ROB 550 Armlab Report

Yilin Ma, Sanyam Mehta, Xinyue Xu
yilinma, sanyam, xxinyue@umich.edu

***Abstract*—This paper presents an autonomous robotic system for object manipulation and basketball launching, utilizing the RX200 robotic arm and Intel RealSense L515 camera. The system integrates precise perception through intrinsic and extrinsic camera calibration, advanced kinematics for manipulation, and efficient path planning. Camera calibration combines chessboard-based intrinsic and PnP-RANSAC extrinsic methods for accurate object detection. The human-robot interface streamlines collaboration, enhancing task execution. Our system successfully performs object sorting, stacking, and launching tasks, demonstrating its potential for industrial and collaborative applications.**

I. Introduction

Robotic arms now perform complex tasks like surgeries and spacecraft inspections. We present an autonomous system that assists in launching basketballs by loading them into a launcher. The system handles object manipulation and perception without direct human intervention using the RX200 robotic arm and Intel RealSense L515 RGB-D camera.

Our system's ability to accurately detect objects is a result of meticulous camera calibration. We employ both intrinsic calibration using a chessboard and extrinsic calibration using AprilTag detection and the Perspective-n-Point algorithm. Furthermore, we utilize color-based detection with filters in the HSV space to minimize noise and false alarms.

Manipulation relies on the Denavit-Hartenberg method to determine kinematics, enhanced by numerical inverse kinematics and decoupling modeling for precise movement. Advanced path planning ensures efficient operation, including obstacle avoidance and steady control.

One of the key design considerations of our system is the human-robot interface. This interface, a direct result of our work on camera calibration and manipulation, enables seamless collaboration between the human operator and the robotic arm. By accurately calibrating the camera and refining manipulation techniques, we have created a system where the human operator can focus on aiming the launcher, while the robotic arm handles the loading of basketballs. This collaboration not only improves efficiency but also reduces task complexity by combining human intuition with robotic precision and speed.

Our system successfully sorts and stacks wooden blocks of various shapes and sizes and loads basketballs into a launcher to aid in their trajectory. While effective, limitations occur when the gripper isn't orthogonal to the ground plane, resulting in unattainable positions. Addressing these challenges advances autonomous robotic manipulation and enhances human-robot collaboration across industries.

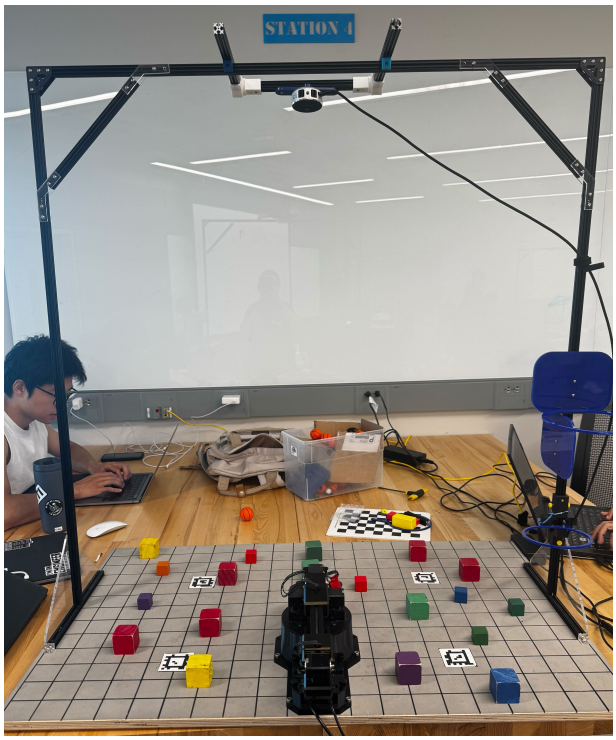


Fig. 1: The workbench is set up with the RealSense L515 positioned above and the RX200 robotic arm mounted at the front lower section.

II. Perception

1) Camera Calibration: Camera calibration estimates the parameters of a camera's lens, image sensor, and its pose relative to the real world. It accurately models how the camera captures 3D scenes and projects them onto the 2D image plane through perspective projection, relying on two independent parameter sets:

- **Intrinsic Parameters:** These constant, camera-specific parameters depend on factors like lens material, refractive properties, focal length, distortion,

and the optical center. The intrinsic matrix K can be represented as:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where f_x and f_y are the focal lengths in pixels, (c_x, c_y) is the optical center, and s is the skew coefficient (usually 0 for most cameras). This matrix can be obtained from the manufacturer or through manual calibration using algorithms like Zhang's [1] or Tsai's [2] methods. Factory intrinsic calibration is generally more reliable due to its higher accuracy, consistency, and specialized techniques. Manual calibration, however, is prone to errors caused by environmental factors, inconsistent measurement techniques, human errors, equipment quality, and improper application of correction factors, all of which can affect calibration accuracy.

- **Extrinsic Parameters:** These non-static parameters define the orientation and position of the camera with respect to the world coordinate frame. The extrinsic matrix $[R|t]$ transforms points from the world coordinate frame $\mathbf{W_w}$ to the camera coordinate frame $\mathbf{W_c}$:

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where R is the 3x3 rotation matrix and t is the 3x1 translation vector. This matrix cane be obtained by manually measuring the lab apparatus, or using Perspective-n-Point (PnP) transformation using ex- ternal landmarks (e.g., AprilTags). Manual tape measurement is often less accurate and more prone to human error compared to using external landmarks with the PnP algorithm. The latter approach provides superior accuracy and consistency by leveraging robust algorithms to minimize outliers and enhance calibration precision.

The instrinsic and extrinsic matrix is shown in the Table I and Table II.

The complete camera projection matrix P combines both intrinsic and extrinsic parameters:

$$P = K[R|t] \tag{3}$$

For optimal camera calibration results, we have employed a two-pronged approach:

Intrinsic Calibration: We used factory-calibrated intrinsic parameters from the Intel RealSense camera, which proved more accurate and consistent than manual calibration over five iterations for this hardware.

Extrinsic Calibration: To determine the camera's pose relative to the world, we applied OpenCV's 'solvePnPRansac' function, combining the Perspective-n-Point (PnP) algorithm for position and orientation estimation with RANSAC to handle outliers and enhance accuracy.

This approach leverages factory precision for intrinsic calibration and robust extrinsic calibration, ensuring reliability and adaptability across various environments.

2) Forward and Inverse Projection: The calibration process enables two crucial operations:

1) **Forward Projection:** This maps 3D world coordinates to 2D image coordinates using the camera matrix P:

$$\mathbf{x} = P\mathbf{X} \tag{4}$$

where $\mathbf{X}$ is a 3D point and $\mathbf{x}$ is its 2D projection.

2) **Inverse Projection:** This attempts to recover 3D world coordinates from 2D image coordinates. However, it's important to note that this process is inherently ambiguous due to the loss of depth information during projection. The inverse projection is accurate only up to a scale factor, as the same image point could correspond to any 3D point along the incoming light ray.

The general form of the inverse projection is:

TABLE I: The measured intrinsic matrices.

Calibration	Intrinsic Matrix (unit: mm)
Checker Board	$\begin{bmatrix} 906.98 & 0 & 618.85 \\ 0 & 913.31 & 345.16 \\ 0 & 0 & 1 \end{bmatrix}$
Factory	$\begin{bmatrix} 892.88 & 0 & 657.04 \\ 0 & 893.41 & 383.22 \\ 0 & 0 & 1 \end{bmatrix}$

TABLE II: The measured extrinsic matrices. DATA UPDATED

Calibration	Extrinsic Matrix
Manual	$\begin{bmatrix} 1.00 & 0.00 & 0.00 & 0.00 \\ -0.00 & -0.98 & 0.17 & 164.04 \\ -0.00 & -0.17 & -0.98 & 1027.65 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$
PnP Ransac	$\begin{bmatrix} 0.99 & 0.00 & -0.02 & 16.57 \\ 0.01 & -0.98 & 0.18 & 187.39 \\ -0.02 & -0.18 & -0.98 & 1017.10 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$

$$\mathbf{X} = R^T(K^{-1}\mathbf{x} \cdot Depth(u, v) - t) \quad (5)$$

where $Depth(u, v)$ is the depth associated with pixel with coordinates u, v. This can be reliably obtained from sensors such as LiDAR, stereo camera setups, structured light systems, or time-of-flight cameras. By incorporating this depth data, we can determine the correct scale factor and accurately reconstruct 3D points from their 2D projections. We get this value from the stereo depth recovered from the Realsense camera.

3) Perspective Correction using Homography and Validation: Due to the non-parallel alignment of the camera with the workspace plane, parallel lines in the real world appear non-parallel in the captured image, creating a noticeable perspective shift. To correct this distortion and validate our calibration, we employ a two-step process: homography-based perspective correction and projection-based validation.

a) Homography-based Perspective Correction: We utilize the known rectangular arrangement of AprilTags on the workspace (see Appendix C) to compute the homography matrix H. The process involves:

a) Identifying the quadrilateral formed by the AprilTag centers in the image.
b) Calculating the minimum bounding rectangle that this quadrilateral should map to.
c) Computing the homography matrix H using OpenCV's perspective transformation function.

$$H = \begin{pmatrix} 0.99 & -0.10 & 20.12 \\ 0.02 & -1.03 & 854.6 \\ 0.00 & -0.00 & 1.0 \end{pmatrix} \quad (6)$$

The resulting homography is then applied to the entire image using OpenCV's warp transform function, effectively correcting the perspective distortion and restoring the parallelism of workspace lines in the image plane. Figure 2 illustrates the effect of the homography transformation. The corrected image (right) shows restored parallelism of the grid lines, facilitating more accurate measurements and analysis.

b) Validation through Grid Projection: To assess the accuracy of our calibration matrices and homography transformation, we leverage the structured grid pattern of our workspace. This pattern provides an ideal setup for a qualitative analysis of the calibration results. Given the known world coordinates of the grid intersections, we project these points onto the image plane using the forward projection equation (Equation 4). We then verify if these projected points align accurately with the actual grid corners visible in the captured image.

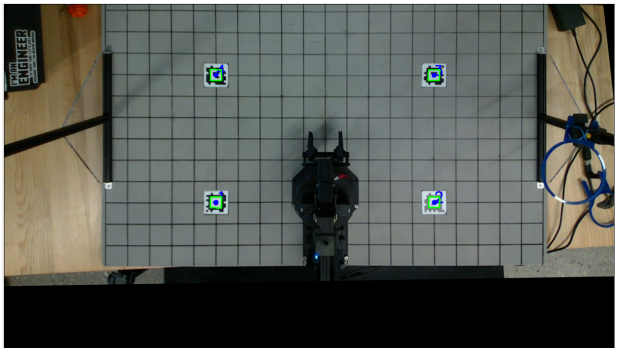
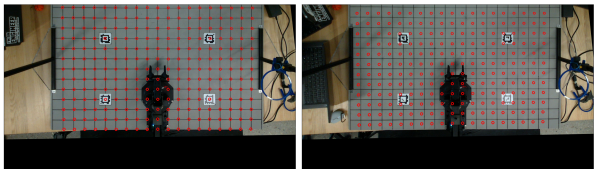


Fig. 2: Perspective correction using homography.



(a) (b)

Fig. 3: (a) PnP Ransac calibrated matrices. (b) Manual calibrated matrices.

Figure 3 compares projections using well-calibrated matrices and manually calibrated matrices. The well-calibrated matrices result in projections that closely align with the actual grid corners, while the manually calibrated matrices show noticeable misalignment. This comparison highlights the superiority of our calibration method over manual techniques, emphasizing the critical role of accurate camera calibration in computer vision. Detailed projection examples and analysis are in Appendix B. The combination of perspective correction and grid projection validation ensures precise calibration and image transformation, forming a reliable basis for further image analysis and feature extraction.

4) Segmenting Objects and Obstacles: For autonomous block manipulation, a crucial step is reliably identifying the location and color of blocks on the workspace while avoiding obstacles. We solve this using object segmentation, classifying each pixel into three categories: *a. blocks*, *b. obstacles/distractors*, and *c. background*

a) Contour Detection and Object Identification: After classification, we employ contour detection

to identify object boundaries. Contour detection extracts continuous curves of (x,y) coordinates outlining distinct regions. We use OpenCV's findContours function on a binary mask, ignoring pixels below 15 mm from the workspace base. Once contours are detected, we use the inverse projection matrix (Equation 5) to obtain real-world coordinates for each block, enabling precise arm manipulation.

b) Color Identification: Color identification is performed in the LAB color space, which is perceptually consistent with human color perception. We use the deltaE* algorithm, a well-researched metric for perceptual color differences in the LAB space. Our approach involves:

- Recording LAB values for different color distributions
- K-Means clustering to identify cluster centers
- Comparing block color means to cluster centers
- Robust identification using multiple samples and maximum voting

This method achieves 100% accuracy in color identification without using neural networks, making it suitable for resource-constrained devices.

c) Distractor Identification: We identify distractors using geometric properties:

- Width-to-length ratio of minimum oriented bounding rectangle
- Area ratio comparison (circular vs. square faces)
- Intersection over union and height-based filtering

d) Block Orientation: Block orientation is determined using the slope of the oriented bounding box, crucial for end effector alignment during manipulation. For detailed algorithms and implementation specifics, refer to Appendix D. This approach provides a reliable block detection system using classical computer vision techniques, avoiding the computational demands of neural networks and enabling deployment on resource-constrained devices.

An image of our block detector is attached in figure 4.

5) Error Calibration and Correction: Figure 5 shows the associated error heatmaps for both x and y directions, visualizing the spatial distribution of errors across the workspace. This calibration and correction process significantly improves the accuracy of our object localization, enabling more precise manipulation tasks.

As discussed earlier, we use the PnP-RANSAC algorithm for camera calibration to project points onto the workspace. While robust, it is not fully error-free. The first type of error, termed *projective transformation error*, arises from inherent limitations in automatic calibration. The second, perspective distortion error, occurs due to the camera's elevated position, causing block side faces to appear larger. To mitigate these issues, we perform error calibration and linear interpolation, generating error heatmaps for x and y directions, and rectifying world coordinates accordingly. The detailed approach is:

a) Calibrating the errors at known points in the workspace.
b) Creating error heatmaps for both x and y directions.
c) Applying linear interpolation to estimate errors at intermediate points.
d) Rectifying the predicted world coordinates for all objects in the scene using these interpolated error values.

Figure 6 illustrates the improvement in block center prediction accuracy after applying our error calibration and correction method.

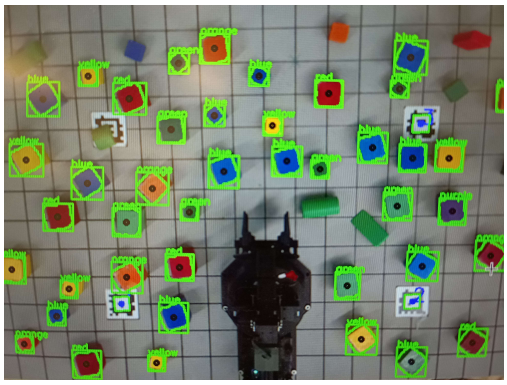
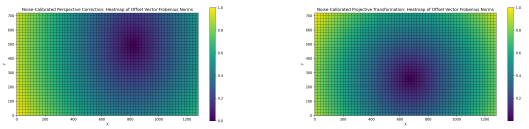


Fig. 4: Block Detector

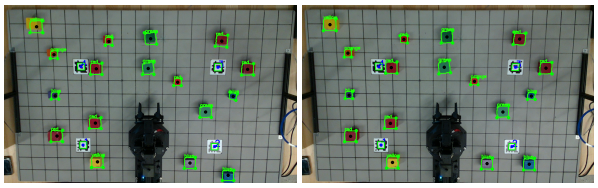


(a) (b)

Fig. 5: (a) Noise Calibrated Perspective Correction. (b) Noise Calibrated Projective Transformation.

III. Control

1) Forward Kinematic: Forward kinematics is the process of determining the pose of the end effector of a robotic arm based on the given joint angles.



(a) (b)

Fig. 6: (a) Without Offset Correction. (b) With Offset Correction.

This process provides the real-time pose of the arm, allowing precise tracking of its movements. To derive the transformations of the robotic arm, the Denavit-Hartenberg (DH) convention is commonly used.

Typically, a general transformation involves six parameters: three for position and three for orientation. The DH convention simplifies these transformations by carefully selecting the frames, links, and joints, and representing the transformation using four parameters: joint angle θ, joint offset d, link length a, and link twist α.

$$A_i = \text{Rot}_{z,\theta_i}\text{Trans}_{z,d_i}\text{Trans}_{x,a_i}\text{Rot}_{x,\alpha_i} \quad (7)$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To apply the DH convention, the first step is to establish a DH table. This requires identifying all the links and joints of the robotic arm. The next step is to assign coordinate frames to each joint, ensuring that all rotations occur about the z axis, while links extend along the x axis as shown in Figure7.

Using the assigned coordinate frames, the DH table can then be derived. As shown in TableIII, additional frame rotations were introduced to ensure the correct positioning of the x and z axes. It should be noted that the joint angles for joints 1 and 5 increase with anticlockwise rotation, whereas the joint angles for joints 2, 3, and 4 decrease in the same direction. Therefore, negative signs are applied to the joint angles for joints 2, 3, and 4 in the DH table.

After implementing all the transformations, we obtain the final transformation matrix of the end effector relative to the base frame,

$$H = A_1A_2...A_n = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} \quad (7)$$

Here, R_n^0 represents the orientation of the end effector, and o_n^0 represents the position of the end effector in the base frame. Using this transformation matrix, we can determine the complete pose of the end effector. Specifically, we extract the Euler angles in the ZYX sequence from R_n^0 to describe the orientation, and obtain the position from o_n^0.

We verified the forward kinematics by using Matplotlib to visualize the pose based on our DH table, and the results met all of our requirements, as shown in Figure8.

2) Inverse Kinematic: Inverse kinematics is another crucial process that determines the joint angles required to achieve a given pose of the end effector. In real-world applications, the robot first determines the pose of the target object using computer vision, and then calculates the required joint angles to reach that pose, allowing it to successfully grasp the object.

We used a geometric approach to compute the joint angles. There are four possible geometric

TABLE III: Link parameters for the arm

1	θ_1^*	103.91	0	0
2	$\frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
3	$\frac{\pi}{2}$	0	0	0
4	$-\theta_2^*$	0	200	0
5	$-\frac{\pi}{2}$	0	50	0
6	$-\theta_3^*$	0	200	0
7	$-\theta_4^*$	0	174.15	0
8	$\frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
9	θ_5^*	0	0	0

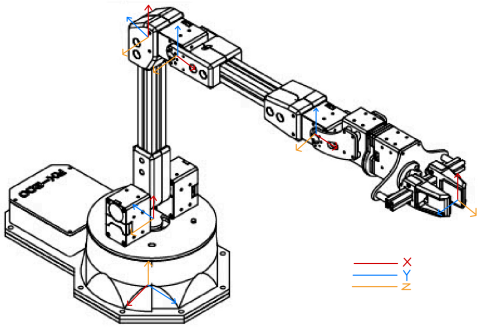


Fig. 7: DH Convention frame for Forward Kinematic

configurations for our robotic arm at the same target position: elbow-up, elbow-down, reaching forward, and reaching backward. Since our robot is positioned at the back of the workspace, we consider only the reaching-forward configuration to maximize its working space. Additionally, as the robot's task involves picking up objects from the ground, we only consider the elbow-down configuration. We denote the position of the end effector Pe as $[px, py, pz]$ and the orientation as $yaw, pictchandroll$.

The first step is to determine the base joint angle θ_1. As shown in Figure 9, the robot's joints lie in the same plane, making θ_1 the only parameter that determines the arm's position in the x-y plane.

$$\theta_1 = arctan(-px, py) \tag{7}$$

We designed two different picking poses, as shown in Figure 10: one for picking objects vertically, and the other for picking objects horizontally. The choice of pose can be set manually, depending on the height of the target object.

Based on the chosen picking pose, we can determine the wrist position. The wrist position Pw is given by,

$$Pw = Pe + a_5 * z \tag{7}$$

Where Pw is the pose of wrist, a_5 is the length from joint 4 to end effector, z is the basis of a_5. If the pose is chose pick vertically,

$$z = [0, 0, 1] \tag{7}$$

If the picking pose is horizontal, then,

$$z = [sin(\theta_1), -(abs(\theta_1)), 0] \tag{7}$$

After deriving the wrist position, we can calculate the positions of joints 2 and 3 using trigonometry.

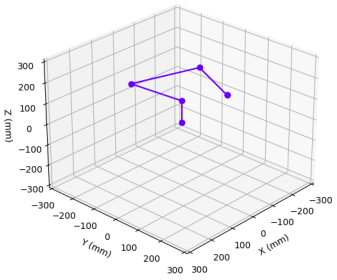


Fig. 8: 3D visualization of robot arm of joint angle [1, -1, 1, 1, 1](rad)

The important parameters are shown in Figure 11, where d_1 is base height from workspace, a_2 is length from joint 1 to joint 2, a_3 is length from joint 2 to joint 3, r is horizontal distance from joint 1 and joint 3, b is vertical distance from joint 1 and joint 3, D is the distance between joint 1 and joint 3, θ is the initial angle between a_2 and a_3, α is the angle between D and a_2, β is the angle between y-axis and D, γ is the angle between a_2 and arm. Joint angle θ_2 can be calculated by,

$$r = \sqrt{P_w[0]^2 + P_w[1]^2} \tag{7}$$

$$s = Pw[2] - d_1 \tag{7}$$

$$D = \sqrt{r^2 + s^2} \tag{7}$$

$$\theta_3 = arccos(\frac{(a_2^2 + a_3^2 - D^2)}{2 \cdot a_2 \cdot a_3} - \theta) \tag{7}$$

And then θ_2 can be derived by,

$$\alpha = \arctan\left(\frac{a_3 \cdot \sin(\theta - \theta_3)}{a_2 - a_3 \cdot \cos(\theta - \theta_3)}\right) \tag{7}$$

$$\beta = arctan(r, s) \tag{7}$$

$$\theta_2 = beta - alpha - gamma \tag{7}$$

Based on the different poses, θ_4 can be calculated from the known parameters. If the picking pose is vertical:

$$\theta_4 = \theta - \gamma - \theta_2 - \theta_3 \tag{7}$$

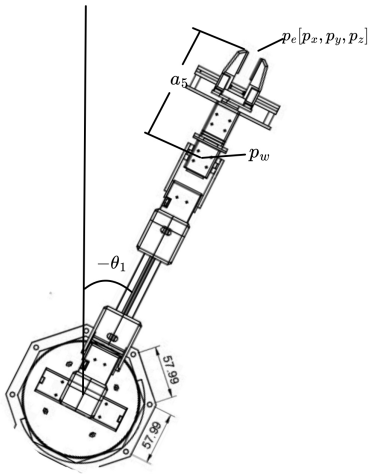
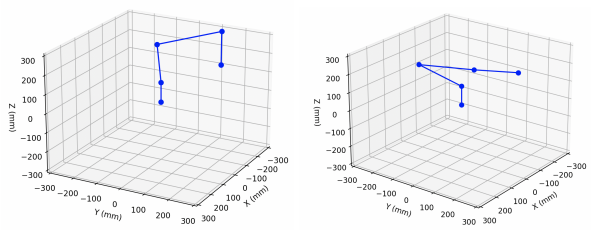


Fig. 9: Top view of the arm



(a) (b)

Fig. 10: Vertical and horizontal pick poses in the same location

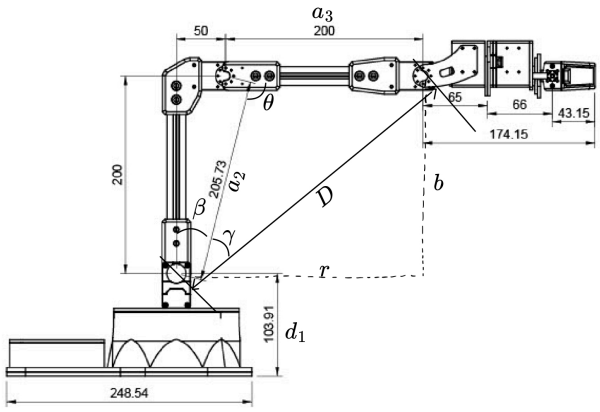


Fig. 11: Side view

If the pose is horizontal,

$$\theta_4 = \theta - \gamma - \theta_2 - \theta_3 - \frac{\pi}{2} \tag{7}$$

Finally, we can compute θ_5. θ_5 is only determined by object's orientation which is yaw here and θ_1,

$$\theta_5 = \pi + \theta_1 - yaw \tag{7}$$

3) Click to Grab/Drop: After completing the forward and inverse kinematics, we tested the execution capabilities of our robotic arm. We added two additional states to the state machine: "pick" and "release." We also introduced a new button

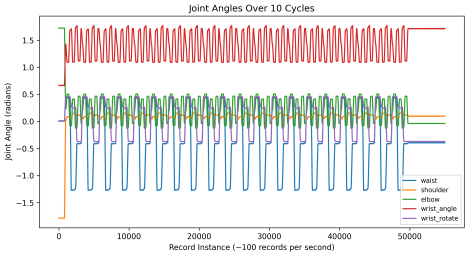


Fig. 12: Joint Angles Over 10 Teach and Repeat Cycles

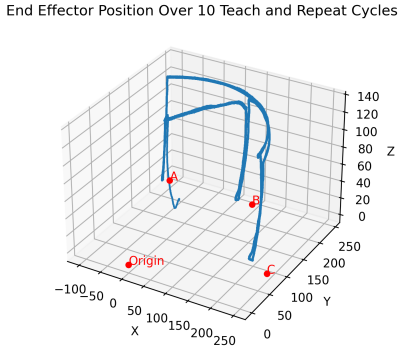


Fig. 13: End Effector Positions Over 10 Teach and Repeat Cycles

labeled "Grab/Drop." When this button is clicked, the state transitions to "pick" and remains there until a mouse click is detected by the computer. Once a mouse click is detected, the mouse coordinates are converted to world coordinates, which are then used in the inverse kinematics calculations, setting the arm to pick vertically. The joint angles are then computed to move the end effector to the target position. To ensure that the gripper grasps the center of the block rather than the bottom, a height offset of 20 mm is added. After computing the joint angles, the robotic arm executes the motion to grasp the block.
Upon completing the grasping action, the state transitions to "release." The system remains in the "release" state until another mouse click is detected. Similar to the "pick" state, the arm moves to the specified position and releases the block. Once the release process is completed, the state returns to "idle."

IV. Competition

1) Sort 'n Stack: We successfully completed Level 3 in this event, achieving full marks. Our strategy involved initially detecting all blocks while in the sleep state. Based on the detected width and height, we categorized the blocks into two groups: large and small, and determined their stacking status. We then unstacked all large blocks into the right side of the negative workspace and all small blocks into the left side of the negative workspace, ensuring all blocks were fully unstacked. Following this, the arm returned to the sleep state, and the camera was used to detect all blocks again, categorizing them in rainbow order and dividing them into large and small groups.

Finally, the blocks were picked vertically one by one and stacked in the negative workspace.

2) Line 'em Up: We completed Level 2 in this event, scoring 350 points. Although we had the capability to complete Level 3, certain blocks were positioned beyond our reach, limiting us to Level 2. Our strategy was similar to the Sort 'n Stack event. First, we unstacked all blocks in the negative workspace and then re-detected them in rainbow order, categorizing them into large and small groups. We used the location and orientation data to line up the large blocks on the right side of the negative workspace and the small blocks on the left side.

3) To the Sky: In this event, we successfully stacked 12 blocks, earning 440 points. Initially, we placed all large blocks in the workspace and detected their location and orientation, storing this information for subsequent stacking. The arm then picked the blocks and stacked them in a designated area one by one. When the number of stacked blocks was less than four, the arm released each block in a vertical pose. Once the number of stacked blocks exceeded three, the arm began stacking horizontally, allowing it to reach higher positions.

4) Free Throw: We completed 21 shoots and got full scores in this event. Our strategy involves using a modified gripper to retrieve the ball from the storage area, position the arm at the desired launch angle, and release the gripper to hold the ball loosely. The arm then accelerates to launch it. The hoop's size introduces some tolerance, allowing slight vertical movement, which affects the shot style—either a *bank shot* (ball bounces off the backboard) or *swish* (directly into the hoop). Perception detects the AprilTag shown in Figure 14 to determine hoop height and size, adjusting the launch parameters for optimal scoring.

5) Bonus Event: In this event, we completed 3 cycles and got total scores. Our strategy is similar to the Free Throw task but with an added constraint that requires the robot arm to shoot in a specific order. This means the arm must follow a predetermined sequence of shots while maintaining accuracy. This restriction adds complexity, as the system must adjust between shots while consistently hitting the target.

V. Conclusion

This paper successfully demonstrates the development of an autonomous robotic system for object manipulation and basketball launching, leveraging advanced camera calibration techniques and robotic control methods. We achieved precise object detection and manipulation by integrating factory-calibrated intrinsic parameters with robust extrinsic calibration using the PnP-RANSAC algorithm. The system's ability to perform complex tasks like sorting, stacking, and basketball launching highlights its potential for industrial applications, where accuracy, efficiency, and human-robot collaboration are essential. While some limitations were identified, such as perspective distortion, our error calibration approach effectively addressed these issues, making the system adaptable to various environments. This work lays the foundation for further improvements in robotic autonomy and collaboration.

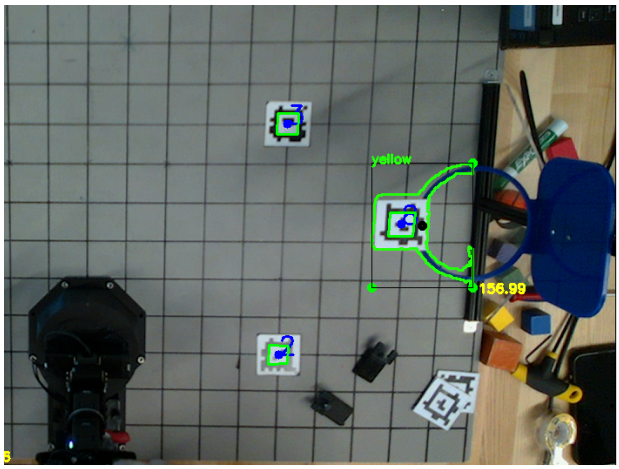


Fig. 14: Hoop Detection

References

[1] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[2] R. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[3] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ

[4] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010.

VI. Appendices

A. Color Identification: Why Use LAB Color Space?

The RGB color space is device-dependent and non-linear, meaning the same color may appear differently across devices and lighting conditions. Additionally, humans do not perceive colors equally across the RGB spectrum—for example, slight variations in green are easier to notice than changes in blue. This makes color classification difficult using raw RGB values.

To address this, we use the **CIE LAB color space**, which is designed to be *perceptually uniform*. In LAB space:

- **L** represents lightness (0 = black, 100 = white).
- **A** and **B** encode green-red and blue-yellow components.

This space aligns more closely with human perception, making color-based clustering more accurate.

B. Delta E* Metric for Color Distance

The **Delta E* metric** quantifies the difference between two colors in LAB space. It is used in industrial applications such as textiles and printing, where precise color matching is crucial. The basic **Delta E* 76** formula is:

$$\Delta E^* = \sqrt{(L_1 - L_2)^2 + (A_1 - A_2)^2 + (B_1 - B_2)^2}$$

Pros:

- Perceptually accurate and consistent.
- Works well for small color variations.

Cons:

- Sensitive to lighting conditions.
- Not robust to noisy images unless preprocessed.

Chosen Threshold: We use a threshold of $\Delta E^* < 10$ to cluster similar colors. A value below 10 generally indicates the color difference is imperceptible to the human eye.

C. Threshold Selection in Block Detection

Our algorithm segments the workspace into blocks and distractors based on size, shape, and depth. The following **threshold values** were chosen based on real-world testing:

- **Contour Area:** Minimum 550 pixels.
  Rationale: Avoids false positives from small contours generated by noise.
- **Bounding Box Width and Height:** Minimum 20 pixels.
  Rationale: Ensures only meaningful objects larger than trivial noise are captured.

D. Algorithm 1: Block Detection with Contour Analysis

Input: Image contours, depth frame
for each contour in contours **do**
  $(x, y, w, h) \leftarrow$ boundingRect(contour)
  $camera\_x \leftarrow \max(0, \lfloor x + w/2 \rfloor - 1)$
  $camera\_y \leftarrow \max(0, \lfloor y + h/2 \rfloor - 1)$
  $depth\_z \leftarrow$ DepthFrameRaw$[camera_y, camera_x]$
  if $depth\_z < 32$ **then**
    $ratio\_multiplier \leftarrow 0.7$
  else if $depth\_z < 52$ **then**
    $ratio\_multiplier \leftarrow 1$
  else
    $ratio\_multiplier \leftarrow 4$
  if $w/h \notin [1 - 0.2 \cdot ratio\_multiplier, 1 + 0.2 \cdot ratio\_multiplier]$ **then**
    continue
  if $w < 20$ or $h < 20$ **then**
    continue
  Perform contour offset, shrink, and expand operations
  Calculate minimum bounding box and its dimensions
  if $length \cdot breadth < 550$ **then**
    continue
  if $length/breadth \notin [1 - 0.2 \cdot ratio\_multiplier, 1 + 0.2 \cdot ratio\_multiplier]$ **then**
    continue

Justification for Block Detection Steps:

- **Bounding Rectangle Calculation:** Extracts the smallest upright rectangle enclosing a contour for shape analysis.
- **Depth Filtering:** Discards out-of-reach objects to improve efficiency.
- **Aspect Ratio Filtering:** Ensures detected objects are likely cubic blocks.

E. Algorithm 2: Distractor Avoidance using IoU and Depth

Input: Contour polygons, depth frame
for each contour polygon **do**
  Compute minimum rotated rectangle and its coordinates
  Calculate side lengths l_1, l_2, l_3, l_4
  $length \leftarrow \max(l_1, l_2, l_3, l_4)$
  $breadth \leftarrow \min(l_1, l_2, l_3, l_4)$
  Calculate IoU between polygon and bounding box

if IoU < 0.7 or (IoU < 0.87 and $depth\_z \in [50, 65]$) **then**
    continue
if $w < 38$ or $h < 38$ and $depth\_z \in [50, 65]$ **then**
    continue

Justification for Distractor Avoidance Steps:

- **Rotated Rectangle Calculation:** Enables more accurate shape comparison for skewed objects.
- **Intersection-over-Union (IoU):** Measures overlap between predicted and actual bounding boxes to identify overlaps.

These values ensure that the robotic arm accurately identifies blocks while avoiding obstacles and maintaining computational efficiency.

F. Block Detection Algorithm

Input: Image contours, depth frame
for each contour in contours **do**
    $x, y, w, h \leftarrow$ boundingRect(contour)
    $camera\_x \leftarrow \max(0, \lfloor x + w/2 \rfloor - 1)$
    $camera\_y \leftarrow \max(0, \lfloor y + h/2 \rfloor - 1)$
    $depth\_z \leftarrow$ DepthFrameRaw$[camera_y, camera_x]$
    if $depth\_z < 32$ **then**
        $ratio\_multiplier \leftarrow 0.7$
    else if $depth\_z < 52$ **then**
        $ratio\_multiplier \leftarrow 1$
    else
        $ratio\_multiplier \leftarrow 4$
    if $w/h \notin [1 - 0.2 \cdot ratio\_multiplier, 1 + 0.2 \cdot ratio\_multiplier]$ **then**
        continue
    if $w < 20$ or $h < 20$ **then**
        continue
    Perform contour offset, shrink, and expand operations
    Calculate minimum bounding box and its dimensions
    if $length \cdot breadth < 550$ **then**
        continue
    if $length/breadth \notin [1 - 0.2 \cdot ratio\_multiplier, 1 + 0.2 \cdot ratio\_multiplier]$ **then**
        continue

G. Distractor Avoidance Algorithm

Input: Contour polygons, depth frame
for each contour polygon **do**
    Compute minimum rotated rectangle and its coordinates
    Calculate side lengths l_1, l_2, l_3, l_4
    $length \leftarrow \max(l_1, l_2, l_3, l_4)$
    $breadth \leftarrow \min(l_1, l_2, l_3, l_4)$
    Calculate IoU between polygon and bounding box
    if IoU < 0.7 or IoU < 0.87 and $depth\_z \in [50, 65]$ **then**
        continue
    if $w < 38$ or $h < 38$ and $depth\_z \in [50, 65]$ **then**
        continue

H. Conclusion on Threshold Choices and Tuning

The threshold values were chosen through an iterative process involving:

- Real-world tests under various lighting conditions and camera angles.
- Balancing between avoiding false positives and ensuring robust block identification.