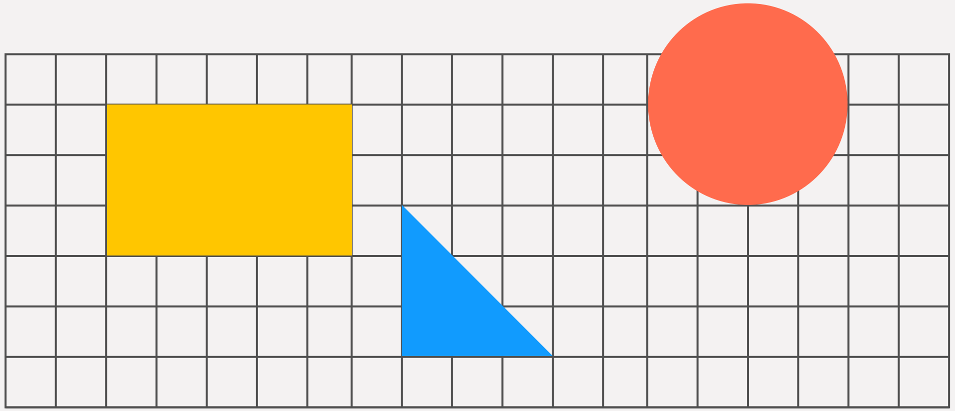
Slotbot: Pick and Place Shapes



Kyle Poulson, Keven Diaz, Akeem Mohammed, and Alan Mascorro

Project Overview

Perception:

- The system scans a camera feed to lock onto the active workspace using corner ArUco markers.
- It then identifies shapes and their orientations via specific ID tags and locates their wooden grasp handles using color thresholding

Decision (Path Planning):

- The software converts 2D pixel data into 3D robot-relative millimeter coordinates using real-time scaling
- It logically pairs each detected “Pickup” shape with its corresponding “Drop” slot based on ArUco ID matching or geometric elimination

Action:

- The system feeds the target coordinates *(X, Y, Z)* into an inverse Kinematics solver to calculate required joint angles and execute the move.

Vision Pipeline Overview

Goal

- Detects **ArUco markers** to define a workspace and also what shape belongs in what tray slot
- Identifies and classifies **geometric shapes** inside that region
- Provides **accurate coordinates** for robotic pick-and-place operations

Method

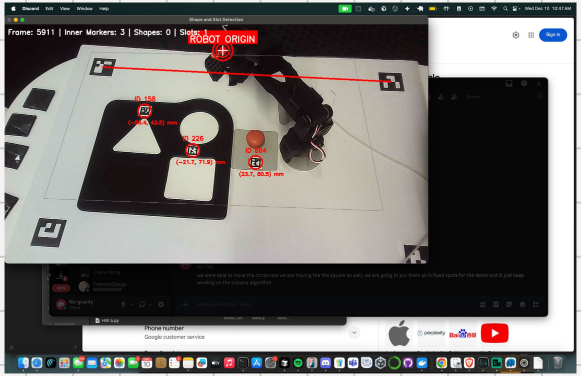
- **Boundary setup:** Four corner ArUco markers (IDs 0–3) form the workspace
- **Shape detection:** Adaptive thresholding + contour approximation for triangles, squares, polygons
- **Coordinate mapping:** Pixel-to-millimeter calibration using reference markers
- **Optimization:** Caching and temporal smoothing for 25–30 FPS real-time performance

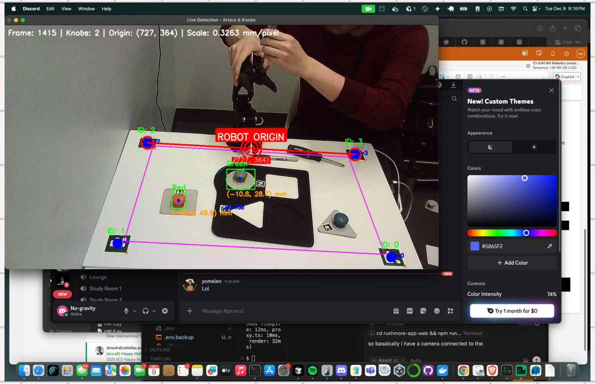
Validation/Results

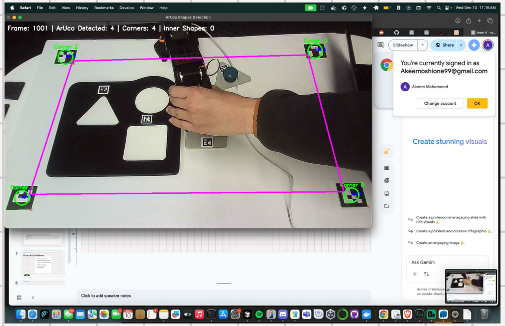
- ✅ **Corner markers (ID 0–3)** detected consistently
- ✅ **Shapes** (Triangle, Square, Pentagon, etc.) correctly classified
- ✅ **Inner ArUco markers** (e.g., ID 987, 190) accurately identified
- ✅ **Real-time performance** confirmed during live testing
- ✅ **Workspace boundary** precisely defined and stable

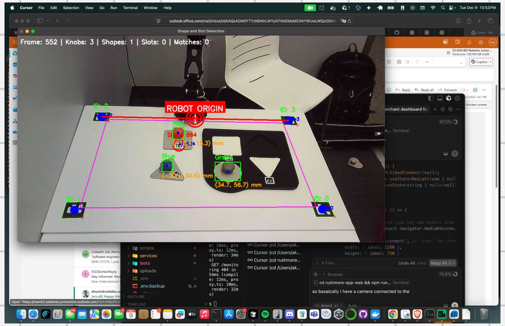
Vision Algorithms

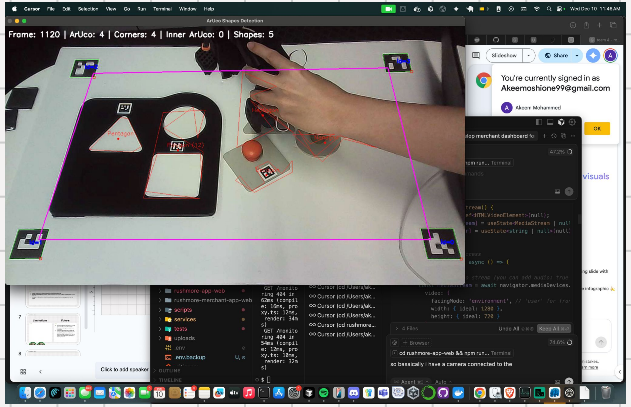
- **Harris Corner Detection** – Locates marker corners by analyzing local intensity gradients.
- **Adaptive Thresholding** – Dynamically separates foreground (shapes/markers) from background under variable lighting.
- **Contour Detection (cv2.findContours)** – Extracts shape outlines from binary masks.
- **Aspect Ratio & Vertex Count Classification** – Classifies shapes (triangle, square, pentagon, etc.) based on geometry
- **Ray Casting Algorithm** – Determines if detected shapes are inside the defined workspace region.
- **Pixel-to-Millimeter Conversion** – Converts image-space coordinates to physical robot coordinates using calibration.
- **Temporal Smoothing & Statistical Filtering** – Stabilizes detections across frames and removes outliers.

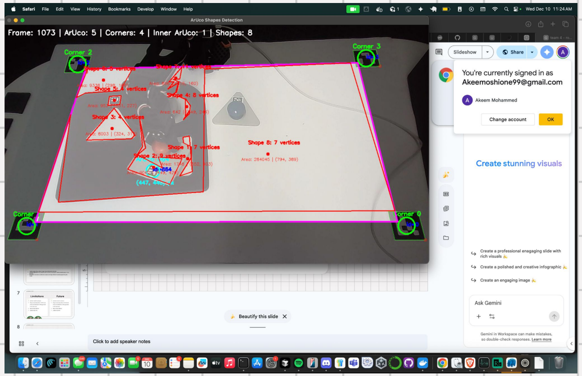

Shape and Slot Detection
Frame: 5911 | Inner Markers: 3 | Shapes: 0 | Slots: 1
ROBOT ORIGIN
ID 158
ID 226
ID 664
(−21.7, 71.9) mm
(23.7, 80.5) mm


Live Detection - ArUco & Knobs
Frame: 1415 | Knobs: 2 | Origin: (727, 364) | Scale: 0.3263 mm/pixel
ROBOT ORIGIN
(−10.8, 28.7) mm
New! Custom Themes
Try 1 month for $0

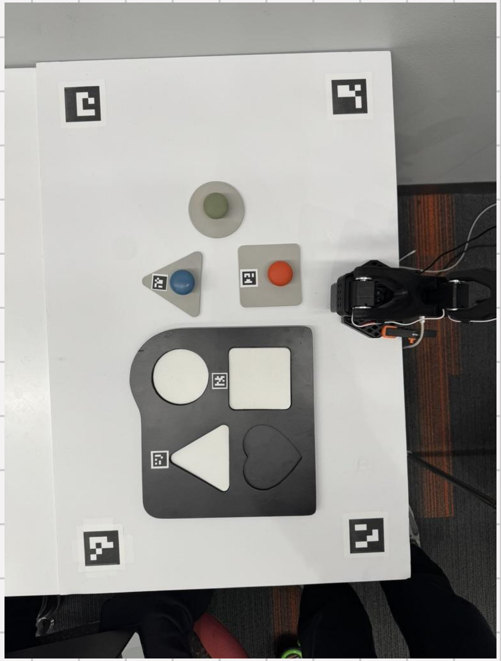

ArUco Shapes Detection
Frame: 1001 | ArUco Detected: 4 | Corners: 4 | Inner Shapes: 0
Corner 2
Corner 3
Create stunning visuals


Shape and Slot Detection
Frame: 552 | Knobs: 3 | Shapes: 1 | Slots: 0 | Matches: 0
ROBOT ORIGIN
ID: 3
(34.7, 56.7) mm


ArUco Shapes Detection
Frame: 1120 | ArUco: 4 | Corners: 4 | Inner ArUco: 0 | Shapes: 5
Pentagon
You're currently signed in as Akeemoshione99@gmail.com


ArUco Shapes Detection
Frame: 1073 | ArUco: 5 | Corners: 4 | Inner ArUco: 1 | Shapes: 8
Corner 2
Corner 3
Shape 8: 7 vertices
Shape 4: 8 vertices
Shape 3: 4 vertices
Shape 1: 7 vertices
Create stunning visuals

Environment Calibration



Why ArUco markers?

- We didn't want to hardcode specific pixel coordinates because if the camera or table got bumped even slightly, the robot would miss every grasp. Instead, we used four corner anchors (IDs 0-3) to define a dynamic workspace.

How the calibration works.

- **Defining the Boundary**: We implemented a masking algorithm that draws a quadrilateral connecting the four corner markers. Our vision systems creates a binary mask from this shape, effectively ignoring all background noise (like cables or hands) outside the play area.
- **Pixel-to-Real-World Scaling:** We automated the unit conversion by calculating a real-time *mm_per_pixel* factor. The system measures pixel distance between our anchor markers and compares it to the known physical distance, allowing us to translate camera pixels into precise millimeter targets for the arm

Robot Control and Kinematics

Simplifying the Motion: Although we are using a 6-motor arm, we treated the movement logic like a SCARA robot. We calculate the Planar Reach (*X, Y*) separately from the Orientation

The Math:

- To reach a specific point, we calculate the Shoulder Pan using simple trigonometry.
- We then use the Law of Cosines to determine the shoulder lift, extending the arm to the correct distance based on the lengths of our upper arm and forearm.

Orientation Logic: For the shapes that need rotation (square and triangle), we calculate the angular difference between the pickup ArUco marker and the slot ArUco marker, sending that value directly to the wrist roll servo.

Limitations

Vision:

- Lighting sensitivity: HSV color thresholding to detect the orange, blue, and green knobs.

Mechanical:

- The robot operates blindly once it starts moving. It calculates a path and executes it without feedback. If the piece slips out of the gripper, or if the robot bumps into something, the system has no way of knowing and will continue its motion as if nothing happened.

Future

- Fine tune the inverse kinematics to move more smoothly to avoid accidental throwing or moving other elements on the board.
- Move to machine learning (YOLO). The current system is heavily reliant on color and lighting. Training a small object detection model would allow the robot to identify the colored knobs regardless of lighting conditions.

DEMO