ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Vision-Guided Robotic Manipulation for Object Interaction.

2nd Year Internship in University of Oldenburg

Robot Manipulator Interbotix ViperX-300s

Aurèle Planchard
Robotics Engineer
FISE 2026
aurele.planchard@ensta.fr

September 21, 2025

// Abstract

This project develops a vision-guided robotic manipulation system combining an Intel RealSense depth camera with an Interbotix ViperX robotic arm for autonomous object handling. The system integrates GPU-accelerated object detection, AprilTag tracking, and liquid level analysis with data fusion techniques that achieve sub-2cm positioning accuracy. A multi-threaded architecture running on NVIDIA Jetson ensures real-time performance across concurrent vision processing, Kalman filtering at 100 Hz, and robot control operations.

Such systems have practical applications in automated warehousing and logistics for package sorting and handling, food and beverage industries for container manipulation and fill-level monitoring, laboratory automation for sample handling, and assistive robotics for elder care or accessibility support. The modular design and scalable data fusion architecture provide a foundation for multi-robot collaborative systems and complex manipulation tasks in dynamic environments.

Acknowledgments

I want to express my gratitude to my supervisor, Prof. Dr. Andreas Rauh for his support and guidance throughout the internship. I also want to thank Seven Ahrens for helping me to kickstart the project and helping me along the way, and for sharing his valuable work.

Contents

List of Figures 3

1 Introduction 4
1.1 Setup . . . 4
1.2 Software, libraries . . . 4
1.3 Objectives . . . 6

2 Data Collection 7
2.1 Camera frame conversion . . . 7
2.2 Tag detection . . . 7
2.3 Object detection . . . 8
2.4 Liquid level detection . . . 8
2.5 Robot sensors . . . 9
2.5.1 Current implementation . . . 9
2.5.2 Limits and improvements . . . 9

3 Calibration 11
3.1 Points sampling . . . 11
3.2 Calibration . . . 12
3.3 Other attempts . . . 12
3.4 Results . . . 13

4 Interactions 14
4.1 Bottle Grabbing . . . 14
4.2 Tracking and Analysis . . . 15

5 Data fusion 17
5.1 Kalman model . . . 17
5.2 Results . . . 18
5.3 Improvements . . . 18

6 Real-time Interface 19
6.1 User interface design . . . 19
6.2 Threading architecture . . . 20
6.3 Data synchronization . . . 21

7 Conclusion 22

References 23

List of Figures

1 Interbotix ViperX-300s robotic manipulator in the experimental workspace. 4
2 Intel RealSense D435i camera mounted for optimal workspace coverage. . . 5
3 NVIDIA Jetson AGX Orin development board serving as the main computing platform. 5
4 AprilTag detection in action, showing the detected fiducial marker with pose estimation. 7
5 Object detection showing a bottle identified by the SSD-MobileNet-v2 neural network. 8
6 Liquid level detection process showing the computer vision pipeline. 9
7 Hand-eye calibration configuration showing the transformation chain from camera to robot base. 11
8 Calibration results showing the spatial alignment between camera and robot coordinate frames and validation of the computed transformation matrix. 13
9 Graphical user interface showing control buttons for robot operation, calibration, detection, and interaction tasks. 19
10 Multi-threaded system architecture showing concurrent processing of camera data, detections, and fusion. 20

1 Introduction

1.1 Setup

The experimental setup consists of three main hardware components that work together to enable vision-guided robotic manipulation.

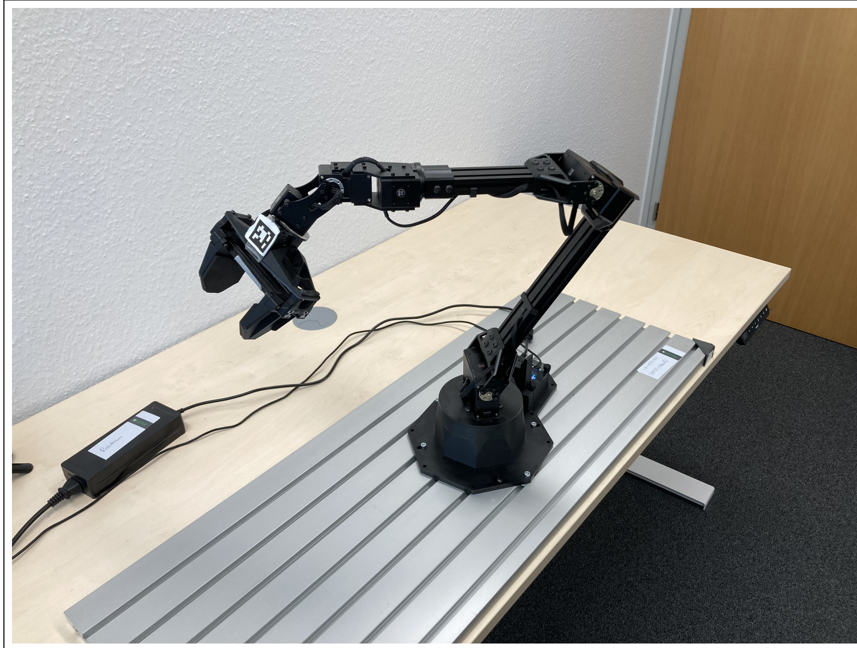

Figure 1: Interbotix ViperX-300s robotic manipulator in the experimental workspace.

The **Interbotix ViperX-300s** serves as the primary robotic manipulator, featuring 6 degrees of freedom and a precision gripper suitable for object manipulation tasks. The robot provides reliable joint position feedback through its integrated servo motors and supports both direct joint control and Cartesian space manipulation through the Interbotix API.

The **Intel RealSense D435i** camera provides both RGB and depth sensing capabilities at 640×480 resolution at 30 FPS. The camera is strategically positioned to capture the robot's workspace and enable real-time object detection and tracking. Its integrated depth sensor allows for accurate 3D localization of objects and AprilTag fiducials.

The **NVIDIA Jetson AGX Orin** acts as the central computing platform, running Ubuntu 20.04 and providing GPU acceleration for real-time computer vision tasks. Its ARM-based architecture and integrated GPU enable efficient processing of multiple data streams while maintaining real-time performance requirements.

1.2 Software, libraries

The software architecture is built on a combination of robotics frameworks, computer vision libraries, and custom Python modules designed for real-time operation.

Operating System and Framework:

- Ubuntu 20.04 LTS providing a stable Linux environment
- ROS2 (Robot Operating System 2) for robot communication and control

Figure 2: Intel RealSense D435i camera mounted for optimal workspace coverage.

Figure 3: NVIDIA Jetson AGX Orin development board serving as the main computing platform.

- Python 3.8 as the primary programming language

Robotics Libraries:

- `Interbotix API` for robot control and kinematics
- `Modern Robotics` library for advanced kinematic calculations

Computer Vision Libraries:

- `OpenCV (cv2)` for image processing and computer vision algorithms
- `pyrealsense2` for Intel RealSense camera interface and depth processing
- `apriltag` for fiducial marker detection and pose estimation
- `jetson-inference` for GPU-accelerated object detection using pre-trained neural networks

Scientific Computing:

- `numpy` for numerical computations and matrix operations
- `scipy` for advanced mathematical functions and optimization
- `scipy.spatial.transform` for rotation and transformation handling
- `threading` for concurrent processing and real-time data handling

The software architecture emphasizes modularity and real-time performance, with separate threaded processes handling camera data acquisition, object detection, robot control, and data fusion operations.

1.3 Objectives

This internship project focuses on developing a comprehensive vision-guided robotic manipulation system with three primary objectives:

1. **Data fusion for object localization:** Implement robust data fusion techniques combining camera-based observations with robot sensor feedback to accurately localize objects in the robot's coordinate frame. This involves developing calibration procedures and real-time transformation algorithms.
2. **Advanced object detection:** Develop and integrate computer vision algorithms for detecting and analyzing objects in the robot's workspace, including bottle recognition, pose estimation, and liquid level detection capabilities for manipulation planning.
3. **Intelligent robot control:** Implement sophisticated control algorithms that enable the robot to perform complex manipulation tasks such as object grasping, following, and analysis based on real-time visual feedback and sensor fusion.

The system aims to demonstrate autonomous robotic manipulation capabilities in realistic scenarios, with particular emphasis on handling containers and analyzing their contents. The integration of multiple sensor modalities and real-time processing creates a foundation for advanced robotic applications in industrial and service robotics domains.

2 Data Collection

The data collection system forms the foundation of our vision-guided robotic manipulation system. Multiple data streams are processed simultaneously to provide comprehensive information about the robot's environment and the objects it needs to manipulate.

Throughout the entire system, all spatial positions and orientations are consistently represented using 4×4 homogeneous transformation matrices. This mathematical framework enables unified handling of translation and rotation operations, simplifies coordinate frame transformations, and ensures compatibility with robotics libraries such as Modern Robotics or the Interbotix API. Each transformation matrix $T \in SE(3)$ encodes both the 3D position and orientation of objects, end-effector poses, and coordinate frame relationships in a computationally efficient format.

2.1 Camera frame conversion

The Intel RealSense D435i camera provides both RGB and depth data streams at 640x480 resolution at 30 FPS. The raw camera data is processed through several conversion steps to extract meaningful spatial information.

The depth frame and color frame are first aligned using the RealSense alignment functionality to ensure pixel correspondence between RGB and depth data. The depth values are then converted from the camera's internal units to metric distances using the camera's depth scale factor.

For spatial positioning, the system converts 2D pixel coordinates (u, v) with depth d to 3D Cartesian coordinates (x_c, y_c, z_c) in the camera frame using the RealSense deprojection function and the camera's intrinsic parameters. This conversion is essential for all subsequent 3D object localization tasks.

2.2 Tag detection

AprilTag detection is implemented using the `apriltag` Python library to provide reliable fiducial markers for calibration and tracking purposes. The system continuously monitors the video stream for AprilTags with configurable IDs.

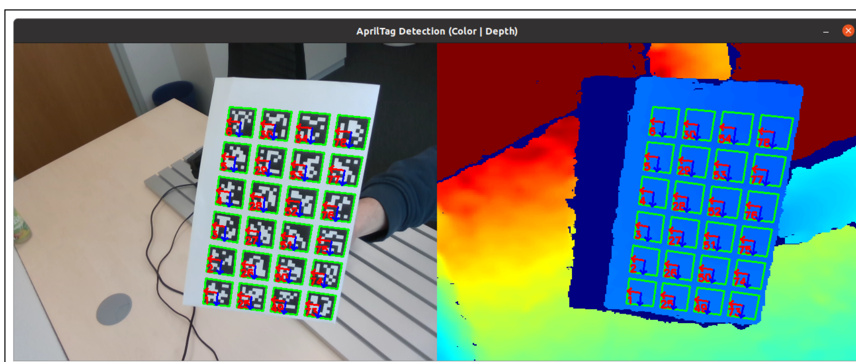

Figure 4: AprilTag detection in action, showing the detected fiducial marker with pose estimation.

The detection pipeline processes each incoming frame by converting it to grayscale and applying the AprilTag detector. When a tag is detected, the system extracts both

the 2D pixel coordinates of the tag corners and computes the 3D pose of the tag in the camera coordinate system.

The tag detection system is designed to run in a separate thread with shared data structures protected by threading locks. This allows real-time tag tracking while other processing continues simultaneously. The system can be configured to detect specific tag IDs and provides transformation matrices for detected tags.

2.3 Object detection

Object detection utilizes the Jetson Inference library with a pre-trained SSD-MobileNet-v2 model to identify and localize objects in the camera feed. The system is specifically configured to detect bottles and cups (class IDs 44 and 47 respectively) with a confidence threshold of 0.5.

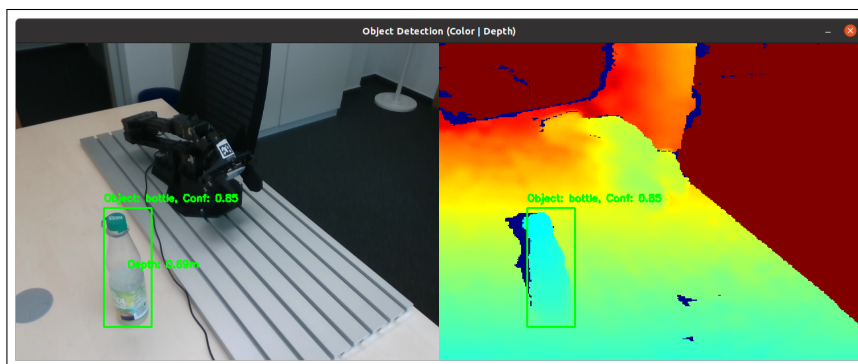

Figure 5: Object detection showing a bottle identified by the SSD-MobileNet-v2 neural network.

The detection pipeline converts the input RGB frame to CUDA format for GPU acceleration, then applies the neural network inference. For each detected object, the system extracts bounding box coordinates and confidence scores. The 3D position of detected objects is calculated by combining the 2D bounding box center with depth information from the aligned depth frame.

The object detection runs continuously in a threaded architecture, updating shared data structures with the latest object positions and transformation matrices. This enables real-time object tracking and manipulation planning.

2.4 Liquid level detection

The following liquid level detection methods are largely adapted from the work of Sven Ahrens.

Liquid level detection is implemented using computer vision techniques to analyze the contents of detected containers. The system focuses on the region of interest defined by the object detection bounding box to perform detailed liquid analysis.

The detection algorithm follows a four-step computer vision pipeline:

1. **Extraction:** Using the bounding box coordinates provided by the object detection system, crop the aligned RGB and depth frames to isolate the region of interest (ROI) containing the detected container.

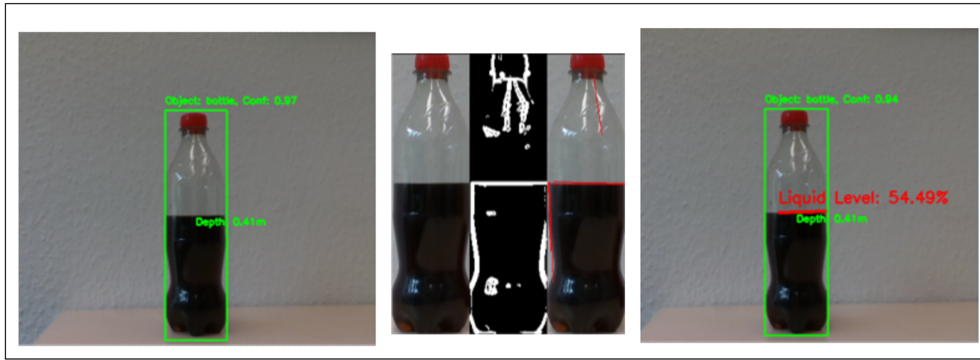

Figure 6: Liquid level detection process showing the computer vision pipeline.

2. **Image processing:** Apply Gaussian blur for noise reduction, compute Sobel operators in both X and Y directions for edge detection, and convert gradient magnitude to binary edge images using adaptive thresholding techniques.
3. **Line detection (Hough):** Apply probabilistic Hough line transformation to detect line segments from the binary edge image and filter for nearly horizontal segments (within $\pm 10°$) that represent potential liquid surfaces.
4. **Level estimation:** Identify the highest detected horizontal line as the liquid level and calculate the fill percentage based on the container height within the region of interest.

This information is continuously updated and made available to the manipulation system in real-time.

2.5 Robot sensors

2.5.1 Current implementation

The robot's internal sensors provide joint position feedback and end-effector pose information through the Interbotix API. Joint states are continuously monitored and used for forward kinematics calculations to determine the precise 3D position and orientation of the robot's end-effector.

The system uses the Modern Robotics library to compute forward kinematics based on the robot's kinematic model and current joint positions. This provides an independent measurement of the end-effector pose that can be fused with camera-based observations for improved accuracy.

Robot sensor data is integrated into the data fusion system where it serves as one input to the Kalman filter alongside camera-based measurements.

2.5.2 Limits and improvements

The current robot sensor integration has several limitations due to working exclusively through the Interbotix API. The API provides access to basic joint positions and end-effector poses, but does not expose comprehensive sensor data such as joint velocities, motor currents, torque measurements, or real-time force feedback from the gripper. This limits the system's ability to perform advanced control strategies or detect contact forces during manipulation tasks.

Future improvements would involve migrating to direct ROS2 integration to access comprehensive sensor data through ROS2 topics, particularly focusing on enhanced data fusion capabilities and force sensing. This would enable real-time acquisition of joint velocities and motor currents to improve the Kalman filter's motion model, while torque and force measurements would allow implementation of force-sensitive manipulation strategies. The additional sensor streams would significantly enhance the data fusion system's accuracy and enable contact-aware robotic operations for more sophisticated object interaction scenarios.

3 Calibration

The calibration process establishes the spatial relationship between the camera coordinate frame and the robot base frame, enabling accurate transformation of detected objects from camera observations to robot-actionable positions. This critical step ensures that visual detections can be reliably converted into robot movements.

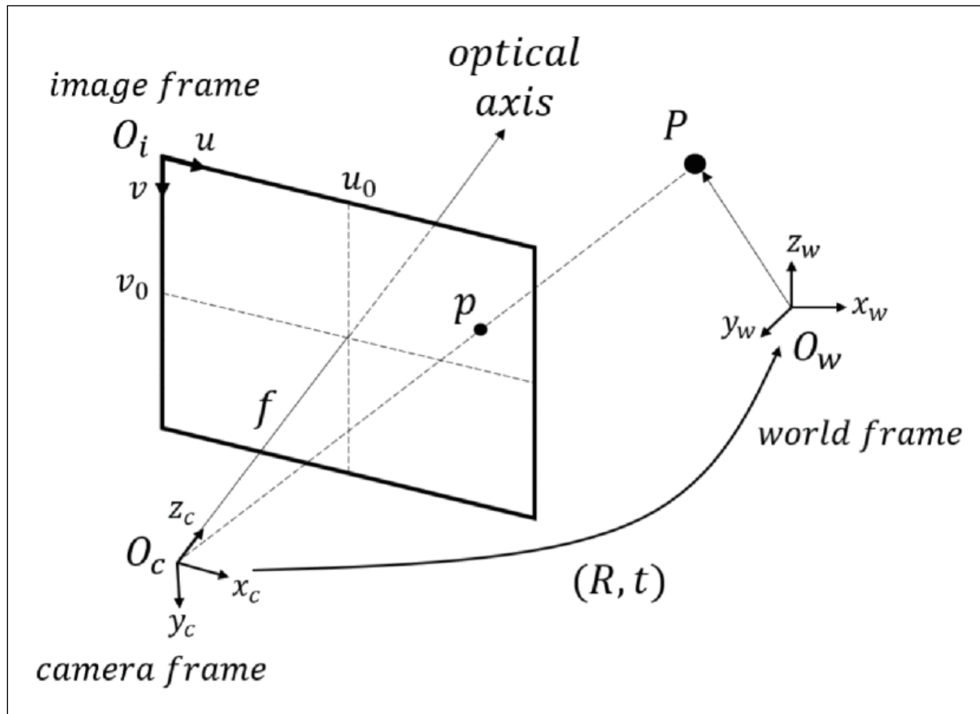

Figure 7: Hand-eye calibration configuration showing the transformation chain from camera to robot base.

3.1 Points sampling

The calibration procedure begins with systematic collection of corresponding point pairs in both the camera and robot coordinate frames. An AprilTag (ID 96) is attached to the robot's end-effector at a known offset from the tool center point, serving as a visual reference marker that can be detected by the camera.

The sampling strategy employs a two-phase approach:

Phase 1: Camera position estimation

The robot end-effector is moved to 8 positions distributed around the robot base in a semicircle at 0.4m radius and 0.2m height. Angular positions range from $-\pi$ to $\frac{3\pi}{4}$ with orientation angled at $\frac{\pi}{3}$ to face the camera. At each position, the system verifies AprilTag visibility to determine the camera's approximate angular position relative to the robot. The camera's angular position is estimated using the circular mean of valid detection angles: $\theta_{cam} = \arctan 2(\sum \sin(\theta_i), \sum \cos(\theta_i))$, which provides a robust estimate even when some positions fail to detect the tag.

Phase 2: Reference points collection

Based on the estimated camera location, 12 reference positions are generated to provide comprehensive spatial coverage. These positions span across 4 azimuthal angles

centered on the camera, ranging from $[\theta_{cam} - \frac{\pi}{2}$ to $\theta_{cam} + \frac{\pi}{2}]$. Each angular position is sampled at 3 different heights (0.1m, 0.2m, 0.3m) to provide vertical diversity in the dataset. At each position, the end-effector orientation is adjusted to face the estimated position of the camera perpendicularly with respect to the optical axis, ensuring optimal AprilTag detection. At each position, the system records both the robot's end-effector pose (computed via forward kinematics) and the AprilTag pose detected by the camera, creating corresponding point pairs in both coordinate frames.

The end-effector poses are corrected for the physical offset between the gripper center point and the AprilTag center ([-0.1m, 0.0m, 0.03m] in the end-effector frame), ensuring accurate correspondence between the two coordinate systems.

A visualization of this calibration process is available: https://youtu.be/-Zu1m5oe0Sw

3.2 Calibration

The transformation matrix $T_{cam}^{robot} \in SE(3)$ is computed using a least-squares approach based on Singular Value Decomposition (SVD), which optimally aligns the two point clouds.

Algorithm steps:

1. **Data preparation:** Extract 3D position vectors from the N collected pose pairs, forming two point clouds P_{cam} and P_{robot}, each containing N points in 3D space
2. **Centering:** Compute centroids $\bar{p}_{cam}$ and $\bar{p}_{robot}$, then center both point clouds:
$$P'_{cam} = P_{cam} - \bar{p}_{cam}, \quad P'_{robot} = P_{robot} - \bar{p}_{robot}$$
3. **Covariance matrix:** Compute the cross-covariance matrix $H = P'_{cam} \cdot P'^{T}_{robot}$
4. **SVD decomposition:** Perform singular value decomposition $H = U\Sigma V^T$
5. **Rotation matrix:** Compute the optimal rotation $R = VU^T$. If $\det(R) < 0$, negate the third row of V to ensure a right-handed coordinate system
6. **Translation vector:** Compute the translation $t = \bar{p}_{robot} - R\bar{p}_{cam}$
7. **Transformation matrix:** Assemble the homogeneous transformation matrix with rotation R and translation t

The calibration accuracy is quantified by computing residual errors: transforming camera points using the computed transformation and measuring Euclidean distances to corresponding robot points. The mean residual error provides a metric of calibration quality.

3.3 Other attempts

Alternative calibration approaches were explored during development:

Kalman filter approach: An Extended Kalman Filter was tested to iteratively refine the transformation estimate using sequential measurements. However, the main reason for not using this approach was the bad results it produced. Additionally, Kalman filtering is designed for real-time computation with streaming data, whereas in this case

the points were already sampled in batch, making the batch optimization with SVD more appropriate.

Non-linear optimization: The hand-eye calibration problem formulated as $AX = XB$ was considered, where A represents robot motion, B represents camera observations, and X is the unknown transformation. However, the simpler point-based approach was preferred for its robustness and direct interpretability.

For a comprehensive review of hand-eye calibration methods and comparative results, see [1].

3.4 Results

The calibration procedure successfully establishes a stable transformation matrix between camera and robot frames. The mean residual error didn't exceed 1.5cm, which is acceptable for the manipulation tasks targeted by this system, considering that the precision of the robot's end-effector is within the same range.

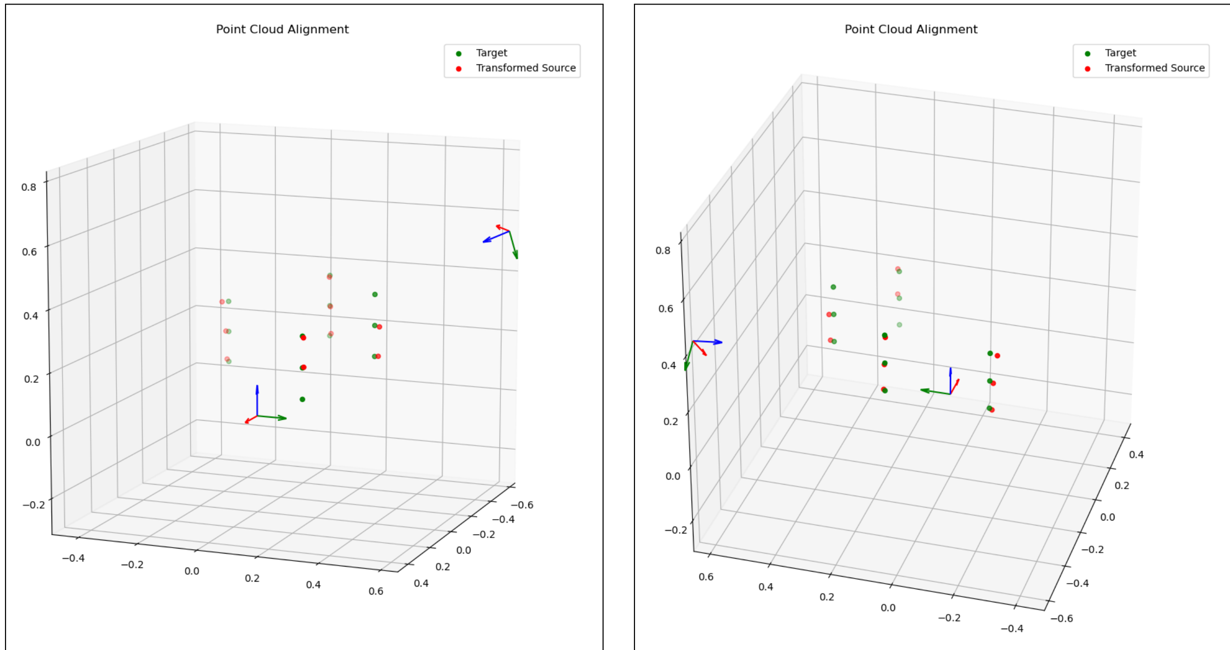

Figure 8: Calibration results showing the spatial alignment between camera and robot coordinate frames and validation of the computed transformation matrix.

The transformation matrix is saved persistently and can be reloaded for subsequent sessions, eliminating the need for recalibration unless the camera mount or robot configuration changes. A recalibration flag in the shared data structure allows the data fusion system to update its transformation matrix when needed.

The calibration quality is validated through visual inspection during manipulation tasks and by monitoring the consistency of object positions when tracked simultaneously by camera-based detection and robot forward kinematics.

4 Interactions

This section describes the robotic manipulation capabilities implemented in the system. Building upon the calibrated camera-robot transformation and real-time data collection infrastructure, the system enables vision-guided manipulation tasks that demonstrate autonomous object interaction. The primary focus is on bottle grasping, which serves as a representative task combining object detection, spatial reasoning, motion planning, and gripper control.

The interaction capabilities leverage the complete perception-action pipeline: object positions detected by the camera are transformed into robot coordinates using the calibration matrix, enabling the robot to accurately reach and manipulate objects in its workspace. The implementation emphasizes safe and reliable operation through multi-stage approach strategies and comprehensive error handling.

4.1 Bottle Grabbing

The bottle grasping function implements autonomous pick-and-place capabilities for detected containers. This task represents a complete manipulation scenario that integrates all previously described system components: camera-based object detection, coordinate transformation, and robot control.

Grasping Strategy:

The grasping algorithm employs a two-stage motion planning approach to ensure safe and reliable object acquisition:

1. **Initialization:** The robot moves to a safe starting pose positioned above the workspace at coordinates ($x = 0.1$m, $y = 0.0$m, $z = 0.65$m) relative to the robot base. The gripper is opened to prepare for grasping, ensuring no objects are held from previous operations.

2. **Intermediate approach:** Rather than moving directly to the detected bottle position, the system computes an intermediate waypoint positioned 15cm away from the bottle along the line connecting the robot base to the target. This two-stage approach reduces the risk of collisions and allows the robot to approach the object from a predictable direction. The intermediate distance $d_{intermediate}$ is calculated based on the bottle distance $d_{bottle} = \sqrt{x_{bottle}^2 + y_{bottle}^2}$ (Euclidean distance from robot base to bottle in the XY plane). If $d_{bottle} \geq 0.15$m, then $d_{intermediate} = d_{bottle} - 0.15$m; otherwise $d_{intermediate} = d_{bottle}$.

3. **Final approach and grasp:** After successfully reaching the intermediate position, the robot moves directly to the bottle's detected position. The end-effector orientation is adjusted to face the bottle by computing the azimuthal angle $\theta = \arctan 2(y_{bottle}, x_{bottle})$ and applying a rotation matrix $R_z(\theta)$ about the Z-axis. Once the end-effector reaches the target position, the gripper closes to complete the grasp.

A demonstration of this bottle grabbing process is available: https://youtu.be/QXpp6NoNTBw

Safety Constraints and Error Handling:

The grasping algorithm incorporates several safety mechanisms to prevent collisions and ensure robust operation:

- **Minimum distance check:** If the bottle is detected closer than 5cm to the robot base ($d_{bottle} < 0.05$m), the grasp attempt is aborted to avoid kinematic singularities and potential collisions with the robot's base.
- **Motion validation:** The system verifies the success of each motion command (using the Interbotix API return values) before proceeding to the next stage. If a commanded pose is unreachable due to workspace limits or kinematic constraints, the system retries after a brief delay.
- **Timeout mechanism:** A 30-second timeout prevents infinite loops in cases where object detection fails or the bottle moves out of the workspace. The algorithm continuously polls for updated bottle positions from the shared data structure maintained by the detection thread.

Upon successful completion of the grasp, the system reports success and maintains the gripper closure. The robot can then proceed to subsequent manipulation tasks such as moving the bottle to a different location or positioning it for content analysis.

In practice, the bottle grasping routine demonstrated a good success rate in informal tests. Although comprehensive quantitative metrics are not available, the system successfully grasped the target bottle in around 9 out of 10 trial runs. This anecdotal result indicates reliable behaviour for the targeted manipulation tasks, while a more systematic evaluation would be required for rigorous performance characterization.

4.2 Tracking and Analysis

Beyond static grasping, the system implements two additional interaction modes that demonstrate continuous vision-guided control: real-time tracking of moving objects and automated bottle content analysis.

AprilTag Following:

The AprilTag following function enables the robot to continuously track and follow a fiducial marker in real-time. This action is originally designed to test the system's tag detection capabilities. This capability is useful for applications requiring the robot to maintain a fixed spatial relationship with a moving reference frame, such as following a moving platform or maintaining alignment with a dynamic workspace.

The tracking algorithm operates in a continuous control loop for a specified duration (typically 20 seconds). At each iteration, the system reads the latest AprilTag pose from the detection thread's shared data structure. If a tag with the specified ID is detected, its pose is transformed from camera coordinates to robot base coordinates using the calibration matrix. The system continuously commands the robot to move toward the tag's current position with a control rate limited by the robot's motion constraints. This creates a tracking behavior where the end-effector follows the tag as it moves through the workspace.

Bottle Following:

Similar to AprilTag tracking, the bottle following function demonstrates continuous tracking of detected containers. This mode reads object poses from the neural network-based detection system rather than fiducial markers, showcasing the system's ability to track arbitrary objects without requiring special markers.

The control architecture mirrors the AprilTag following implementation: the system continuously polls the shared data structure for updated bottle positions, transforms coordinates to the robot frame, and commands the end-effector to move toward the detected

object. The main difference lies in the detection source (SSD-MobileNet-v2 neural network versus AprilTag detector) and the handling of detection uncertainties inherent to learned object detection models.

Limits

The main limitation of the tracking implementations is the robot's controlling techniques, which rely on sequential blocking commands through the Interbotix API. This approach limits the control frequency and responsiveness of the tracking behavior, as each motion command must complete before the next can be issued. As a result, the robot's ability to smoothly follow fast-moving objects is constrained, leading to potential lag and overshoot in dynamic scenarios. A solution would be to implement a non-blocking control loop with velocity commands, allowing for higher-frequency updates and more fluid tracking performance.

5 Data fusion

The purpose of the data fusion stage is to combine measurements from heterogeneous sensors (camera-based pose estimates and robot kinematics) into a single, more accurate estimate of object and end-effector state. By fusing camera observations with motor/encoder readings, the system reduces the effect of drift and bias present in the robot's proprioceptive sensors, leading to more stable long-term tracking. This fused estimate will also be essential when scaling to multi-camera and multi-robot setups: coordinated fusion across several sensors and platforms reduces cross-device inconsistencies and enables robust distributed perception. This section describes the Kalman-based filtering approach used to fuse noisy depth/vision measurements with the robot's forward-kinematics estimates, discusses the chosen state and measurement models, and outlines practical improvements and limitations observed during experiments.

5.1 Kalman model

The Kalman filter state-space model combines the system dynamics and the two measurement sources (robot encoders and camera) as follows:

$$\begin{cases} \underline{x}_{k+1} &= A_k \underline{x}_k + u_k + \alpha_k \\ y_1 &= C_1 \underline{x} + \beta_1 \\ y_2 &= C_2 \underline{x} + \beta_2 \end{cases}$$

where α_k represents process noise, and β_1, β_2 represent measurement noise from the encoders and camera respectively. In the implementation, the measurement noise covariance matrices are set to:

$$\Gamma_{\beta_1} = \Gamma_{\beta_2} = \mathrm{diag}(10^{-4}, 10^{-4}, 10^{-4}, 10^{-12})$$

where the first three diagonal elements correspond to position measurement uncertainties in x, y, z. Assuming a standard deviation of 1 cm = 0.01 m, the variance is $(0.01)^2 = 10^{-4}$ m^2. The last element is effectively zero for the homogeneous coordinate.

The state vector includes position, velocity, and a homogeneous coordinate:

$$\underline{x}_k = \begin{pmatrix} x \\ y \\ z \\ 1 \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \in \mathbb{R}^7$$

The state transition matrix A_k implements a constant-velocity model with time step dt:

$$A_k = \begin{pmatrix} I_3 & 0_3 & dt \cdot I_3 \\ 0_3^T & 1 & 0_3^T \\ 0_3 & 0_3 & I_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The control input is zero (the robot's commands are not known): $u_k = 0$.
The two measurement models are:

$$y_1 = \begin{pmatrix} x_{enc} \\ y_{enc} \\ z_{enc} \\ 1 \end{pmatrix}, \quad C_1 = \begin{pmatrix} I_4 & 0_{4\times3} \end{pmatrix}, \quad y_2 = \begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{pmatrix}, \quad C_2 = \begin{pmatrix} T^{-1} & 0_{4\times3} \end{pmatrix}$$

where T^{-1} is the inverse of the camera-to-robot calibration transformation matrix computed in Section 3.

5.2 Results

The Kalman filter fusion approach was implemented and tested in the 'fusion.py' module. The filter runs in a dedicated thread at approximately 100 Hz (10 ms time step) and continuously updates the state estimate based on available sensor measurements.

The fusion algorithm alternates between two measurement sources depending on data availability: when camera detections are available, the filter updates using the camera measurement (y_2) transformed through the calibration matrix; otherwise, it relies on robot forward kinematics from encoder readings (y_1). This adaptive strategy ensures continuous state estimation even when visual tracking is temporarily lost.

In practice, the fused estimate provides smoother and more consistent position tracking compared to using either sensor alone. It is difficult to provide quantitative error metrics without ground truth data, but qualitative observations indicate that the Kalman filter effectively tracked the robot's end-effector position, and the speed components of the state vector converges to 0 when the robot is stationary, demonstrating the filter's ability to model motion dynamics.

The state covariance matrix Γ_x maintained by the filter provides real-time uncertainty estimates, which could be used for confidence-weighted control decisions or to detect tracking failures when uncertainty exceeds predefined thresholds.

5.3 Improvements

Several improvements to the current data fusion implementation could enhance performance and robustness:

Dynamic process noise adaptation: The current implementation uses a fixed process noise covariance Γ_α. Adaptive tuning based on observed motion characteristics or measurement residuals could improve tracking performance for objects with varying dynamics.

Multi-object tracking: The current filter tracks a single object or end-effector pose. Extending to multiple objects would require implementing separate filter instances or a more sophisticated joint state representation with association logic.

Integration of force/torque sensing: As mentioned in Section 2.5.2, incorporating force and torque measurements from the robot would enable contact-aware manipulation and improve state estimates during physical interaction with objects.

6 Real-time Interface

The real-time interface ties together all system components into a unified framework for human-robot interaction and autonomous operation. The system architecture is designed around concurrent processing: multiple threads run simultaneously to handle camera data acquisition, object detection, robot control, data fusion, and user interaction without blocking each other. This section describes the user interface design that enables manual control and task initiation, the threading architecture that ensures responsive real-time performance, and the data synchronization mechanisms that maintain consistency across concurrent processes.

6.1 User interface design

A graphical user interface (GUI) was implemented using the Tkinter library to provide interactive control over the robotic system. The interface serves as the primary means for human operators to initiate tasks, monitor system state, and manually control robot functions.

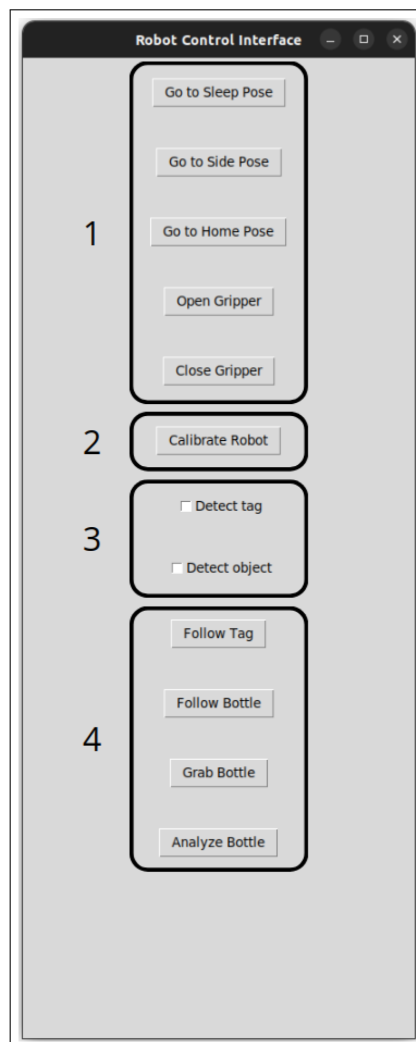

Figure 9: Graphical user interface showing control buttons for robot operation, calibration, detection, and interaction tasks.

The GUI includes several categories of controls:

1. **Basic robot control:** Buttons for homing the robot, moving to sleep pose, and manual gripper control (open/close) provide fundamental manipulation capabilities and safe initialization/shutdown procedures.
2. **Calibration:** A dedicated button initiates the automatic calibration procedure described in Section 3, collecting pose pairs and computing the camera-to-robot transformation matrix.
3. **Detection toggles:** Checkboxes enable or disable AprilTag detection and object detection threads independently, allowing selective activation of perception modules based on task requirements.
4. **Interaction commands:** Task-specific buttons trigger high-level behaviors such as "Follow Tag," "Follow Bottle," "Grab Bottle," and "Analyze Bottle," which execute the autonomous manipulation sequences described in Section 4.

The interface design prioritizes simplicity and real-time feedback, with all actions executing asynchronously to prevent GUI freezing during long-running operations.

6.2 Threading architecture

The system employs a multi-threaded architecture to achieve real-time performance and responsive operation. Each major system component runs in a dedicated thread, allowing parallel execution of computationally intensive tasks:

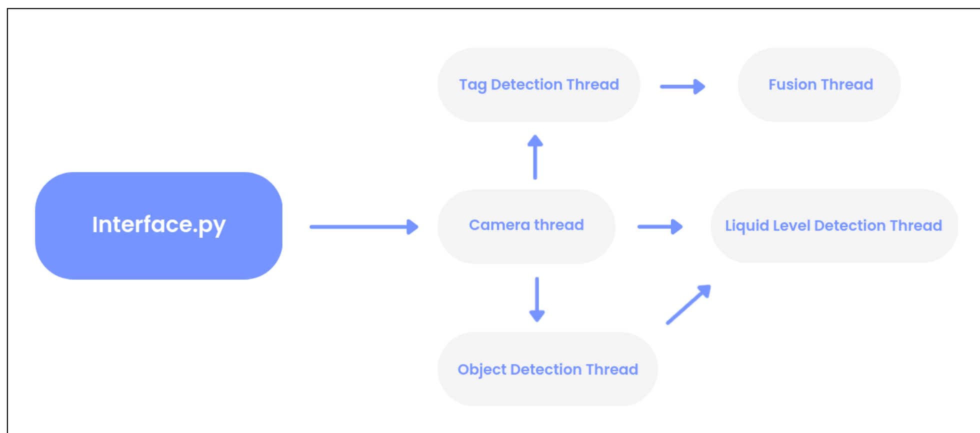

Figure 10: Multi-threaded system architecture showing concurrent processing of camera data, detections, and fusion.

GUI thread: Handles user input events and updates interface elements, running the Tkinter event loop independently of computation-heavy tasks. It's the main thread that spawns other worker threads.

Camera acquisition thread: Continuously captures RGB and depth frames from the Intel RealSense camera at 30 FPS, performing frame alignment and preprocessing before making data available to detection threads.

Object detection thread: Runs the Jetson Inference neural network to detect bottles and cups in the camera feed, updating shared data structures with detected object poses transformed to robot coordinates.

AprilTag detection thread: Processes camera frames to detect and localize AprilTag fiducials, computing 3D poses for calibration or tracking purposes.

Liquid level detection thread: Analyzes regions of interest within detected containers to estimate fill levels using the computer vision pipeline described in Section 2.4.

Data fusion thread: Executes the Kalman filter at 100 Hz to fuse camera observations with robot encoder measurements, providing smooth state estimates as described in Section 5.

This architecture ensures that slow operations (e.g., neural network inference, robot motion) do not block critical real-time processes or user interaction.

6.3 Data synchronization

Coordinating data access across multiple concurrent threads requires careful synchronization to prevent race conditions and ensure data consistency. The system employs two primary mechanisms:

Thread-safe shared data structures: A global `shared_data` dictionary stores information exchanged between threads, including detected object poses, transformation matrices, sensor measurements, and control flags. Access to this dictionary is protected by a `threading.Lock` object (`data_lock`), ensuring that only one thread can read or write at a time.

Lock acquisition protocol: Threads acquire the lock using Python's context manager (`with data_lock:`), perform their read or write operations, and automatically release the lock upon exiting the context. This pattern prevents deadlocks and ensures minimal lock hold times.

Non-blocking reads: Detection and control threads are designed to gracefully handle cases where expected data is not yet available (e.g., `None` values), allowing the system to continue operating even when some sensors are temporarily inactive or processing delays occur.

Motion queues: Robot control commands are passed through thread-safe `queue.Queue` objects rather than direct function calls, decoupling command generation from execution and enabling buffering of sequential motion primitives.

The synchronization strategy balances real-time responsiveness with data integrity, ensuring that the system operates reliably under concurrent load while maintaining deterministic behavior for safety-critical robot control operations.

7 Conclusion

This internship project successfully developed a comprehensive vision-guided robotic manipulation system that integrates multiple perception modalities with intelligent control algorithms to achieve autonomous object interaction. The system demonstrates effective coordination between camera-based vision, real-time object detection, data fusion, and robotic control components.

The implementation achieved several key technical milestones: robust hand-eye calibration with sub-2cm accuracy, real-time multi-threaded processing architecture enabling concurrent operation of vision, fusion, and control systems, and successful autonomous bottle grasping. The Kalman filter-based data fusion approach effectively combines heterogeneous sensor measurements, providing smoother and more reliable state estimates than individual sensors alone.

The modular software architecture proved valuable for system development and debugging, with clear separation between perception, fusion, and control modules. The threading design ensures responsive real-time performance while maintaining data consistency through careful synchronization mechanisms. The graphical user interface provides intuitive access to system functions and facilitates rapid prototyping of new manipulation behaviors.

Several opportunities for future enhancement were identified during development. Migrating to full ROS2 integration would provide access to comprehensive robot sensor data including joint velocities, motor currents, and force/torque measurements, enabling more sophisticated control strategies and contact-aware manipulation. Extending the data fusion system to handle multiple objects simultaneously would enable more complex scenarios such as sorting or assembly tasks. Adaptive process noise tuning could improve tracking performance for objects with varying dynamics, while integration of force sensing would enable compliant manipulation and better handling of delicate objects.

The project establishes a solid foundation for advanced vision-guided manipulation applications. The scalable architecture is well-suited for extension to multi-camera and multi-robot configurations, while the demonstrated capabilities in object detection, pose estimation, and data fusion provide the essential building blocks for autonomous robotic systems in industrial, laboratory, and service robotics domains.

References

[1] I. Enebuse, M. Foo, B. S. K. K. Ibrahim, H. Ahmed, F. Supmak, and O. S. Eyobu, "A comparative review of hand-eye calibration techniques for vision guided robots," *IEEE Access*, vol. 9, pp. 113143–113155, 2021.

[2] K. H. Strobl and G. Hirzinger, "Optimal hand-eye calibration," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Beijing, China), pp. 4647–4653, IEEE/RSJ, Oct. 2006.