpoints and feature matching to compute accurate object location	Initial perception
coarse	Moves directly to coarse target (hybrid map/base_link coordinates) without refinement	Initial detection
center_xy	Iteratively centers object in end-effector image using visual servoing; adjusts X and Y while maintaining Z	After 3D refinement
final_descend	Final approach to grasp height; descends to pre-computed Z position while maintaining XY alignment	After centering

The master state machine implements recovery strategies for common failure modes, including perception errors, motion planning failures, and object loss during manipulation. The chosen recovery action depends on the failure type and current pipeline stage, and all retry paths are bounded to avoid endless execution loops. Appendix D, Table D.1 details the implemented failure cases and their corresponding recovery actions. Additional error cases and unhandled situations can be covered in future work.

In addition, several aspects of the pipeline behavior are controlled through configuration parameters instead of hard-coded values. These parameters define retry limits, safety thresholds, search patterns, and positional offsets that are reused across multiple states. The selected values were determined empirically during experimental testing and can be adjusted without modifying the state machine logic. The most relevant parameters used in the experiments are summarized in Appendix D, Table D.2.

For a complete overview of the individual state machines and their hierarchical interconnections, refer to Appendix D.

6. Object Localization and Visual Odometry Considerations

As previously mentioned in Section 2.3.1, object localization in this work relies on an existing perception pipeline that was adapted to support headless execution and reuse across multiple camera sources. Object detection is performed using both the RealSense depth camera and the monocular camera mounted on the robotic arm end-effector.

Originally, the autonomous navigation stack used the RealSense camera frame with a pure translational offset, without any rotational component, relative to the rover's body frame (`base_link`). This setup allowed the system to detect objects at a distance while exploring the environment. However, at this stage of the autonomous pipeline, the rover has already detected an object and must position itself close to it in order to be within reach for grasping. As a result, the RealSense depth camera observes the target under a tilted angle (measured at approximately 50°). The vision processing pipeline then estimates the object position in the camera's optical frame[1]. This change in camera orientation resulted in significant discrepancies in the visual odometry stack, further discussed in Section 6.2.

Object detection outputs the detected class, a confidence score, and the corresponding bounding box in image coordinates (see 5.1). The model was originally trained using 5×5×5 cm 3D-printed cubes with distinct colors. For this work, the cubes were reprinted at a smaller size (3×3×3 cm) while maintaining the same colors. This size reduction was necessary to ensure that the objects fit within the gripper's grasping range. Despite this change in scale, the model still performed reliably. Results are further discussed in Section 9.1. The distinctive color features remained consistent between the training and deployment object sets. The bounding box of the detected object is defined by the top-left and bottom-right pixel coordinates in the image.

Object detection occurs at three stages of the autonomous grasping sequence:

1. Using the depth camera for initial localization.

2. Using the monocular camera for re-localization after the rotation.

3. Using the monocular end-effector camera for close-range visual refinement.

This chapter focuses on the first step; depth-based localization using the RealSense

[1]The `camera_color_optical_frame` follows the OpenCV frame convention [9], where the x-axis points to the right in the image plane (width), the y-axis points downward in the image (height), and the z-axis points in the viewing direction of the camera.

camera. Monocular detection is discussed in Chapter 7.

Two headless classes are used within the vision package: `ObjectDetector`, which wraps the detection inference pipeline, and `ObjectLocalizer`, which implements object center estimation, depth-based 3D reconstruction, coordinate transformations, among other helper functions.

6.1. RealSense Depth-Based Localization



Figure 6.1: Orange cube (object) detected with a confidence score by the RealSense camera.

Using the headless package, the camera model (RealSense or monocular) can be set beforehand by the master state (see 5.2) to save the image accordingly and run the inference on the selected image. After detection, the center of each bounding box is computed in pixel coordinates. This point corresponds to the center of the bounding box rather than the true geometric center of the object. When viewing a cube from an angle, this introduces a systematic localization error, as the bounding box center does not necessarily coincide with the physical center of the object. Figure 6.1 shows a saved image after running inference on the RealSense camera, with a blue bounding box around the detected object and the corresponding class label and confidence score displayed above it.

Using the pixel coordinates (u, v) together with the associated depth value, the three-

dimensional object coordinates are reconstructed in the camera optical frame following the projection model described in Section 2.2.1.2. The resulting (x, y, z) coordinates must then be expressed in a frame suitable for robotic arm control. Figure 6.2 visualizes the system state before any rover motion is applied. On the right side of the figure, the RealSense RGB image is displayed, showing the detected orange cube in the scene. On the left side, the RViz 3D view shows the robotic arm TF tree, and the reconstructed object position visualized as an orange cube marker published in the `map` frame. The spatial alignment between the marker and the object observed in the camera view validates the accuracy of the detection, depth-based back-projection, and coordinate transformation before the rover rotation.

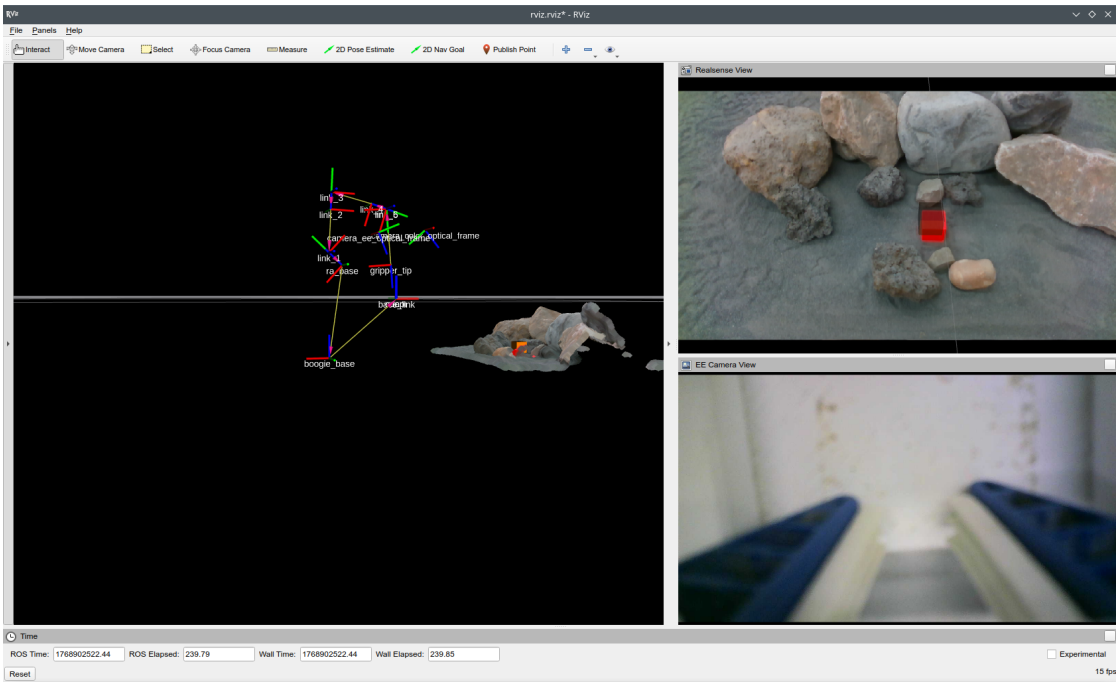


Figure 6.2: RViz visualization of the object marker (orange cube) shown together with the current RealSense view before rover rotation.

6.1.1. Publishing Object Coordinates

The RealSense camera is rigidly mounted at the front of the rover and oriented such that its field of view covers the object during the perception phase. In contrast, the robotic arm is mounted at the back of the rover. As a result, a 180°rotation of the rover is required to bring the detected object into the reachable workspace of the arm. After this rotation, the object is no longer visible to the RealSense camera, since the camera orientation is fixed and does not change relative to the rover chassis.

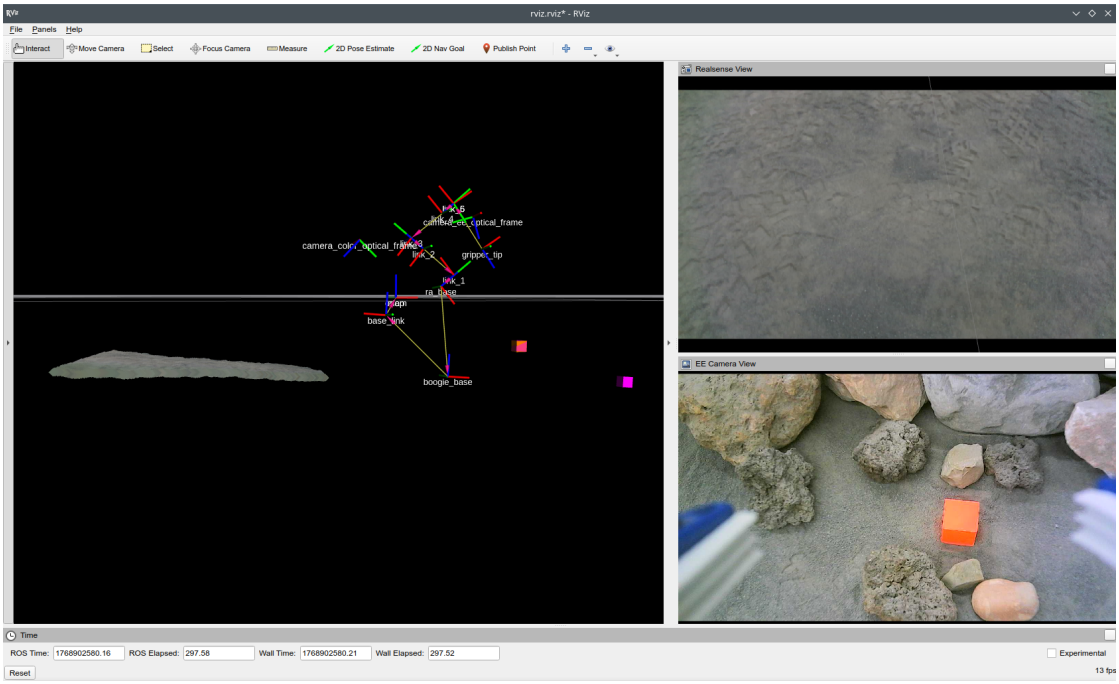


Figure 6.3: RViz visualization of the object markers after a 180° rover rotation.

Initial experiments published object coordinates in RViz using the `map` frame (Figure 6.3). Testing revealed that visual odometry drift corrupts the TF tree after rover motion. The system was changed to publish coordinates in the `base_link` frame instead, which is rigidly attached to the rover chassis.

Figure 6.3 shows the RViz visualization after a 180° rotation. The orange marker shows the object position before rotation, published in `map` and fixed as a global reference. Two magenta debug markers show the same position transformed to `map` after rotation (first marker) and expressed in `base_link` (second marker). Transforming coordinates through static transforms to `boogie_base` should isolate the manipulation pipeline from odometry drift. The significant difference between markers in both frames demonstrates that odometry errors propagate through the TF tree even with static transform chains.

6.2. Localization Observations and Discussion

Visual odometry limitations show clearly during the rover's 180° rotation, which positions the robotic arm toward the detected object. After rotation, the depth camera no longer observes the object, and accumulated odometry drift corrupts the TF transforms.

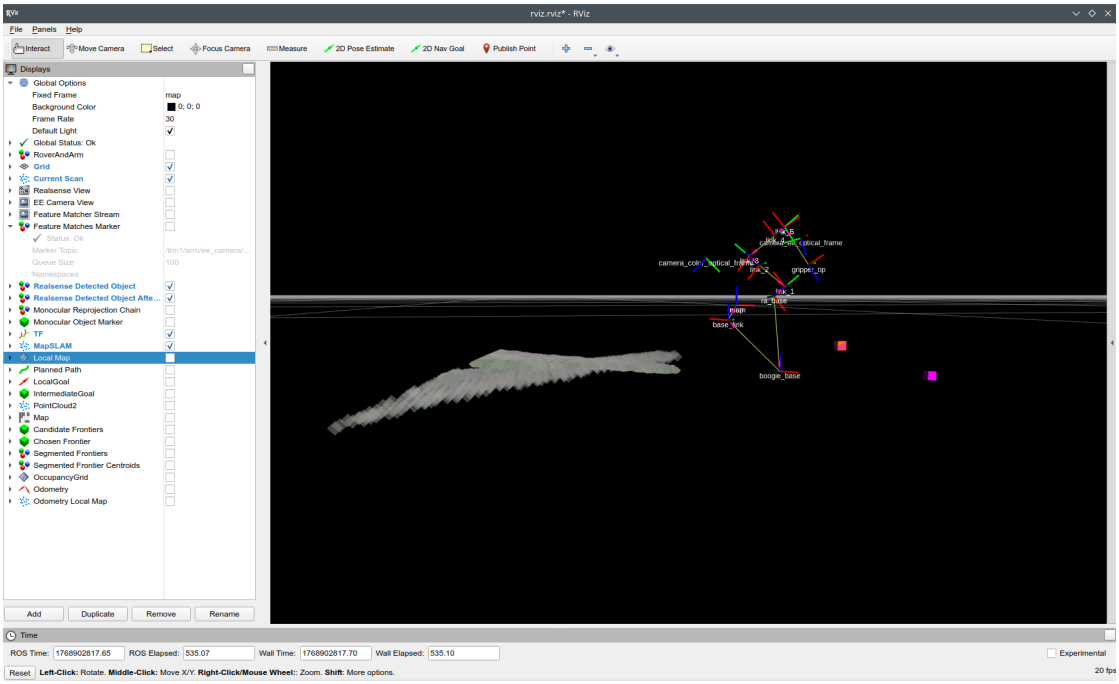


Figure 6.4: RViz visualization of the TF tree with two markers and the visual odometry output (seen tilted) after the rotation.

Object markers published in the `map` frame show clear offsets, particularly in the vertical direction, matching De Castro's [8] findings on z-axis drift from visual odometry. Markers expressed in the `base_link` frame stay consistent with the robotic arm workspace, but the X and Y coordinates still show large errors compared to the `map` frame position.

Since all grasping operations assume the rover remains stationary after rotation, improving visual odometry itself falls outside the scope of this thesis. The focus lies on manipulation rather than navigation. Chapter 7 explains the proposed approach to obtain an accurate object position after rotation, bypassing the visual odometry drift.

7. Visual Pose Estimation and Target Refinement

This chapter details the approach used to update the object position after the 180° rotation of the rover. As discussed in Section 6.2, odometry drift from the visual odometry stack introduces significant errors into the *TF* chain over time. Since the RealSense camera is rigidly mounted at the front of the rover and the robotic arm is located at the back, the required 180° rotation causes the object to leave the RealSense field of view. Therefore, an alternative method is required to update the object position for grasping.

In this work, feature-based monocular pose estimation is used to re-localize the target object from the viewpoint of the end-effector camera. Visual features extracted from the initial RealSense image are matched with features from the current monocular image, allowing the pose of the end-effector camera to be estimated. Based on this estimated pose, the target coordinates are updated in the robotic arm workspace. This approach avoids relying on accumulated odometry drift and provides object position estimates that are sufficiently accurate for grasp execution when combined with the visual refinement stage described in Section 7.4.

The chapter is structured as follows. Section 7.1 describes the feature extraction and matching pipeline. Section 7.2 explains the *PNP*-based camera pose estimation method. Finally, Section 7.3 details how the estimated camera pose is used to update the target coordinates in the robotic arm workspace, expressed in the `boogie_base` frame.

7.1. Feature Extraction and Matching

7.1.1. Reference Image Acquisition

To re-localize the object after the rover rotation, a reference image is required for feature matching. This image is stored during the RealSense perception stage of the pipeline (see Section 5.2), before the rover rotates. Visual features extracted from this reference image serve as the basis for matching against the monocular end-effector camera view.

In addition to the *RGB* image, the corresponding depth image captured at the same time provides metric depth information for each pixel. This allows known 3D coordinates to be associated with the detected feature points, which is required for the *PNP* pose estimation step described in Section 7.2.1. In this way, the reference image captures the spatial relationship between the object and its surrounding environment, which allows the same features to be found again when the scene is observed from a different viewpoint by the end-effector camera.

7.1.2. Re-localization Using the End-Effector Camera

After the rover rotation, the robotic arm is moved to a predefined viewpoint referred to as `pantilt_like_view`. This pose places the end-effector camera in a perspective as similar as possible to the original RealSense camera view, as illustrated in Figure 7.1.



Figure 7.1: Viewpoint comparison between the RealSense scene (left) and the monocular end-effector camera scene (right).

Physical constraints of the robotic arm prevent exact replication of the RealSense camera pose.[1] Nevertheless, this viewpoint maximizes the overlap between the fields of view of both cameras, increasing the likelihood of detecting common visual features. It also reduces the baseline distance and angular difference between viewpoints, improving pose estimation reliability.

From this position, the monocular end-effector camera captures an image of the scene. Although this camera does not provide depth information, it observes many of the same visual features present in the reference RealSense image. By matching these features, correspondences can be established between 3D points obtained from the reference image and 2D pixel locations in the monocular image. A feature detected in the RealSense image can be matched to the monocular image by comparing their *SIFT* descriptors.

The *PNP*-based pose estimation algorithm uses these 3D–2D correspondences as input, as described in Section 7.2, to estimate the pose of the end-effector camera relative to the reference frame.

The estimated camera pose transforms object coordinates from the original RealSense frame into the current end-effector camera frame, and finally into the robotic arm workspace, as detailed in Section 7.3. This update compensates for accumulated odometry drift and provides a more accurate target position for grasp planning.

[1] The arm kinematics and mounting geometry prevent the end-effector camera from reaching the exact RealSense position and orientation.

7.1.3. SIFT Feature Detection

Feature extraction is performed using the *SIFT* algorithm, introduced in Section 2.4.2. *SIFT* was selected over other feature detectors such as *ORB* due to its robustness to scale and rotation changes, which are expected when comparing images from the RealSense and end-effector cameras.

The two cameras have different intrinsic parameters (see Sections 2.2.2 and 4.2), resulting in differences in scale, field of view, and image resolution. In addition, the `pantilt_like_view` pose only approximates the RealSense viewpoint, introducing perspective differences between the images. *SIFT*'s scale and rotation invariance handles these variations.

SIFT feature detection [16] is implemented using a wrapper class in the vision headless package, based on OpenCV's implementation [21]. For each image, the detector identifies a set of keypoints and computes a corresponding 128-dimensional descriptor for each keypoint. These descriptors capture the local gradient structure around keypoints and remain stable under moderate changes in viewpoint and illumination [16].

Feature extraction using SIFT [16] is performed independently on the reference *RGB* image from the RealSense camera and on the current image from the end-effector camera. The feature sets from each image are denoted as:

$$\mathcal{F}_{\text{rs}} = \{f_1, f_2, \ldots, f_m\} \tag{7.1}$$

$$\mathcal{F}_{\text{ee}} = \{f'_1, f'_2, \ldots, f'_n\} \tag{7.2}$$

where $\mathcal{F}_{\text{rs}}$ represents the feature set from the RealSense reference image and $\mathcal{F}_{\text{ee}}$ represents the feature set from the end-effector image. Each feature consists of a keypoint location and a 128-dimensional descriptor vector [16].

Most *SIFT* parameters use OpenCV defaults [16, 21], with the following exceptions. The maximum number of features was increased from 1200 to 2000 to ensure sufficient keypoints for robust matching, particularly when the object occupies only a small portion of the image. The contrast threshold of 0.04 filters out low-contrast keypoints by rejecting candidate keypoints with contrast values below this level, removing features that are more sensitive to noise [16]. The edge threshold of 10.0 suppresses poorly localized features along strong edges by comparing the ratio of principal curvatures at each keypoint location; values above 10.0 indicate edge-like structures that are discarded [16]. The Gaussian smoothing parameter ($\sigma = 1.6$) controls the scale of the initial Gaussian blur applied to the image before keypoint detection and remains at the OpenCV default [21]. No additional filtering occurs at this stage since correspondence quality is handled during feature matching and outlier rejection.

Figure 7.2 and Figure 7.3 show the extracted *SIFT* keypoints for the RealSense reference image and the end-effector monocular camera image, respectively.

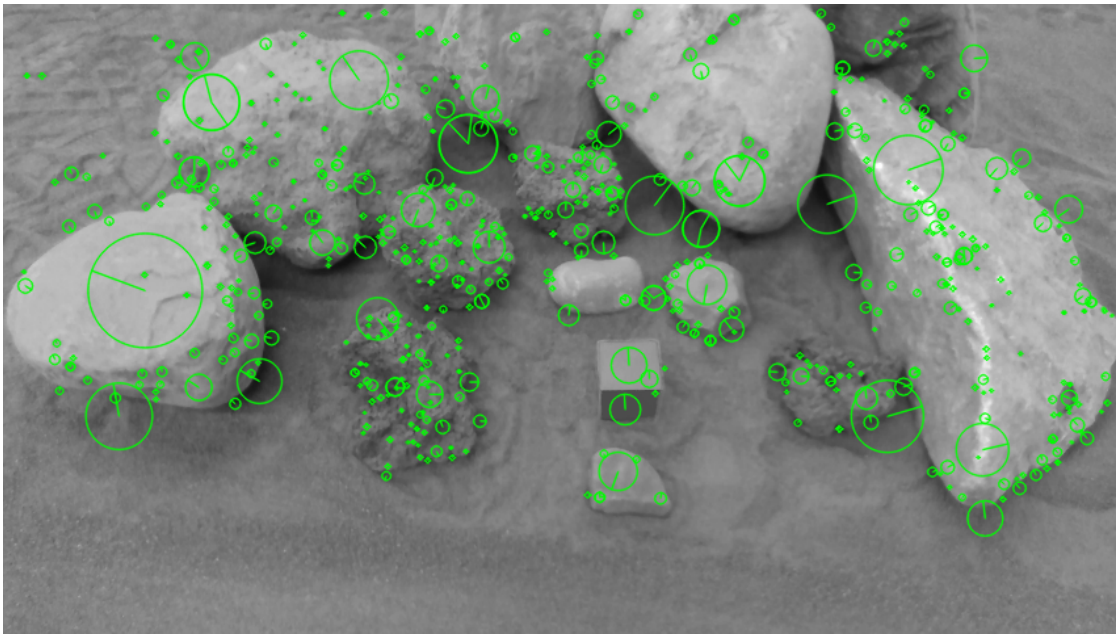


Figure 7.2: SIFT *keypoints detected in the RealSense reference image.*

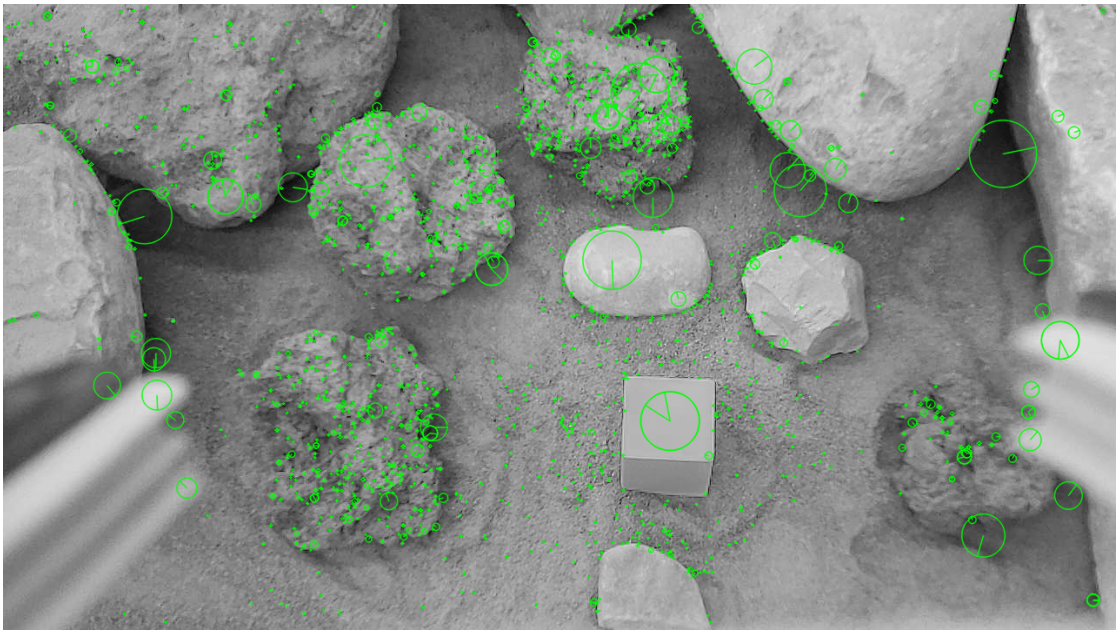


Figure 7.3: SIFT *keypoints detected in the end-effector monocular camera image.*

7.1.4. Feature Matching and Correspondence

Feature matching uses *FLANN* [22], which is commonly used with *SIFT* descriptors due to its computational efficiency for high-dimensional feature vectors. The *FLANN* matcher is configured with a KD-tree index using the default parameters: 5 randomized trees and up to 50 checks during the search [22]. The KD-tree structure enables efficient nearest neighbor search in the 128-dimensional *SIFT* descriptor space, while multiple randomized trees improve robustness across different data distributions.

For each descriptor in the end-effector image, the matcher performs a k-nearest neighbor search with $k = 2$, retrieving the two closest matches from the reference image. The value $k = 2$ allows Lowe's ratio test [16], which compares the best match against the second-best match to assess match quality.

Lowe's ratio test [16] filters out ambiguous matches. Let d_1 and d_2 denote the distances to the first and second nearest neighbors, respectively. A match is accepted only if:

$$\frac{d_1}{d_2} < \tau \tag{7.3}$$

where τ is the ratio threshold. The ratio test rejects matches where the best match is not significantly better than the second-best, which typically corresponds to an unreliable match.

The threshold τ was tuned experimentally across multiple test runs, starting from the commonly used value of 0.75 [16]. Values of 0.75, 0.8, 0.82, and 0.85 were tested by running the complete perception and PnP pipeline on different object positions and viewpoints. For each threshold value, the number of matches, inlier ratio after RANSAC, and final object center reprojection error were recorded. Lower thresholds (0.75, 0.8) produced fewer matches, occasionally falling below the minimum required for reliable pose estimation. Higher thresholds (0.85) resulted in too many false correspondences that decreased pose estimation accuracy, as measured by increased reprojection errors. The value $\tau = 0.82$ provided the best balance, yielding fewer but more accurate matches that improved PnP solver reliability and reduced reprojection errors compared to other tested values.

The ratio test produces a set of putative correspondences. These matches may contain outliers from repetitive textures, occlusions, or descriptor ambiguity. *RANSAC*-based homography estimation removes these outliers through geometric verification. The complete pipeline is illustrated in Figure 7.4, with each step explained in detail in the following subsections.

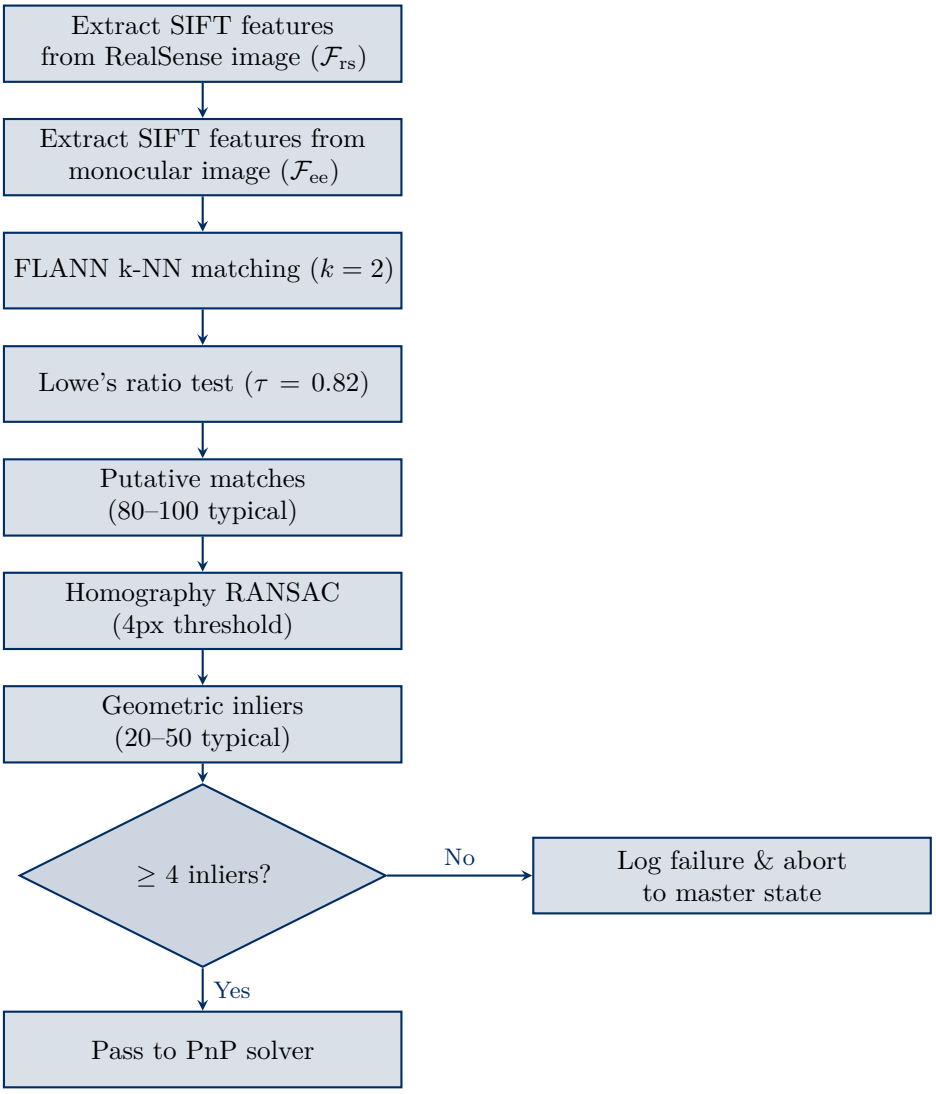


Figure 7.4: Feature matching and correspondence pipeline showing the complete flow from SIFT feature extraction through ratio test filtering, homography-based RANSAC verification, and minimum inlier check before passing correspondences to the PnP solver. Each step is detailed in the following subsections.

7.1.4.1. Homography-based Outlier Rejection

A homography is a projective transformation that relates two views of a planar surface. For a point (x, y) in the reference image and its corresponding point (x', y') in the end-effector image, the relationship is given by [10]:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7.4}$$

where H is a 3×3 homography matrix with 8 degrees of freedom (up to scale), and $\sim$ denotes equality up to a scale factor.

The *RANSAC* algorithm [24] iteratively estimates H by randomly selecting minimal sets of correspondences (4 point pairs) and computing the homography that best fits the data. For each hypothesis, the algorithm classifies matches whose reprojection error falls below a threshold as inliers. The homography with the largest number of inliers is selected as the final model.

The reprojection error for a match is computed as the Euclidean distance between the transformed point from the reference image and the observed point in the end-effector image:

$$e = \left\| \mathbf{x}' - \frac{1}{w'} H\mathbf{x} \right\|_2 \tag{7.5}$$

where $\mathbf{x} = [x, y, 1]^T$ is a point in the reference image, $\mathbf{x}' = [x', y', 1]^T$ is the corresponding point in the end-effector image, and w' is the homogeneous scaling factor obtained from the third row of $H\mathbf{x}$. The algorithm classifies matches with reprojection error e below the pixel threshold as inliers.

The *RANSAC* implementation uses a maximum of 2000 iterations and a confidence level of 99.5%. The confidence parameter controls the probability of finding at least one iteration with only inlier samples, given an expected outlier ratio [10]. With these settings, *RANSAC* can tolerate high outlier ratios and still reliably estimate a valid homography [10].

In this work, the pixel threshold was initially set to 8.0 pixels but was later reduced to 4.0 pixels after testing. The tighter threshold enforces stricter geometric consistency, filtering out more outliers and improving the quality of matches passed to the *PNP* solver. While this reduces the number of inliers, the remaining matches are more geometrically consistent, which leads to lower reprojection errors in the final pose estimation (see Section 7.2).

The lunar-like environment used during the tests contains a flat sand surface with rocks. The homography model fits features on the sand plane, while *RANSAC* filters out matches on the rocks.

In typical operation, the ratio test produces approximately 80–100 putative matches, of which 20–50 survive the *RANSAC* geometric verification as inliers depending on scene complexity and feature distribution. For example, in one representative run, 84 matches passed the ratio test, and 25 were classified as inliers by *RANSAC*, yielding an inlier ratio of approximately 30%. The *PNP*-based pose estimation uses these inliers as input.

7.1.4.2. Visualization and Quality Assessment

For debugging and validation, the system visualizes matched features by drawing lines between corresponding keypoints in the reference and end-effector images. The visualization distinguishes between all matches (after ratio test, Figure 7.5) and inliers (after *RANSAC*, Figure 7.6), with match statistics overlaid on the image. The system saves these visualizations to disk and publishes them on a *ROS* topic for real-time monitoring in RViz.

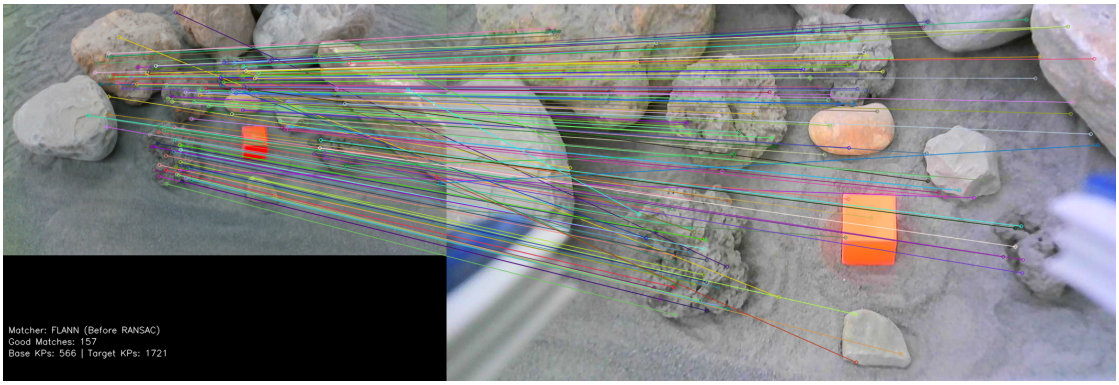


Figure 7.5: Matching results before RANSAC *filtering, with 157 good matches between the RealSense reference image (left) and end-effector monocular camera image (right).*

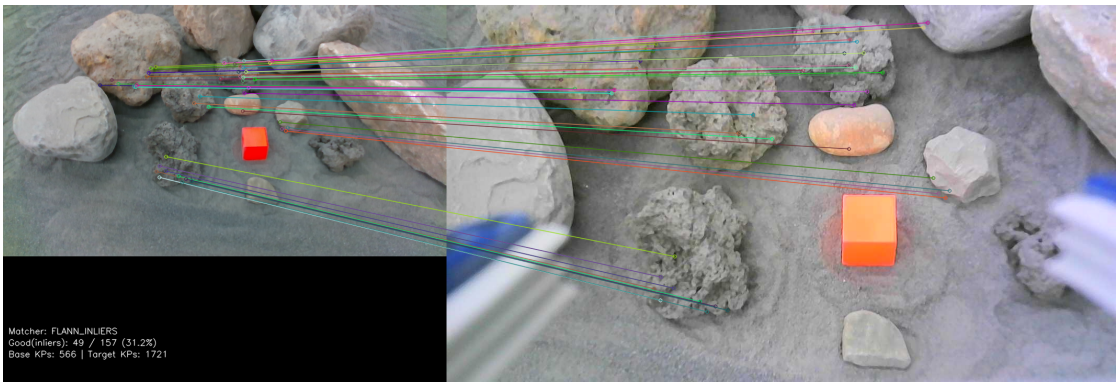


Figure 7.6: Matching results showing the same scene after geometric verification, where 49 inliers remain. Colored lines connect geometrically consistent SIFT *keypoint pairs.*

A minimum of 4 matches is required to proceed with pose estimation, as this is the minimum needed for the *PNP* solver with *RANSAC*. In practice, if fewer than 4 inliers remain after geometric filtering, the state machine logs a detection failure and either retries monocular perception with a different viewpoint or transitions to a recovery state, depending on the current retry count (see Section 5.2).

7.2. Camera Pose Estimation

7.2.1. PnP Formulation with Known Depth

The *PNP* problem consists of estimating the pose of a camera given a set of 3D points and their corresponding 2D projections in the image plane. In this work, the 3D coordinates of feature points are obtained from the RealSense depth image, while the corresponding 2D projections are the matched keypoints in the end-effector camera image.

Let $\{P_i\}_{i=1}^{N}$ denote the 3D coordinates of matched features expressed in the RealSense camera frame, and $\{p'_i\}_{i=1}^{N}$ their corresponding 2D pixel coordinates in the end-effector camera image. The goal is to estimate the rigid transformation $T = [R \mid t]$ that relates the end-effector camera frame to the RealSense camera frame, where $R \in \mathbb{R}^{3\times3}$ is the rotation matrix and $t \in \mathbb{R}^3$ is the translation vector.

Under the pinhole camera model with known intrinsic parameters K_{ee} (see Section 2.2), the projection relationship is given by [10]:

$$s_i \begin{bmatrix} u'_i \\ v'_i \\ 1 \end{bmatrix} = K_{\text{ee}} \, [R \mid t] \begin{bmatrix} P_i \\ 1 \end{bmatrix} \tag{7.6}$$

where s_i is a scale factor and (u'_i, v'_i) are the pixel coordinates in the end-effector camera image.

7.2.1.1. 3D Point Reconstruction from Depth Image

The 3D coordinates P_i are computed from the RealSense depth image using the inverse pinhole projection, which produces a dense `xyz_map` in the RealSense camera optical frame. For a pixel at coordinates (u, v) with depth value Z, the corresponding 3D point is computed as:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Z \begin{bmatrix} \frac{u-c_x}{f_x} \\ \frac{v-c_y}{f_y} \\ 1 \end{bmatrix} \tag{7.7}$$

where f_x and f_y are the focal lengths, and (c_x, c_y) is the principal point of the RealSense intrinsic matrix [10].

This operation processes all valid pixels in the depth image, producing a dense 3D map of shape $(H \times W \times 3)$. Pixels with invalid or missing depth values are marked as *Not a Number (NAN)* and excluded from further processing.

For each matched feature with pixel coordinates (u_i, v_i) in the reference *RGB* image, the corresponding 3D point P_i is taken from the precomputed `xyz_map`. Since depth measurements can be noisy from surface properties or sensor limitations. To handle this,

a local neighborhood is evaluated: a 3×3 window is used first, and the median of valid depth values is selected. If fewer than five valid values are found, the window is expanded to 5×5 and then 7×7 until sufficient data are available.

This process gives the 3D points P_i in the RealSense camera frame. Together with the corresponding 2D observations p'_i from the end-effector image (Section 7.1.4), these form the 3D–2D correspondences required for *PNP* pose estimation.

7.2.2. RANSAC-Based Robust Estimation

The homography-based filtering described in Section 7.1.4.1 removes many geometric outliers, but incorrect correspondences can still remain. These come from repetitive textures, residual descriptor ambiguities, or depth measurement errors. To handle such cases, the *PNP* problem is solved using OpenCV's `solvePnPRansac` function [11], which combines pose estimation with *RANSAC*-based outlier rejection.

The *RANSAC* procedure operates as follows [24]:

1. Randomly select a minimal subset of correspondences (four point pairs).
2. Estimate a candidate pose $[R \mid t]$ using the *PNP* solver.
3. Project all 3D points P_i into the image using Equation 7.6 and compute the reprojection error [10]:
$$e_i = \|p'_i - \pi(K_{\text{ee}}, R, t, P_i)\|_2 \tag{7.8}$$
4. Classify correspondences with $e_i < \epsilon$ as inliers.
5. Repeat the process for a fixed number of iterations and retain the pose with the largest inlier set.

The reprojection error threshold ϵ and iteration count were tuned experimentally in multiple test runs. Initial tests with $\epsilon = 8.0$ pixels and 100 iterations resulted in higher final reprojection errors. Increasing the threshold to 10.0 pixels proved too permissive. The final configuration uses $\epsilon = 5.0$ pixels with 500 iterations. This stricter threshold works because the input correspondences are already filtered by homography-based *RANSAC*, while the higher iteration count improves robustness against residual outliers.

7.2.3. Pose Refinement

After *RANSAC* selects the best pose hypothesis, the solution is refined using the Levenberg–Marquardt algorithm via OpenCV's `solvePnPRefineLM` function [11]. This non-linear optimization minimizes the reprojection error over all inliers:

$$\sum_{i \in \text{inliers}} \|p'_i - \pi(K_{\text{ee}}, R, t, P_i)\|_2^2 \tag{7.9}$$

This refinement typically reduces the mean reprojection error by approximately 0.5–1.0 pixels compared to the initial *RANSAC* estimate.

7.2.4. Pose Quality Assessment

The estimated pose is evaluated using two criteria:

- **Object center reprojection error**: The pixel distance between the detected object center in the end-effector image and the reprojected object center from the RealSense depth map using the estimated pose.
- **Number of inliers**: A minimum of four correspondences is required for pose estimation.

7.2.5. Object Center Reprojection Validation

The system validates the estimated pose before arm motion by transforming the 3D object center point obtained from the RealSense depth `xyz_map` to the monocular camera frame using the estimated transformation $[\mathbf{R}|\mathbf{t}]$:

$$\mathbf{p}_{\text{ee}} = \mathbf{R} \cdot \mathbf{p}_{\text{rs}} + \mathbf{t} \tag{7.10}$$

where $\mathbf{p}_{\text{rs}}$ is the object center in the RealSense camera frame (obtained from the depth value at the *YOLO* bounding box center) and $\mathbf{p}_{\text{ee}}$ is the transformed point in the monocular end-effector camera frame.

This transformed point is reprojected to the monocular image plane using the end-effector camera intrinsics $\mathbf{K}_{\text{ee}}$ [10]:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \mathbf{K}_{\text{ee}} \cdot \mathbf{p}_{\text{ee}} \tag{7.11}$$

The pixel distance between this reprojected point (u, v) and the detected object center from the *YOLO* bounding box in the monocular image is the object center reprojection error. If this error exceeds an experimentally determined threshold, the system rejects the *PNP* estimate and repeats the perception cycle with a different viewpoint.

This validation differs from the standard feature-level reprojection error used internally by `solvePnPRansac` [11]. The object center reprojection error directly measures whether the estimated pose correctly aligns the detected object centers across both camera views. This metric is more relevant for arm motion than feature-level residuals.



Figure 7.7: Reprojection error visualization after pose refinement. The green circle marks the detected object center (YOLO bounding box center), and the red circle shows the reprojected 3D point position based on the estimated camera pose. The pixel error of 30.96 pixels indicates good pose accuracy.

Figure 7.7 shows a representative reprojection result. If the pose quality criteria are not met, the state machine triggers a retry using a different viewpoint or a new *PNP* run, as described in Section 5.2. Otherwise, the estimated pose is used to update the object coordinates for grasp planning. Results obtained during the multiple test runs are evaluated and analyzed further in Section 9.2.

7.3. Coordinate Transformation to Arm Workspace

The *PNP*-based pose estimation described in Section 7.2 computes the object position in the end-effector camera frame (`camera_ee_optical_frame`). To command the robotic arm to grasp the object, this point need to be transformed to the arm workspace, which is defined relative to the `boogie_base` frame. This section describes the coordinate transformation pipeline that converts the estimated object position into the manipulation frame.

When this transformation is performed, the robotic arm is static. The rover has stopped moving after the 180° rotation, and the arm has been commanded to the pantilt_like_view position, but is not actively moving joints. All transforms in the chain from the end-effector camera to the arm base are therefore static and can be queried from the *TF*

tree.

The transformation chain follows the frame hierarchy established in Section 4.1:

$$\texttt{camera_ee_optical_frame} \rightarrow \texttt{gripper_tip} \rightarrow \texttt{ra_base} \rightarrow \texttt{boogie_base} \tag{7.12}$$

Each link in this chain represents a known static transform:

- `camera_ee_optical_frame` → `gripper_tip`: static transformation from Equations 4.3 and 4.4.
- `gripper_tip` → `ra_base`: Current arm configuration from forward kinematics (Section 2.7.1).
- `ra_base` → `boogie_base`: Static mount offset from Equation 4.2.

The implementation queries the complete transformation directly from the *TF* tree using the *ROS* `tf.TransformListener` interface, rather than manually computing each matrix multiplication. This approach simplifies the code and maintains consistency with the rest of the system, which relies on the same *TF* tree for coordinate transformations.

The object position in the `boogie_base` frame is computed as:

$$^{\texttt{boogie_base}}\mathbf{P}_{\text{obj}} = \mathbf{T}^{\texttt{boogie_base}}_{\texttt{camera_ee_optical}} \cdot {}^{\texttt{camera_ee_optical}}\mathbf{P}_{\text{obj}} \tag{7.13}$$

where $\mathbf{T}^{\texttt{boogie_base}}_{\texttt{camera_ee_optical}}$ is the 4×4 homogeneous transformation matrix obtained from the *TF* lookup, and $^{\texttt{camera_ee_optical}}\mathbf{P}_{\text{obj}}$ is the object position estimated by *PNP*.

The transformation is applied to the object position as a homogeneous coordinate:

$$\begin{bmatrix} {}^{\texttt{boogie_base}}x \\ {}^{\texttt{boogie_base}}y \\ {}^{\texttt{boogie_base}}z \\ 1 \end{bmatrix} = \mathbf{T}^{\texttt{boogie_base}}_{\texttt{camera_ee_optical}} \begin{bmatrix} {}^{\texttt{camera_ee_optical}}x \\ {}^{\texttt{camera_ee_optical}}y \\ {}^{\texttt{camera_ee_optical}}z \\ 1 \end{bmatrix} \tag{7.14}$$

The resulting coordinates $({}^{\texttt{boogie_base}}x, {}^{\texttt{boogie_base}}y, {}^{\texttt{boogie_base}}z)$ define the target position in the arm workspace and are passed to the motion planning module for commanding the arm.

The system evaluates the quality of the *PNP* estimate using the mean reprojection error computed during pose estimation (Section 7.2.2). The reprojection error quantifies how well the estimated pose explains the observed feature correspondences. Large reprojection

errors indicate unreliable pose estimates which can come from poor feature distribution or insufficient inliers.

If the reprojection error exceeds an experimentally determined threshold, the system does not immediately accept the transformed position. Instead, the *PNP* estimation process is re-executed with a new image capture. This retry mechanism uses the fact that different images of the same scene produce different *SIFT* features and feature matches. A new image may result in a better-conditioned feature set that results in a more accurate pose estimate with lower reprojection error.

The retry loop continues until the reprojection error falls below the threshold. The retry logic is managed by the master state machine described in Section 5.2.

Once a pose estimate with acceptable reprojection error is found, the system stores the transformed target position ${}^{\texttt{boogie_base}}\mathbf{P}_{\text{obj}}$ and uses it for subsequent motion planning execution. At this stage, residual positioning errors were observed and likely originate from calibration uncertainties, feature localization precision, and mechanical elasticity in the arm structure. Although the pose estimation provides a significantly improved position estimate compared to the initial coarse localization after rotation, the residual error is still sufficient to cause grasp failures if the gripper descends directly to the estimated position. The image-based position refinement mechanism described in Section 7.4 addresses this limitation by performing a final alignment step before the gripper descends to grasp the object.

7.4. Image-Based Position Refinement

The *PNP*-based pose estimation described in Section 7.2 provides significant improvement over the odometry-affected localization. However, experimental testing revealed that residual positioning errors still prevented reliable grasping. In test runs without additional correction, only 3 out of 30 grasp attempts succeeded (10% success rate), with the gripper typically landing approximately 30 mm away from the target object.

To enable experimental validation of the complete grasping pipeline, this section describes a simple image-based correction approach developed as a proof-of-concept. This method uses visual feedback from the end-effector camera to iteratively adjust the gripper position before the final grasp attempt.

7.4.1. Operating Constraints

The end-effector camera is mounted with an 18° downward tilt (Equation 4.6), causing objects to appear at different pixel locations depending on gripper height. Additionally, testing revealed that *YOLO*v7 object detection only functions reliably at heights of 70 mm or above, as closer distances exceed the camera's minimum focus distance and

produce blurred images. For these reasons, the system performs all visual refinement operations at a fixed height of 100 mm.

7.4.2. Calibration and Target Pixels

To determine where the object should appear in the image when properly centered below the gripper, a simple calibration procedure was performed. The orange cube was placed directly below the gripper at known heights from 50 mm to 100 mm in 10 mm steps. At each height, the pixel coordinates corresponding to a centered object were manually recorded. Table 7.1 shows the measured target pixel locations.

Table 7.1: Target pixel coordinates for object centering at different gripper heights.

Height [mm]	Target U [px]	Target V [px]
50	708	324
60	697	282
70	694	243
80	688	228
90	685	219
100	679	153

Figure 7.8 shows representative calibration images with the target pixels marked.



Figure 7.8: Calibration images at 50 mm (left) and 100 mm (right) showing target pixel locations (red crosshair) for centered object positioning.

7.4.3. Refinement Procedure

After the arm moves to the *PNP*-estimated target position, the object is typically not visible in the camera due to the positioning error. The system then performs a grid search pattern at 100 mm height, moving the gripper in 20 mm steps around the estimated position (center, then cardinal and diagonal directions). The system detected the object within 1-3 search positions (median 1 attempt).

Once detected, an iterative correction loop adjusts the gripper position to align the object's bounding box center with the target pixel coordinates from the calibration table.

At each iteration:

1. The system captures an image and detects the object bounding box.
2. The pixel error between the bounding box center and target pixel is computed.
3. If the error is below 35 pixels (experimentally determined threshold) in both axes, centering is complete.
4. Otherwise, the gripper position is adjusted based on the pixel error using correction gains (tuned during experimentation).
5. The process repeats until convergence or a maximum iteration limit (see Section 5.2).

Figure 7.9 shows a typical iteration with the target pixel (green), detected object center (red), and pixel error vector (yellow arrow). Appendix E.2 shows a complete run demonstrating how the error decreases through successive iterations.

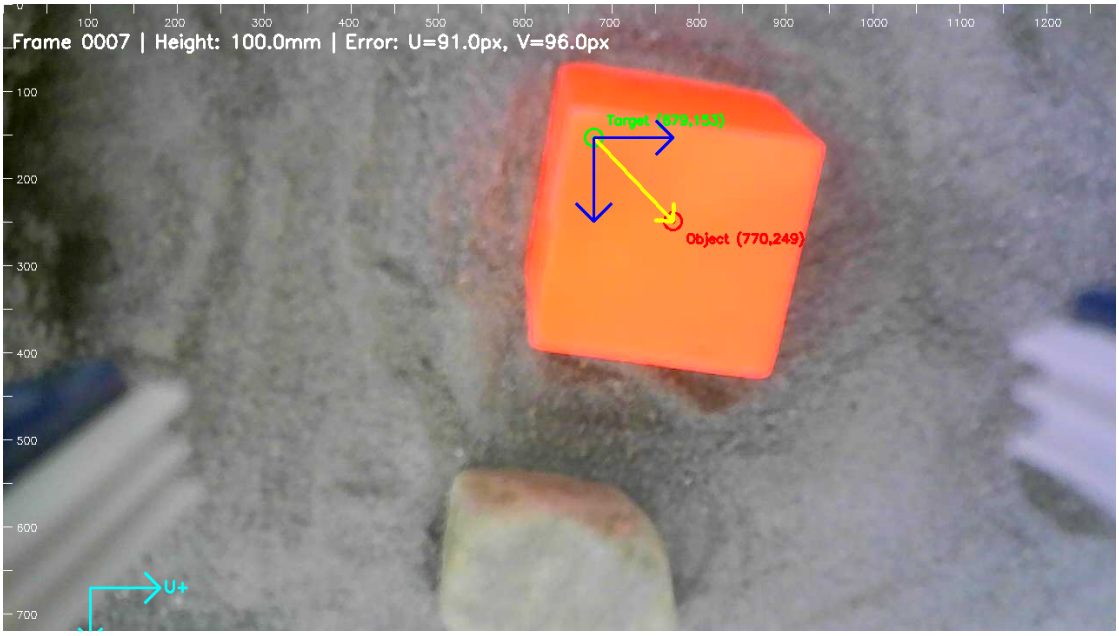


Figure 7.9: Visual refinement iteration showing target pixel (green circle), detected object center (red circle), and correction vector (yellow arrow). On the top of the image an overlay with the image number (represents the iteration of the loop), height (100 mm), and the pixel error in both u (91px) and v (96px) axis.

Trial-and-error testing tuned the correction gains and step limits to achieve acceptable convergence behavior. This approach is not based on a rigorous control law, but rather serves as a pragmatic solution to enable grasp testing within the project timeline.

7.4.4. Performance and Limitations

This visual refinement approach completes the autonomous grasping pipeline and demonstrates successful object retrieval within the thesis timeline. The minimal centering implementation achieved the end goal: all runs that reached the centering stage converged successfully, even though some iterations required extended convergence times (up to 28 iterations). Systematic evaluation (Section 9.3) shows a 100% success rate across five end-to-end runs, demonstrating that the approach is sufficient for proof-of-concept validation of the complete grasping pipeline.

The main limitations observed during testing were:

- **Oscillation:** During development testing with manually placed initial conditions, oscillation occurred in approximately 15-20% of attempts, with the correction loop alternating between positions on either side of the target. However, in systematic end-to-end evaluation (Section 9.3), no oscillation failures occurred. This suggests that the PnP motion naturally avoids configurations that lead to centering oscillation.

- **Fixed operating height:** The 100 mm operating height constraint (required for reliable detection) limits final positioning accuracy.

- **Empirical tuning:** The correction parameters were tuned for a specific test scenario and may not generalize to different object sizes, positions, or lighting conditions.

This method validated the complete autonomous grasping pipeline. A more robust implementation requires proper *IBVS* control with online Jacobian computation [27]. However, implementing velocity-level control was outside the scope of this thesis. Potential improvements are discussed in Section 10.3.

8. Control and Manipulation

This chapter describes the control and manipulation components required to execute autonomous grasping. Section 8.1 presents the workspace validation approach that ensures detected targets are physically reachable before motion planning. Section 8.2 describes the path planning interface and wrapper implementation used to command collision-free arm motions. Section 8.3 details the motor current-based grasp detection method developed to confirm successful object grasping without additional sensors.

The inverse kinematics formulation used by the system was already covered in Section 2.7.2 and is not repeated here.

8.1. Reachability-Based Target Validation

Before commanding the arm to move toward a detected object, the system validates whether the target position is reachable. This prevents the motion planner from attempting trajectories to unreachable targets.

The reachability validation uses the workspace analysis from Conenna [25]. In that work, the reachable workspace of the *LRM* robotic arm was evaluated by solving inverse kinematics over a Cartesian grid covering the ground-level grasping area and the sample collection basket. For each grid point, multiple end-effector orientations were tested while enforcing joint limits and checking for self-collisions with the rover structure.

This work implements a simple reachability check at the planning level. When an object is detected and its 3D position is computed in the arm base frame, the target is validated against the reachable workspace. Targets that fail this validation are rejected before being passed to the motion planner.

8.1.1. Workspace Constraints

Based on Conenna's reachability analysis [25], this work approximates the workspace using a Cartesian bounding box in the arm base frame:

$$x \in [-0.25,\, 0.38]\ \text{m}, \quad y \in [-0.30,\, 0.30]\ \text{m}, \quad z \in [0.00,\, 0.65]\ \text{m}.$$

In addition to these Cartesian bounds, a radial distance constraint is enforced:

$$0.16\ \text{m} \leq \sqrt{x^2 + y^2 + z^2} \leq 0.38\ \text{m}.$$

The minimum radial distance (0.16 m) enforces Conenna's collision-avoidance constraint, requiring the end-effector to maintain clearance from the arm base and rover structure [25].

The maximum radial distance (0.38 m) corresponds to the arm's maximum reach from Conenna's analysis.

Joint limits are enforced separately by the low-level servo controller (Section 4.3.2), with each joint constrained to:

$$\theta_i \in [-100^\circ, 100^\circ].$$

This two-stage validation checks Cartesian bounds at the planning level and joint limits at the control level.

8.2. Path Planning

Path planning for the robotic arm is handled by the *RMPL*, discussed in Section 2.7.2, an in-house tool developed at *DLR*. The library requires only the *URDF* model as input and provides functions to compute collision-free trajectories in both joint space and Cartesian space.

8.2.1. RMPL Integration

The *RMPL* planner handles inverse kinematics computation internally and generates collision-free trajectories between the current arm state and the target pose while respecting joint limits and avoiding self-collisions. The library was developed prior to this work and is used without modification through its provided *Application Programming Interface (API)*.

To integrate *RMPL* into the autonomous grasping pipeline, a wrapper class called `MotionPlanning` was developed to provide a unified interface for path planning across different system components, including *RAFCON* state machines and the high-level *GUI*. This wrapper implements a Singleton pattern to ensure consistent planner state across all subsystems and provides two main planning modes:

- **Joint space planning:** Plans a trajectory from a given joint configuration to a target joint configuration.
- **Cartesian space planning:** Plans a trajectory to reach a target position (x, y, z) in the `boogie_base` frame, with optional orientation constraints.

For Cartesian planning, the wrapper supports two orientation control modes. In goal-only orientation mode, the end-effector orientation is constrained only at the final target position. In continuous orientation mode, the orientation is maintained throughout the entire trajectory. The continuous orientation mode is used during grasping operations to keep the gripper pointing downward toward the ground throughout the approach motion.

8.2.2. Path Interpolation

The *RMPL* planner outputs a sequence of joint states representing the planned trajectory. To achieve smooth and steady arm motion, these waypoints are interpolated using a configurable step size. Through experimental testing, an interpolation delta of 0.1 radians ($\approx 5.7°$) was found to produce smooth motion without excessive trajectory points. This value was tuned empirically during field tests to balance motion smoothness with computational overhead.

The interpolated trajectory is then executed by commanding the arm servo motors sequentially through each waypoint, as described in Section 3.1.2.

8.3. Autonomous Grasp Execution Using Motor Current Feedback

Marco Conenna's work developed a 6-*DOF* robotic arm with validated kinematics. To demonstrate the arm's functionality, a *Tool Center Point (TCP)* equipped with a visual tracker was used. This thesis replaces the original *TCP* with a physical gripper mounted after the sixth link. The gripper uses a linear gear mechanism and is actuated by a single A20CLS servo motor (Appendix B.3.4), following the low-cost design philosophy of the system.

With the introduction of a physical gripper, the manipulation task requires more than simply positioning the end-effector at the target location. After object detection and arm pose correction using visual feedback to place the gripper around the target, the system must determine whether the object has actually been grasped. Without grasp confirmation, the system cannot reliably lift or transport the object.

Typical robotic systems achieve grasp confirmation using tactile, force, or torque sensors located at the gripper fingers, as discussed in Section 2.9. However, the current Middleboard design has no available *Input / Output (IO)* pins, which prevents the integration of such sensors. As a result, grasp confirmation must rely on sensing capabilities already available in the system.

The onboard current sensor on the Middleboard addresses this limitation. This sensor measures the total electrical current drawn by all seven motors of the robotic arm, including the gripper actuator. Although this measurement does not isolate the gripper servo current directly, experimental testing showed that grasp detection is still possible when the arm remains stationary during closing.

When the arm is in a static configuration, the measured total current remains approximately constant around a baseline value, with small fluctuations of about ± 0.2 A. During grasp execution, only the gripper joint is commanded to move, while all other joints remain stationary. Under these conditions, variations in the measured current can be

attributed primarily to the gripper motor.

8.3.1. Observed Current Behavior

When the gripper closes without contacting an object, the measured arm current remains close to a nominal value, with small oscillations. When the gripper fingers make contact with an object, the current increases as additional torque is required to continue closing. Once the object becomes mechanically constrained between the fingers, further closing does not result in a significant increase in current; instead, the current stabilizes at a higher value compared to the initial nominal level.

This behavior allows autonomous grasp detection using motor current.

8.3.2. Initial Grasp Detection Logic

Grasp execution uses an incremental closing process, where the gripper angle is reduced in steps of 2° while continuously monitoring the measured current. The logic is divided into two phases:

1. **Contact detection:** The gripper closes until the measured current exceeds a predefined lower threshold, indicating initial contact between the gripper fingers and an object.

2. **Stable current detection:** After contact is detected, the gripper continues closing while the current is monitored over a range of 15°. If the current remains approximately constant during this motion, the object is considered grasped.

During initial testing of this approach, it became evident that the absolute value of the measured arm current depends not only on gripper interaction, but also on the static joint configuration of the robotic arm prior to grasping. Since the arm configuration varies depending on object location, kinematically unfavorable poses (such as extended configurations or elevated servo temperatures) resulted in significantly higher baseline currents compared to more favorable configurations.

Fixed absolute current thresholds proved insufficiently robust. In some arm configurations, the baseline current already exceeded the thresholds used for contact detection, leading to premature detection. Increasing the thresholds to compensate this resulted in missed grasps when the arm was in low-load configurations.

8.3.3. Pre-Grasp Current Calibration

A pre-grasp current calibration step addresses this issue by computing a dynamic baseline current. Before closing the gripper, the arm reaches a static pre-grasp pose. In this state, the total motor current is measured over a time window of 3.0 s with a sampling period of 0.05 s. The median of these samples (after rejecting outliers beyond $3.0 \times$ *Median Absolute Deviation (MAD)*) gives the baseline current value I_{base}.

This baseline current represents the steady-state electrical load of the arm for the current configuration and serves as a reference for subsequent grasp detection.

8.3.4. Delta-Current Based Detection

Instead of using absolute current values, grasp detection is performed using the difference between the instantaneous current and the calibrated baseline current. Let $I(t)$ denote the measured current during grasp execution. The delta current is defined as:

$$\Delta I(t) = I(t) - I_{\text{base}} \tag{8.1}$$

The system detects contact and grasp events when $\Delta I(t)$ exceeds dynamically computed thresholds. To maintain robustness across different arm configurations, the system computes these thresholds as:

$$\Delta I_{\text{lower}} = \max(0.4 \text{ A},\ 0.5 \cdot I_{\text{base}}) \tag{8.2}$$

$$\Delta I_{\text{upper}} = \max(0.8 \text{ A},\ 1.0 \cdot I_{\text{base}}) \tag{8.3}$$

This hybrid approach works in both low-load and high-load joint configurations without manual retuning. The fixed offsets (0.4 A, 0.8 A) provide sensitivity when the baseline current is low, while the proportional terms (0.5× and 1.0× I_{base}) scale the thresholds for high-load configurations.

8.3.5. Stable Current Detection and Post-Grasp Tightening

Following initial contact detection, the gripper continues closing while monitoring the evolution of $\Delta I(t)$. Once the object is mechanically constrained between the fingers, further closing does not lead to a significant increase in current, and $\Delta I(t)$ stabilizes.

The system confirms a grasp when $\Delta I(t)$ remains within ± 0.15 A over a closing range of 15° (validated through experimental testing). Once the system detects a grasp, it closes the gripper by an additional 4°. This post-grasp tightening step secures the hold without excessive servo load and current draw.

8.3.6. Failure Classification

Multiple failure states in the grasp execution loop improve robustness. The system classifies unsuccessful grasp attempts based on observed current behavior and gripper motion:

- **Empty grasp:** The gripper reaches its minimum angle without detecting any contact, indicating that no object is present between the fingers.
- **Contact without grasp:** The system detects contact but observes no stable current behavior before reaching the angle limit.

- **System error:** Current sensor read failures or arm error states occur during execution.

These failure modes are propagated to the master state machine, which determines the appropriate recovery strategy, as described in Section 5.2.

This approach enables autonomous grasp detection using only motor current feedback already available in the system. By introducing baseline current calibration and delta-current based detection, the method becomes robust to variations in arm configuration and joint load. While this approach does not measure grasping force directly, it enabled reliable grasp execution in field tests. Section 9.4 presents detailed quantitative results.

9. Experimental Evaluation

This chapter evaluates the autonomous grasping pipeline described in Chapters 6, 7 and 8. All experiments were conducted at the *DLR* Planetary Exploration Lab, which simulates a lunar-like environment with basaltic grey sand and rocks of various sizes and shapes (Figure 9.1).

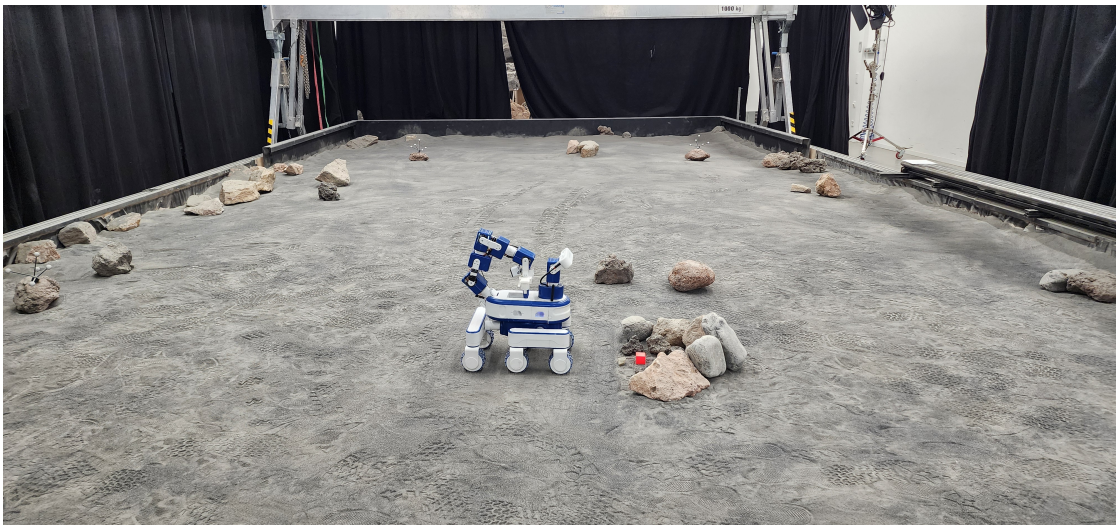


Figure 9.1: Experimental setup at the DLR Planetary Exploration Lab showing the Lunar Rover Mini operating in a simulated lunar environment.

For all experimental runs, the pre-mission requirements from Section 4.4 were enforced: only one target object in the field of view of both cameras, rover positioned approximately 30 cm from the object, all *ROS* processes running, stable communication to the Middleboard, and the robotic arm in home position. The target object was an orange 3×3×3 cm cube placed on the rocky surface, selected based on the detection robustness analysis in Section 9.1.

The evaluation covers individual subsystem performance and integrated end-to-end operation. Section 9.1 evaluates *YOLO*v7-based object detection with both RealSense and monocular cameras. Section 9.2 examines *PNP* pose estimation convergence under the object center reprojection error criterion. Section 9.3 analyzes the image-based visual refinement loop. Section 9.4 validates the current-based grasp detection method. Section 9.5 assesses the complete pipeline from detection through object placement.

9.1. Object Detection Results

This section presents experimental results from the object detection module used in the autonomous grasping pipeline. Detection performance was tested under different lighting conditions and camera configurations using both the RealSense depth camera and the monocular end-effector camera.

The object detection model is based on *YOLO*v7, pretrained by Plácido de Castro [8] on 5×5×5 cm cubes and reused in this work without retraining or modification. All results were obtained from system logs and recorded images during experimental testing.

9.1.1. RealSense Depth-Based Object Detection

Object detection with the RealSense depth camera was evaluated using three 3×3×3 cm cube colors: green, blue, and orange. For each color, four configurations were tested by varying lighting (lights on/off) and object position in the camera field of view (centered/shifted right), resulting in twelve experimental runs.

For each detection, the system outputs the object class, confidence score, and bounding box in image coordinates. The depth value at the bounding box center is used to reconstruct the 3D object position in the camera optical frame, then transformed to the `base_link` and `map` frames and published as visualization markers.

Tables 9.1 and 9.2 show results extracted from system logs. Table 9.1 presents detection success rates and confidence scores for all twelve tested configurations. Table 9.2 shows the reconstructed 3D positions for the ten successful detections across different coordinate frames. The complete dataset is in Appendix E.1.

Table 9.1: RealSense object detection results: detection success and confidence scores under varying lighting and object position conditions.

Detected Class	**Lighting**	**Position**	**Detected**	**Confidence [%]**
Green	On	Center	No	–
Green	Off	Center	Yes	92.20
Green	Off	Right	Yes	94.95
Green	On	Right	Yes	79.99
Blue	On	Center	No	–
Blue	Off	Center	Yes	88.59
Blue	Off	Right	Yes	93.00
Blue	On	Right	Yes	92.76
Orange	On	Center	Yes	89.76
Orange	Off	Center	Yes	94.29
Orange	Off	Right	Yes	89.91
Orange	On	Right	Yes	91.95

Table 9.2: RealSense 3D position reconstruction for successful detections. Coordinates are shown in camera optical frame, base_link frame, and map frame.

Class	**Lighting**	**Position**	**Camera** (x, y, z) **[m]**	**Base_link** (x, y, z) **[m]**	**Map** (x, y, z) **[m]**
Green	Off	Center	(0.0432, -0.0153, 0.407)	(0.364, -0.010, -0.121)	(0.365, -0.012, -0.123)
Green	Off	Right	(0.1164, 0.0286, 0.373)	(0.309, -0.083, -0.122)	(0.310, -0.085, -0.124)
Green	On	Right	(0.1156, 0.0286, 0.373)	(0.309, -0.082, -0.122)	(0.309, -0.082, -0.122)
Blue	Off	Center	(0.0559, -0.0026, 0.398)	(0.348, -0.022, -0.121)	(0.348, -0.022, -0.121)
Blue	Off	Right	(0.1189, 0.0294, 0.372)	(0.308, -0.086, -0.122)	(0.308, -0.086, -0.122)
Blue	On	Right	(0.1185, 0.0290, 0.371)	(0.307, -0.085, -0.121)	(0.308, -0.086, -0.123)
Orange	On	Center	(0.0520, -0.0060, 0.408)	(0.358, -0.018, -0.127)	(0.358, -0.019, -0.129)
Orange	Off	Center	(0.0512, -0.0059, 0.402)	(0.354, -0.018, -0.123)	(0.353, -0.018, -0.122)
Orange	Off	Right	(0.1197, 0.0294, 0.376)	(0.310, -0.086, -0.125)	(0.310, -0.086, -0.125)
Orange	On	Right	(0.1236, 0.0296, 0.387)	(0.317, -0.090, -0.134)	(0.316, -0.091, -0.133)

Figure 9.2 shows detection of the green cube under different lighting conditions. The cube is detected in both cases, but confidence varies due to surface reflections. Figure 9.3 demonstrates robust detection of the orange cube under strong illumination with the object shifted laterally, showing insensitivity to moderate position changes within the field of view.



Figure 9.2: Green cube detection under lights-off (top) and lights-on (bottom) conditions. Detection fails when the light is on due to the cube's reflective surface.



Figure 9.3: Orange cube detection under lights-on with object shifted laterally in field of view. Detection confidence remains high (0.92) despite position offset, confirming robustness to moderate lateral shifts. The matte orange surface shows minimal reflection artifacts compared to the green and blue cubes.

9.1.1.1. Analysis

Table 9.1 shows that green and blue cubes both failed detection when centered under lights-on conditions, but were detected successfully in all other configurations. When detected, confidence scores ranged from 79.99% to 94.95%. The green cube achieved its lowest confidence (79.99%) under lights-on with lateral position shift.

The orange cube was detected successfully in all four tested configurations, with confidence scores from 89.76% to 94.29%. Unlike the green and blue cubes, the orange cube showed no detection failures under any lighting or position combination.

The detection failures for green and blue cubes under centered, lights-on conditions are caused by specular reflections from their partially reflective painted surfaces. Under strong illumination, these reflections create bright spots that reduce feature contrast and interfere with object boundary identification. The orange cube's matte surface minimizes specular highlights, allowing consistent detection across all lighting conditions.

Table 9.2 shows reconstructed 3D coordinates for all successful detections across three reference frames: RealSense camera optical frame, `base_link` frame (rover body), and `map` frame (global odometry reference). In the camera frame, centered objects appear at approximately $x \approx 0.05\,\text{m}$ and $z \approx 0.40\,\text{m}$, while right-shifted objects appear at

approximately $x \approx 0.12\,\text{m}$ and $z \approx 0.37\,\text{m}$. The negative y coordinates for centered objects and positive y coordinates for right-shifted objects confirm the expected lateral displacement.

After transformation to `base_link`, all objects are located at approximately $z \approx -0.12\,\text{m}$, corresponding to ground level relative to the rover base. The x coordinates range from 0.31 m to 0.36 m, placing objects in the forward operating region. The close correspondence between `base_link` and `map` coordinates (typically 1-2 mm difference) shows minimal visual odometry drift during these static detection tests.

The orange cube was selected as the reference object for the autonomous grasping pipeline due to its 100% detection success rate and consistently high confidence scores across all tested conditions.

Variations in object size were previously investigated by Plácido de Castro [8]. Since the detection model and training procedure were reused without modification, further size-dependent experiments were considered outside the scope of this work.

9.1.2. Monocular End-Effector Camera Object Detection for Centering

Object detection with the monocular end-effector camera was evaluated as a function of camera-to-object distance during the visual centering operation (Section 7.4). The camera was positioned directly above the object with fixed x and y coordinates, while height z was varied from 50 mm to 100 mm in 10 mm increments.

This experiment identified detection limits caused by optical effects such as blur and depth-of-field constraints rather than geometric misalignment.

Table 9.3: Monocular object detection results as a function of camera height during centering operation.

Height z [mm]	**Detected**	**Class**	**Confidence [%]**
50	No	–	–
60	No	–	–
70	Yes	Orange	90.12
80	Yes	Orange	92.47
90	Yes	Orange	93.85
100	Yes	Orange	94.10

Figure 9.4 illustrates the detection limit due to optical constraints. At 70 mm, the object is in focus and detected with 90% confidence. At 50 mm, the image is blurred beyond the camera's depth of field, causing detection failure.

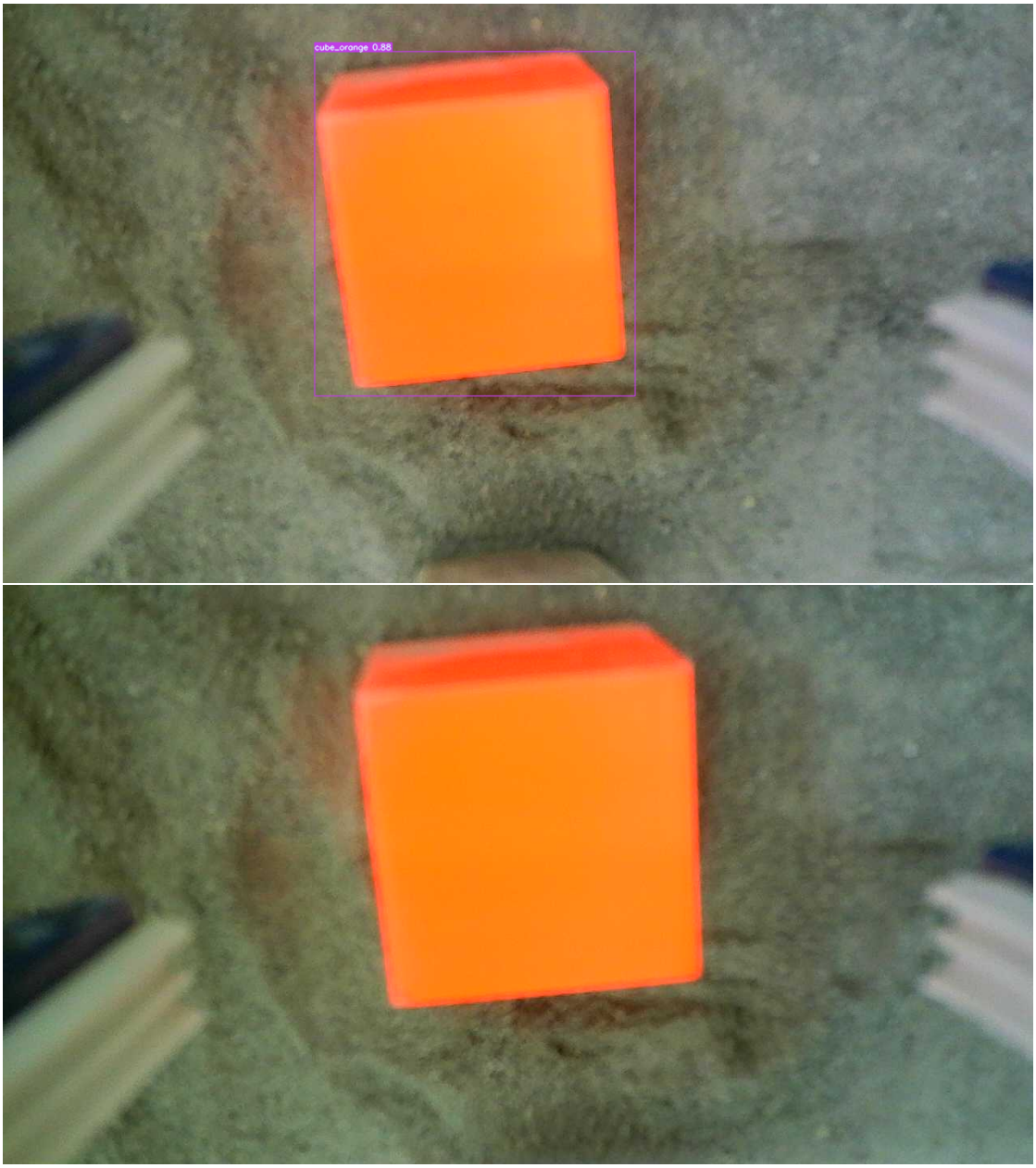


Figure 9.4: Monocular detection at 70 mm (top) and detection failure at 50 mm (bottom) due to image blur. At 70 mm, the cube is in focus and detected with 90.12% confidence. At 50 mm, the image exceeds the camera's minimum focus distance, resulting in blur that prevents feature extraction. No bounding box is drawn when detection fails.

9.1.2.1. Analysis

Detection failures below approximately 70 mm were consistently caused by image blur, which prevented reliable feature extraction. No false-positive detections were observed; the model failed conservatively when image quality degraded.

These results show that monocular detection performance is limited by optical constraints rather than model misclassification. Based on this observation, monocular detection in the centering loop (Section 7.4) is only performed at 100 mm height, where detection is reliable and the operating point is well within the camera's focus range.

The complete detection image dataset (RealSense and monocular) is in Appendix E.1. Potential improvements to the object detection module are discussed in Section 10.3.1.

9.2. PnP Pose Estimation Evaluation

This section evaluates the feature-based *PNP* pose estimation pipeline described in Chapter 7. The analysis is based on per-run logs and debug frames saved during experiments.

In the implementation, the acceptance criterion for a *PNP* estimate is the *object center reprojection error*. The value is set to 50 px and is the pixel distance between the detected object center in the monocular image and the object center point from the RealSense depth map transformed and reprojected using the estimated pose, as described in Section 7.2.5. The retry logic uses this metric to accept a pose or trigger additional attempts and viewpoint changes.

Even when the same input images are used, repeated executions of the pipeline may produce different pose estimates. *RANSAC*-based pose estimation does not follow a fixed computation path. The algorithm tests randomly selected hypotheses and keeps the one that best fits the data. As a result, different runs may rely on different subsets of correspondences, resulting in different pose solutions.

9.2.1. Feature Matching and PnP Inlier Statistics

The pose estimation relies on 3D–2D correspondences derived from *SIFT* feature matches (Section 7.1). After ratio test and homography-based *RANSAC* filtering, the remaining correspondences are passed to `solvePnPRansac`, which returns a robust inlier subset. Table 9.4 shows the correspondence and inlier statistics at the accepted attempt for each run where *PNP* converged. Acceptance depends on the object center reprojection error, as detailed in Section 9.2.2.

Across the accepted runs (R1–R7), the correspondence count at the accepted attempt is 35.43 ± 11.76, and the inlier ratio is $60.25 \pm 17.07\%$. Table 9.4 shows that accepted

Table 9.4: Feature matching and PnP inlier statistics at the accepted attempt. The inlier ratio is computed as inliers divided by total correspondences.

Run	Accepted error [px]	Correspondences	PnP inliers	Inlier ratio [%]
R1	42.75	25	15	60.0
R2	46.03	40	35	87.5
R3	30.96	43	21	48.8
R4	48.93	25	20	80.0
R5	40.48	46	20	43.5
R6	47.08	49	23	46.9
R7	45.08	20	11	55.0

runs exhibit different inlier ratios while reaching comparable object center reprojection errors. This occurs because acceptance is evaluated on the reprojected object center, not on feature-level residuals.

Figure 9.5 shows the geometrically consistent feature correspondences for an accepted attempt. The matched keypoints represent the 3D–2D correspondences used by `solvePnPRansac` to estimate the object pose.

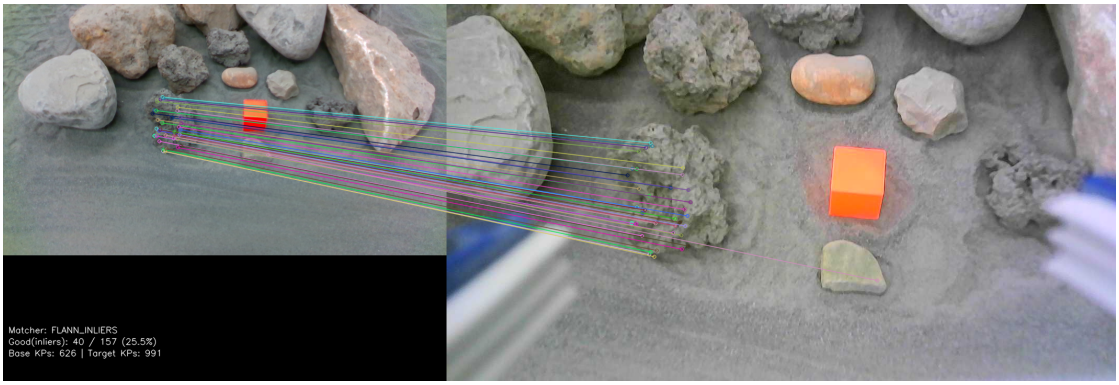


Figure 9.5: Inlier feature correspondences for the accepted attempt of run R2. The lines connect geometrically consistent SIFT keypoint matches between the reference RealSense image (left) and the monocular end-effector image (right). These 3D–2D correspondences are used by solvePnPRansac to estimate the object pose.

9.2.2. PnP Convergence Outcome Based on Object Center Reprojection Error

Table 9.5 summarizes the *PNP* outcome for all evaluated runs. Each row shows one run with the number of *PNP* attempts executed, the minimum observed object center reprojection error within the run, and the final status (accepted or rejected). For accepted runs, the minimum error is the accepted attempt. For rejected runs, it is the best attempt before termination.

Table 9.5: PnP outcome summary using the object center reprojection error (green detected center vs. red reprojected center). "Accepted" means the run reached the 50 px acceptance threshold. "Rejected" means the system terminated after extended retries without meeting the threshold.

Run	Attempts	Final error [px]	Status	Notes
R1	1	42.75	Accepted	Successful grasp
R2	3	46.03	Accepted	Successful grasp
R3	1	30.96	Accepted	Successful grasp
R4	1	48.93	Accepted	Successful grasp
R5	1	40.48	Accepted	Successful grasp
R6	2	47.08	Accepted	Planner failure (out of workspace)
R7	1	45.08	Accepted	Object lost after PnP
NC1	20	62.76	Rejected	Termination after extended retries
NC2	20	42.67	Rejected	Threshold tightened to ≈20 px for testing

Across the accepted runs (R1–R7), the accepted object center reprojection error is 43.04 ± 6.01 px (mean $\pm$ standard deviation), with a minimum of 30.96 px (R3) and a maximum of 48.93 px (R4). The number of attempts required until acceptance is 1.43 ± 0.79, and most runs converged on the first attempt (Table 9.5).

Figure 9.6 illustrates the acceptance metric. In both images, the green marker shows the detected object center in the monocular image and the red marker shows the object center from the RealSense depth map transformed and reprojected using the estimated pose. The pixel distance between these two markers is the object center reprojection error used by the retry logic.



Figure 9.6: Object center reprojection error used for PnP acceptance. Left: accepted run example with smaller pixel offset between the detected object center (green) and the reprojected object center (red). Right: rejected attempt example with a larger pixel offset, which triggered additional retries.

9.2.3. Retry Behavior Above Object Center Reprojection Error Threshold

The retry logic is triggered when the object center reprojection error exceeds 50 px. The system then repeats the perception and estimation cycle, potentially under a different monocular viewpoint. Table 9.6 shows an accepted run with multiple attempts, where

the object center reprojection error decreases across consecutive attempts until the acceptance condition is met. Acceptance is evaluated on the reprojection error of the current attempt, not on the minimum error observed over all intermediate attempts. A run may temporarily reach lower reprojection errors during retries without converging if manual terminated before acceptance.

Table 9.6: Example of retry behavior in an accepted run (R2). Each attempt is a new estimation cycle. Acceptance depends on the object center reprojection error.

Attempt	Object center error [px]	Correspondences	PnP inliers	Inlier ratio [%]
1	53.85	41	30	73.2
2	51.19	44	40	90.9
3	46.03	40	35	87.5

The progression from attempt 1 (53.85 px) to attempt 3 (46.03 px) demonstrates how the retry logic reduces the reprojection error through additional observations. Figure 9.7 visualizes the pose quality for the accepted attempt through the monocular reprojection with refined inlier correspondences.



Figure 9.7: Monocular reprojection with refined inlier correspondences for the accepted attempt of run R2 (46.03 px error). The visualization shows the quality of the pose estimate through the reprojected features and object center alignment.

For rejected runs, the system executed many attempts and viewpoint changes without meeting the 50 px acceptance threshold before manual termination. Table 9.7 summarizes

three representative attempts of one rejected run: the first attempt, the best attempt (minimum object center reprojection error), and the final attempt before termination.

Table 9.7: Representative attempts of a rejected run (NC1): first attempt, best attempt (minimum error), and final attempt before manual termination.

Attempt	Object center error [px]	Correspondences	PnP inliers	Inlier ratio [%]
1	53.56	32	19	59.4
8	51.92	32	20	62.5
20	62.76	46	23	50.0

Figure 9.8 shows the best observed attempt of the rejected run NC1 [1]. The top panel shows the basic object center reprojection (green detected center vs. red reprojected center), while the bottom panel shows the monocular reprojection with error visualization. The best observed attempt (51.92 px) remained above the 50 px threshold.

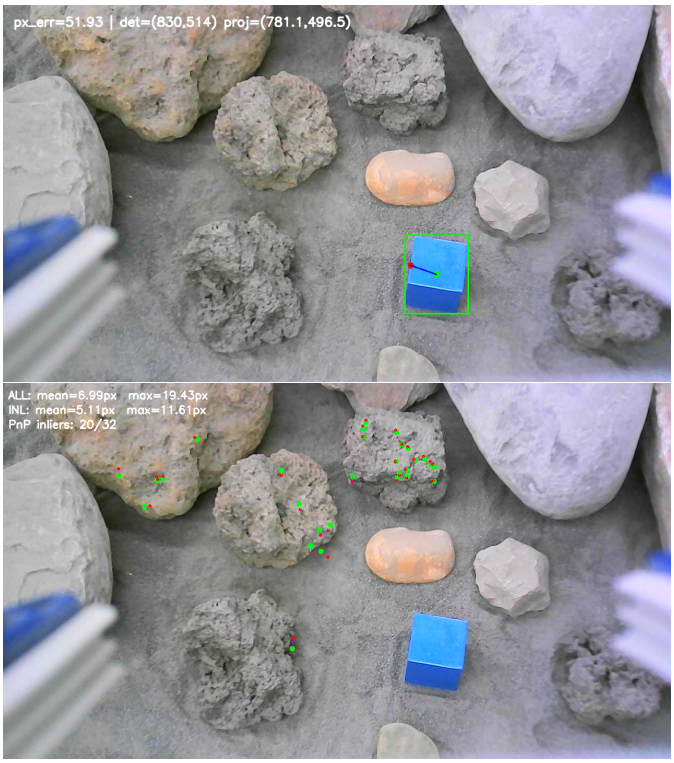


Figure 9.8: Object center reprojection frames for the best observed attempt of rejected run NC1 (51.93 px). Top: detected object center (green) vs. reprojected center (red). Bottom: monocular reprojection with error visualization showing pose quality.

[1] Note that run NC1 used a blue cube instead of the orange cube used in all other runs; however, object detection succeeded in this case, and the reported results reflect purely the *PNP* pose estimation performance independent of the target object's appearance.

Figure 9.9 shows the final attempt before manual termination. The error increased to 62.76 px, and both visualization types reveal poor pose quality. The offset between the green and red markers in the top panel is clearly larger than in the best attempt, and the bottom panel shows degraded alignment in the reprojected features.

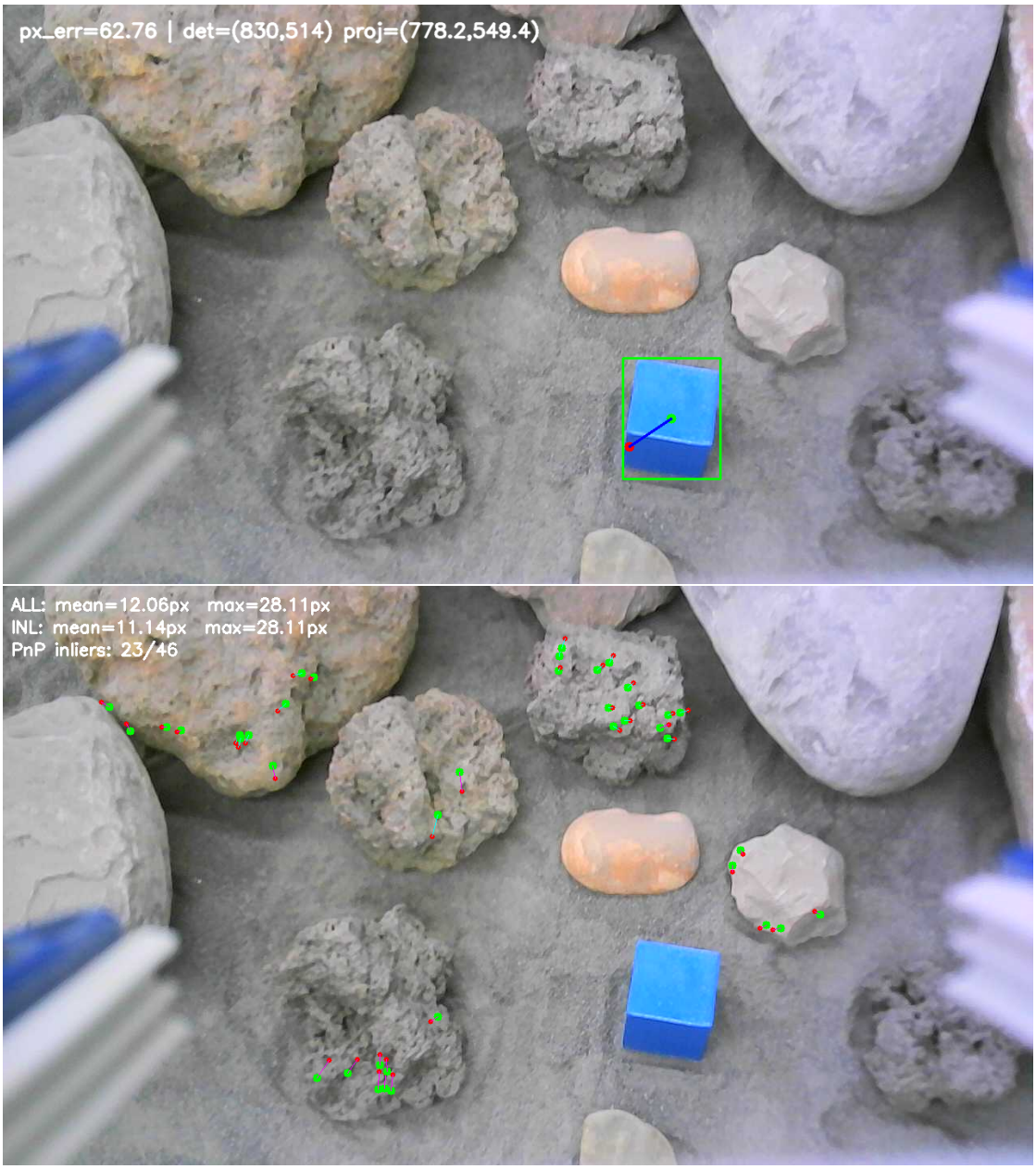


Figure 9.9: Object center reprojection frames for the final attempt of rejected run NC1 before manual termination (62.76 px). Top: detected object center (green) vs. reprojected center (red) with larger offset. Bottom: monocular reprojection showing degraded pose quality.

9.2.4. PnP Accepted but Run Failed in Later Stages

Two runs converged in *PNP* according to the acceptance criterion but failed later due to reasons outside the scope of pose estimation (Table 9.8). This separates downstream failures from PnP estimation errors. The object-loss case is further analyzed in Section 9.3, where re-detection and image-based alignment are addressed.

Table 9.8: Runs where PnP converged (accepted) but the overall run did not complete successfully due to downstream causes.

Run	Accepted error [px]	Downstream outcome (out of scope here)
R6	47.08	Motion planning failed (no collision-free solution; target outside reachable workspace)
R7	45.08	Object lost after PnP (handled in visual refinement section)

9.2.5. Analysis

Iterative experimental testing determined the 50 px acceptance threshold as a trade-off between pose accuracy and convergence reliability. Lower thresholds (e.g., 20 px as tested in NC2) improved theoretical accuracy but reduced convergence rates. During early testing, isolated cases achieved a minimum object center reprojection error of approximately 5 px. However, systematic evaluation could not reproduce this result, with accepted errors ranging from 30.96–48.93 px. The spatial distribution of feature correspondences explains this discrepancy. The detector only finds *SIFT* keypoints in high-texture regions—in this case, predominantly on the lunar rock simulant surrounding the cube. The smooth, uniform surface of the cube itself produces few or no reliable keypoints. As a result, features located in the rock regions primarily constrain the *PNP* optimization, while the cube center position is extrapolated from this spatially limited feature set. Small angular errors in the estimated pose, which have minimal effect on reprojection accuracy at the feature locations (typically 2–3 px as reported by `solvePnPRansac`), propagate as larger positional errors at the cube center because the object center is spatially separated from the feature cluster used for pose estimation. The early 5 px result likely occurred when features spread more evenly around the object center, while the systematic evaluation represents the typical case where features cluster in textured regions. The 50 px threshold accommodates this extrapolation uncertainty while still allowing successful grasps, representing the consistently achievable performance under realistic feature distribution conditions.

The evaluated runs show that the *PNP* stage converges using the 50 px object center reprojection error threshold (Table 9.5). For the runs where the full grasp sequence succeeded (R1–R5), accepted object center reprojection errors lie in the range 30.96–48.93 px, all below the 50 px threshold. In the rejected runs (NC1, NC2), the system terminated after extended retries without meeting the configured acceptance threshold. NC2 is a configuration-driven case where the threshold was tightened to approximately 20 px for testing; under this setting, the best observed attempt (42.67 px) remained above the acceptance criterion and did not converge.

The feature matching and *RANSAC* statistics in Table 9.4 show the feature correspondence quality for the accepted poses. The system achieved convergence with correspondence counts on the order of a few tens and with robust inlier subsets, while acceptance depended on the object center reprojection error. The retry example in Table 9.6 and Figures 9.5 and 9.7 show that repeating the estimation cycle can reduce the object center reprojection error and lead to acceptance. In contrast, Table 9.7 and Figures 9.8 and 9.9 show a representative rejected run where retries did not achieve consistent convergence before manual termination.

The special-case runs in Table 9.8 show that *PNP* convergence is necessary but not sufficient for overall task completion. A converged pose can still lead to downstream failure when the target is outside the arm workspace (R6) or when the object is not found by subsequent perception stages (R7). These downstream aspects are treated in the corresponding sections of the thesis.

During repeated testing, part of the reprojection error variability came from image quality degradation when the end-effector camera lens became contaminated (dust on the lens surface). After cleaning the lens, the mean pixel reprojection error returned to approximately 30–50 px, without modifying the camera intrinsics, distortion parameters, feature matching settings, or the *PNP* configuration. Other factors likely contribute to the observed variability, but this shows that reduced image quality affected the accuracy of the detected 2D keypoints and the reprojection error.

Dust contamination is a relevant issue for lunar surface applications, but detailed treatment of dust effects and mitigation strategies is outside the scope of this work. The experiments show that the performance of the feature-based pose estimation pipeline depends strongly on image quality. Possible improvements to the *PNP* stage are discussed in Section 10.3.3.

The pose estimated by the *PNP* stage represents the transformation between the RealSense depth camera frame and the monocular end-effector camera frame. To obtain a Cartesian target for motion planning, this estimated transformation is composed with the static and kinematic transformations defined in the system's *TF* tree. Through this composition, the object position is expressed in the robotic arm base frame and used as the target for the subsequent approach motion. Residual errors from this transformation chain and from the pose estimation stage itself are then compensated by the image-based visual refinement described in the next section.

9.3. Visual Refinement Evaluation

This section evaluates the image-based centering refinement described in Section 7.4. The goal is not to outperform the *PNP* pose estimation, but to quantify how reliably the

fallback centering mechanism reduces residual image error before the final descend.

A total of $N = 5$ complete autonomous grasp runs were evaluated. All runs resulted in successful grasps. Four runs were executed at a hover height of 100 mm, and one additional run was executed at 80 mm to validate the height-dependent calibration table.

9.3.1. Per-Run Summary

Table 9.9 and Table 9.10 summarize the centering performance for all evaluated runs. The first table reports the hover height and grid search behavior, while the second table presents the centering convergence results.

The final pixel error norm $\|e\|$ is computed as:

$$\|e\| = \sqrt{du^2 + dv^2} \tag{9.1}$$

This norm is included only as a compact measure of the remaining image offset at convergence. The stopping condition, however, is evaluated per axis.

Table 9.9: Hover height and grid search behavior for all successful runs.

Run	Height [mm]	Grid Attempts
1	100	1
2	100	3
3	100	1
4	100	2
5	80	1

Table 9.10: Centering convergence results. The reported final errors correspond to the last iteration before the stopping condition was satisfied.

Run	Iterations	Final du [px]	Final dv [px]	$\|e\|$ [px]
1	10	-26.5	-24.0	35.75
2	11	13.5	11.0	17.41
3	2	18.5	5.5	19.30
4	28	13.0	17.5	21.80
5	13	-25.0	-17.0	30.23

Table 9.9 shows that the object was re-acquired within at most three grid positions in all runs. Most cases required only a single grid attempt, indicating that the residual *PNP* positioning error remained within the defined search window.

Table 9.10 shows that the number of centering iterations varies considerably between

runs. Run 3 converged after only two iterations, while Run 4 required 28 iterations. Differences in the initial image offset mainly cause this variation, since the control gains and stopping criteria remained constant across all experiments. The gains received no further tuning, as this image-based correction serves as a proof-of-concept to enable validation of the complete grasping pipeline rather than as a fully optimized control strategy, and additional tuning was outside the available time scope.

The final pixel errors remain below the per-axis threshold of 35 px in all cases. Although the error norm in Run 1 reaches 35.75 px, both $|du|$ and $|dv|$ individually satisfy the convergence condition. Overall, the results indicate that the implemented correction step consistently reduces the residual image error to within the defined tolerance before the final descend.

9.3.2. Object Re-acquisition Grid Search

After the *PNP* motion, the object is not always immediately visible in the end-effector camera. A grid search at the fixed hover height is therefore executed before centering.

The number of grid search attempts per run was: [1, 3, 1, 2, 1]. Table 9.11 summarizes the statistics.

Table 9.11: Grid search statistics for object re-acquisition.

Mean attempts	1.6
Standard deviation	0.89
Median	1
Range	1–3
Success rate	100%

In all evaluated runs, the object was successfully re-detected within at most three grid positions. In three out of five runs, the object was found at the first grid position. The median value of one attempt therefore reflects that, in most cases, the *PNP* estimate already places the end-effector sufficiently close for immediate re-detection. The maximum of three attempts indicates that the residual positioning error remained limited and consistently within the predefined 3×3 search window.

The grid search performance indicates that the *PNP* stage typically places the gripper within the immediate field of view of the end-effector camera, despite object center reprojection errors of 30–50 px reported in Section 9.2.2. This is consistent with the *PNP* analysis in Section 9.2.5, where the object center error reflects spatial extrapolation from features located on the surrounding rocks rather than fundamental pose estimation failure. The 20 mm grid step size proved sufficient to recover from worst-case *PNP* positioning errors within three search positions.

9.3.3. Centering Convergence

The number of centering iterations per run was: [10, 11, 2, 28, 13]. The aggregated statistics are shown in Table 9.12.

Table 9.12: Convergence statistics for the visual centering loop across all successful runs ($N = 5$).

Mean iterations	12.8
Standard deviation	9.47
Median	11
Minimum	2
Maximum	28
Cases >10 iterations	3

The iteration count depends strongly on the initial image offset. For example, Run 3 required only two iterations, while Run 4 required 28 iterations before convergence.

Figure 9.10 shows the convergence trajectory of Run 4, representing the longest successful case. The pixel error norm decreases progressively over the iterations until the stopping condition is met.

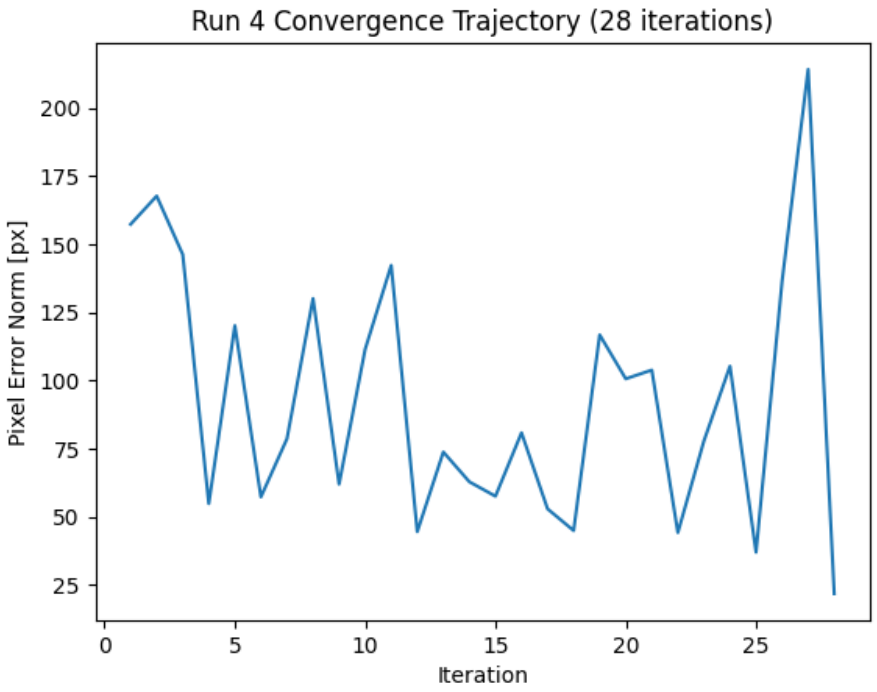


Figure 9.10: Convergence trajectory for Run 4 (28 iterations). The plotted value corresponds to the pixel error norm $\|e\| = \sqrt{du^2 + dv^2}$ at each iteration. Convergence occurs when both $|du|$ and $|dv|$ fall below 35 px.

The error oscillates during convergence rather than decreasing steadily. This behavior occurs because the fixed control gains (k_x=0.2, k_y=0.08) do not adapt to the magnitude of the pixel error. When the error is large, the correction step may overshoot the target, causing the gripper to oscillate between positions on opposite sides of the target pixel. The lower gain in the y-axis reflects the camera's downward tilt, which causes pixel positions to be more sensitive to lateral motion than to forward/backward motion. Despite this oscillation, the error envelope gradually decreases as the gripper approaches the target, eventually satisfying the per-axis stopping condition.

In contrast, Figure 9.11 shows Run 3, where the initial offset was much smaller and the system achieved convergence after only two iterations.

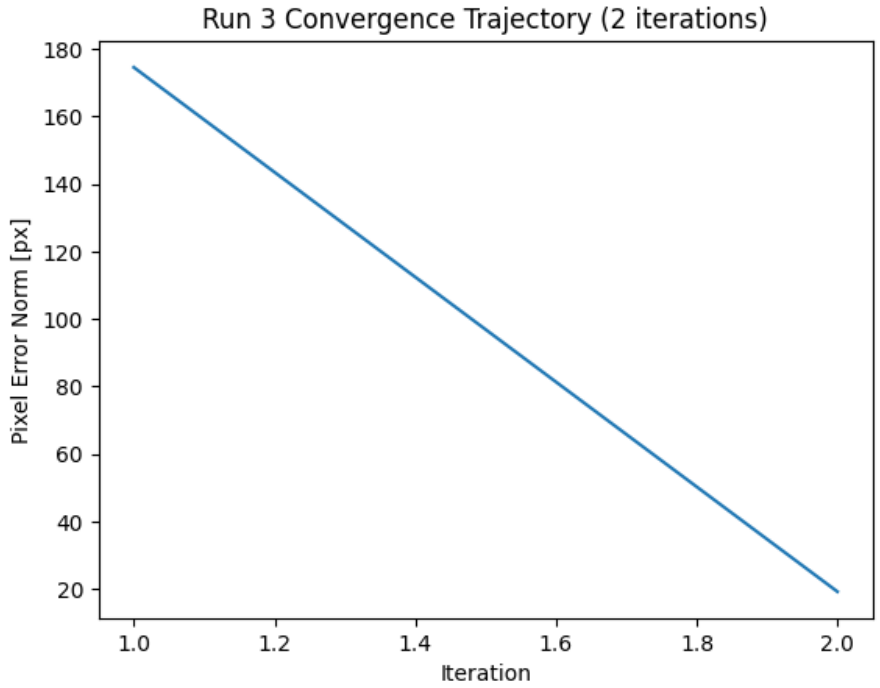


Figure 9.11: Convergence trajectory for Run 3 (2 iterations). Due to a smaller initial image error, only one corrective step was required before satisfying the convergence condition.

Run 3 demonstrates ideal convergence behavior when the initial offset is small enough that the fixed gains produce appropriate correction steps without overshoot. The single large correction step (174.45 px → 19.30 px) places the gripper nearly centered, requiring only minor final adjustment.

No runs exhibited oscillatory divergence. Even in the longest case, the corrections eventually reduced the image error below the defined threshold. The fact that all

runs converged despite oscillation demonstrates robustness of the per-axis stopping criterion, which prevents premature termination that could occur with a pure norm-based threshold.

9.3.4. Pixel Error at Convergence

The final pixel error norms at convergence were: [35.75, 17.41, 19.30, 21.80, 30.23] px. Table 9.13 summarizes the statistics.

Table 9.13: Final pixel error statistics at convergence ($N = 5$).

Mean $\|e\|$	24.90 px
Standard deviation	7.80 px
Median	21.80 px
Minimum	17.41 px
Maximum	35.75 px

All runs satisfied the per-axis stopping condition of 35 px. The final error norm therefore remains bounded by this threshold, although the norm itself may exceed 35 px if both axes are close to the limit.

9.3.5. Analysis

Across all five evaluated runs, the image-based centering successfully reduced the residual image error to within the defined tolerance and enabled successful grasp execution. The wide variation in iteration counts (2–28 iterations) demonstrates that fixed control gains cannot adapt to varying initial conditions, but the system proved robust enough to converge in all cases.

The grid search performance (median 1 attempt, maximum 3 attempts) validates the *PNP* positioning quality. Despite object center reprojection errors of 30–50 px (Section 9.2.2), the *PNP* stage places the gripper close enough that immediate object re-detection succeeds in the majority of cases. This supports the analysis in Section 9.2.5: the *PNP* object center error reflects spatial extrapolation from features on the surrounding rocks rather than fundamental estimation failure. The visual centering stage compensates for this residual positioning error through iterative refinement.

The 100% grasp success rate after visual refinement (Section 9.4) indicates that the final positioning accuracy achieved by centering is sufficient for the gripper geometry and object size. The 35 px per-axis threshold corresponds to approximately 10–15 mm positioning error at 100 mm height, which is within the gripper's capture range for the $3 \times 3 \times 3$ *cm* cube. This demonstrates that precise object center alignment is not required—only sufficient proximity for the gripper fingers to contact the object during descent.

The centering approach uses only the 2D bounding box center from *YOLO* detection rather than feature-based visual servoing or depth-based positioning. This choice trades precision for simplicity and robustness. Bounding box detection is more stable under varying lighting and viewpoint than individual feature tracking, and avoids the depth measurement noise observed in the *PNP* stage. The fixed-height operation eliminates the need for online Jacobian computation, at the cost of limiting final approach accuracy.

The additional run at 80 mm demonstrates that the calibration table interpolation remains consistent at a different hover height. The system automatically used the corresponding target pixel $(688, 228)$ and achieved convergence without modifying the control gains. This validates the height-dependent calibration approach and suggests that the method could be extended to variable-height operation with online target pixel computation.

Overall, the visual centering mechanism provides a practical and repeatable refinement step that compensates for residual positioning errors after *PNP*. This image-based correction served as a proof-of-concept to enable validation of the complete grasping pipeline rather than as a fully optimized control strategy. While not formulated as a formal *IBVS* controller with proper Jacobian computation, the implemented approach proved sufficient to reliably enable grasp execution in the evaluated scenarios. The observed oscillation in Run 4 and the wide variation in iteration counts indicate that adaptive gain scheduling and velocity-level control (discussed in Section 10.3.4) would improve convergence behavior, particularly for large initial errors.

9.4. Grasp Detection Evaluation

This section evaluates the current-based grasp detection method presented in Section 8.3. The evaluation examines how arm configuration affects baseline current calibration, the observed motor current behavior during grasp execution, and the resulting detection outcomes for successful and empty grasp attempts.

Two representative arm configurations with respect to the `boogie_base` frame were used throughout the experiments: an *easy position* with target coordinates approximately $[160, 0, 5]$ and a *hard position* with target coordinates approximately $[260, 0, 10]$. These positions simulate common observed positions throughout the tests of the pipeline. The hard position is a near-reachability configuration in which the arm is more extended and static joint loads are higher, particularly at the first and second joints.

9.4.1. Experimental Protocol

A total of $N = 5$ grasp attempts were executed using an orange cube as target object. Each attempt used the same procedure:

1. The robotic arm was moved to a static target pose corresponding to either the easy

or hard Cartesian configuration.

2. The gripper was commanded to the predefined `fully_open` state.

3. The autonomous grasp routine was executed, including baseline current calibration (Section 8.3.3) followed by incremental gripper closure with continuous delta-current monitoring (Section 8.3.4).

Testing considered the failure mode *contact without grasp*, but it did not occur in practice. The observed outcomes therefore included only *grasped* and *not grasped*.

A grasp attempt succeeded if the system reported `Grasp successful` and the object was mechanically constrained at the end of the closing sequence.

9.4.2. Baseline Current Calibration

The baseline current I_{base} is measured before the gripper starts closing and represents the static electrical load of the robotic arm for a given configuration. Table 9.14 summarizes the measured baseline currents and the corresponding dynamically computed delta-current thresholds for the evaluated grasp attempts. ΔI_{lower} is the contact detection threshold; ΔI_{upper} is the upper bound of the stable current range. Higher baseline currents in the hard configuration result in proportionally increased thresholds via the hybrid max (fixed, proportional) formula (Equations 8.2 and 8.3).

Table 9.14: Baseline current and computed delta-current detection thresholds for easy and hard arm configurations.

Group (N=5)	I_{base} [A]	ΔI_{lower} [A]	ΔI_{upper} [A]
Easy (target $[160, 0, 5]$), $N = 3$	0.71 ± 0.08	0.40 ± 0.00	0.80 ± 0.00
Hard (target $[260, 0, 10]$), $N = 2$	1.45 ± 0.20	0.73 ± 0.10	1.45 ± 0.20
All attempts, $N = 5$	1.00 ± 0.39	0.53 ± 0.19	1.06 ± 0.39

9.4.3. Delta-Current Detection Performance

Table 9.15 shows per-attempt details of the evaluated grasp trials, including arm configuration, baseline current, computed delta-current thresholds, and final detection outcome. Outcomes are classified as grasped or not grasped. “Not grasped” indicates empty grasp attempts where the gripper reached its experimentally defined closing limit without detecting a valid grasp condition. An empty grasp means the object was intentionally not placed within the gripper.

Table 9.15: Summary of grasp detection trials.

Test	Config.	Target [mm]	I_{base} [A]	$(\Delta I_{\text{lower}}, \Delta I_{\text{upper}})$ [A]	Outcome
1	Easy	[160, 0, 5]	0.67	(0.40, 0.80)	Grasped
2	Hard	[260, 0, 10]	1.31	(0.66, 1.31)	Grasped
3	Easy	[160, 0, 5]	0.80	(0.40, 0.80)	Not grasped (empty)
4	Hard	[260, 0, 10]	1.59	(0.80, 1.59)	Not grasped (empty)
5	Easy	[160, 0, 5]	0.65	(0.40, 0.80)	Not grasped (empty)

9.4.4. Grasp Detection Accuracy

Table 9.16 summarizes detection performance using a binary confusion matrix, distinguishing between grasped and not grasped outcomes. "Held" means the object was mechanically constrained at the end of the grasp routine, while "Not Held" indicates empty grasp termination.

Table 9.16: Confusion matrix for grasp detection across the evaluated trials ($N = 5$).

Detected	Actual	
	Held	Not Held
Grasp Detected	2	0
No Grasp Detected	0	3

9.4.5. Failure Mode Classification

Within the evaluated dataset, all unsuccessful grasp attempts were empty grasps. Testing did not observe the failure mode *contact without grasp*. No system-level errors occurred.

Table 9.17: Observed failure mode distribution for unsuccessful grasp attempts ($N = 3$).

Failure Mode	Count	Percentage
Empty Grasp	3	100%
Contact Without Grasp	0	0%
System Error	0	0%

9.4.6. Analysis

The results demonstrate that motor current feedback, despite measuring total arm current rather than isolated gripper current, provides sufficient information for reliable grasp detection when the arm remains stationary during closure. The 100% detection accuracy (5 out of 5 attempts correctly classified) validates the delta-current approach across both easy and hard arm configurations.

Figure 9.12 shows the complete detection sequence from baseline calibration through grasp confirmation for a successful grasp in the hard arm configuration.

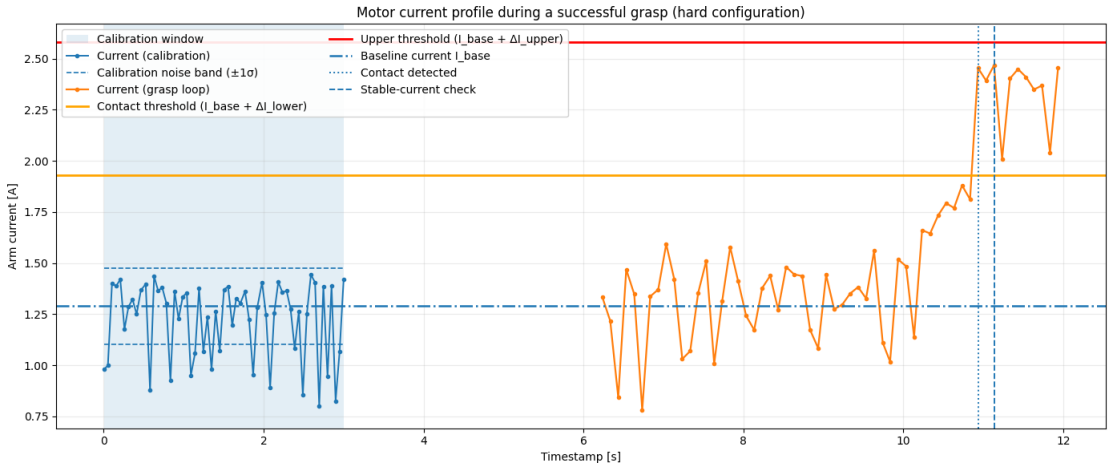


Figure 9.12: Current profile during a successful grasp (hard configuration). Light blue region (0–3 s): baseline calibration with blue trace and noise band (dashed). seeking region (5–11 s): grasp execution with orange trace oscillating during seeking, then rising at contact detection (blue vertical dotted line). Baseline I_{base} (teal dash-dot), lower threshold (yellow), and upper threshold (red) shown constant.

The calibration phase (0–3 s) shows measurements fluctuating around $I_{\text{base}} = 1.31$ A. The gap between 3–5 s reflects threshold computation, during which gripper current is not logged. The computed thresholds are $\Delta I_{\text{lower}} = 0.66$ A (contact threshold at 1.97 A) and $\Delta I_{\text{upper}} = 1.31$ A (upper bound at 2.62 A), matching Table 9.15 (Test 2).

During grasp execution (5–11 s), the current initially oscillates between 0.75–1.60 A as the gripper closes without object contact. Each 2° step creates brief current spikes. The current remains below the contact threshold during seeking, indicating no sustained mechanical resistance.

At approximately 11 s, the gripper fingers make contact with the cube. The current increases sharply and consistently exceeds the lower threshold ($I_{\text{base}} + \Delta I_{\text{lower}}$), triggering contact detection after 3 consecutive samples above threshold (marked by the dotted vertical line). This sudden increase from baseline-level oscillations to sustained elevated current provides a clear contact signature despite the noisy measurements.

Following contact detection, the system monitors current stability over the subsequent gripper closing range. The region between 11 and 12 seconds shows the stable-current check phase, where the current fluctuates between approximately 2.0–2.5 A but remains within the required ±0.15 A stability band over 15° of closing. This stable elevated current indicates that the object has become mechanically constrained—further closing

compresses the object slightly but does not significantly increase the required torque. The stability criterion distinguishes true mechanical constraint from brief current spikes that might occur during gripper motion without object contact.

The baseline current exhibits strong configuration dependence, with the hard configuration (mean $I_{\text{base}} = 1.45$ A) requiring 104% more current than the easy configuration (mean $I_{\text{base}} = 0.71$ A) to maintain the same static pose. This substantial difference reflects the increased static joint loading when the arm operates near its reachability limits. A fixed absolute current threshold would either produce false contact detection in easy configurations (where baseline already approaches the threshold) or fail to detect contact in hard configurations (where the detection margin becomes too small). The pre-grasp calibration step addresses this by establishing a configuration-specific baseline, which allows robust detection across the tested workspace.

The hybrid threshold formulas (Equations 8.2 and 8.3) maintain appropriate detection margins across configurations. In the easy configuration, the lower threshold ($\Delta I_{\text{lower}} = 0.40$ A) provides a margin of approximately 56% above the mean baseline current (0.71 A). In the hard configuration, the proportionally scaled threshold ($\Delta I_{\text{lower}} = 0.73$ A) maintains a similar relative margin of 50% above the mean baseline (1.45 A). This proportional scaling ensures that the detection criterion adapts to the configuration-dependent baseline while maintaining sufficient margin to avoid false detection from baseline current fluctuations. Figure 9.12 shows these fluctuations during the seeking phase (3–11 s), where current oscillates by approximately ±0.4 A around baseline due to incremental gripper motion. The contact threshold remains safely above this noise floor throughout execution.

Empty grasp attempts showed no stable current region—the current either remained consistently below the lower threshold or showed inconsistent fluctuations that did not satisfy the stability criterion over the required angular range. The discrete sampling visible in Figure 9.12 (individual data points at each 2° gripper step) illustrates why the consecutive-sample requirement (3 samples above threshold) provides robustness against single-point noise spikes during the seeking phase.

The absence of *contact without grasp* failures in the evaluated dataset likely reflects the specific object geometry and gripper design. This failure mode would occur if the gripper contacted the object sufficiently to trigger current increase but failed to achieve mechanical constraint before reaching the angular limit—for example, with very small objects that slip between finger gaps, or with objects whose geometry prevents stable grasping despite contact. For the tested cube, the gripper finger spacing and closing range proved sufficient to consistently achieve either full mechanical constraint (successful grasp) or no contact at all (empty grasp), with no intermediate cases.

Vision-based grasp confirmation was looked into as an alternative approach, since a

camera is mounted on the end-effector. However, during gripper closure the object moves into a very close-range configuration, where blur and limited field of view reduce detection reliability, as discussed in Section 9.1.2. Training the object detection model primarily used images captured at operational distances (70–100 mm) and did not include sufficient near-contact examples. During experimental testing, detection confidence dropped significantly during closure, particularly when gripper fingers partially blocked the object or when the object moved closer than the camera's minimum focus distance. Therefore, this work selected motor current sensing as the primary grasp confirmation method, as it remains effective regardless of visual occlusions or focus limitations.

The method does not measure grasping force directly, and therefore cannot detect slip after grasp confirmation or adjust grip strength based on object properties. The post-grasp tightening step (additional 4° closure after detection) provides a fixed increase in constraint force, but empirical testing determined this value for the tested object and it may not generalize to objects of different sizes, weights, or compliance. Despite these limitations, motor current feedback using only the existing onboard sensor proved sufficient for reliable grasp confirmation without requiring additional hardware or manual parameter retuning across the tested arm configurations. Further improvements for grasp force control and slip detection are discussed in Section 10.3.6.

9.5. End-to-End Pipeline Performance

This section evaluates the complete autonomous grasping pipeline from initial object detection to final placement of the object in the collection basket. The analysis is based on the nine full autonomous runs described in Section 9.2. Each run includes RealSense object detection, *PNP* pose estimation, motion planning, visual centering, grasp execution, and object placement.

The subsystem evaluations presented earlier (12 detection experiments and five grasp-detection tests) were conducted independently to characterize individual components. In contrast, the present section analyzes only the nine integrated autonomous executions. In all nine integrated runs, RealSense object detection was successful and therefore does not limit progression to subsequent stages. The five successful end-to-end runs (R1–R5) include successful *PNP* convergence, motion planning, visual centering, grasp execution, and object placement.

9.5.1. Overall Success Rate

Table 9.18 summarizes the outcome distribution across all evaluated autonomous runs.

Table 9.18: End-to-end pipeline outcomes across the nine evaluated autonomous runs ($N = 9$).

Outcome	**Count**	**Percentage**
Successful grasp + object placed in basket	5	55.6%
Failures:		
PNP did not converge (NC1, NC2)	2	22.2%
PNP converged, motion planning failed (R6)	1	11.1%
PNP converged, object lost after *PNP* (R7)	1	11.1%
Total	**9**	**100%**

Five of the nine runs completed the full task without intervention, corresponding to a 55.6% end-to-end success rate. In these runs (R1–R5), the system detected the object, estimated its pose, positioned the gripper, performed visual centering, grasped the object, and placed it in the basket.

Two runs (NC1 and NC2) were terminated during the *PNP* stage after repeated retries without meeting the 50 px object center reprojection error threshold. One run (R6) reached a valid pose estimate but failed during motion planning. One additional run (R7) converged in *PNP* but lost the object before successful centering.

9.5.2. Pipeline Stage Success Breakdown

Table 9.19 reports the success rate at each major pipeline stage. Each stage is conditional on the previous stage succeeding.

Table 9.19: Stage-wise success rates for the nine autonomous runs. Each stage is conditional on the previous stage succeeding.

Pipeline Stage	**Attempted**	**Succeeded**	**Success Rate**
RealSense object detection	9	9	100%
PNP pose estimation convergence	9	7	77.8%
Motion planning (collision-free path found)	7	6	85.7%
Visual centering convergence	6	5	83.3%
Grasp execution	5	5	100%
Object placement in basket	5	5	100%
End-to-end (all stages)	**9**	**5**	**55.6%**

RealSense object detection succeeded in all nine integrated runs. This is consistent with the subsystem evaluation in Section 9.1, where the orange cube showed robust detection behavior under laboratory conditions.

PNP pose estimation converged in seven of nine runs (77.8%). The two non-convergent runs correspond to NC1 and NC2, as analyzed in Section 9.2. In both cases, the object

center reprojection error remained above the acceptance threshold despite repeated retries.

Of the seven runs with successful *PNP* convergence, motion planning succeeded in six cases (85.7%). The remaining case (R6) failed due to workspace constraints, even though a valid pose estimate was available.

Six runs reached the visual centering stage. Five of these converged successfully (83.3%). In run R7, the object was not successfully maintained in view after *PNP* motion and the run did not proceed to a completed centering sequence.

All five runs that reached grasp execution successfully grasped the object and placed it in the basket. Within this dataset, no failures occurred after successful centering. After successful grasp confirmation, the arm is commanded to the predefined `safe_transition` position and then to the `home` configuration. Upon reaching the home pose, the gripper is opened, releasing the object into the collection basket. This sequence defines the object placement stage reported in Table 9.19.

9.5.3. Analysis

The overall success rate of 55.6% is mainly limited by *PNP* convergence reliability. Table 9.19 shows that only seven of nine runs progressed beyond the pose estimation stage. The two non-convergent runs therefore account for a significant portion of the total failures.

However, downstream stages also contributed to performance reduction. Among the seven runs with successful *PNP* convergence, two did not complete the full task (R6 and R7). Conditional on *PNP* convergence, the end-to-end success rate is therefore $5/7 = 71.4\%$.

The results indicate that pose estimation remains the dominant bottleneck in the integrated system. Once a valid pose estimate is available and the target lies within the reachable workspace, the subsequent stages—centering, grasp execution, and placement—operated reliably in the evaluated runs.

The experiments demonstrate that the integrated pipeline can autonomously detect, localize, approach, and retrieve objects under laboratory conditions when *PNP* convergence is achieved and motion planning succeeds. Improvements in pose estimation robustness and viewpoint selection during retries would directly increase the overall task completion rate.

9.6. Summary

This chapter evaluated the autonomous grasping pipeline at the *DLR* Planetary Exploration Lab, covering both subsystem performance and integrated end-to-end operation.

Object Detection (Section 9.1) was tested across 12 RealSense configurations (three cube colors under four lighting and position conditions). The orange cube achieved 100% detection success (4/4), while the green and blue cubes each failed once under centered, lights-on conditions due to specular reflections. Monocular detection for centering succeeded at heights of 70–100 mm and failed below 70 mm due to focus limitations.

***PNP* Pose Estimation** (Section 9.2) converged in 7 of 9 integrated runs (77.8%) under the 50 px object center reprojection error threshold. Accepted errors ranged from 30.96–48.93 px (mean 43.04 px). Two runs that converged at the *PNP* stage did not complete the task: one due to motion planning constraints and one due to object loss after *PNP* motion. *PNP* convergence was the primary bottleneck.

Visual Refinement (Section 9.3) converged in all 5 runs that reached the centering stage. Grid search re-acquisition required 1–3 attempts (median 1). Centering iterations ranged from 2 to 28 (median 11). All runs satisfied the 35 px per-axis stopping criterion.

Grasp Detection (Section 9.4) correctly classified all 5 independent grasp tests. Baseline currents varied between 0.71 A (easy configuration) and 1.45 A (hard configuration). The hybrid threshold formulation accommodated this variation without retuning.

End-to-End Performance (Section 9.5) achieved 55.6% task completion (5/9 runs). Conditional on *PNP* convergence, the completion rate was $5/7 = 71.4\%$. Failures were dominated by pose estimation non-convergence, with additional losses due to workspace constraints and object loss after motion.

Overall, the system can autonomously detect, localize, and grasp objects under laboratory conditions when *PNP* convergence is achieved. Improving pose estimation robustness would directly increase overall task success (Section 10.3.3).

10. Conclusion and Future Work

10.1. Summary of Contributions

This thesis developed and experimentally validated an autonomous grasping pipeline for the *LRM* using vision-based localization and current-based grasp detection. The system performs object detection, pose estimation, visual refinement, and grasp execution without requiring force sensors or tactile feedback.

The main contributions are:

Integration of end-effector camera and coordinate transformation chain. The monocular camera was integrated into the robotic arm end-effector with complete kinematic modeling through the *ROS TF* tree (Chapter 5). Camera calibration and extrinsic parameter estimation enabled consistent transformation of detected features between camera frames and the robotic arm workspace. The coordinate architecture handles transformations from the RealSense camera at the rover front, through the 180° rotation, to the end-effector camera mounted on the rear arm.

Autonomous state machine and integrated pipeline. The complete grasping pipeline was implemented using *RAFCON* state machines with modular Python components (Chapter 5). The state machine coordinates perception, motion planning, visual refinement, and grasp execution, including retry logic for *PNP* failures and handling of common error conditions. End-to-end evaluation achieved 55.6% task completion (5/9 runs) from initial detection through object placement, demonstrating autonomous execution of the integrated system.

Feature-based *PNP* pose estimation for object localization. A vision-based pose estimation pipeline was implemented using *SIFT* feature extraction, ratio test filtering, homography-based *RANSAC* outlier rejection, and *PNP* solving with Levenberg–Marquardt refinement (Chapter 7). Depth information from the RealSense camera was used to construct 3D–2D correspondences with the monocular end-effector camera. An object center reprojection error metric was introduced to validate pose estimates before arm motion. Experimental evaluation showed convergence in 77.8% of integrated runs (7/9) under a 50 px acceptance threshold. The larger threshold compared to feature-level reprojection errors (2–3 px) accounts for extrapolation uncertainty when the object center lies outside the spatial distribution of detected features.

Image-based visual refinement for grasp positioning. A centering mechanism was implemented to compensate for residual positioning errors after *PNP* motion (Chapter 7). The method uses calibrated target pixels at a fixed operating height (100 mm) and

iteratively adjusts gripper position based on pixel error feedback. Experimental results showed 100% convergence in all runs that reached the centering stage (5/5), confirming that the method provides sufficient positioning accuracy for grasp execution.

Current-based grasp detection without force sensors. A grasp detection method was implemented using motor current feedback from the existing servo interface (Chapter 8). The method includes pre-grasp baseline calibration to compensate for configuration-dependent static loads, hybrid threshold computation (maximum of fixed and proportional values), and stable current detection during gripper closure. Experimental validation showed 100% accuracy in distinguishing successful grasps from empty attempts across different arm configurations.

The experimental evaluation (Chapter 9) identified *PNP* convergence as the primary bottleneck of the integrated system. Once pose estimation converged and the object remained within the reachable workspace and field of view, downstream stages—visual refinement, grasp execution, and placement—completed without failure in the evaluated runs.

The results indicate that autonomous manipulation on resource-constrained platforms is feasible using low-cost components. The monocular camera (approximately €50), the RealSense depth camera (approximately €200), and servo motors without position encoders were sufficient to achieve reliable visual refinement and grasp detection once pose estimation converged. This supports the use of vision-based compensation strategies for manipulation in planetary exploration scenarios where sensing and hardware resources are limited.

10.2. Limitations

Although the implemented system demonstrates autonomous grasping using low-cost hardware and without servo position feedback, the current implementation has several limitations that restrict its operating envelope. These limitations result mainly from the chosen algorithmic design and the limited range of experimental scenarios considered in this work.

Single-object grasping scenarios. The pipeline processes one target object at a time. Multi-object environments would require additional logic for object prioritization, occlusion handling, and collision avoidance with non-target objects.

Limited object diversity. The *YOLO*v7 detector was trained on 3D-printed cubes under controlled lighting conditions. Detection robustness decreases for glossy surfaces due to specular reflections and fails when images are blurred below the camera's minimum focus distance. Deployment in realistic planetary environments would require training

and validation on a broader set of objects, including natural rocks and irregular geological samples.

***PNP* convergence bottleneck.** The object center reprojection error criterion resulted in 77.8% convergence (7/9 runs) under a 50 px acceptance threshold. Lower thresholds reduce convergence reliability, while accepted reprojection errors (30–48.93 px) indicate residual positioning error that must be corrected by visual refinement. The threshold is significantly larger than feature-level reprojection errors (2–3 px) because *SIFT* features cluster in high-texture regions of the rock simulant rather than on the smooth cube surface. Pose estimation therefore requires spatial extrapolation from limited feature distributions to the object center. If the target object itself provided sufficient surface texture (e.g., natural rocks instead of smooth cubes), feature correspondences would likely distribute more evenly across the object surface. This could reduce extrapolation uncertainty and potentially improve *PNP* convergence reliability. However, this hypothesis was not experimentally validated within the scope of this work.

Fixed-height visual refinement. The centering procedure operates at a fixed hover height of 100 mm due to camera focus constraints and height-dependent pixel calibration. This restricts grasping to predefined approach conditions and limits flexibility in trajectory planning. The empirically tuned control gains ($k_x = 0.2$, $k_y = 0.08$) are not adaptive and may not generalize to different object geometries or imaging conditions. Fixed gains led to oscillatory convergence behavior for large initial pixel errors, with up to 28 iterations observed in evaluation.

Top-down grasp constraint. The gripper approaches the object from above with a fixed 180° orientation constraint. This reduces the effective reachable workspace and prevents side grasps or angled approaches that may be kinematically advantageous.

Indirect grasp force measurement. The current-based detection method provides only indirect information about contact and constraint. The system cannot detect slip after grasp completion or regulate grip force according to object properties such as weight or compliance.

Controlled test environment. All experiments were conducted indoors at the *DLR* Planetary Exploration Lab under controlled lighting and prepared terrain conditions. Performance under outdoor illumination, strong shadows, or highly irregular terrain has not been characterized. Dust contamination was observed to increase *PNP* reprojection error during testing, but systematic analysis of dust effects and mitigation strategies was outside the scope of this work.

These limitations define the present operational envelope of the system and motivate the improvements discussed in Section 10.3.

10.3. Future Work

10.3.1. Object Detection Improvements

The *YOLO*v7 detection model used in this work was pretrained on a limited dataset under controlled lighting conditions and sharp, in-focus images. Experimental results in Section 9.1 showed reduced performance under strong illumination from reflections on glossy surfaces and complete failure when images are blurred below the end-effector camera's minimum focus distance.

To improve detection robustness, future work could retrain the model using an augmented dataset that includes:

- Images captured under varying lighting conditions, including strong directional light, shadows, and low-light scenarios
- Blurred training images to simulate out-of-focus conditions
- Multiple surface finishes (matte, glossy, reflective)
- Diverse object classes beyond 3D-printed cubes, such as natural rocks and irregular geological samples

Another possible improvement is the introduction of temporal filtering across consecutive frames. During the visual centering loop (Section 7.4), temporary detection loss due to occlusion or lighting changes causes the grid search to restart. A Kalman filter or frame-to-frame tracker could maintain short-term object position estimates during brief detection dropouts.

10.3.2. Multi-Object Scenarios and Object Selection

The current pipeline is limited to single-object scenarios (Section 1.3). Extending the system to multi-object environments would require:

- An object selection strategy (e.g., based on proximity or task relevance)
- Occlusion handling
- Motion planning that accounts for non-target objects
- Re-planning when selected targets become unreachable

Such functionality would be necessary for autonomous sample collection tasks involving multiple candidate objects.

10.3.3. PNP Pose Estimation Improvements

The results in Section 9.2 show that the current *PNP* pipeline supports the grasping task under the selected acceptance criterion, but variability in object center reprojection error indicates room for improvement.

Isolated tests achieved reprojection errors of only a few pixels, while systematic evaluation resulted in accepted errors between 30–50 px. This suggests that achievable accuracy depends on data-dependent factors such as feature distribution, illumination, image sharpness, and depth noise. As discussed in Section 9.2.5, *SIFT* features cluster predominantly in high-texture regions rather than on the smooth cube surface, leading to spatial extrapolation when estimating the object center pose.

Future work could investigate methods to increase feature coverage on the object surface, such as structured lighting, artificial texture patterns, or active markers. A systematic parameter study may help determine under which imaging conditions lower reprojection errors can be achieved consistently.

The current acceptance logic relies on a fixed object center reprojection error threshold. Alternative acceptance criteria that incorporate additional information—such as correspondence count, *PNP* inlier ratio, or spatial feature distribution—could provide a more nuanced reliability measure.

The pose estimated by the *PNP* stage is composed with additional transformations from the system's *TF* tree to obtain the object pose in the robotic arm base frame. Residual errors from kinematic modeling, *URDF* parameters, and inverse kinematics may accumulate. Therefore, improvements at the perception level may not translate linearly into manipulation accuracy.

10.3.4. Visual Refinement Improvements

The current centering implementation uses fixed control gains ($k_x = 0.2$, $k_y = 0.08$) at a fixed height of 100 mm. Adaptive gain scheduling based on pixel error magnitude could reduce oscillatory behavior for large initial offsets while preserving convergence speed for smaller errors.

The fixed-height constraint could be relaxed through live calibration of target pixels as a function of measured gripper height. This would allow centering at different heights within the camera focus range.

A formal *IBVS* controller with Jacobian estimation could replace the empirical pixel-based correction. This would provide a more systematic control formulation and enable velocity-level control if future hardware revisions support velocity commands.

10.3.5. Motion Planning and Grasp Pose Selection

The current system restricts grasping to a top-down approach with a continuous 180° orientation constraint (Section 8.2). This simplifies planning but reduces the reachable workspace and contributed to the failure observed in run R6 (Section 9.2.4).

Future work could implement grasp pose selection based on object geometry and reachability analysis. For rectangular objects, multiple candidate grasp orientations (top, side, corner) could be evaluated, and the planner could select the most feasible configuration.

This would require:

- A grasp pose sampling or database strategy (e.g., antipodal grasp generation [45])
- Orientation-aware reachability analysis (Section 8.1)
- Path planning without fixed orientation constraints
- Visual refinement for non-vertical approaches

The current-based grasp detection method (Section 8.3) is orientation-independent and would not require modification for alternative grasp directions.

More advanced collision checking that includes geometric models of the rover body and arm links could further reduce planning failures in cluttered scenes.

10.3.6. Grasp Force Control and Tactile Feedback

The present grasp detection relies on motor current feedback to infer contact and stability. While effective, it provides only indirect force information and cannot detect slip after grasp completion.

Integration of wrist-mounted force/torque sensors or tactile sensors on the gripper fingers would enable closed-loop force control. This could allow:

- Controlled grasp force independent of object compliance
- Slip detection during manipulation
- Adaptive gripping for fragile or heavy objects
- Direct assessment of grasp quality

Such extensions would require additional hardware integration and modification of the Middleboard interface.

Appendices

A. User Interface Development

A.1. Low-Level GUI

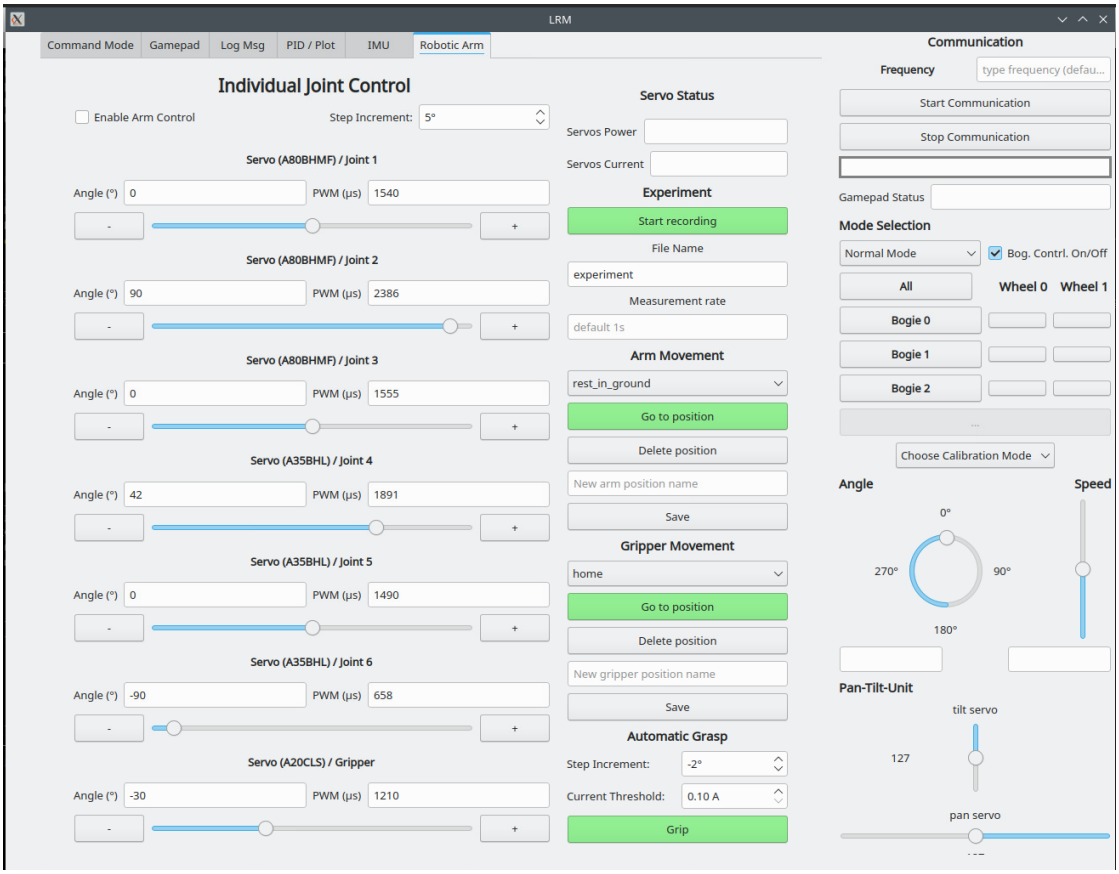


Figure A.1: Low-level GUI: Robotic arm panel.

A.1.1. Overview

The low-level *GUI* provides direct control over individual robotic arm joints, enabling precise manual positioning and testing during development. This interface was designed for debugging, calibration, and situations that require explicit joint-level control rather than multi-joint operations. An adjustable step size option enables fine or coarse positioning depending on the required precision.

A.1.2. Communication Setup

The communication panel on the right side is used to establish a connection between the *NUC* and the Middleboard. The chosen frequency for the robotic arm is 100 Hz, which was determined through empirical testing.

A.1.3. Enabling Arm Control

Arm control can be enabled through the *GUI* interface. Enabling arm control engages the servos and simultaneously disables wheel movement. This mutually exclusive operation is intentional—the system allows control of either the wheels or the arm at any given time, but not both simultaneously.

A.1.4. Joint Slider Interface

The slider section displays information for each joint, including the joint name, servo type, current angle, and the corresponding *PWM* value. Joints can be positioned using either the slider controls or the increment/decrement buttons adjacent to each slider.

An important design consideration is that the gripper and arm servos are controlled separately. This ensures that when selecting a predefined arm position, the gripper maintains its current state. This separation is reflected in the distinct arm movement and gripper movement sections of the interface.

The interface restricts transitions between predefined arm positions when communication is active. The position selection button is disabled (displayed in red) during active communication. To change between predefined positions, communication must first be stopped, then the new position can be selected. This restriction prevents direct joint transitions during active communication, since the low-level *GUI* does not perform path planning and could otherwise generate unsafe trajectories. For smooth, planned transitions between positions, the high-level *GUI* (Section A.2) should be used instead.

A.1.5. Predefined Position Management

Two dedicated sections handle predefined positions for the arm and gripper, respectively. Users can select saved positions from dropdown menus, navigate to selected positions, delete existing positions, or save new configurations. The save function captures the current slider values and stores them with a user-specified name in a YAML configuration file.

A.1.6. Servo Status Monitoring

The servo status panel displays current draw and power consumption for the servo line. The displayed current and power values represent the aggregate measurement for all seven servos, as the current sensor is positioned at the end of the servo power line.

A.2. High-Level GUI

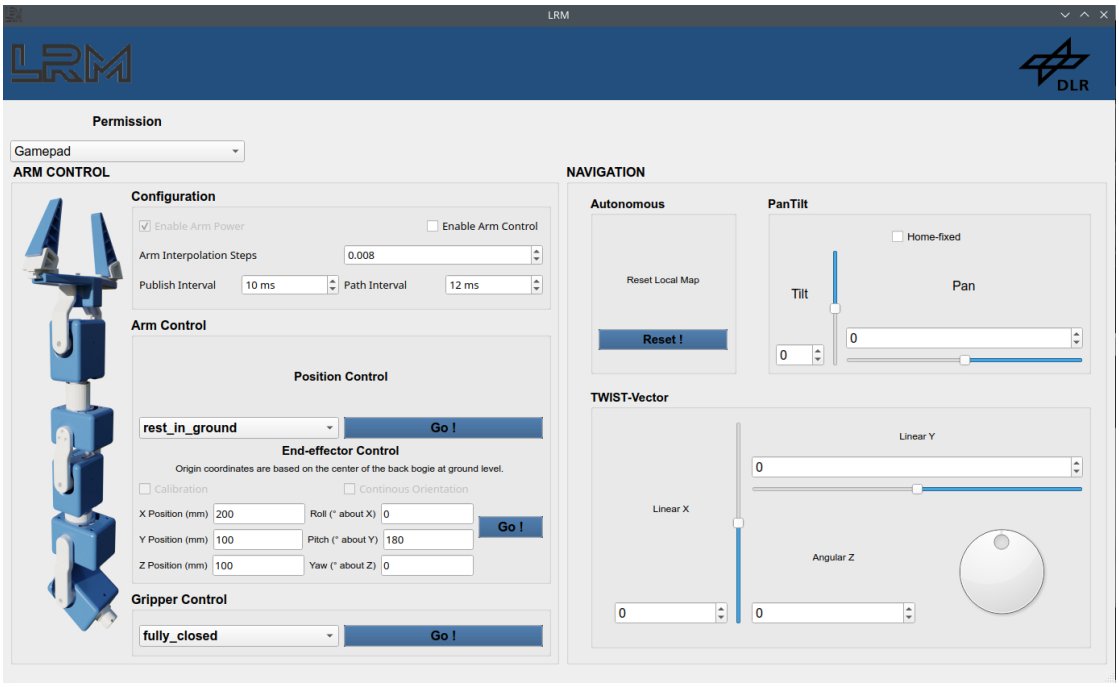


Figure A.2: High-level GUI*: Robotic arm control interface.*

A.2.1. Overview

The high-level *GUI* is designed for **manual** multi-joint robotic arm operations with integrated path planning. Unlike the low-level *GUI*, which provides direct joint control, the high-level *GUI* generates collision-free trajectories and manages system-wide permissions for different control modes. The interface is divided into three main sections: permission management, robotic arm control, and navigation control. This appendix focuses on the first two sections, as they are most relevant to the autonomous grasping implementation.

A.2.2. Permission Management

The permission section is a dropdown menu with three available control modes: *GUI*, gamepad, and autonomous. The *GUI* mode enables manual control through the high-level interface, the gamepad mode allows rover control via an external gamepad controller, and the autonomous mode is reserved for *RAFCON* state machine operation. When *RAFCON* initiates the autonomous grasping loop, it automatically sets the system to autonomous mode. During autonomous operation, manual robotic arm control through the *GUI* is disabled to prevent conflicting commands.

A.2.3. Robotic Arm Control Panel

The robotic arm control section includes several features developed to support system testing and operation. Two primary control toggles manage the arm state: enable arm control and enable power. These toggles set the status of the arm for movement and power respectively, and are linked to the `ArmControl` message type used throughout the system for state synchronization.

Path Planning Parameters

Three configurable parameters control the path planning and execution behavior:

- **Interpolation delta:** Defines the spacing between waypoints during path interpolation after optimization, controlling trajectory smoothness. Best found value after testing: 0.008
- **Publish timer:** Sets the rate at which joint states are published to *LN*. Best found value after testing: 10 ms
- **Iterate path timer:** Determines the time interval between each step during path execution, controlling movement speed. Best found value after testing: 12 ms

Arm Control Methods

The interface provides two methods for commanding arm motion:

Predefined Positions

Predefined joint configurations are loaded from the YAML configuration file and displayed in a dropdown menu. This list is read-only in the high-level *GUI*, as all position definitions and modifications must be performed through the low-level *GUI*. Users select a desired target position from the dropdown and execute the motion using the "Go" button.

Cartesian Coordinate Input

Users can specify target coordinates (x, y, z) for the end-effector in the arm's base frame (boogie base). An optional orientation constraint allows the end-effector to maintain a fixed pitch angle in its local coordinate frame. For example, setting the pitch to 180 degrees ensures the gripper always faces downward toward the ground, which is useful for grasping operations.

A.2.4. Path Planning and Execution

Both control methods (predefined positions and Cartesian coordinates) serve as inputs to the path planning module. The path planner generates trajectories while enforcing three primary constraints: joint position limits, arm workspace boundaries (bounding box), and self-collision avoidance. After generating a valid path, the planner applies optimization, then interpolates the trajectory using the specified interpolation delta to produce a smooth, continuous motion path suitable for execution on the robotic arm.

B. Calibration Data

B.1. SQ11 Camera Specifications

Table B.1: Technical specifications of the SQ11 mini DV camera

Parameter	**Specification**
Video resolution	1920×1080, 1280×720
Frame rate	15, 30 frames per second
Video file format	AVI
Photo resolution	4032×3024
Photo file format	JPEG
Viewing angle	140°
Night illumination	6 IR LEDs (range up to 5 m, non-visible light)
Motion sensor	Yes (range up to 5 m)
Recording while charging	Supported
Operating time (continuous)	Approximately 100 minutes (1 h 40 min)
Operating time (motion detection)	Approximately 5 hours
External power operation	Continuous (24/7) when externally powered
Supported memory cards	MicroSD up to 64 GB
Cyclic recording	Supported
Operating temperature	−10°C to +60°C
Operating humidity	15–85%
Dimensions and weight	23 × 23 × 23 mm, 15 g

Information taken from [37].

B.2. D435i RealSense Camera Specifications

Table B.2: Technical specifications of the Realsense D435i

Parameter	**Specification**
Depth technology	Active stereo depth sensing
Depth field of view (HxV)	Approximately 87° × 58° (wide FOV)
Depth output resolution	Up to 1280 × 720
Depth frame rate	Up to 90 fps
Minimum depth distance (Min–Z)	0.28 m
Depth accuracy	< 2% at 2 m
RGB camera resolution	1920 × 1080
RGB frame rate	30 fps
RGB sensor field of view (HxV)	69° × 42°
RGB sensor technology	Rolling shutter
Depth sensor shutter	Global shutter
IMU	Integrated 6-axis IMU (gyro + accelerometer)
Ideal operating range	0.3 m to 3 m
Maximum range	Up to 10 m (varies with lighting)
Vision processor	RealSense Vision Processor D4
Camera module	RealSense Module D430 + RGB camera
Connectivity	USB-C 3.1 Gen 1
Dimensions	90 mm × 25 mm × 25 mm
Mounting	1 × 1/4-20 UNC + 2 × M3 mount points

Information taken from [42].

B.3. Servo Specification Data

The information in tables B.3, B.4, and B.5 is available at [46] [47], and [48] respectively.

B.3.1. A80BHMF Servo Datasheet

Table B.3: Specifications of the AGFRC A80BHMF Servo

Parameter	**Value**
Operating Voltage	4.8–8.4 V
Operating Speed	0.140 s/60° @ 4.8 V; 0.120 s/60° @ 6.0 V; 0.100 s/60° @ 7.4 V; 0.085 s/60° @ 8.4 V
Stall Torque	27 kg cm @ 4.8 V; 31 kg cm @ 6.0 V; 36 kg cm @ 7.4 V; 40 kg cm @ 8.4 V
Gear Material	Steel + Titanium
Motor Type	Brushless
Dimensions	$40 \times 20 \times 37.5$ mm
Weight	79 g
Connector Wire	TYU 300 mm / 22AWG
Pulse Width Range	500–2500 μs
Operating Travel	$180° \pm 10°$
Neutral Position	1520 μs
Frequency	333 Hz

B.3.2. A35BHL Servo Datasheet

Table B.4: Specifications of the AGFRC A35BHL Servo

Parameter	**Value**
Operating Voltage	4.8–8.4 V
Operating Speed	0.125 s/60° @ 4.8 V; 0.110 s/60° @ 6.0 V; 0.090 s/60° @ 7.4 V; 0.080 s/60° @ 8.4 V
Stall Torque	10 kg cm @ 4.8 V; 12 kg cm @ 6.0 V; 15 kg cm @ 7.4 V; 16 kg cm @ 8.4 V
Gear Material	Titanium
Motor Type	Brushless
Dimensions	$35.5 \times 15 \times 29.2$ mm
Weight	45 g
Connector Wire	JR 180 mm / 22AWG
Pulse Width Range	500–2500 μs
Operating Travel	$180° \pm 10°$
Neutral Position	1520 μs
Frequency	333 Hz

B.3.3. A35BHL Servo Configuration

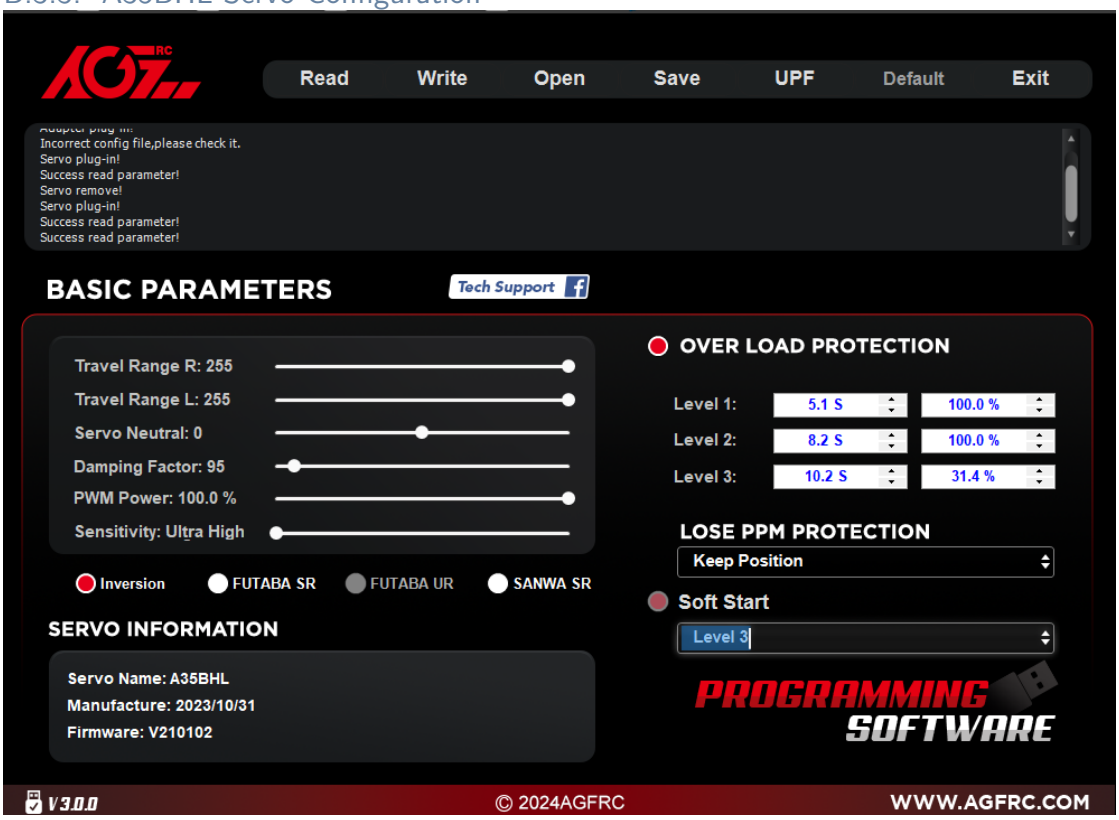


Figure B.1: Representative hardware-level servo configuration results for the A35BHL servos used in joints 4–6.

B.3.4. A20CLS Servo Datasheet

Table B.5: Specifications of the AGFRC A20CLS Servo

Parameter	**Value**
Operating Voltage	4.8–8.4 V
Operating Speed	0.110 s/60° @ 6.0 V; 0.090 s/60° @ 7.4 V; 0.080 s/60° @ 8.4 V
Stall Torque	5.8 kg cm @ 6.0 V; 6.8 kg cm @ 7.4 V; 7.5 kg cm @ 8.4 V
Gear Material	Metal
Motor Type	Coreless
Dimensions	$23 \times 12 \times 27.5$ mm
Weight	21 g
Connector Wire	TYU 190 mm / 26AWG
Pulse Width Range	500–2500 μs
Operating Travel	$180° \pm 10°$
Neutral Position	1520 μs
Frequency	333 Hz

B.3.5. A20CLS Servo Configuration

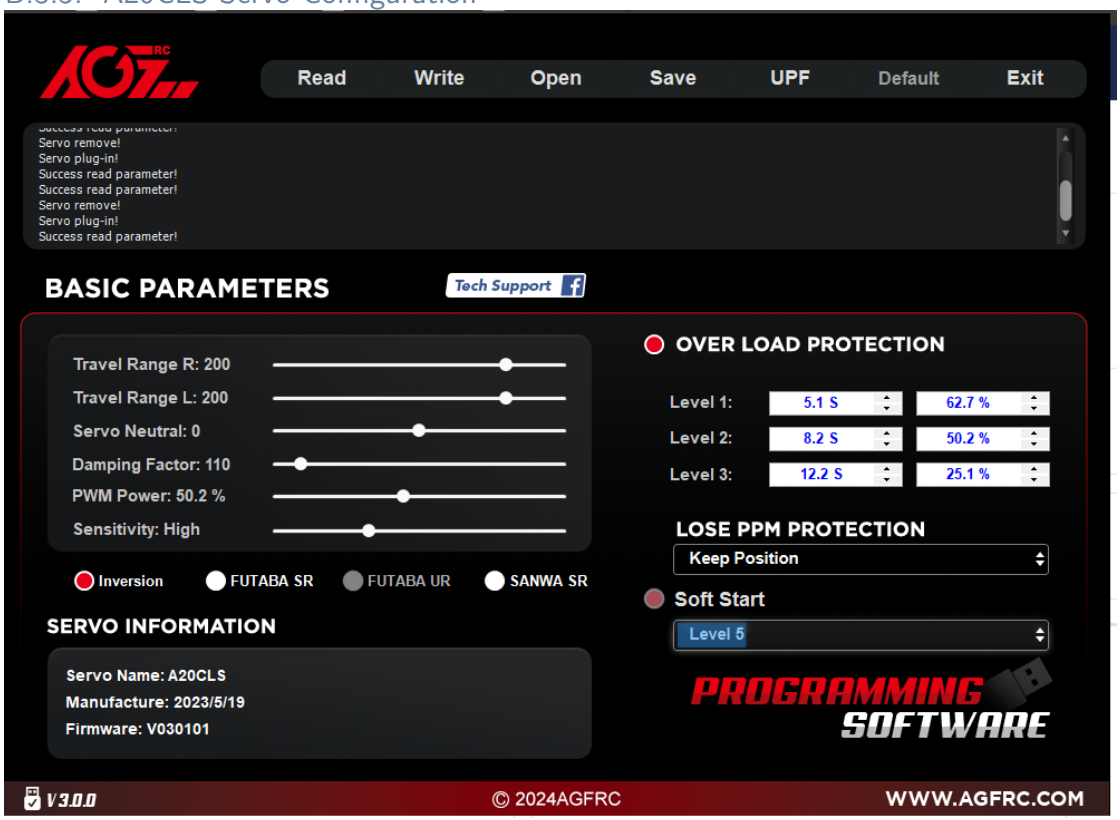


Figure B.2: Representative hardware-level servo configuration results for the A20CLS servo used in the gripper.

C. ROS Topic and Message Definitions

Listing C.1: ROS message definition for robotic arm control

```
# msg/ArmControl.msg
# Message for arm control in LRM1

std_msgs/Header header
bool enable_arm
bool arm_power_state
string arm_state
```

Listing C.2: ROS message definition for gripper state feedback

```
# msg/GripperState.msg

std_msgs/Header header
string position_name
int32 degree_value
```

The `JointState` message is used directly from *ROS* [38].

D. RAFCON State Machines and Tables

This appendix provides additional implementation details for the autonomous grasping state machine architecture described in Section 5.2.

D.1. Master State Machine

Figure D.1 shows the master state machine that coordinates the complete autonomous grasping pipeline. This state machine controls the high-level execution flow, including initialization, perception, motion planning, visual refinement, and error recovery. State transitions and failure handling logic are detailed in Section 5.2.

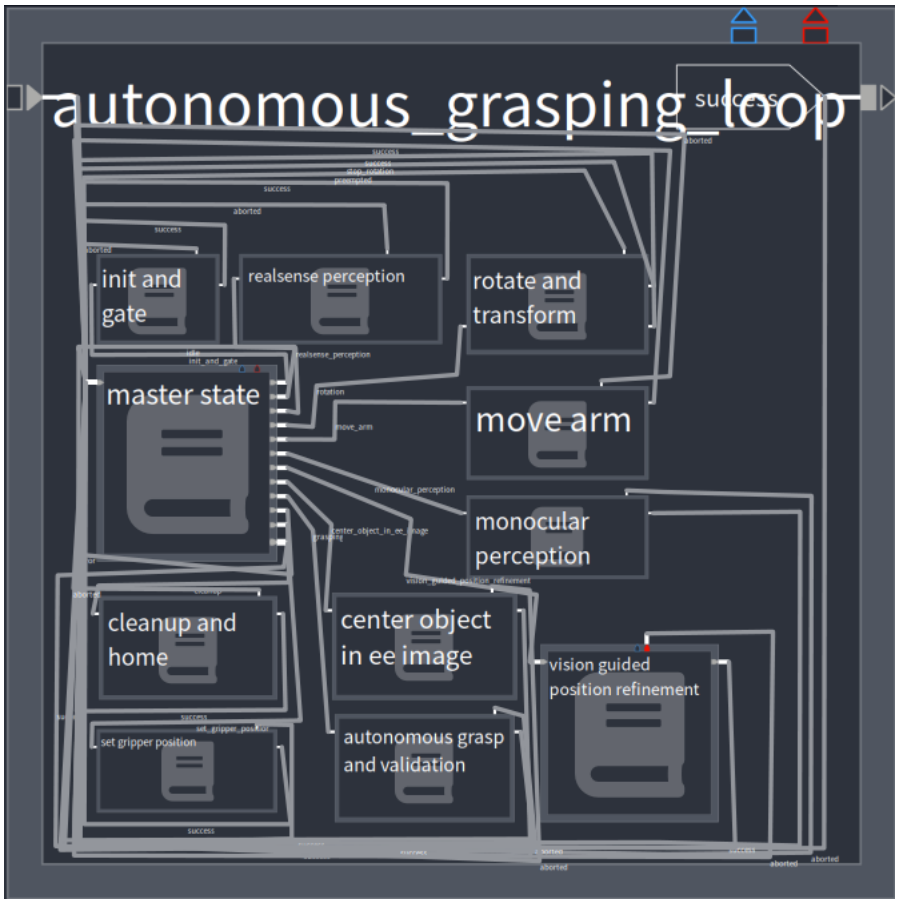


Figure D.1: Master state machine showing the complete autonomous grasping pipeline structure with initialization, perception, motion, and error recovery states.

D.2. Object Detection State Machine

Figure D.2 shows the nested object detection state machine used by both the RealSense perception and monocular perception states. This state machine executes image acquisition, YOLO-based detection, and feature extraction. The same structure is reused for both depth-based and monocular detection by parameterizing the camera source through a *RAFCON* global variable manager.

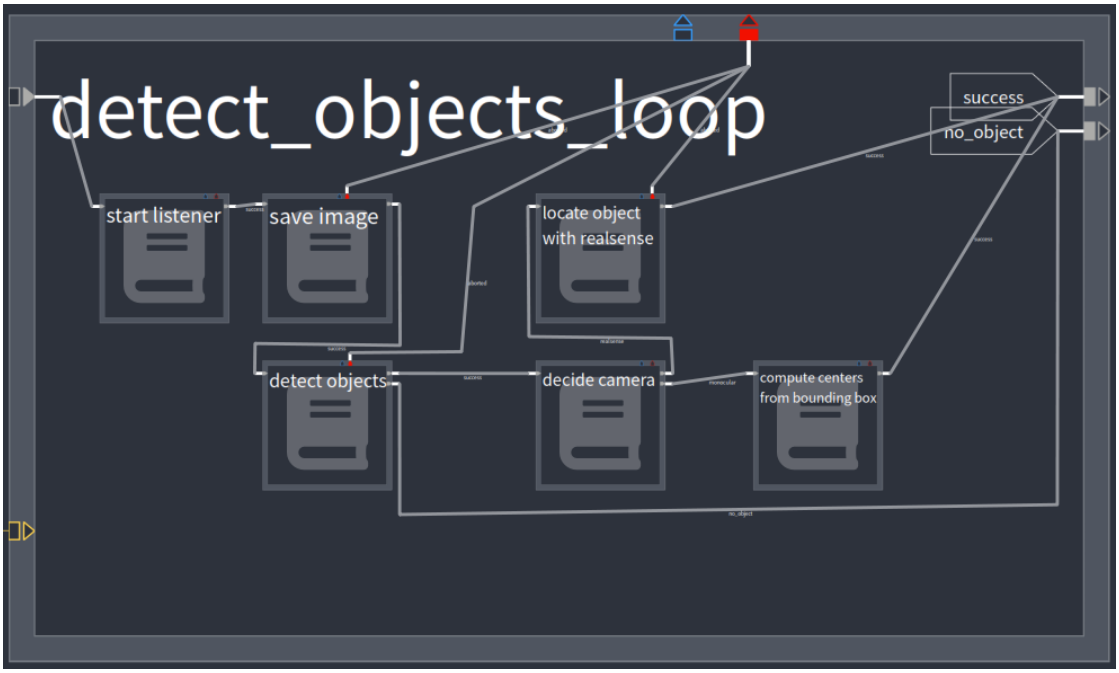


Figure D.2: Object detection state machine shared between RealSense and monocular perception modes, showing image acquisition, YOLO inference, and detection validation logic.

D.3. Vision-Guided Position Refinement State Machine

Figure D.3 shows the vision-guided position refinement state machine that implements the feature matching, *PNP*-based pose estimation and retry logic described in Section 7. This state machine captures images from multiple viewpoints, performs the matching, executes *PNP* pose estimation with *RANSAC*-based outlier rejection, and validates the result using the object center reprojection error metric. If the reprojection error exceeds the acceptance threshold, the state machine commands a new viewpoint and repeats the estimation cycle until convergence or the viewpoint limit is reached.

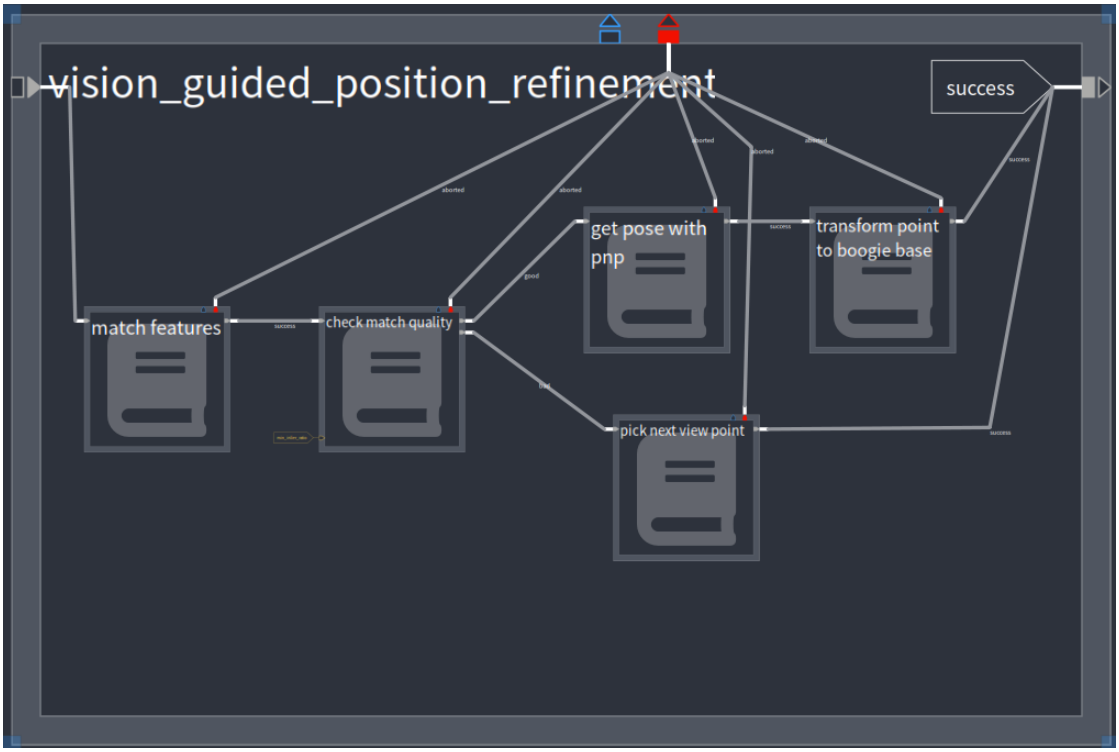


Figure D.3: Vision-guided position refinement state machine implementing multi-viewpoint PnP pose estimation with RANSAC outlier rejection and object center reprojection error validation.

D.4. Additional State Machines

The complete RAFCON implementation includes additional nested state machines for arm motion control, gripper commands, initialization sequences, visual centering, among others. These substates are not shown here as they do not contain pipeline-specific logic beyond what is described in Chapters 5 and 8.

Table D.1: Recovery and retry mechanisms implemented in the grasping pipeline

Failure Type	**Recovery Action**	**Parameters / Limits**
No object detected in RealSense	Retry depth-based perception	Up to 3 attempts, then return to `IDLE`
Object lost during monocular perception	Z-scan followed by grid search	2 Z heights (90 mm, 100 mm), 9-point grid with 20 mm spacing
Object not found after search exhaustion	Abort grasp attempt and cleanup	Failure logged after $2 \times 9 = 18$ search attempts
Insufficient data for 3D refinement	Request additional viewpoint	Iterates through predefined viewpoint list
No viewpoints available	Abort refinement	Transition to `CLEANUP`
Grasp attempt failed	Retry grasp sequence	Configurable, default: 3 attempts via centering phase
Grasp retries exhausted	Abort grasp and cleanup	Return to `IDLE` after cleanup
Large jump from coarse estimate	Fall back to coarse position	Threshold: 150 mm XY distance
Critical system error	Require manual operator reset	Reset command issued from `ERROR` state
Arm reports error or abort	Force transition to `ERROR`	Immediate override of current state

Table D.2: Key configuration parameters used in the autonomous grasping pipeline

Parameter	**Default**	**Description**
`grasp_retry_max`	3	Maximum number of grasp attempts before aborting
`max_jump_from_coarse_mm`	150.0	Maximum allowed XY deviation (mm) between refined and coarse target
`hover_z_offset_mm`	70	Vertical offset above object used for hover position
`z_search_heights`	[90, 100]	Z heights (mm) used during object search
`grid_step_mm`	20	Step size (mm) for grid search pattern
`grid_search_pattern`	9-point spiral	Center point followed by surrounding spiral positions
`realsense_perception_retries`	3	Maximum number of retries for RealSense detection
`debug_states`	false	Enables step-by-step execution using manual state advance

E. Results

E.1. All Object Detection Results



Figure E.1: Post-inference results for the blue cube with centered object position under lights-off (left) and lights-on (right) conditions.



Figure E.2: Post-inference results for the blue cube with object positioned to the right under lights-off (left) and lights-on (right) conditions.

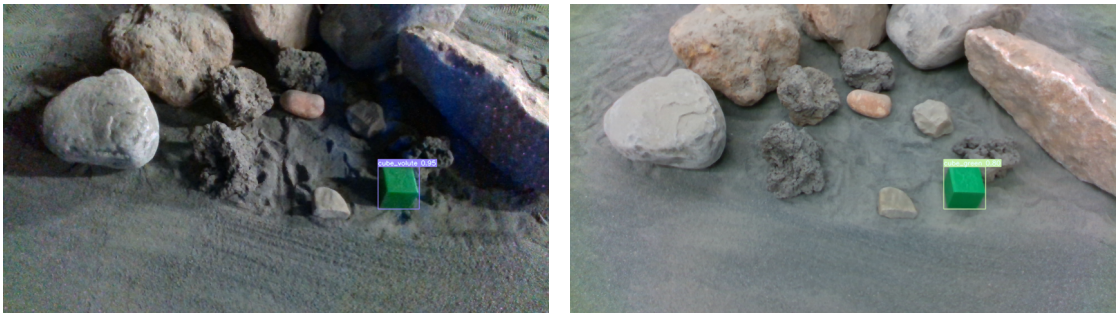


Figure E.3: Post-inference results for the green cube with object positioned to the right under lights-off (left) and lights-on (right) conditions.



Figure E.4: Post-inference results for the orange cube with centered object position under lights-off (left) and lights-on (right) conditions.



Figure E.5: Post-inference results showing the orange cube positioned to the right under lights-off conditions.



Figure E.6: Post-inference results for monocular camera detection at heights of 60 mm (left) and 80 mm (right).



Figure E.7: Post-inference results for monocular camera detection at heights of 90 mm (left) and 100 mm (right).

E.2. Image-Based Position Refinement Results

This section shows the complete visual centering sequence for Run 5 from Table 9.10. This run was executed at 80 mm height to validate the calibration table interpolation at a different operating height. The run required 13 iterations to converge, starting from an initial pixel error and progressively reducing it until the stopping condition was met ($|du| < 35$ px and $|dv| < 35$ px).

Each frame shows the target pixel location (green circle), the detected object center (red circle), and the pixel error vector (yellow arrow). The overlay displays the iteration number, operating height (80 mm), and the current pixel errors in both axes.

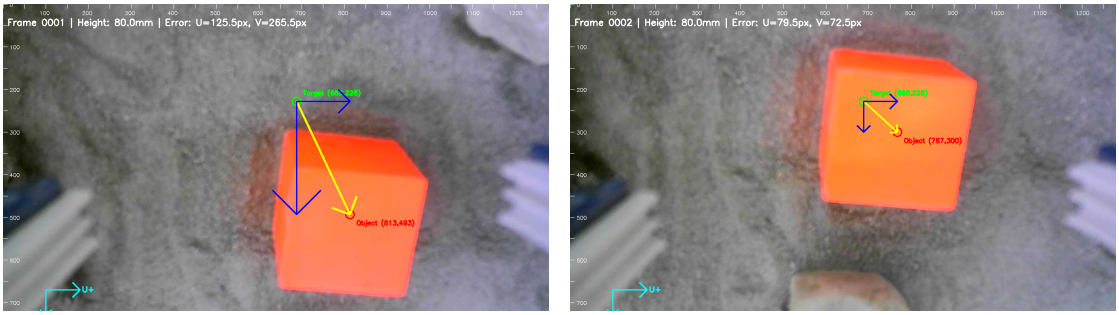


Figure E.8: Visual centering iterations 1-2. Left: Initial detection with large pixel offset. Right: After first correction step.

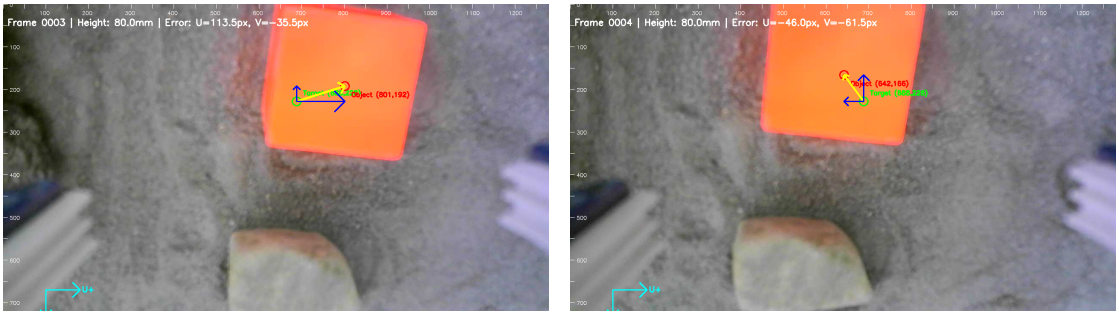


Figure E.9: Visual centering iterations 3-4. The pixel error decreases as the gripper position is adjusted based on visual feedback.

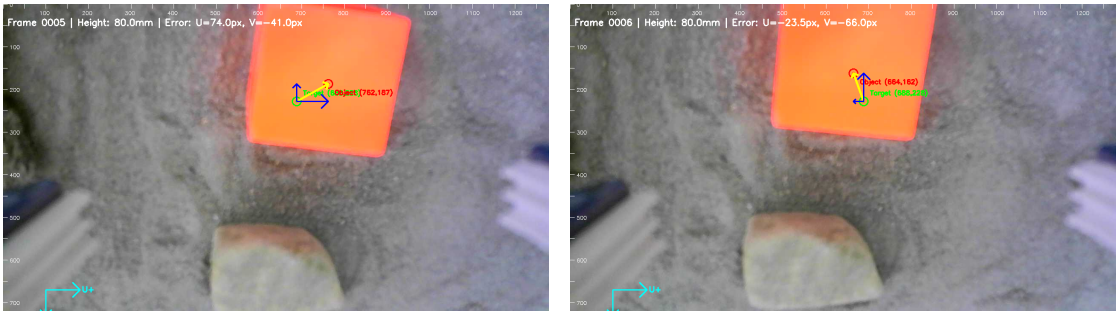


Figure E.10: Visual centering iterations 5-6. Continued convergence toward the target pixel location.

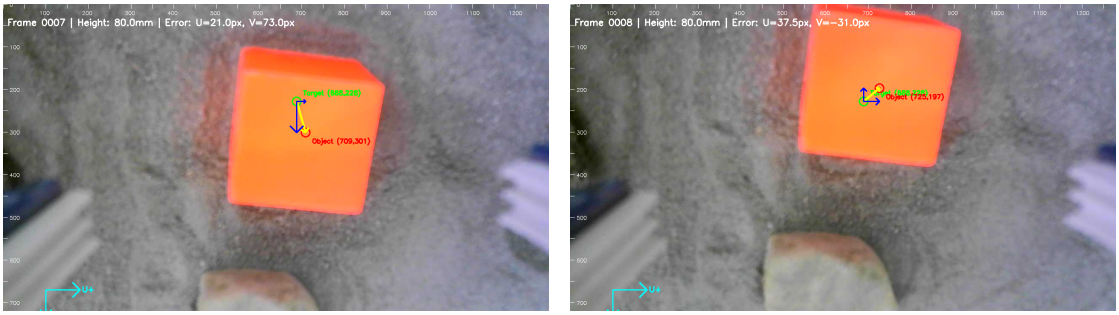


Figure E.11: Visual centering iterations 7-8. The correction vector shortens as the object center approaches the target.

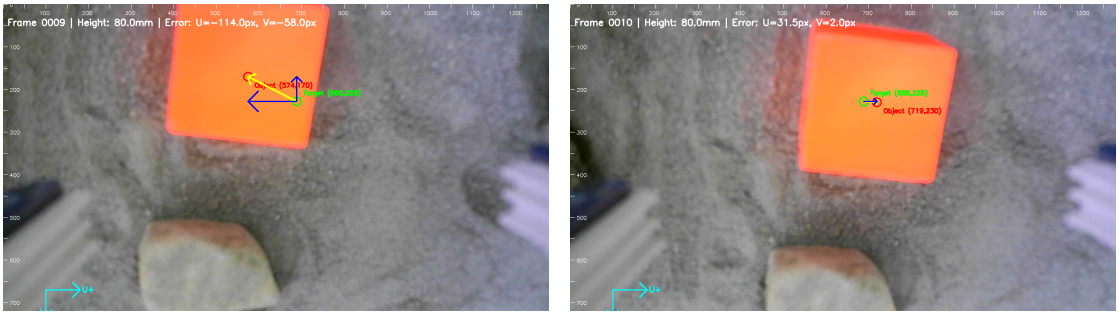


Figure E.12: Visual centering iterations 9-10. Progressive error reduction continues.

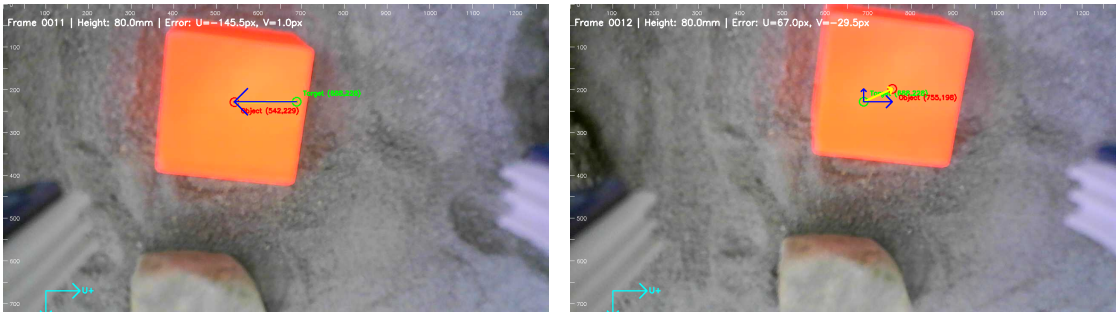


Figure E.13: Visual centering iterations 11-12. Approaching convergence.



Figure E.14: Visual centering iteration 13 (final). The pixel error satisfied the stopping condition (du = -25.0 px, dv = -17.0 px), resulting in a final error norm of 30.23 px.

The sequence demonstrates the iterative nature of the visual refinement approach at a different operating height than the standard 100 mm. Despite the initial large offset in iteration 1, the correction loop successfully reduced the error below the acceptance threshold within 13 iterations. The successful convergence at 80 mm height validates that the calibration table interpolation works correctly across different heights, enabling successful grasp execution.

Use of Generative AI Tools

Generative AI tools (ChatGPT by OpenAI and Claude by Anthropic) were used exclusively for linguistic review, text clarity improvement, and structural refinement of drafted sections. All scientific content, implementation, experimental design, data analysis, and conclusions were developed and validated independently by the author. The author assumes full responsibility for the correctness and integrity of this work.

F. Contact

Candidate

Ivan Gilberto Martin Enciso

ivan.martinenciso@dlr.de

i.martin.enciso@tu-berlin.de

Organization and Supervisors

Deutsches Zentrum für Luft- und Raumfahrt (DLR)

Dr.-Ing. Armin Wedler

armin.wedler@dlr.de

Technische Universität Berlin (TUB)

Prof. Enrico Stoll

e.stoll@tu-berlin.de

M.Sc. Maximilian von Unwerth

maximilian.von.unwerth@tu-berlin.de

References

[1] Planetary Society. Cost of the Mars Exploration Rover. `https://www.planetary.org/space-policy/cost-of-the-mars-exploration-rovers`. Accessed: 2025-07-22.

[2] Planetary Society. Cost of MSL Curiosity. `https://www.planetary.org/space-policy/cost-of-msl-curiosity`. Accessed: 2025-07-22.

[3] Planetary Society. Cost of Perseverance. `https://www.planetary.org/space-policy/cost-of-perseverance`. Accessed: 2025-07-22.

[4] Sam Bekkers, Michelle Bettendorf, Josef Reill, Riccardo Giubilato, and Armin Wedler. The Lunar Rover Mini: Towards a Versatile, Open-Source Mobile Robotic Platform for Educational and Experimental Purposes. In *17th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, Leiden, The Netherlands, October 2023. German Aerospace Center (DLR), Institute of Robotics and Mechatronics.

[5] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an Open-Source Robot Operating System. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, Kobe, Japan, 2009.

[6] The MathWorks, Inc. *Simulink*. The MathWorks, Inc., Natick, Massachusetts, United States, 2025. Version R2025a.

[7] Andreas Steck et al. RAFCON: A Graphical Tool for Task Programming and Mission Control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[8] T. F. Plácido de Castro. Design of an Utility-Frontier Based Exploration Strategy Coupled with 3D Object Localization for Modular Open-Source Autonomous Rovers: A State Machine Framework Approach Implemented and Tested on the Lunar Rover Mini. Master's thesis, Delft University of Technology, Delft, The Netherlands, 2025.

[9] OpenCV Developers. *OpenCV Camera Calibration and 3D Reconstruction*. Open Source Computer Vision Library, 2024. Accessed: 2026-01.

[10] Richard Hartley and Andrew Zisserman. *Preface*, pages xiii–xvi. Cambridge University Press, Cambridge, United Kingdom, 2004.

[11] OpenCV Developers. OpenCV: Open Source Computer Vision Library. `https://opencv.org/`, 2026. Accessed: 2026-01-27.

[12] Duane C. Brown. Close-Range Camera Calibration. *Photogrammetric Engineering*, 37(8):855–866, 1971.

[13] Juho Kannala and Sami S. Brandt. A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1335–1340, 2006.

[14] University of Toronto. Camera Parameters. `https://ftp.cs.toronto.edu/pub/psala/VM/camera-parameters.pdf`. Accessed: 2026-01-27.

[15] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7 Official Implementation. `https://github.com/WongKinYiu/yolov7`, 2022. Accessed: 2026-01-27.

[16] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[17] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Tim Kadir, and Luc Van Gool. A Comparison of Affine Region Detectors. *International Journal of Computer Vision*, 65(1):43–72, 2005.

[18] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[19] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[20] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[21] OpenCV Developers. Introduction to SIFT (Scale-Invariant Feature Transform). `https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html`, 2026. Accessed: 2026-01-27.

[22] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[23] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal of Computer Vision*, 81(2):155–166, 2009.

[24] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[25] Marco Conenna. Elasto-Kinematic Calibration of the Lunar Rover Mini 6DOF Robotic Arm. Technical report, Delft University of Technology, 2024. Accessed: 2025-05-14.

[26] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer, 2009.

[27] François Chaumette and Seth Hutchinson. Visual Servo Control, Part I: Basic Approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006.

[28] Antonio Bicchi and Vijay Kumar. Robotic Grasping and Contact: A Review. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 348–353, 2000.

[29] Ravinder S. Dahiya, Giorgio Metta, Maurizio Valle, and Giulio Sandini. Tactile Sensing—From Humans to Humanoids. *IEEE Transactions on Robotics*, 26(1):1–20, 2010.

[30] Hanna Yousef, Mehdi Boukallel, and Kaspar Althoefer. Tactile Sensing for Dexterous In-Hand Manipulation in Robotics—A Review. *Sensors and Actuators A: Physical*, 167(2):171–187, 2011.

[31] Zhanat Kappassov, Juan-Antonio Corrales, and Véronique Perdereau. Tactile Sensing in Dexterous Robot Hands—Review. *Robotics and Autonomous Systems*, 74:195–220, 2015.

[32] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *The International Journal of Robotics Research*, 37(4–5):421–436, 2018.

[33] Beom-Sahng Kim and Jae-Bok Song. Sensorless Grasping Force Control of a Robotic Gripper Using Current Measurement. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3326–3331, 2013.

[34] An De Greef, Pierre Lambert, and Alain Delchambre. Towards Flexible Medical Instruments: Review of Flexible Fluidic Actuators. *Precision Engineering*, 33(4):311–321, 2009. Includes discussion of current-based force estimation.

[35] Intel Corporation. Intel® NUC – Next Unit of Computing. `https://www.intel.com/content/www/us/en/products/docs/boards-kits/nuc/`, 2025. Accessed: 2025-01-15.

[36] Simply NUC. Leveraging Mini PCs for Robotic Applications. *Simply NUC Technical Blog*, November 2024. Accessed: 2025-01-15.

[37] Quelima. *SQ11 Mini DV Camera User Manual*, 2018. Accessed: 2026-01-27.

[38] ROS Documentation. sensor_msgs/JointState Message. `https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/JointState.html`, 2025. Accessed: 2025-12-07.

[39] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA, 1994.

[40] Chemnitz University of Technology. Transformation Matrices in Robotics. `https://www.tu-chemnitz.de/informatik/KI/edu/robotik/ws2017/trans.mat.pdf`. Accessed: 2026-01-27.

[41] ROS Enhancement Proposals. Rep 103 – standard units of measure and coordinate conventions. Technical report, Open Source Robotics Foundation, 2014. Accessed: 2026-01.

[42] Intel Corporation. Intel RealSense Depth Camera D435i Specifications. `https://www.intelrealsense.com/depth-camera-d435i/`, 2026. Accessed: 2026-01-27.

[43] AGFRC. AGFRC Servo Software and Firmware Downloads. `https://www.agf-rc.com/pages/Download-11925505.html`. Accessed: 2026-01-27.

[44] Intel Corporation. OpenVINO Toolkit: Optimizing and Deploying AI Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[45] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. Grasp Pose Detection in Point Clouds. *The International Journal of Robotics Research*, 36(13–14):1455–1473, 2017.

[46] AGFRC. AGFRC A80BHMF Digital Servo Specifications. `https://www.agfrc.com/index.php?id=2681`. Accessed: 2026-01-27.

[47] AGFRC. AGFRC A35BHL Digital Servo Specifications. `https://www.agfrc.com/index.php?id=2428`. Accessed: 2026-01-27.

[48] AGFRC. AGFRC A20CLS Digital Servo Specifications. `https://www.agfrc.com/index.php?id=2429`. Accessed: 2026-01-27.