Visarm: Vision based Pick-and-Place Robotic Arm

Jaspreet Singh Chhabra and Krupal Shah

***Abstract*—This project was aimed to create a 4DOF vision based pick-and-place system to color sort objects, enabling autonomous object localization, grasping, and placement. The process starts with correcting lens distortion to obtain a geometrically accurate view of the scene. A checkerboard workspace is then detected and used as a reference plane for mapping image-space object locations into robot coordinates. Once objects within the workspace are detected, their positions are transformed into the robot frame solved using inverse kinematics. Inverse kinematics generates multiple solutions, which are evaluated through forward kinematics to identify a reachable solution. The robot executes the selected trajectory to grasp the object, followed by a verification step that confirms whether the pickup was successful. Successful grasps are subsequently placed into predefined bin locations. The system combines calibrated geometric vision with closed-loop grasp verification to ensure robust and consistent manipulation in a structured environment.**

***Index Terms*—Robotic arms, Object Detection, Camera Calibration**

I. Introduction

PICK-AND-PLACE systems using computer vision on robotic manipulators represent a core capability in modern automation, enabling robots to recognize, localize, and manipulate objects with minimal human intervention. In our final project, we investigate a vision-guided pick-and-place task to color sort objects using a 4-degree-of-freedom (4DOF) robotic arm. Such systems are widely used in real-world applications such as automated sorting in manufacturing lines, warehouse item classification, recycling facilities where materials are separated by type or color, and food processing environments that depend on visual inspection and sorting.

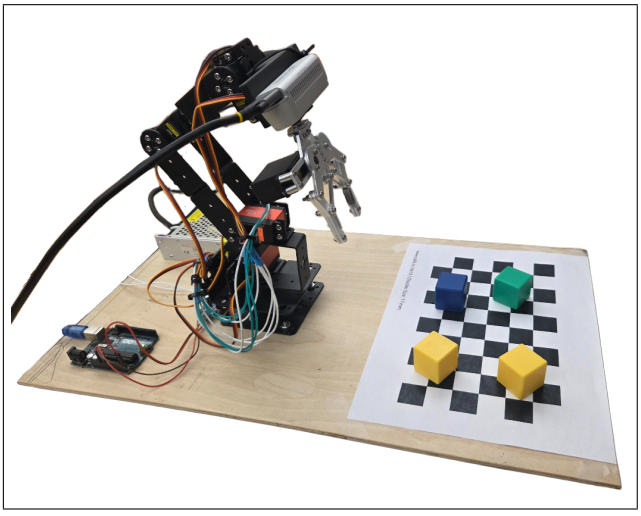

Fig. 1: Visarm Robotic Arm for Pick and Place

The use of a 4DOF robotic arm is particularly interesting because it offers a balance between mechanical simplicity and functional capability. With four independent joints, the robot can achieve a wide range of poses suitable for planar manipulation tasks while still presenting manageable kinematic complexity for analysis and control. The motivation for this project is to extend the knowledge gained from the CMPUT 312 course material and lab exercises to a more complex and sophisticated real-world robotic platform. Implementing vision-based control on a physical 4DOF robot provides practical experience with sensor integration, calibration challenges, and real-time motion execution, thereby deepening our understanding of robotic concepts and preparing for more advanced research or industrial applications.

II. Related Works

Previous research has demonstrated that even relatively simple manipulators, such as 3-DOF or 4-DOF robotic arms, can be effectively combined with computer vision systems to perform sorting and pick-and-place tasks. For instance, a 4-DOF robotic arm employing color-based image processing was used to sort colored balls in [1], while Gashi et al. [2] utilized a 3-DOF arm for object sorting. Despite the rise of deep learning approaches, many object detection and grasping tasks continue to rely on classical image processing techniques due to their lower computational cost, faster execution, greater predictability, and reduced data requirements [3]. These methods also offer easier calibration and deterministic behavior, particularly when the objects are simple, the background is controlled, and task speed is not critical.

Common techniques include color-based segmentation in HSV or other color spaces combined with contour detection and masking [4], edge detection methods such as the Hough transform, and hybrid pipelines that integrate 2D camera data with depth information to improve object localization [3]. These techniques have been studied extensively and applied in the CMPUT 312 course which have been extended in this project.

III. Methodology

A. Hardware

The core of the system is a low-cost robotic arm purchased from Amazon for approximately $80. Although advertised as a 6-DOF platform, the arm provides 5 functional degrees of freedom and is constructed primarily from sheet metal. The stock MG966R/MG996R servos offer approximately 13 kg·cm of torque and are driven by standard PWM signals without any onboard feedback mechanisms.

An Intel RealSense D435i camera was mounted on Joint 4 using a custom 3D-printed bracket to provide full visibility of the workspace. Due to the added weight of the camera, the base and shoulder servos were unable to provide sufficient torque for stable manipulation. These servos were replaced with higher torque Miuzei units rated at 22.8 kg·cm.

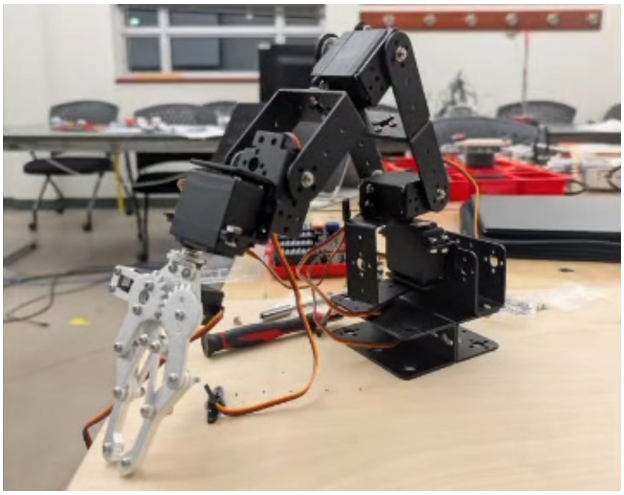

Fig. 2: Assembled Arm

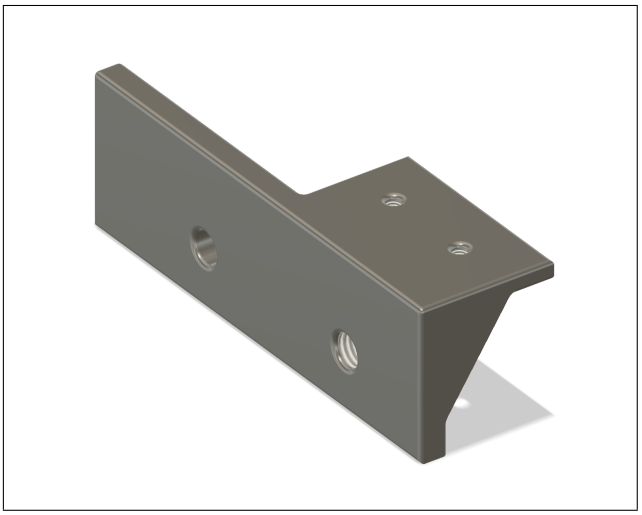

Fig. 3: 3D Printed Camera Mount for Intel RealSense

For controlling the motors, an Arduino UNO together with a PCA9685 16-channel PWM servo driver was used. The UNO receives serial commands from the host computer and generates the corresponding PWM signals for the servos. Because six servos draw a considerable amount of current under load, a 5 V switching-mode power supply capable of providing 10 A was sourced. All components were mounted on a 1/4" plywood board. The overall wiring layout, including the servo driver and power distribution, is illustrated in Figure 4. The workspace was defined using a 6×10 checkerboard with 23 mm tiles, which was also affixed to the board.

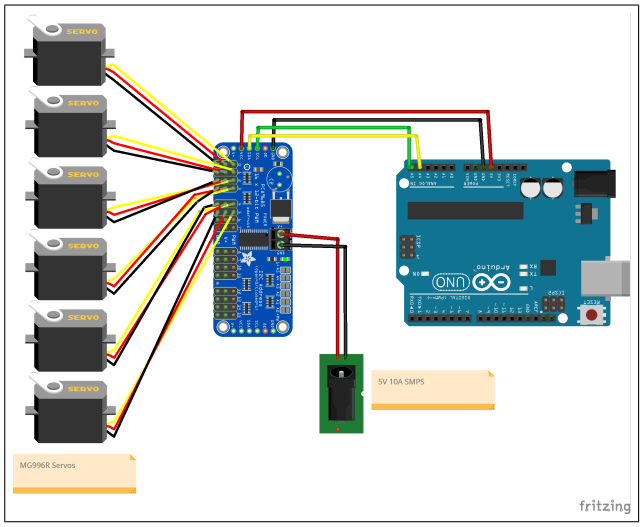

Fig. 4: Circuit Diagram for Visarm.

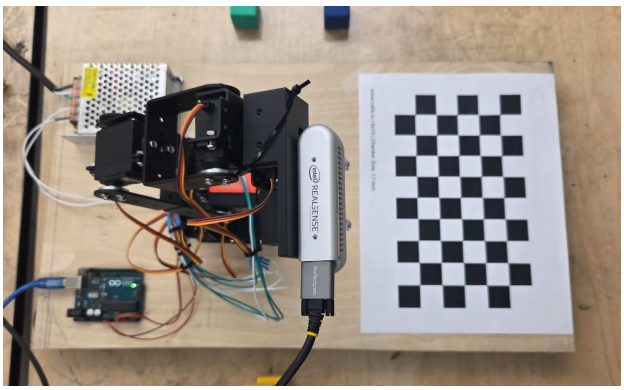

Fig. 5: Visarm final setup

The overall system operates using a client–server architecture. The host computer executes all high-level tasks, including camera-based perception, workspace calibration, coordinate transformation, and forward and inverse kinematics. The Arduino functions as the client, handling all low-level actuation tasks by converting the received joint angles into PWM signals for the servo controller.

Communication between the host and the Arduino is hosted over a USB serial connection using a simple ASCII-based command protocol, illustrated in Figure 6. This lightweight message structure enables transmission of joint commands and retrieval of the current arm state.

SET a1 a2 a3 a4 a5 a6	# Sets specified joint angles
GET	# Returns joint angles

Fig. 6: Arduino-Client Serial Message Structure.

B. Forward Kinematics

In order to control our robotic arm and move it around the workspace, we needed to solve the forward kinematics of our robot so we could determine the end-effector position given the current joint angles. We used the Denavit-Hartenberg (DH) convention to model the forward kinematics of our arm [5]. We started by assigning coordinate frames to each joint position $(F_1, F_2, F_3, F_4, F_5, F_6)$, including the base frame F_B and the end-effector frame F_E. From these frames, we found the DH parameters as defined in Table I.

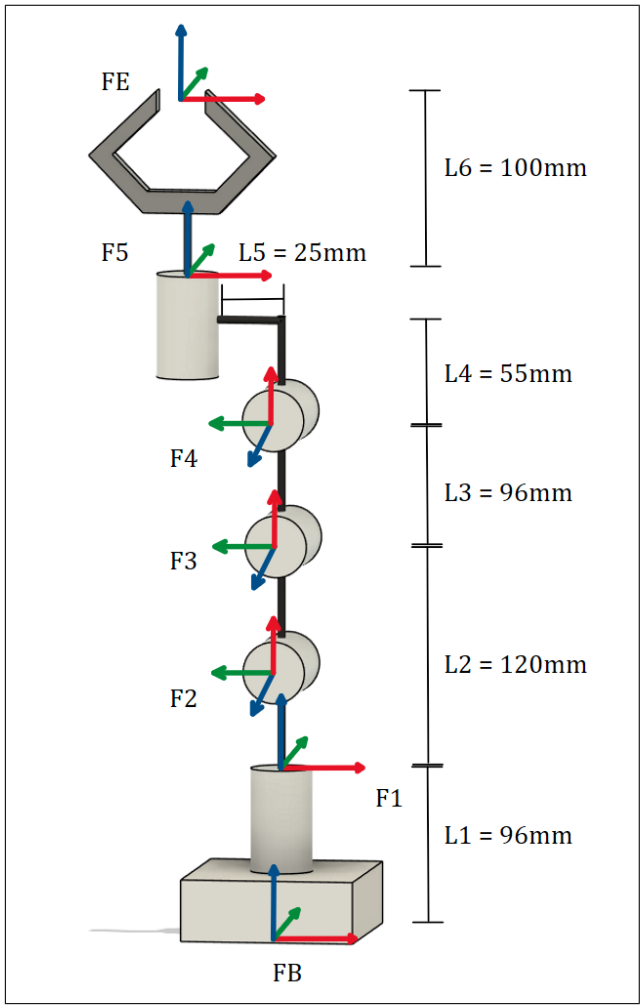

Fig. 7: Visarm Coordinate Frames

Link (i)	θ_i (rad)	α_{i-1} (rad)	d_i (mm)	a_{i-1} (mm)
1	θ_1	$\pi/2$	96	0
2	θ_2	0	0	120
3	$-\theta_3$	0	0	96
4	$-\theta_4 + \pi/2$	$\pi/2$	0	25
5	θ_5	0	55 + 100	0

TABLE I: DH Parameters of the 5-DOF Arm

Multiplying all of the transformation matrices together will produce a single transformation matrix between the base frame of the robot to the end-effector. This final transformation matrix is the forward kinematics solution for our robot. Equation 1 shows the change of transformations and Equation 2 shows the position of the end-effector from the base frame of the robot.

$$ {}^{B}T_{E} = {}^{B}T_{1}\,{}^{1}T_{2}\,{}^{2}T_{3}\,{}^{3}T_{4}\,{}^{4}T_{5}\,{}^{5}T_{E}. \tag{1} $$

$$ p_{EE} = \begin{bmatrix} -12\,s_2c_1 - 9.6\,s_{23}c_1 + 15.5\,s_{-2+3+4}c_1 - 2.5\,c_{-2+3+4}c_1 + 9.6 \\ -12\,s_1s_2 - 9.6\,s_1s_{23} + 15.5\,s_1s_{-2+3+4} - 2.5\,s_1c_{-2+3+4} \\ 12\,c_2 + 9.6\,c_{23} + 15.5\,c_{-2+3+4} + 9.6 \end{bmatrix} \tag{2} $$

where

$$ s_i = \sin\theta_i, \qquad c_i = \cos\theta_i, $$
$$ s_{ij} = \sin(\theta_i - \theta_j), \qquad c_{ij} = \cos(\theta_i - \theta_j), $$
$$ s_{-2+3+4} = \sin(-\theta_2 + \theta_3 + \theta_4), \qquad c_{-2+3+4} = \cos(-\theta_2 + \theta_3 + \theta_4). $$

C. Inverse Kinematics

Inverse kinematics (IK) refers to the computational process of determining the joint parameters required to achieve a desired position and orientation of a robot's end-effector. Unlike forward kinematics, which is easy to calculate from transformation poses, inverse kinematics is much more complicated and has no straight-forward general solution because there might be multiple solutions or even no solutions to a specific point.

In order to map world coordinates to joint angles, we make use of trigonometry. We break down our robot into multiple subproblems and we try to find solutions to the subproblems. We are given the position for the end-effector to reach (x, y, z). Even though the robot is 5DOF, we only consider a 4DOF robot for simplicity, i.e., joint 5 is fixed.

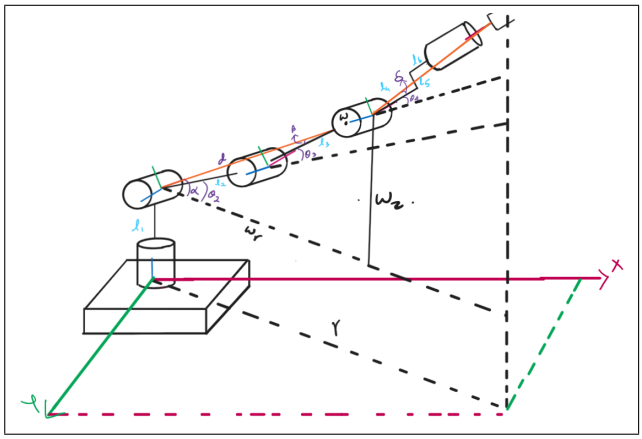

Fig. 8: Inverse kinematics Diagram

From Figure 8:

$$ \theta_1 = \arctan\left(\frac{y}{x}\right) \tag{3} $$

We shift the entire plane by l_1, in order to solve for the subproblem $\theta_2, \theta_3, \theta_4$. We find that three 3 angles from a planar arm and hence can be solved using the planar arm equations.

$$
\begin{aligned}
L_{ee} &= \sqrt{L_5^2 + (L_4^2 + L_6^2)} \\
r &= x\cos\theta_1 + y\sin\theta_1 \\
w_r &= r - L_{ee}\cos\phi \\
w_z &= z - L_{ee}\sin\phi \\
d &= \sqrt{w_r^2 + w_z^2}
\end{aligned}
$$

$$
\theta_3 = \arccos\left(\frac{-(\mathrm{d}^2 - L_2^2 - L_3^2)}{2L_2L_3}\right) \tag{4}
$$

Using the Law of Cosines:

$$
\begin{aligned}
\alpha &= \arctan\left(\frac{w_z}{w_r}\right) \\
\beta &= \arccos\left(\frac{L_2^2 + d^2 - L_3^2}{2L_2 d}\right) \\
\theta_2 &= \begin{cases} (\alpha+\beta) - \frac{\pi}{2}, \\ (\alpha-\beta) - \frac{\pi}{2} \end{cases}
\end{aligned} \tag{5}
$$

Finally,

$$
\begin{aligned}
\delta &= \arctan\left(\frac{L_5}{L_4 + L_6}\right) \\
\theta_4 &= \theta_2 + \theta_3 + \delta - \phi + \frac{\pi}{2}
\end{aligned} \tag{6}
$$

There are many combinations possible and not all candidates are valid solutions for our robot configuration. To determine which combinations of joint angles are actually valid, we plug the candiates back into the forward kinematics to see if the joint angles bring the end-effector to the desired position.

D. Camera Calibration and Mapping

Camera calibration is performed to accurately recover a camera's intrinsic parameters, extrinsic parameters, and lens distortion. Using MATLAB's Camera Calibrator App, this is done by capturing roughly 10–15 checkerboard images from varied viewpoints, making sure the pattern covers diverse orientations and positions across the camera's field of view.

Checkerboard calibration works by detecting the corners of the black-white grid, whose geometry is known a priori. Since the spacing between corners is fixed, the app solves for the camera intrinsics (focal length, principal point, and distortion coefficients) and extrinsics (rotation and translation of the camera relative to the checkerboard) by minimizing the reprojection error-the difference between observed corner locations and the projection of their corresponding 3D points.

Figure 9 shows the estimated camera poses in the checkerboard (world) coordinate frame, confirming good spatial coverage during image collection. The reprojection error plot in Figure 10 indicates a mean reprojection error of approximately 0.20 pixels, demonstrating accurate calibration. As the Intel RealSense D435i does not use a fisheye lens, distortion is relatively modest; however, a small amount of radial distortion was detected and corrected using the estimated distortion coefficients.

The calibrated intrinsic matrix and radial distortion coefficients obtained from the MATLAB Camera Calibrator are:

$$
K = \begin{bmatrix} 898.3306 & 0 & 633.6178 \\ 0 & 898.7783 & 379.6450 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{d}_{\text{radial}} = \begin{bmatrix} 0.1402 \\ -0.2897 \end{bmatrix} \tag{7}
$$

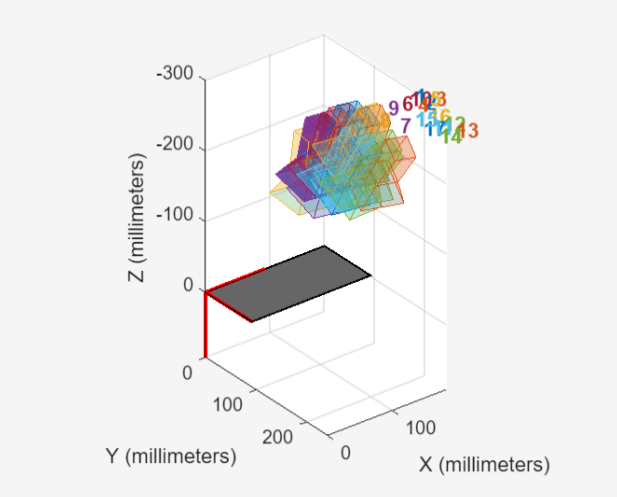

Fig. 9: Estimated Camera Positions from Camera Calibrator App

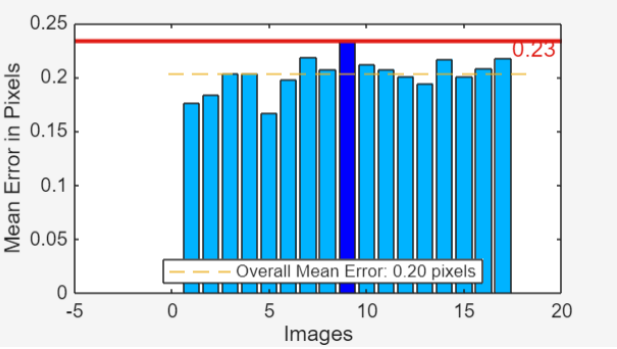

Fig. 10: Reprojection Errors in Calibration

Using these results, we can map image pixel coordinates to world coordinates by constructing a transformation $T_{\text{checker}\to\text{camera}}$. After undistorting the pixel locations, the intrinsic matrix is inverted to obtain normalized camera-frame rays. These rays are then intersected with the checkerboard plane, whose pose relative to the camera is determined from the extrinsic calibration.

Since the workspace is planar and the objects handled by the robot have predetermined heights, we apply a planar assumption by fixing $Z = 0$ in the world frame. This reduces the 3D mapping problem to a 2D homography transformation between the image plane and the checkerboard plane, allowing each detected object centroid in pixels to be mapped directly

into metric coordinates in the robot's workspace.

The transformation $T_{\text{checker}\to\text{robot}}$ can be visually determined from the physical setup, as shown in Figure 11. This homogeneous transform can be decomposed into a rotation matrix R and a translation vector T:

$$T_{\text{checker}\to\text{robot}} = \begin{bmatrix} 0 & -1 & 0 & 305 \\ -1 & 0 & 0 & 122 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

where the rotation and translation components are

$$R = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \qquad T = \begin{bmatrix} 305 \\ 122 \\ 0 \end{bmatrix}. \tag{9}$$

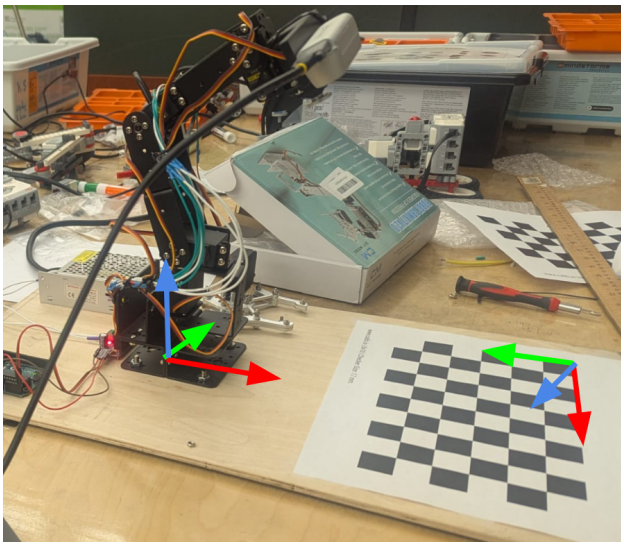

Fig. 11: Coordinate Frames of Robot and Checkerboard

Using the two transformations defined above, a pixel in the image frame can be mapped directly into the robot frame. In homogeneous coordinates, the mapping is expressed as

$$p_{\text{robot}} = T_{\text{checker}\to\text{robot}}\, T_{\text{image}\to\text{checker}}\, p_{\text{image}}. \tag{10}$$

Equivalently, starting from a point in the robot frame, its projection into the image plane is given by

$$p_{\text{image}} = T_{\text{checker}\to\text{image}}\, T_{\text{robot}\to\text{checker}}\, p_{\text{robot}}. \tag{11}$$

E. Object Detection

After computing the camera calibration parameters and determining the pixel to robot transformation, the next stage involves detecting colored objects placed within the workspace defined using the checkerboard. The robot arm is first moved to a fixed *survey position* as in Figure 12. This configuration provides a consistent and unobstructed workspace view and is defined using the following joint angles:

$$\text{POS}_{survey} = [-6,\ 29,\ 77,\ 90,\ 0,\ -10].$$

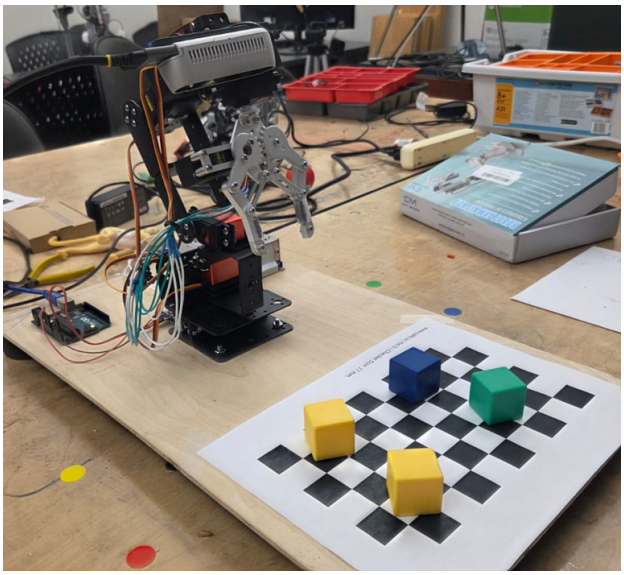

Fig. 12: Visarm Survey Position.

1) Workspace Initialization: The workspace defined using the checkerboard is detected using the four outer vertices. These bounds are then stored in the camera parameter file and subsequently used to mask all frames coming from the camera. A comparison of unmasked and masked workspace can be seen in Figure 13.

2) Undistortion and Contrast Enhancement: Each frame from the camera is first undistorted using the intrinsic matrix K and distortion vector $\mathbf{d}$:

$$I_{\text{undist}} = \text{undistort}(I_{\text{raw}},\, K,\, \mathbf{d})\,. \tag{12}$$

To improve robustness to lighting variation, contrast-limited adaptive histogram equalization (CLAHE) is applied to the V-channel of the HSV representation:

$$I_{\text{eq}} = \text{CLAHE}(\text{HSV}_V(I_{\text{undist}}))\,. \tag{13}$$

3) Workspace Masking: A binary mask corresponding to the checkerboard outline is applied to isolate the sorting area:

$$I_{\text{masked}} = I_{\text{eq}} \odot \text{Mask}_{\text{board}}. \tag{14}$$

Only objects fully contained within this workspace are considered for detection (see Figure 13).

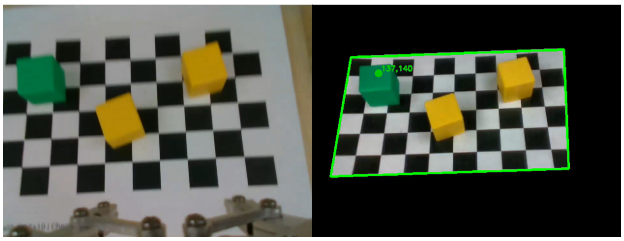

Fig. 13: Unmasked and Masked Workspace.

4) Color Based Segmentation and Contour Extraction: Objects are detected using color thresholding in HSV space:

$$M = \text{inRange}(\text{HSV}(I_{\text{masked}}), \text{lower}, \text{upper}). \quad (15)$$

This is then followed by median filtering and morphological opening to remove noise. External contours are then extracted, and each contour is filtered by area and shape to remove hand intrusions and noise. The centroid of each valid contour is computed and is considered as a candidate object location.

5) Vertical Offset Correction: Slight adjustments are made to the position of the object as the object detection pipeline finds the center of the blob rather than the top face of the cube. A vertical offset proportional to the normalized y-position is subtracted to correct this error. A max err of 1cm was found which is corrected here:

$$c_y^{\text{corrected}} = c_y - \left(\text{max\_err} \cdot \frac{c_y - y_{\text{top}}}{H}\right). \quad (16)$$

where H is the vertical span of the workspace in pixels.

After performing all the above steps, this is how a detection looks like in the workspace:

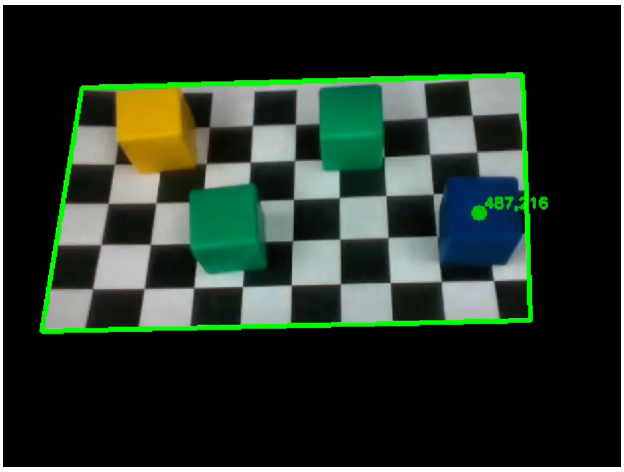

Fig. 14: Object Detection after Correction.

After grasping has been performed, the robot returns to the survey position, and a verification step is executed to determine whether an object is present in the gripper. A fixed region of interest (ROI), corresponding to the expected projection of a grasped cube, is first extracted from the incoming frame. To this mask, an HSV threshold based on saturation is applied to isolate brightly colored regions, and the largest contour in the workspace is selected. A median blur is applied to suppress noise before extracting external contours. A pickup is only considered successful if the following conditions are satisfied: (i) the contour area exceeds a predefined threshold, (ii) the bounding box exhibits a reasonable aspect ratio, and (iii) the detected region is horizontally centered and located in the lower portion of the ROI.

F. Pick-and-Place

At this point, we now had all the necessary components to complete the pick-and-place task. First, the arm is moved to a survey position from which the entire checkerboard is visible. The bounding box of the checkerboard is calculated from this position. Then, the objects are placed on the checkerboard to sort them. The image processing pipeline runs on the images to pick out the colored cubes (Section III-E). The camera parameters are used to project the pixel coordinates of the cubes into checkerboard coordinates (Section III-D). Then the checkerboard coordinates are converted into robot coordinates. The position is corrected using the error mechanism defined in Section III-E.

Inverse Kinematics is used to send the robot up to pick up the object (Section III-C). The object height is 3cm, with an added 2cm offset for robot error, giving a hardcoded height of 5cm. Then the arm comes back to the survey position to check if the arm has successfully picked up the object. It does this using the camera mounted on the arm. If the arm has not detected any, then it will try picking the object again; otherwise it will successfully move to place the object to the appropriate bin position based on its color. With all these steps combined, we finally had a fully-fledged design for picking up and sorting the colored objects. Also, an angular offset is applied to θ_1 based on the target position to compensate for the motor's physical configuration: -6° when the y-position is below 6.7 cm, and 0° otherwise.

IV. Results

The performance of the 4-DOF robotic arm for pick-and-place and color-sorting tasks was evaluated through physical measurements and multiple experimental trials. The arm exhibits a positional error of approximately 1 cm in both the x and y axes (see Figure 15) when executing the inverse kinematics. As a result, the arm successfully picks up objects about 70% of the time. Failures generally occur when the object is oriented unfavorably, primarily because joint 5 is fixed at 0°, which limits the wrist's ability to adjust the end-effector orientation.

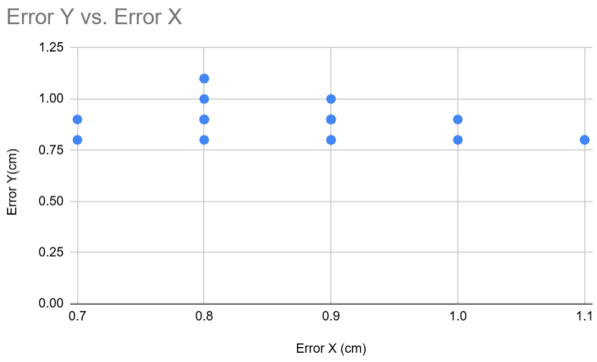

Fig. 15: Positional Errors in X vs Y

Color recognition was accurate in approximately 90% of trials. Most errors stem from the use of HSV color space, which is sensitive to lighting variations and occasionally

leads to misclassification of object colors.

In addition to these computational errors, the arm's joints were not mechanically aligned when assembled, which introduced consistent angular biases in the kinematic chain. To compensate for these assembly related errors, fixed joint offsets were applied during calibration, that is, θ_1 required a 6° adjustment. This alignment offset is distinct from the deflection-related compensations described below and are incorporated directly into the joint angles prior to any kinematic computation.

Beyond these alignment errors, the additional weight of the camera, its mount, and the grasped object caused the arm to exhibit noticeable structural deflection. This resulted in systematic positioning errors during pick-and-place operations. To compensate for these effects, two corrective offsets were introduced (see Figure 16).

1) Origin Offset: A correction was applied to θ_2 to shift the robot's effective origin slightly behind the true physical origin.

2) Position-Dependent Offset: A secondary correction was applied based on the detected object position. If the detected object's X-coordinate exceeded a defined threshold, an additional bias was introduced in θ_1.

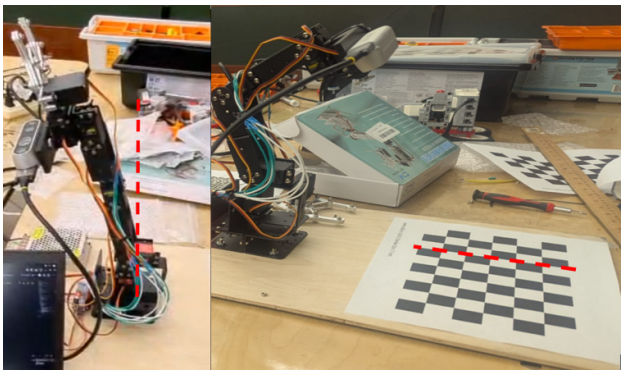

Fig. 16: Origin [Left] and Position-Dependent [Right] Offset

V. Conclusion and Future Work

This project successfully implemented a 4-DOF vision-based pick-and-place system capable of autonomous sorting objects of different colors through localization, grasping, and placement. The system combines classical image processing, HSV-based color detection, and analytical inverse kinematics to perform these tasks. Experimental evaluation through physical measurements and multiple trials showed that the arm achieves approximately 1 cm positional accuracy in both x and y directions, with a successful grasp rate of around 70% and color recognition accuracy of about 90%.

In the future, we plan to enhance the robustness and accuracy of the system by replacing the checkerboard reference with ARUCO markers, which are more tolerant to detection errors. Unlike the checkerboard, where the reference may shift and errors accumulate over multiple iterations, especially since the bounding box cannot always be reliably detected (see Figure 17), ARUCO markers allow recalibration at each iteration, preventing error accumulation and ensuring more accurate results.

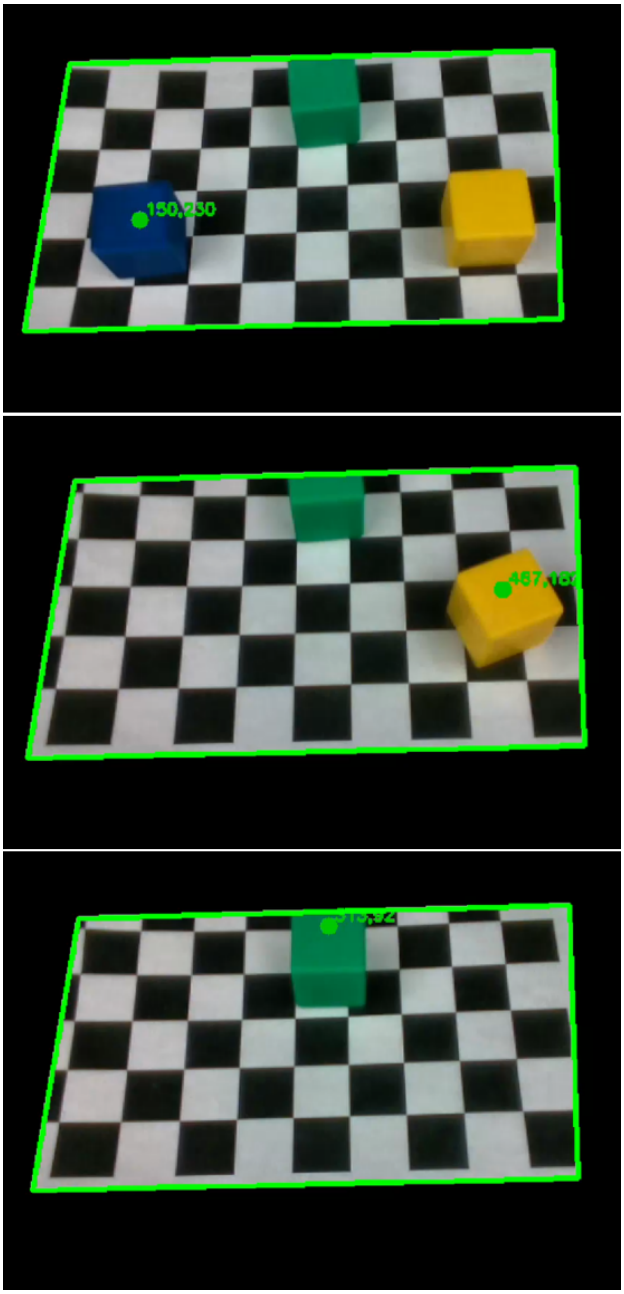

Fig. 17: Cumulative Error due to Checkerboard Drift

In addition to improving workspace definition, we also aim to introduce visual servoing to address end-effector misalignment observed during grasping. Due to the arm's positional inaccuracies and the limited span and surface area of the gripper, the end-effector often approached the cube with small lateral or angular deviations, resulting in unsuccessful grasps. A visual servoing approach would enable feedback to iteratively align the gripper with the object before closing. Specifically, the alignment error can be defined using a point-to-line metric, where the two endpoints of the gripper serve

as point features, and the two opposite vertical edges of the cube are modeled as reference lines (see Figure. 18). By minimizing this error, the gripper can be dynamically adjusted to the correct orientation and position, reducing the reliance on corrective offsets.

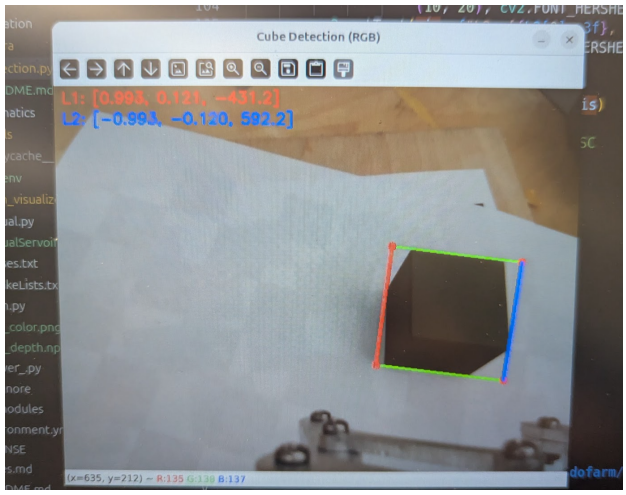

Fig. 18: Point-to-line Error

Lastly, the gripper servo experienced significant overheating and ultimately required replacement. The main cause behind this was that the servos used in the system provide no force feedback-or any feedback mechanism in general while a high gripping force was required to secure the cube. The gripper was commanded to close to a fixed angle that ensured a strong grasp, but this also caused the servo to stall under load, accelerating wear. A way to fix it would be to check for cube detection during the pickup instead of waiting till the arm returns to the survey position. This could be achieved either through a visual check as we are doing above or by installing limit switches to detect contact. If the object is confirmed to be grasped, the command being sent to close the gripper can be relaxed to avoid stalling the servo.

The code and additional documentation are available on GitHub to encourage collaboration and further development: https://github.com/jaz404/visarm

Acknowledgment

We would like to thank our professor Martin Jagersand and our TAs Riley Zilka and Justin Valentine for their continuous support and guidance to help complete this project successfully. We also gratefully recognize the project by Nakamura et al. [1] as having provided valuable inspiration in shaping aspects of this work

References

[1] K. Nakamura, O. Sullivan, and E. Carmody, *Kainakamura/rbe3001: A 4-dof serial robotic arm designed for pick-and-place*. [Online]. Available: https://github.com/KaiNakamura/RBE3001.

[2] A. Gashi, D. Krasniqi, E. Shala, X. Bajrami, and R. Likaj, "Automated robotic arm for object detection and classification," in *2025 14th Mediterranean Conference on Embedded Computing (MECO)*, 2025, pp. 1–5. DOI: 10.1109/MECO66322.2025.11049290.

[3] G. Du, K. Wang, S. Lian, and K. Zhao, *Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: A review*, Oct. 2020. [Online]. Available: https://arxiv.org/abs/1905.06658.

[4] H. Q. T. Ngo, "Using hsv-based approach for detecting and grasping an object by the industrial mechatronic system," *Results in Engineering*, vol. 23, p. 102 298, 2024, ISSN: 2590-1230. DOI: https://doi.org/10.1016/j.rineng.2024.102298. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S259012302400553X.

[5] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, Jun. 1955. DOI: 10.1115/1.4011045.