GROUP 2

Object Detection and Pose Estimation with ROS2 Integration

TEAM MEMBERS:
OLUWATUNMISE SHUAIBU
ARASH BAZRAFSHAN
MYU WAI SHIN

PDE3802 AI IN ROBOTICS
MIDDLESEX UNIVERSITY LONDON
DEC 2025

Project Aim & Objectives

The aim of this project is to design and implement a vision-based perception system on a Raspberry Pi that can detect multiple objects, estimate their poses, and publish their positions as TF frames relative to the camera in ROS.

Objectives

- Detect five different objects using a camera connected to a Raspberry Pi
- Estimate object pose using two different approaches:
  - 2 objects with ArUco markers
  - 3 objects without markers
- Compute object coordinates relative to the camera frame
- Publish object poses as TF frames for real-time visualisation in RViz
- Evaluate detection reliability and pose accuracy under embedded hardware constraints

Problem Statement

While fiducial markers such as ArUco tags make pose estimation reliable, they are not always available or practical in real-world environments.

Marker-less pose estimation is more flexible but significantly more challenging, especially when deployed on resource-limited platforms like the Raspberry Pi.

This project tackles these challenges by combining marker-based and marker-less pose estimation methods within a single ROS-based pipeline, and by publishing all detected object poses as TF frames, allowing consistent spatial representation and visual validation in RViz.

Git-Based Development Workflow:

Initial Setup:

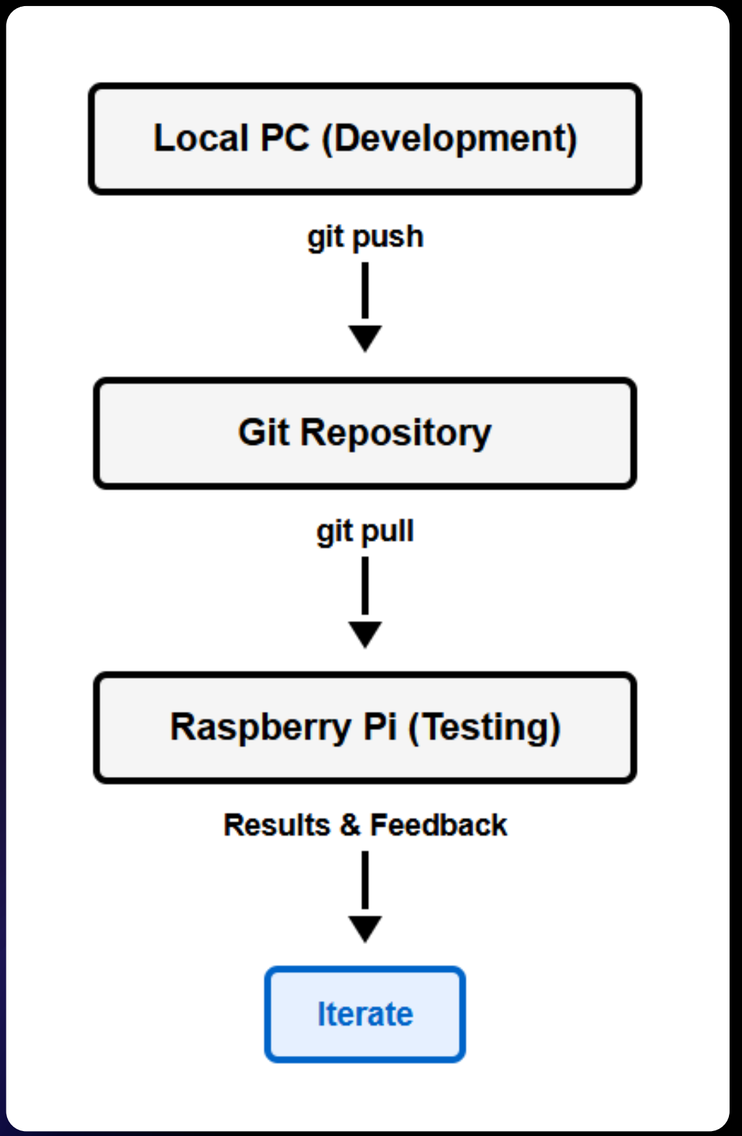
- Shared Git repository created at project start
- All collaborators added with SSH access configured
- Remote deployment enabled from development PC to Raspberry Pi

Iterative Development Process:

- Code developed and tested on local PC
- Changes pushed to repository
- Raspberry Pi pulls latest code for hardware testing
- Results inform next iteration

Benefits:

- Consistent codebase across devices
- Easy rollback when optimizations failed
- Rapid testing cycles with synchronized environments



Identifying High-Risk Components

Our Analysis

After reviewing the assessment brief and all required components, we identified markerless pose estimation as the most challenging aspect of the project. Unlike ArUco-based detection, markerless objects have no predefined reference pattern and are highly sensitive to texture, lighting, and object geometry.

Why This Became Our Focus

- Markerless pose estimation posed the highest risk of failure.
- Early progress here was critical to prevent downstream bottlenecks.
- We began by evaluating which objects were suitable based on texture richness, shape consistency, and lighting robustness.

Approach 1 — Keypoint Based Pose Estimation

What Is Keypoint Detection?

- A computer vision technique that identifies distinct, repeatable points on an object (e.g., corners, edges, textured regions).
- These keypoints are then described using feature descriptors (e.g., ORB, SIFT) and matched across images.
- Once we match 2D image keypoints to known 3D keypoints, we can compute the object's 6-DoF pose using PnP (Perspective-n-Point).

Why Keypoint Detection Suits Our Project

- Works well for markerless objects that have enough texture or geometric features.
- Lightweight algorithms like ORB run efficiently on Raspberry Pi.
- Provides both object detection and pose estimation without needing markers.
- Robust to partial occlusion and changes in orientation.
- Allows us to build a feature-based 3D model of the object for accurate pose recovery.

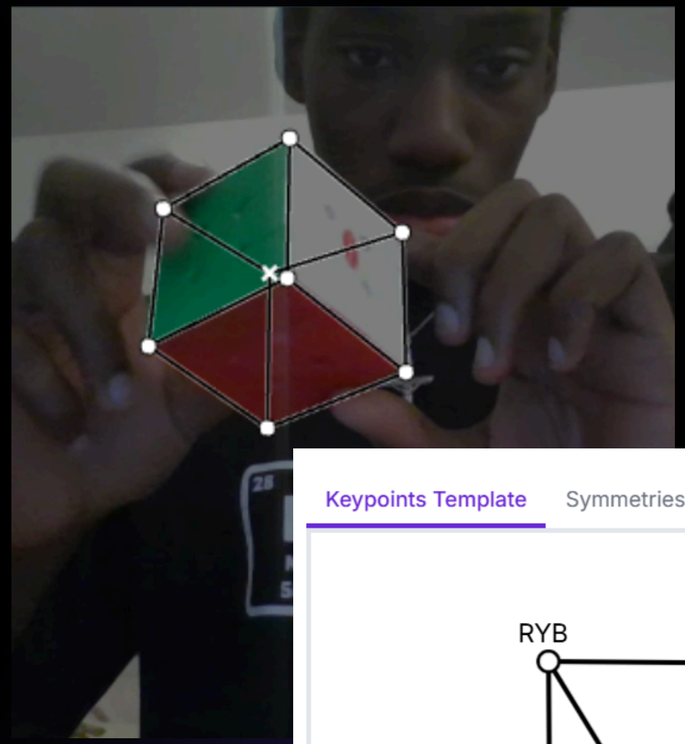
Keypoint Detection: Experimental Setup

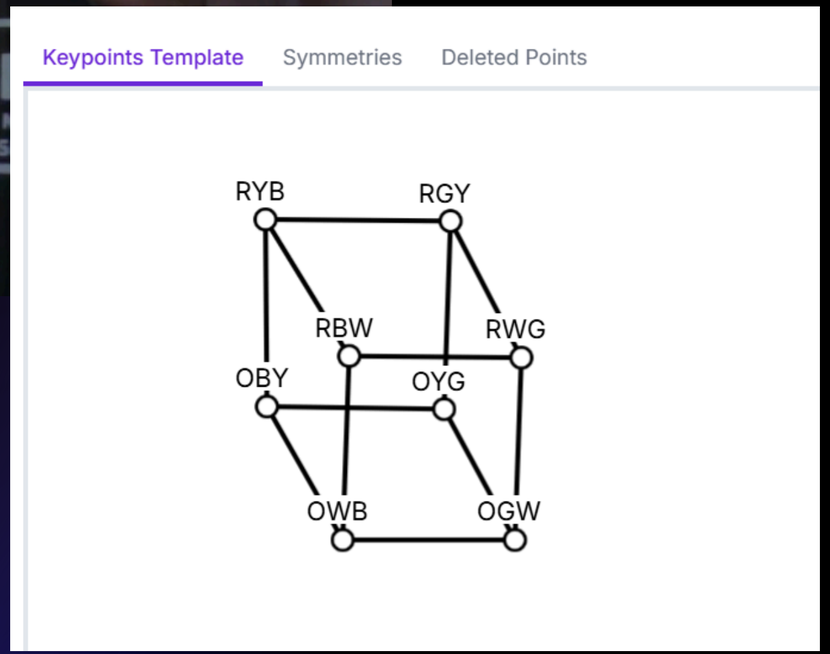
Object Chosen: Rubik's Cube

- Chosen because it has a strong geometric pattern and distinct features that should theoretically produce stable keypoints.
- Despite being a good candidate, we found the method still failed.

Dataset Preparation:

- Manually annotated the Rubik's cube using RoboFlow.
- Performed augmentation to expand dataset to 120 training images.
- Manual annotation was extremely time-consuming, especially for fine keypoint definition.





Keypoint Detection: Training Results

Action

We trained the keypoint model on our Rubik's Cube dataset, but both detection and pose estimation performed poorly.

Detection Outcome

- The model frequently misidentified unrelated objects as the Rubik's cube.
- Bounding box predictions were inconsistent and unstable across test images.
- The model demonstrated clear signs of overfitting to the small dataset.

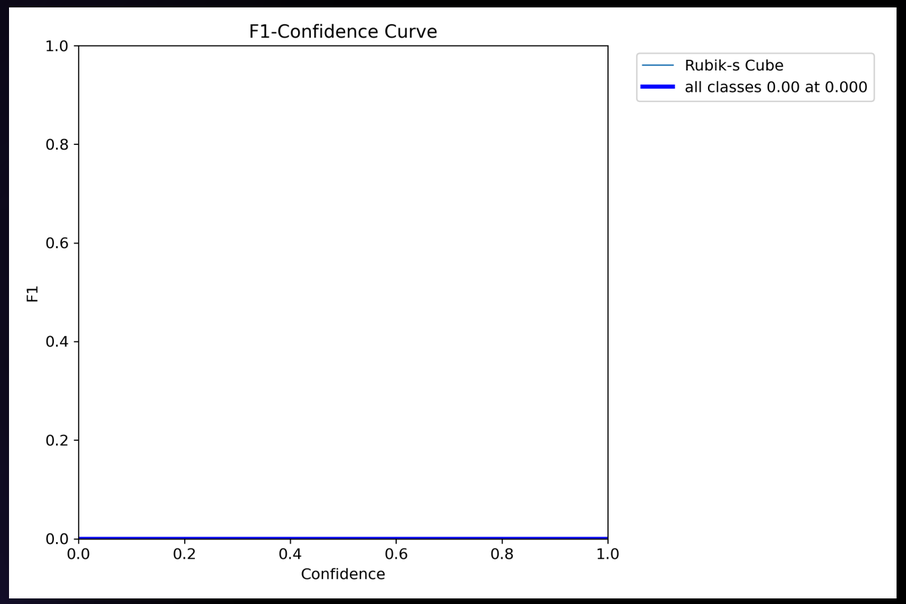
Keypoint/Pose Outcome

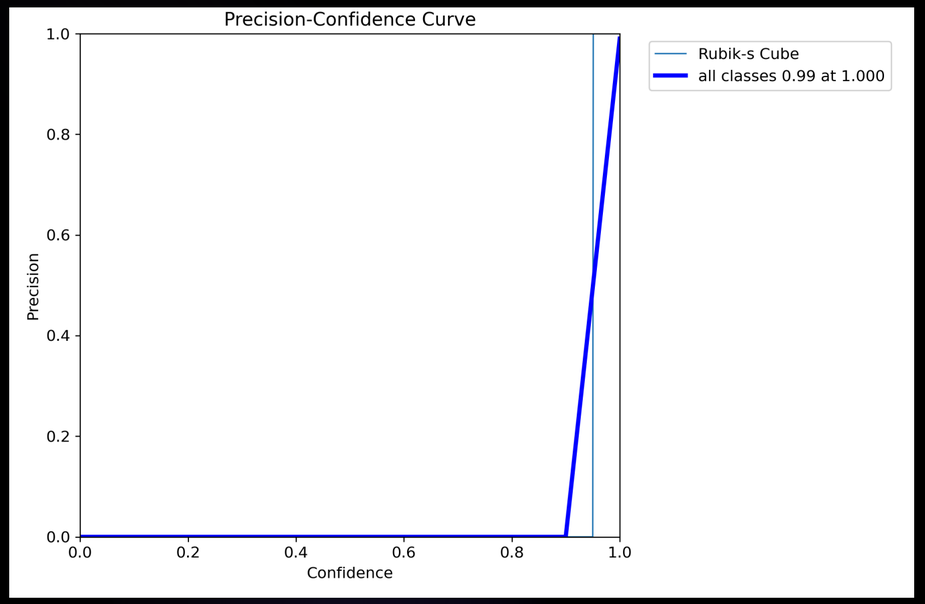
- The pose/keypoint branch failed to learn meaningful geometric patterns.
- F1, precision, and recall scores for pose estimation were near-zero across all thresholds.
- Even in cases where the cube was detected, the predicted pose was completely inaccurate.

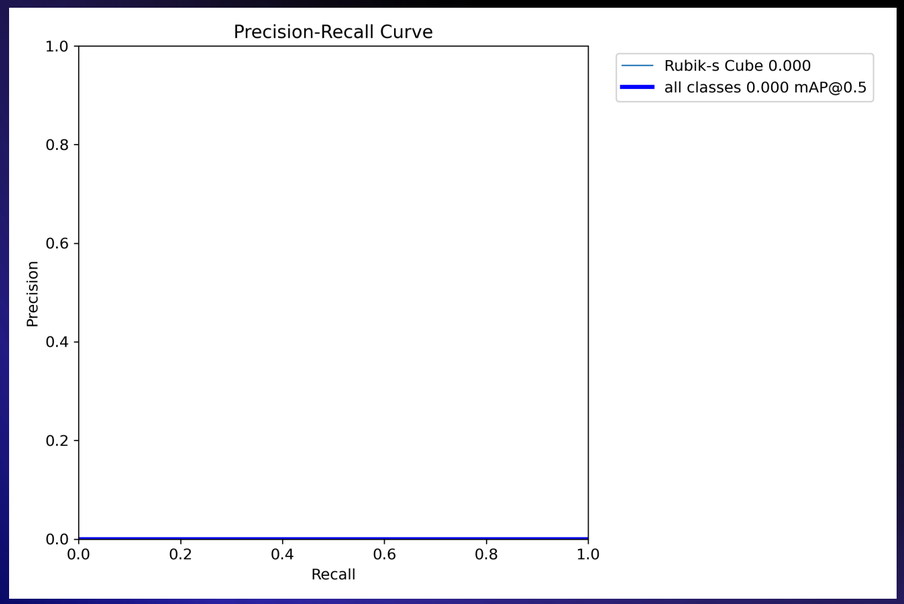
Why This Happened

- 120 images (even with augmentation) is far too small for pose learning.
- Rubik's cube has complex, repeating patterns, causing confusion.
- Manual annotation errors accumulate quickly in pose estimation tasks.
- YOLO's keypoint head requires highly varied data to learn stable correspondences.

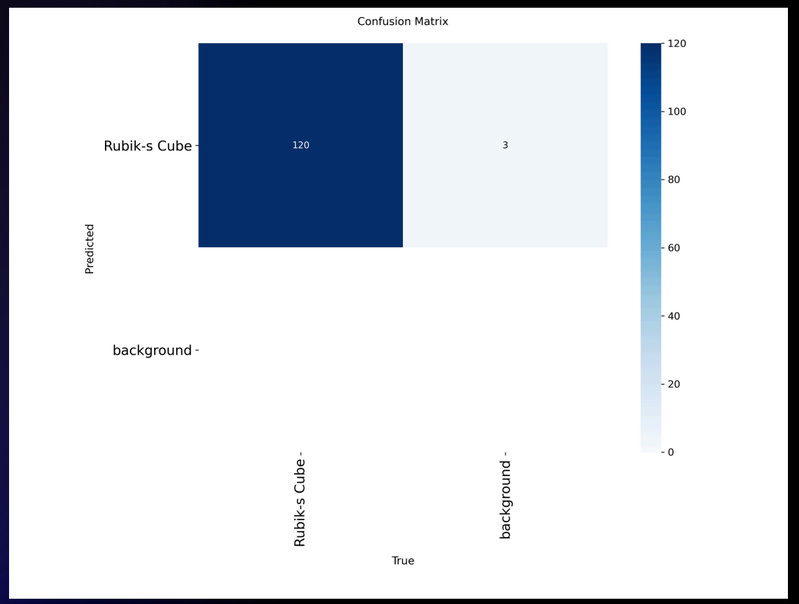
Keypoint Detection: Training Results







Both detection and pose estimation showed poor generalization, with many false positives and near-zero pose accuracy.



Keypoint Detection — Evaluation & Pivot

Overall Evaluation

Despite training the model and running experiments, the overall system performance was not usable:

- Detection was unstable, with many false positives.
- Keypoint/pose estimation produced no reliable results.
- The model struggled to generalize across new test images.

Limitation Identified

To improve these results, we would have needed:

- Significantly more data
- Much more diverse viewpoints
- Extensive manual annotation, which was not feasible within the project timeframe

At this stage, it became clear that continuing with keypoint detection would not be an efficient use of time.

Decision

We decided to pivot to alternative markerless pose estimation approaches that would require less annotation effort and deliver more reliable performance.

Exploring New Approaches

After pivoting from keypoint detection:

We restarted our research to find a simpler, more reliable method for markerless pose estimation — ideally one that doesn't require heavy annotation or large datasets.

What We Found

During this exploration, we came across homography, a geometry-based approach that uses point correspondences on a planar object to estimate its pose.

Why This Was Promising

- No dataset needed
- No manual keypoint labelling
- Works well for planar, structured objects

Camera Calibration

Why Calibration is Essential:

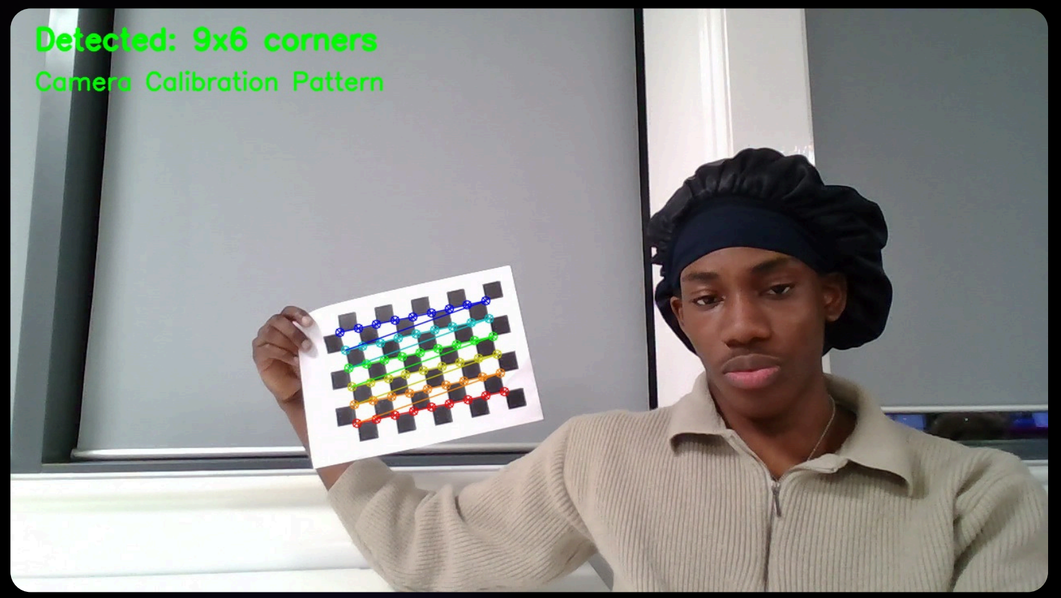
- Homography computation requires accurate camera intrinsic parameters
- Distortion correction ensures reliable feature correspondence
- Enables transformation from 2D image plane to 3D world coordinates

Calibration Process:

- Used checkerboard pattern with known dimensions
- Captured multiple images at various angles and distances
- Computed camera matrix and distortion coefficients using OpenCV

Calibration Parameters Obtained:

- Camera intrinsic matrix (fx, fy, cx, cy)
- Radial and tangential distortion coefficients



Camera Calibration Results

Intrinsic Parameters:

f_x = 888.03 px

f_y = 884.10 px

c_x = 639.41 px

c_y = 354.66 px

Distortion Coefficients:

k_1 = 0.0945

k_2 = -0.2264

p_1 = 0.0005

p_2 = -0.0027

k_3 = 0.2021

Approach 2 — Homography-Based Pose Estimation

What Is Homography?

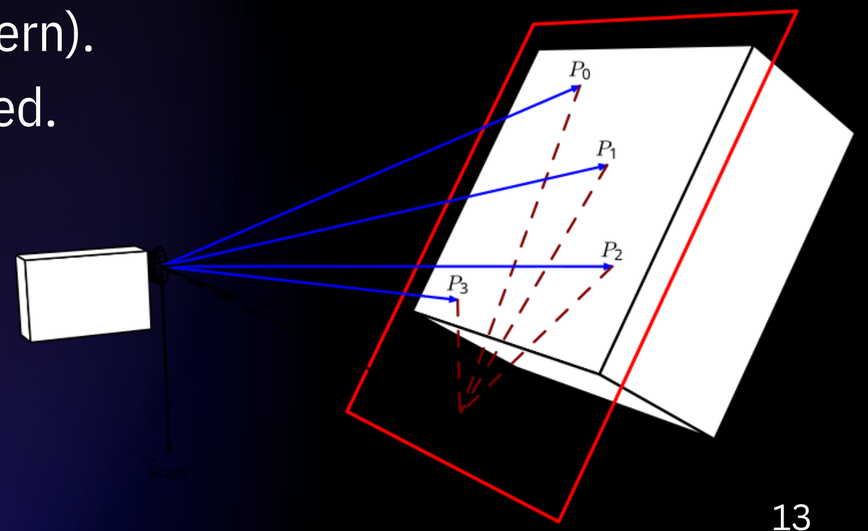
Homography is a geometric transformation that maps one plane to another using four or more point correspondences.

In simple terms:

If an object has a flat, identifiable surface, we can use homography to understand how it's rotated and positioned in the camera view.

Why It Works for Our Project

- Our chosen object has a planar region (e.g., a face with a clear pattern).
- We only need to detect a few corner points — no big dataset required.
- Much easier than training a model for keypoints.
- Provides a stable mathematical way to estimate object pose.
- Runs efficiently on the Raspberry Pi.
- Required no manual annotation
- Allowed faster iteration and testing
- Was realistic to implement within the assessment timeframe



Link to Homography Wiki

Approach 2 – Homography-Based Pose Estimation

With an approach we were finally confident in, we moved on to properly attempting the assessment requirements:

Object Selection

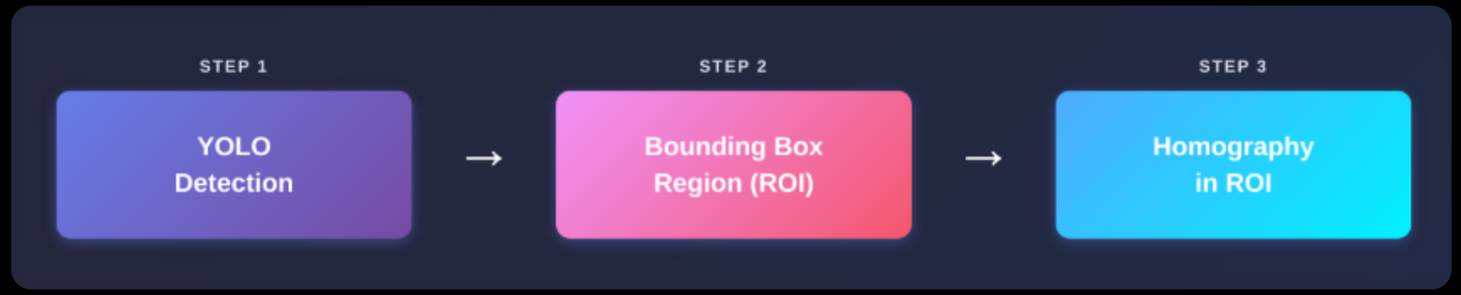
- ArUco objects: headset, wallet
- Marker-less objects: repair mat, notebook, game box

Detection Model Training

YOLO detection model trained with:

~2,800 images per class

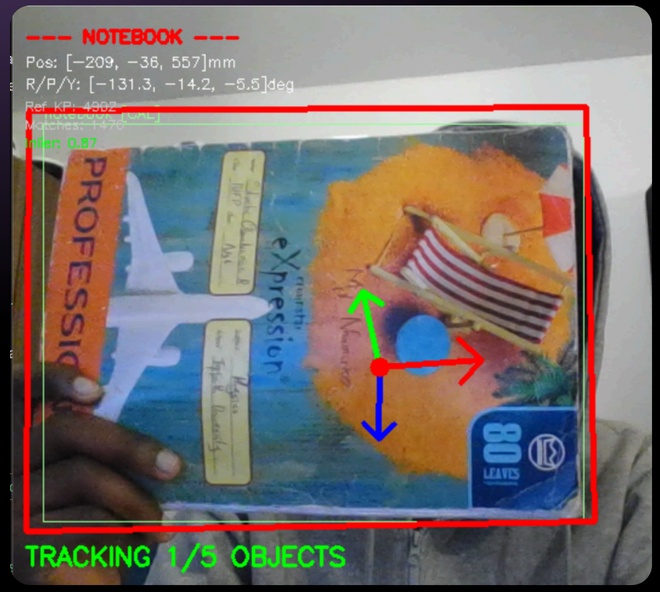
~14,000 images in total

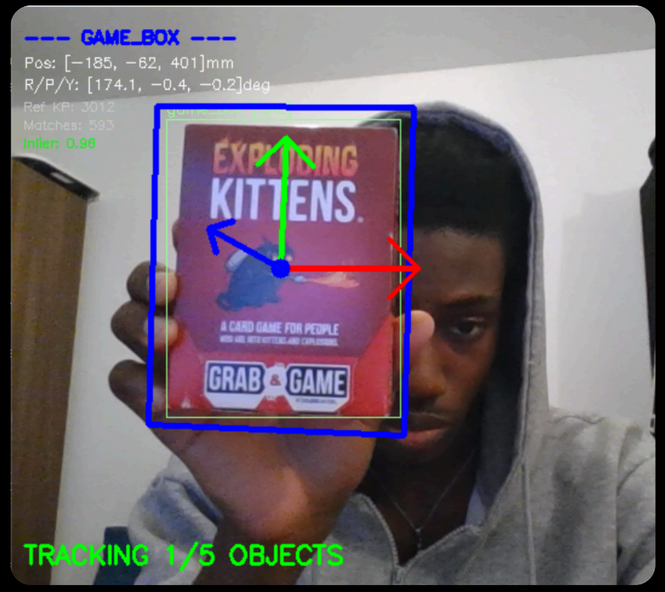


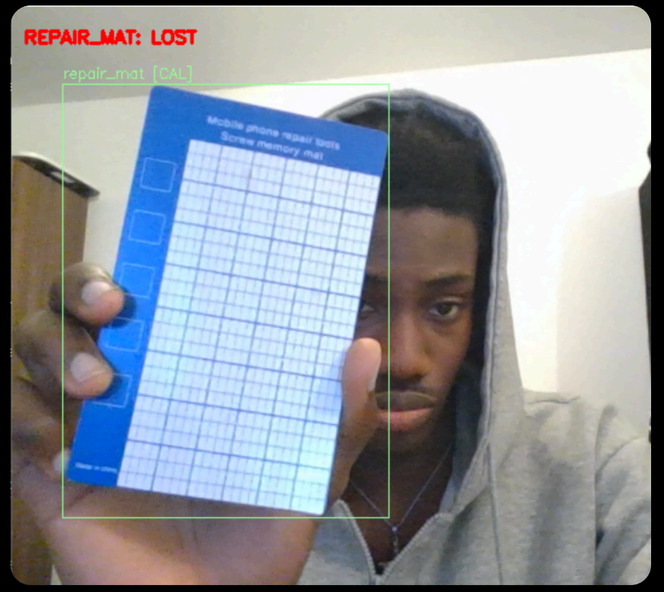
Pipeline Implementation

- YOLO detection identifies objects and defines bounding box regions
- Homography computed within detected regions for pose estimation

Approach 2 — Homography Results – First Iteration







Detection Performance:

- All 5 objects detected successfully by YOLO

Markerless Objects:

- Notebook: Homography computed successfully, pose estimation functional
- Game Box: Homography computed successfully, pose estimation functional
- Repair Mat: Homography failed - insufficient texture density for reliable feature correspondence

Understanding Homography Requirements

Why Did Repair Mat Fail?

Although the repair mat appeared suitable (planar surface with grid pattern), homography failed due to insufficient texture contrast.

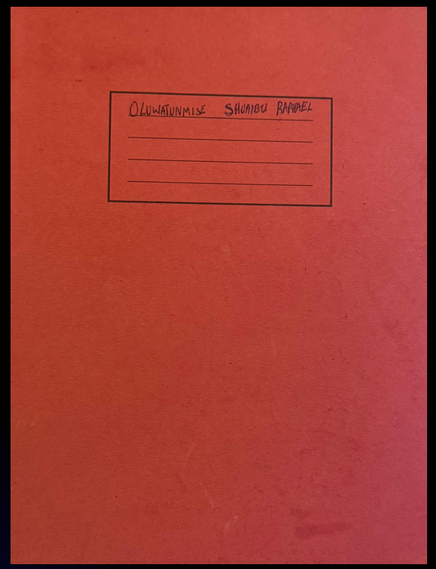
What is "Texture" in Computer Vision?

- Distinctive, high-contrast features that can be reliably detected and matched across images
- Variation in intensity, color, or patterns at multiple scales
- Not just visible patterns - must have strong gradients for feature detection

Critical Requirements for Homography-Based Pose Estimation:

1. Planar Surface: Object must have a flat, identifiable face
2. Rich Texture: High-contrast features with distinctive patterns
3. Camera Calibration: Accurate intrinsic parameters essential for pose computation

Textured Object



Low Textured Object

System Refinement – Modular Architecture

Since homography showed promising results, further refinement was required to improve stability. To support rapid testing and future improvements, the system was redesigned to be modular, allowing different pose estimation methods to be swapped easily.

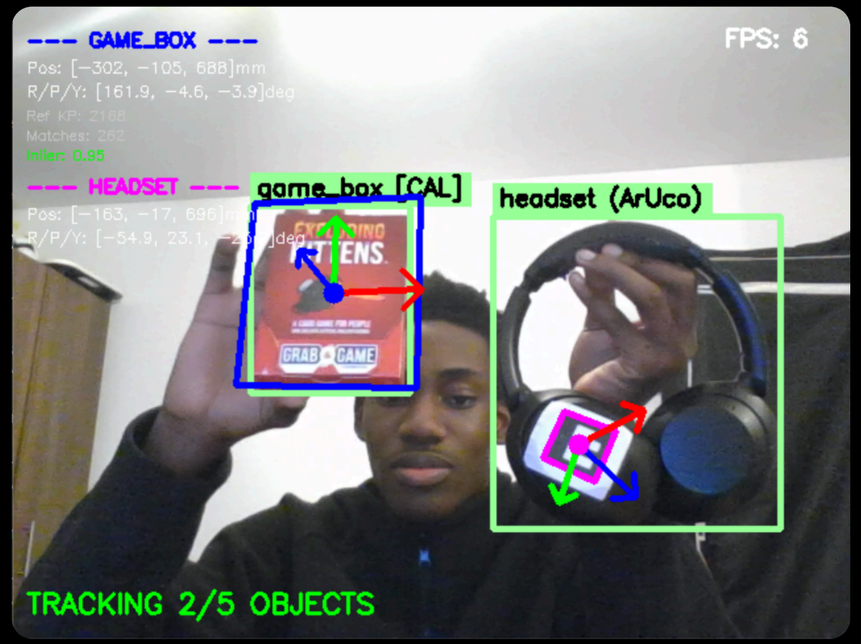
Key Features:

- Centralized object configuration (dimensions, method, thresholds)
- Multi-method support (homography + ArUco in one pipeline)
- Swappable YOLO models with automatic class mapping
- Independent parallel tracking per object

Benefits:

- Rapid iteration and A/B testing
- Simplified debugging and collaboration

System Refinement — Modular System Results



System Performance:

- Both homography and ArUco methods working simultaneously
- Multiple objects tracked in parallel
- Stable pose estimation across different method types

Key Achievement: The modular architecture successfully demonstrated multi-object, multi-method tracking in a unified system, enabling flexible testing and rapid iteration.

Object Refinement – Second Iteration

Based on initial testing results, objects were refined to better suit their respective tracking methods.

Objects Swapped Out:

- Wallet → E-stop button
- Headset → Phone
- Repair mat → Circuit board
- Game box → Card game

Rationale for Changes:

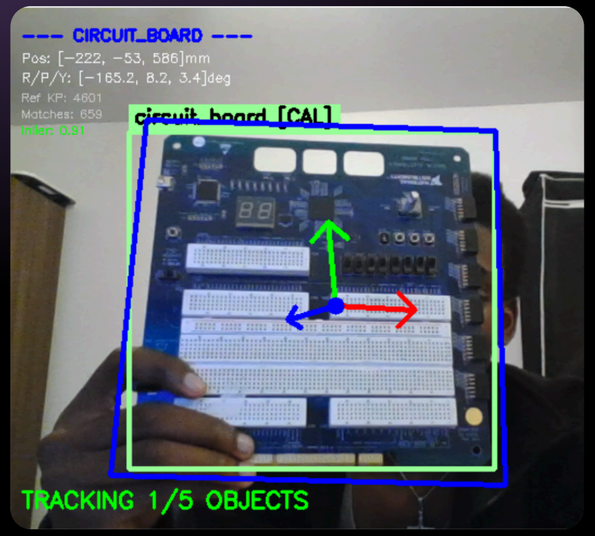
- E-stop and phone: Better ArUco marker placement, more stable surfaces
- Circuit board & card game: Richer texture pattern, improved feature density for homography

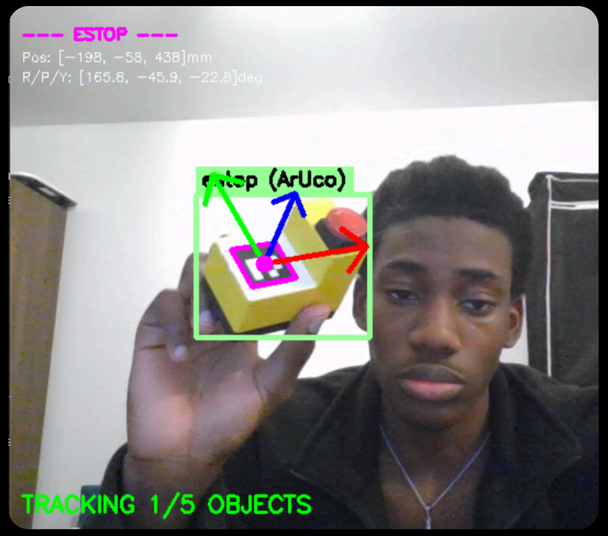
Training Process:

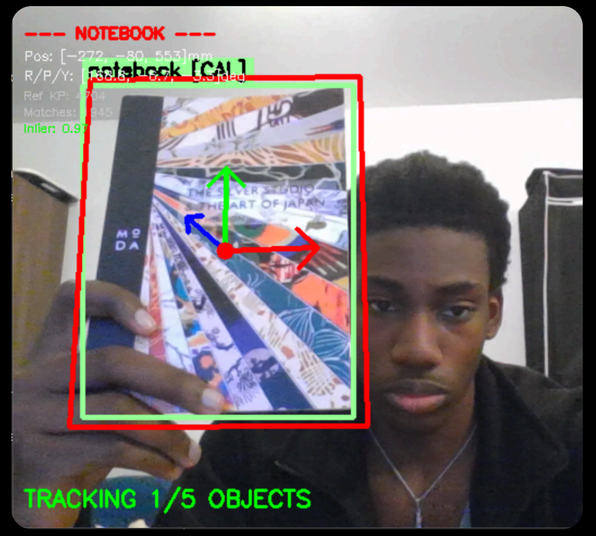
- New YOLO model trained with updated object set
- ~2,800 images per class maintained
- Total dataset: ~14,000 images

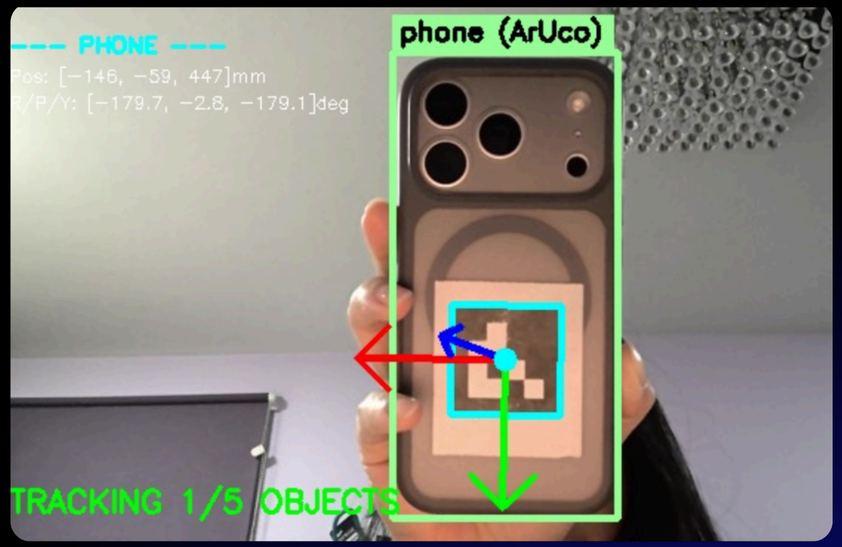
Successfully integrated new objects into modular system with no code restructuring required - demonstrating the benefit of the modular architecture.

Object Refinement – Second Iteration Results









System Validation: All 5 objects successfully tracked with 6DOF pose estimation. New object selection increased feature detection rates and tracking reliability.

Next Step: With the system demonstrating promising results, focus shifted to ROS2 integration for embedded deployment on Raspberry Pi.

Raspberry Pi Deployment

Transition to Embedded System: Code transferred to Raspberry Pi 4 for real-time embedded operation using Git for version control and consistent iteration across development environments.

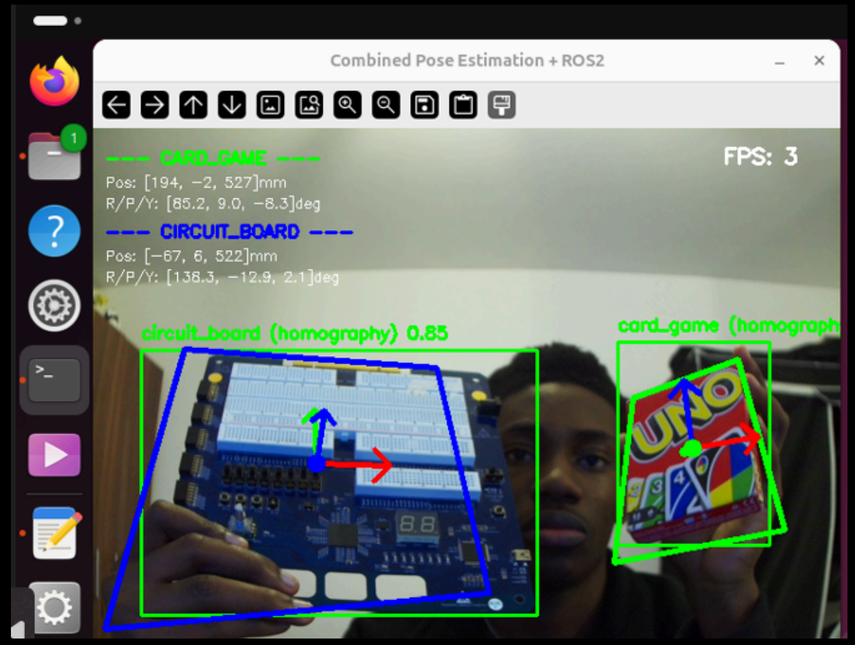
Camera Pipeline Adaptation:

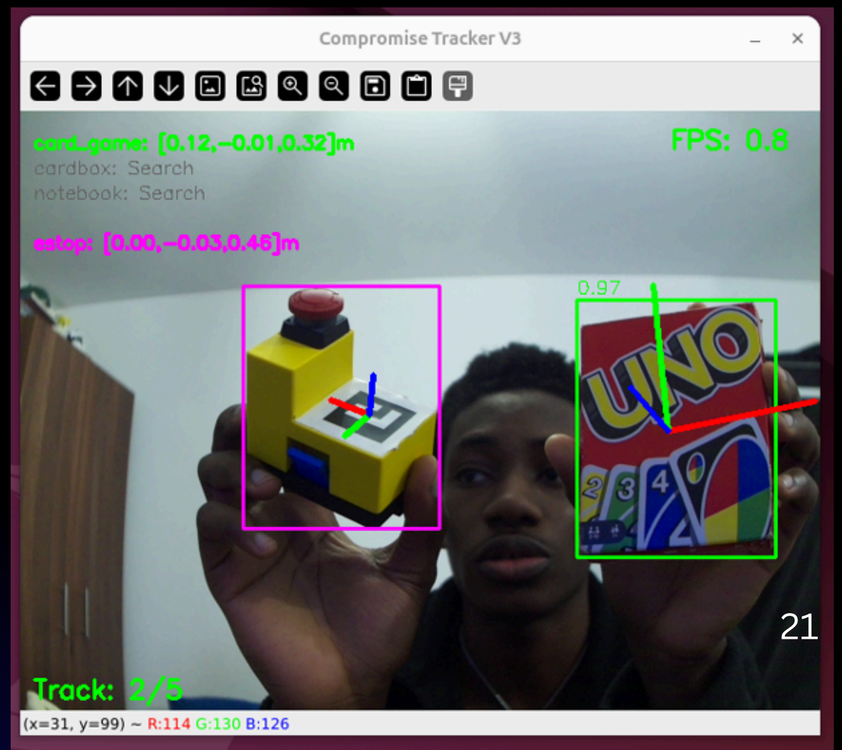
- Standard OpenCV VideoCapture → GStreamer pipeline with libcamera
- Raspberry Pi CSI Camera integration
- Configuration: 640×480

System Optimizations:

- Adaptive frame skipping for YOLO (1-3 frames based on detection state)
- Optimized ORB feature computation pipeline
- Memory-efficient buffering for real-time performance

Performance: Multi-object tracking successfully maintained on embedded hardware, though processing limitations resulted in reduced frame rates. Pose estimation axes exhibited jitter due to frame-to-frame variations in feature matching and homography computation.





ROS2 Integration & TF Publishing

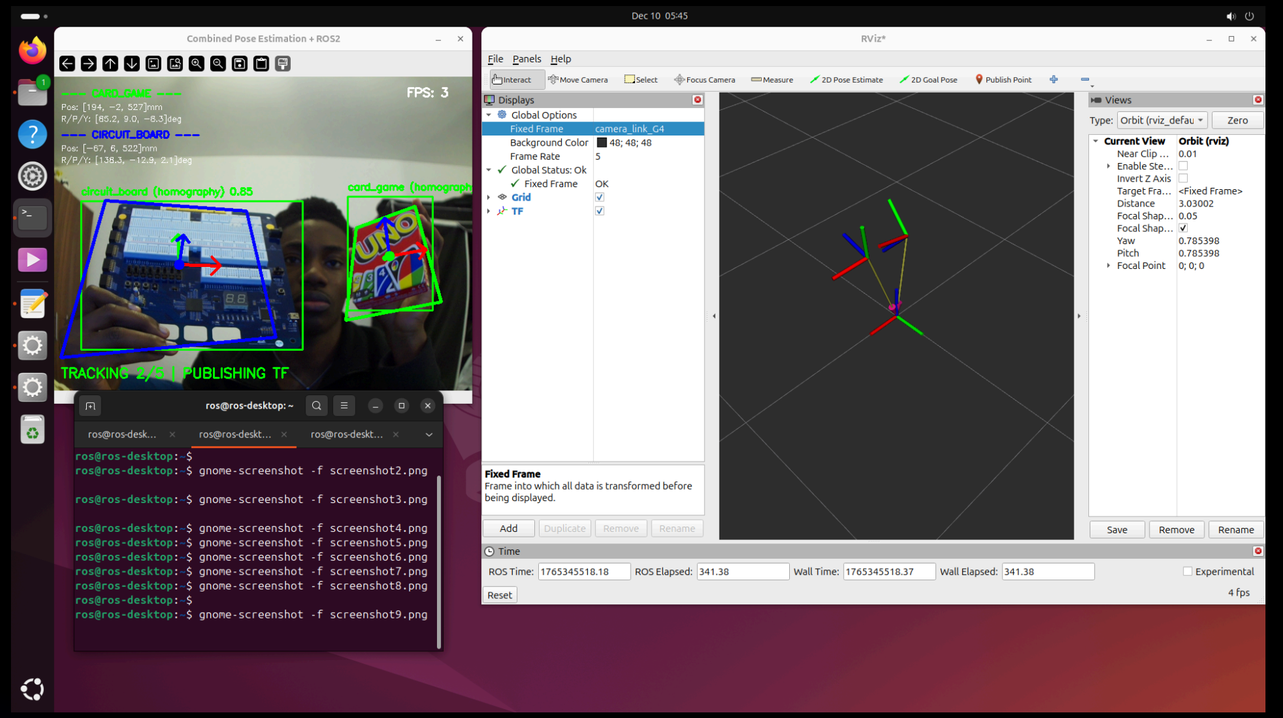
ROS2 Framework Integration: System extended to publish pose data as TF frames, enabling integration with the broader robotics ecosystem.

TF Publishing Architecture:

- Static transform: map → camera_link_G4 (fixed reference frame)
- Dynamic transforms: camera_link_G4 → {object}_frame for each tracked object
- Real-time broadcasting at detection frequency

Published Information:

- 6DOF pose (position + orientation) for all 5 objects
- Position in meters (converted from mm)
- Orientation as quaternions (converted from rotation vectors)



System Performance & Challenges

Current System Status: Multi-object tracking operational with ROS2 TF publishing successfully implemented across all 5 objects.

Performance Issues:

- Low frame rates on Raspberry Pi (~1-3 FPS)
- Jittery pose estimation axes due to frame-to-frame feature matching variations
- Intermittent homography failures on circuit board tracking

Design Decision: Frame rate optimization deferred in favor of prioritizing system stability and reliability. Focus shifted to improving pose estimation robustness.

Solution Approach: Research into alternative feature-based methods revealed a hybrid tracking

Technique combining:

- Direct feature-to-3D plane mapping (instead of homography transformation)
- Temporal smoothing using previous pose as initial guess for PnP
- RANSAC-based outlier rejection with stricter inlier requirements

This approach showed promise for more stable pose estimation with reduced jitter.

ORB Feature Detection with Direct 3D Mapping vs Homography

Previous Approach (Homography):

- 2D feature matching → 2D homography → corner transformation → PnP for pose

New Approach (Direct 3D Mapping):

- Features mapped directly to 3D plane during calibration → PnP on 3D-to-2D matches
- Previous pose used as initial guess for temporal smoothing

Key Improvement: Eliminates intermediate homography step, reducing geometric errors and jitter.

Result: More stable pose estimation with improved tracking consistency.

This makes our 3rd approach to fulfilling the assessment

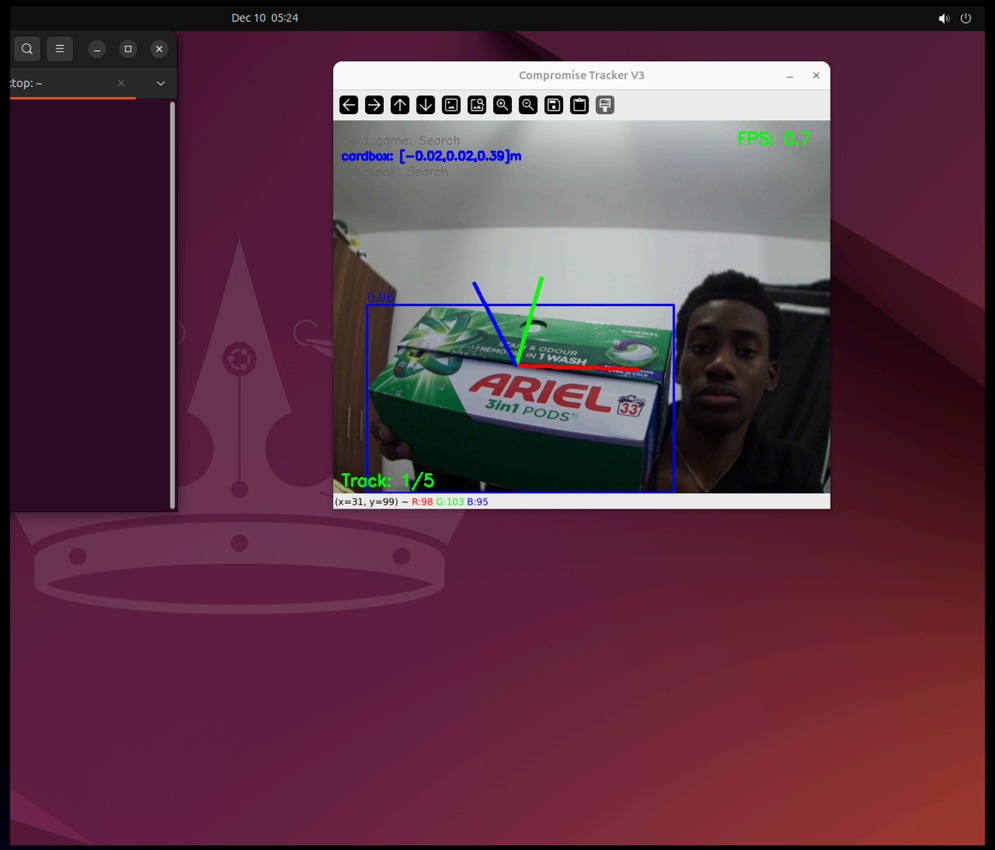
Object Suitability Testing & Final Selection

Automated Testing Implementation: Introduced suitability testing script to systematically evaluate objects before full training cycle, reducing iteration time and ensuring optimal object selection.

Testing Results:

- Notebook: Excellent feature density and stability
- Card game: Strong performance with consistent tracking
- Circuit board: Insufficient texture contrast, intermittent failures

Object Replacement: Circuit board replaced with green cardbox, selected for superior texture characteristics and improved feature detection rates.



Final System Training & Deployment

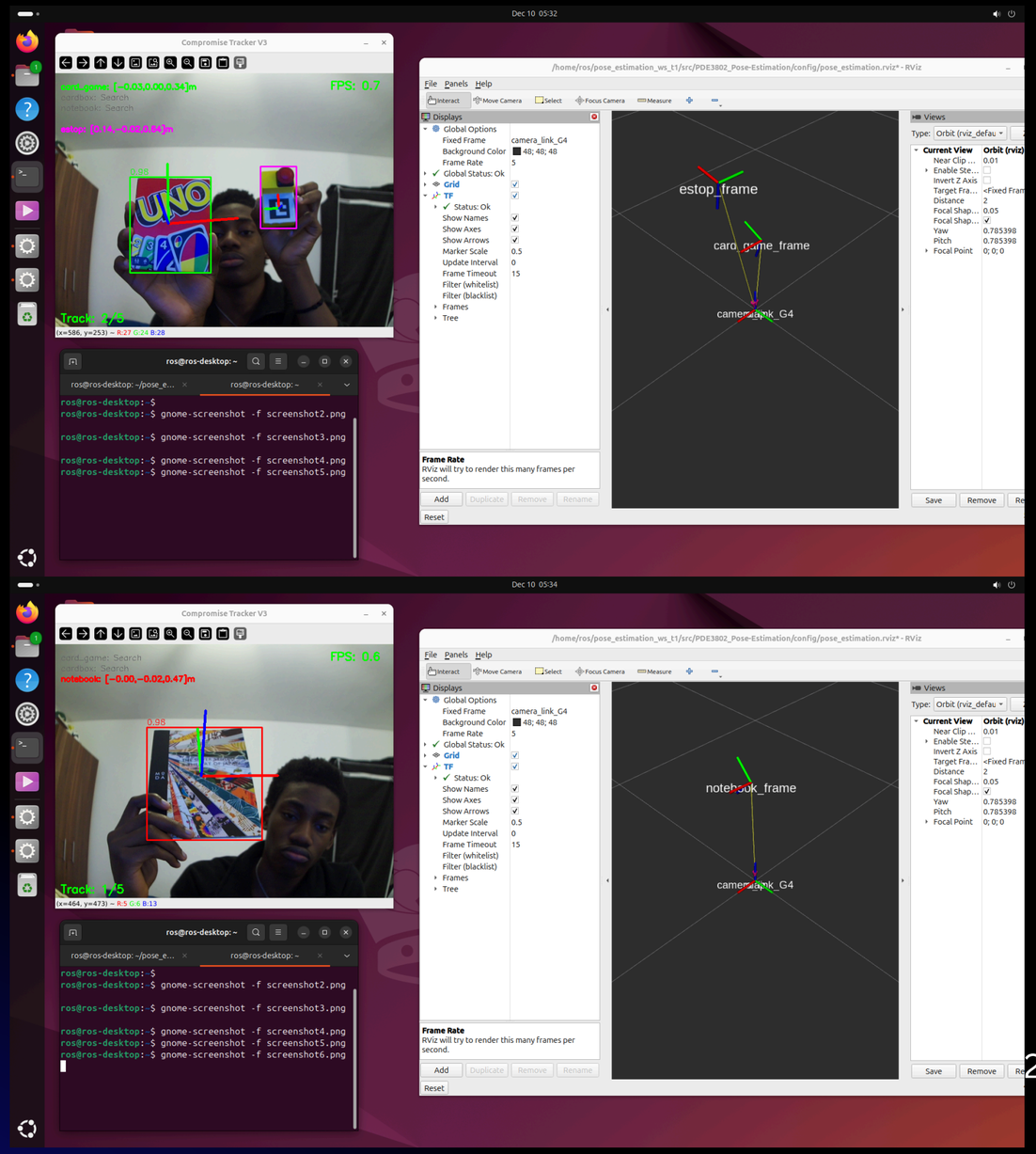
Model Retraining:

- Detection model retrained with updated object set
- Training dataset: ~22,900 images (~4,580 per class)
- Full pipeline deployed and tested on Raspberry Pi

Results:

- Improved pose stability for feature-based objects
- Consistent TF frame visualization in RViz
- Minimal pose estimation jitter achieved
- Detection and tracking operating reliably

Remaining Challenge: Low frame rate performance requiring optimization.



Performance Optimization Attempts

Bottleneck Identification: YOLO inference identified as primary bottleneck (150-300ms per frame on RPi CPU)

Optimization Strategies Attempted: Multiple optimization techniques tested including:

- Multi-threading architecture
- Model format conversion (ONNX, TFLite, NCNN) [ONNX Successful]
- Camera resolution reduction
- YOLOv5 alternative model trial
- ORB feature count reduction
- RANSAC iteration optimization
- Adaptive frame skipping (SUCCESSFUL)

Critical Finding: YOLO is both the bottleneck AND essential—tracking cannot function without accurate detections.

Solution Implemented: Adaptive frame skipping reduces YOLO frequency when objects are stable, caching detections between frames while maintaining accuracy.

Result: FPS improved from 0.4 → 1.8-2.0 FPS through intelligent detection caching without compromising tracking reliability.

Final System Performance & Conclusions

Current Performance:

- System operates reliably at 1.8-2.0 FPS with adaptive frame skipping
- All 5 objects tracked with stable 6DOF pose estimation
- ROS2 TF publishing functional
- Pose estimation jitter minimized through direct 3D mapping approach

Hardware Limitation: Raspberry Pi 4 CPU reaches practical limits for real-time YOLO inference—further optimization yields diminishing returns.

Future Improvements Require:

- Hardware acceleration (Coral TPU, Jetson Nano)
- C++ implementation with optimized libraries
- Alternative lightweight detection architectures

Achievement: Fully functional multi-object 6DOF pose estimation system with ROS2 integration. System demonstrates effective optimization within embedded hardware constraints, achieving 5× FPS improvement through adaptive processing strategies

GROUP 2

Thank You

10/DEC/2025