Rob 550: Armlab

Jacob Jancek, Yung Hu, Yang Zhang
{jjancek, youhu, zhanya}@umich.edu

***Abstract*—This report presents an overview of a 5-degree-of-freedom (DOF) autonomous robotic arm. The problem of autonomous robotic arm control is to develop a system capable of accurately perceiving its environment, making informed decisions, and executing tasks reliably and flexibly without human intervention; innovations in this area can not only improve safety for employees in high-risk work environments but also be used to perform lower value and repetitive tasks. For a 5-DOF robotic arm to autonomously stack blocks of different sizes, colors, and positions, it must use computer vision techniques, kinematics, and path planning to manipulate the desired objects. The arm is equipped with sensors and algorithms that allow it to perceive and respond to changes in its surroundings in real time. There are numerous approaches to automating a robotic arm, including utilizing Convolutional Neural Networks to continually learn and train from the data it receives, thereby enabling it to improve its performance over time. Our arm's design incorporates inverse kinematics to determine appropriate end effector waypoints and computer vision techniques to complete the desired tasks accurately. The report also covers the testing and validation of the arm, including experiments that demonstrate its ability to perform a range of tasks, including grasping, reaching, and orienting objects in a controlled environment. The results show that the 5-DOF autonomous arm has a high level of accuracy and efficiency, making it suitable for a variety of applications in fields such as manufacturing, service robots, and areas of collaborative robotics.**

I. Introduction

The 550 Arm Lab project aims to leverage the power of robotics to fully automate a 5-degree-of-freedom robotic arm, with the ultimate goal of completing various tasks such as block stacking. The project incorporates the three core principles of robotics: sensing, acting, and deciding. Our approach, which is described in detail later in this report, leverages the latest advancements in computer vision and kinematics to locate and manipulate blocks without any human intervention accurately. Controlling a robotic arm begins with opening the system and gaining a foundational understanding of its operation in space. This knowledge is critical for accurately calibrating the camera, which is crucial in determining the arm's workspace coordinates. The stored calibration matrix provided by the camera may not always be precise; therefore, it is necessary to perform a custom calibration. This calibration is done by capturing a series of images of a checkerboard at various angles and heights within the camera's field of view to obtain the x, y, and z coordinates. By repeating this process multiple times and computing a weighted average, the accuracy of the calibration can be improved beyond that of the stored matrix. This step ensures the camera is calibrated to the workspace and provides accurate coordinate information. Implementing the teach-and-repeat process is straightforward. We can achieve this by using the graphical user interface to set the motors' torque to zero, physically move the arm to the desired positions, and finally, record the joint angles and gripper status to teach the robot the pattern it should repeat. In this stage, we will utilize the four AprilTags on the board and their predefined locations to estimate the extrinsic parameters. AprilTags are a visual identification system that accurately computes the 3D position, orientation, and identity of the tags relative to the camera. However, since the camera is tilted at an angle in the workspace, a perspective transform must be applied to the image for precise capture of the AprilTag's x, y, and z coordinates. This transformation changes the image from a trapezoidal shape to a rectangular shape. The entire calibration and transformation process can be automated directly from the graphical user interface. The aim of forward kinematics (FK) is to calculate the joint positions in the global reference frame given the joint positions in the robot's frame. To achieve this, we utilized the Denavit-Hartenberg (DH) table method. The DH table takes in the five joint angles of the RX200 arm and outputs the location of the wrist in global coordinates. A detailed explanation of the DH table method will be provided in the methodology section, where most of the effort is put toward enabling autonomous control of the arm's desired locations. To complete the final step before integrating all elements into a seamless autonomous motion plan, we need to detect the blocks and calculate their joint positions through Inverse Kinematics. This can be accomplished using OpenCV, a highly efficient Computer Vision library in Python. With OpenCV, we can process images and identify various features such as color, orientation, and size of the blocks. Additionally, by pairing the processed image with the depth information from the camera, we can accurately determine the block's location in the world space. The Inverse Kinematics calculation will then provide the joint angles for the end effector to reach the desired

position and orientation in coordination with the block detection process. The final stage of the project involves crafting a streamlined motion planner to control the arm's movements and implementing various states in the state machine to handle the pickup and placement of blocks. Then, drawing on our experience from the teach and repeat sections of the project, we can automatically generate waypoints based on the information provided by our block detector. The Inverse kinematics will then come into play, allowing the arm to grab and place the blocks at their intended locations precisely. At this point, it becomes a matter of devising the most effective and efficient strategy for executing the motion planner, given the specific objectives at hand.

II. Methodology

A. Calibration

We must first find the world coordinates to create an autonomous arm. To do this, we must obtain the intrinsic and extrinsic calibration matrix. The purpose of the intrinsic calibration matrix is that it converts points from the camera coordinate system to the pixel coordinate system. Better put, the intrinsic matrix transforms 3D camera coordinates to 2D homogeneous image coordinates and vice versa. The intrinsic parameters describe the internal properties of the camera, while the extrinsic parameters describe the relationship between the camera and the world. The flow diagram below illustrates the interaction between these matrices and how they perform the camera calibration.

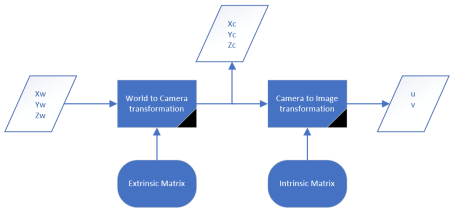

Fig. 1. Overarching flow diagram of how the matrices interact to perform camera calibration.

The extrinsic calibration matrix is found using the graphical user interface (GUI) and a 10x12-inch checkerboard held at different angles and heights in the camera range. To increase the accuracy of the matrix, we perform the calibration several times and then take an average of the matrices to form a final intrinsic matrix. An example of the intrinsic matrix is expressed below (1).

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1}$$

f_x and f_y represent the focal length of the camera lens and should be the same, or else the pixels will not be square. x_0 and y_0 are the offset, and s is skew, which causes shear distortion in the projected image. We will choose to use our own intrinsic matrix for the manual calibration portion of the lab. The second step to obtaining the world coordinates is obtaining a rough extrinsic matrix for the camera. The camera's extrinsic matrix describes its location in the world and in what direction it is pointing. This matrix can be easily obtained with a tape measure and by calculating the camera angle. We will then calculate The x,y, and z coordinates using the measurements from the camera. We have these coordinates in the extrinsic matrix formula below (2), with R and T representing the extrinsic matrix we are solving.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} & R & & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2}$$

The next step is implementing teach and repeat to familiarize ourselves with the software. Given a set of joint positions, we can implement a function to make the robot move to those positions. First, we will set the torque to zero on the GUI to get our positions and move the robot to the set waypoints. Then, to record the joint positions of the robot arm, we create a function that adds a button to the GUI that records the joint positions every time we press the button. Likewise, another function is also necessary to open and close the gripper at the appropriate time, and it is created similarly to how we record the joint positions. Next, we integrate the repeat mode into execute in the GUI, which allows the user to execute the recorded information, causing the arm to repeat the taught motion. We can now use the button that records the waypoint to play back the waypoints we recorded. Calibrating the workspace manually is time-consuming and lacks the accuracy required for an autonomous arm. We will use Apriltags to calibrate our workspace automatically every time to fix this problem. Four AprilTags are placed in the field of view of the camera to calibrate a camera, and images of the tags are captured. The position of each tag can be obtained from the tag-detections, which is a property in the class camera. These estimates can be used to compute the extrinsic parameters of the camera. In this scenario, we will take a single image and use a package that finds and identifies unique Apriltags. The first step in using the Apriltags to calibrate is by accessing the given package and specifying the ID on each of the four tags visible in our grid to work with them in the workspace. After specifying the ID, we can read data of the tags ID and positions from tag-detections that belong to the camera class. Tag-detections will give

us the x,y, and z positions to calibrate the camera. We will use the echo command to see this data live in the terminal. As previously mentioned, our camera is tilted at an angle, so the perspective is trapezoidal. In order to change this perspective, we will apply a projective transform in the form of a homography matrix. We will compute this matrix from the x,y, and z positions we find in the terminal. Likewise, the points on the board have known world coordinates. Therefore, we will also use those to transform our world coordinates into image coordinates using the cv2.warpPerspective from the OpenCV library combined with the homography matrix and the trapezoidal image.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \mathbf{I} & | & 0 \end{bmatrix}}_{\text{projection}} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad \text{Camera to Image Frame (Intrinsic)}$$

Fig. 2. Equations that turn [u,v,d] coordinates in the image frame to [x,y,z] coordinates in the camera frame. (Depth Calibration Function)

The initial stage in computing the homography matrix involves determining the pixel coordinates of the Apriltags. In the above equation, the pixel coordinates are represented by u and v, while Z_c denotes the depth information. Furthermore, the left-hand side of the equation comprises a 3x3 intrinsic matrix that we have already computed. From the image frame, we gather the reference coordinates: $[900, 210, 400, 510, 400, 210, 900, 510]$. Using these, we will now begin computing the homography matrix using the Apriltags' coordinate in trapezoidal and rectangle images. The coordinates in the rectangle image are decided in a way that is compatible with our project, which will take the form in the figure below.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \equiv \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{3}$$

Likewise, d,e, and f represent the scaling, rotation, and translation of the y-coordinate in the second image. G and h define the mapping of the x- and y-coordinates of the points in the first image to the third coordinate (depth) of the points in the second image. Finally, i is used as a scaling factor set to one that normalizes the homogenous coordinates. The homographic matrix can then apply to the equation below, which converts the coordinates in the camera frame to coordinates in the world frame. This will be used along with cv2.WarpPerspective(), which returns the rectangular image.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \frac{1}{H} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{4}$$

In this equation, $\frac{1}{H}$ represents the inverse homographic matrix we calculate. X_c, Y_c, and Z_c represent the camera coordinates we obtain in the above figure. The overarching process of automatic calibration we follow is described below. The algorithm is iterative, meaning we can run it many times until we achieve the desired accuracy.

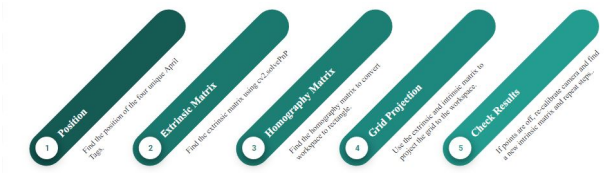

Fig. 3. Overarching camera calibration process.

B. *kinematics*

The objective of forward kinematics (FK) is to determine the position of the joints in the global frame, given the position of each of the joints in the robot frame. You can approach the implementation of the FK using either the Denavit-Hartenberg (DH) table. Before any code can be implemented into the script, we will first derive the DH table necessary for finding the forward kinematics. The first step in deriving these parameters is accurately representing our system to find the DH table. The parameters are: θ_i (Joint Angle), d_i (Joint Offset), a_i (Link Length), and α_i (Link Twist).

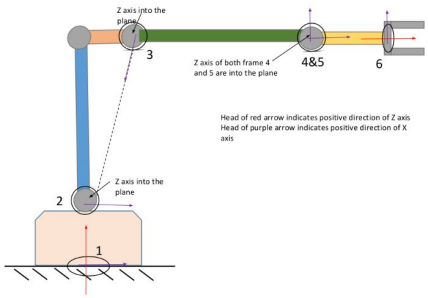

Fig. 4. Schematic of the arm with DH frames indicated.

To establish the kinematic model of a robotic arm, it is necessary to find the DH parameters. The first step is identifying the links and joints that comprise the robot's structure. Then, for each joint, the z-axis must be determined, as it is the axis of rotation for a revolute joint or the direction of translation for a prismatic joint.

The next step is to assign the base frame, which serves as the reference coordinate system for the entire robot. With these steps completed, the DH parameters can be calculated and arranged in a DH table, a matrix containing the parameters that describe the relationship between adjacent links and joints. Using the DH table, it is possible to calculate the forward kinematics of the robot, i.e., the position and orientation of the end-effector concerning the base frame, given the joint angles or link lengths.

•	Zθ	Z_d	X_d	X_θ
1	θ_1	103.91	0	$\frac{-\pi}{2}$
2	$\theta_2 + 104°$	0	-205.73	0
3	-$\theta_3 - 104°$	0	200	0
4	$\frac{-\pi}{2} - \theta_4$	0	0	$\frac{-\pi}{2}$
5	θ_5	131	0	0

Each row in the DH table represents a homogeneous transformation matrix. These thetas and measurements can be applied using the equation listed below, which constructs the forward kinematics equation.

$$
\begin{aligned}
A_i &= \mathrm{Rot}_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}\mathrm{Rot}_{x,\alpha_i} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Fig. 5. Forward kinematics equation.

This formula will be applied in the kinematics.py script as a function. By finding the forward kinematics using the DH method, one can now take in the five joint angles of the RX200 arm and output the wrist location in the global coordinates. After obtaining the forward kinematics solution, the next step is determining the joint angles and link lengths required to achieve a desired end-effector position and orientation. To accomplish this, inverse kinematics is employed, working with forward kinematics to control the robot's movement. Several methods exist to solve the inverse kinematics problem, such as analytical, numerical, and hybrid methods. Analytical methods derive a closed-form solution for the joint angles, which is only feasible for some robotic arm configurations with simpler geometries or fewer joints. We will opt for this method as we use a relatively simple robotic arm.

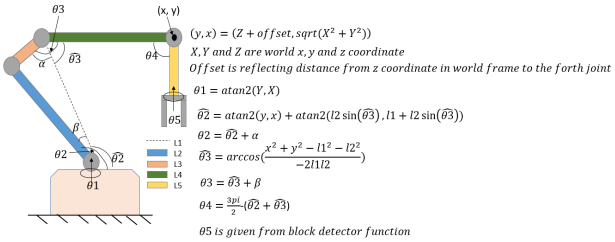

Fig. 6. Inverse kinematics solution.

Solving inverse kinematics involves taking in the referenced equations from figure 8 and using the previously found x and y world coordinates to determine the desired position and orientation.

With the closed-form inverse kinematic expressions, the team implemented the click and grab function. At each click and grab event, the current joint position $POS_{t=i-1}$ will be recorded at beginning. Then $\theta 4$ will be set equal to π to avoid collision with block if there is any. This is following by changing $\theta 1$ to match with that in next joint position $POS_{t=i}$. Before reaching $POS_{t=i}$, the gripper will be moved to 50 mm above the target place. After that, the gripper will descend gradually by 15 mm for twice and a 20 mm descend. Finally the gripper will be closed to grab the block.

C. Block Detection

Kinematics provides the mathematical foundation necessary to determine the arm's movement, while block detection enables the robotic arm to perceive and interact with the environment. Our next step will be implementing this block detector. Building a block detector involves multiple steps. Firstly, the image is segmented based on color using the OpenCV library. This includes thresholding the image and converting it from RGB to HSV to account for varying lighting conditions. The segmented images are then processed to remove noise, and individual colors of the blocks are isolated using lower and upper bounds. The objects are then dilated to eliminate any remaining noise. Then we threshold the areas of the blocks to differentiate block size. Subsequently, the contours of the blocks are found using OpenCV, and bounding boxes are drawn around the objects to differentiate between big and small blocks. The moments of the contours are calculated to determine the centroids of the blobs in the image. Furthermore, we utilize OpenCV's cv2.minAreaRect() to find the block's orientation, which is crucial for picking it up with the gripper. Overall, this process enables the accurate location and pickup of the blocks on the board. The next challenge in robotic block manipulation is to determine the coordinates and depth of the block accurately. To find the depth, pixel values are analyzed, and a threshold is

created. The camera receives the depth of each pixel in the frame, and the presence of a block changes the pixel depth in that area. With the help of the threshold, the camera can identify the block's location. To determine its coordinates, the inverse intrinsic and inverse extrinsic matrices are multiplied to obtain the x and y coordinates in the world frame. With knowledge of the block's depth and world coordinates, the centroid and orientation previously found can be used to determine the accurate position of the block in the x, y, and z-axis. This process enables the robotic arm to locate and manipulate the block precisely.

D. Automation

Our approach involves a modular design, where each function is created to perform specific tasks based on the competition's requirements. We utilize the block detector for the first event to differentiate between large and small blocks. Next, we obtain world frame positions by using intrinsic and extrinsic matrices to determine the centroid of each block. We then use inverse kinematics to grasp the block and place it at a predetermined location, either on the right negative plane for large blocks or the left negative plane for small blocks. This can be handled with a simple for loop that makes x equal to the block height multiplied by the number of blocks. To ensure proper block handling, we have adjusted the world frame's z-axis by subtracting 20mm and 10mm for large and small blocks. We follow the same approach for the second event as event one, but this time we stack the blocks, so we manipulate the z-axis rather than the x-axis.

In event three, we line up the blocks according to the color order of (ROYGBV). We utilize the centroids of the blocks and a color dictionary to determine the order of the blocks. A loop is created to adjust the block size, and the block is placed in the correct location. For the fourth event, we stack the blocks again, but this time we must also consider the depth of the end effector when grabbing the blocks. To avoid collisions with the stacked blocks, the end effector is positioned above the block and then lowered to place the block. We maintain the same angle while moving up the z-axis to avoid knocking down the stacked blocks. By using a modular design and adjusting the world frame, we ensure proper block handling and avoid collisions with the stacked blocks.

III. Results

The experiments aimed to assess the accuracy, robustness, and efficiency of the system in various scenarios. In this section, we present the results of our evaluations, including the quantitative measures of the system's performance and the qualitative analysis of its behavior. The initial procedure involved acquiring the intrinsic and extrinsic matrices, which enabled the projection of 3D points onto a 2D image. The intrinsic matrix was accessible in the ROS system as an internal factory calibration and could also be obtained through a weighted average of multiple calibrations using the checkerboard method. Shown below, on the left is the stored camera intrinsic matrix, and on the right is the intrinsic matrix computed.

$$\begin{bmatrix} 907.9 & 0 & 641.0 \\ 0 & 908.3 & 354.8 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 952.4 & 0 & 650.0 \\ 0 & 963.9 & 344.8 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

As depicted, the matrices exhibit substantial similarity, with the factory calibration showcasing a lesser difference in focal lengths. The focal lengths differ in that our final calibrated matrice is 98.8% the same, whilst the given camera matrice is 99.8% similar. To form square pixels, the focal lengths ought to be equal. The greater the deviation in focal lengths, the more distorted the pixels appear. Thus, in an effort to minimize errors in the system, factory calibration was selected for use for our manual calibration. Following the intrinsic matrix, we collected our nominal extrinsic matrix to complete our manual calibration procedure. The results of this matrix are shown below.

$$K = \begin{bmatrix} 1.00 & -0.033 & -0.011 & 0 \\ -0.0349 & -0.945 & -0.325 & 365 \\ 0 & 0.326 & -0.946 & 870 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

With both of these matrices collected our system could now determine the position and orientation of objects in a 3D environment based on the 2D images it received from the RGB-D camera. Having established the capability to determine the location of objects within the workspace, the next step involved training the robot to perform block swapping via an intermediate location. The block-swapping procedure highlighted the impact of minor calibration errors that accumulate over time. The robotic arm could complete approximately five cycles before the accumulation of errors in the x and y-axis prohibited further progress. Despite being relatively small (approximately 3-5mm), the error continuously accumulated over time as the blocks were placed farther from their designated locations. The aforementioned calibration failure emphasizes the requirement for an improved calibration mechanism to reduce error. We first use the realsense-viewer to change the mode from max range to min range in order to reduce noise in the image. Then the calibration was achieved by utilizing AprilTags and the AprilTag package available in ROS, which provided more accurate readings of the x, y, and z coordinates. The known centers of the AprilTags were compared to the coordinates obtained from computer mouse clicks

on the centers. Utilizing this new data, a new intrinsic matrix and depth calibration were established, resulting in a reduction of error in all axes.

$$\begin{bmatrix} 932.79 & 0 & 628.22 \\ 0 & 938.58 & 354.62 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

As a result of the updated intrinsic matrix, the depth calibration function used in (3) of the methodology section has been updated. By incorporating the new intrinsic matrix values, the depth calibration function produces more accurate and reliable results, allowing for more precise object detection and localization in various robotic tasks. Subsequently, a perspective transform, also known as a homogeneous transform, was performed to alter the image perspective. The center of the four April tags was utilized as the reference for the perspective transform, as these coordinates are well-known and located at the center of the workspace.

$$\begin{bmatrix} 1.049 & -0.125 & -3.783 \\ 0 & 0.985 & 354.62 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Upon application of the homography transform to the image, the calibration accuracy was evaluated qualitatively through the projection of grid points. The alignment of the grid points with the workspace coordinates indicates the relative precision of the calibration procedure.

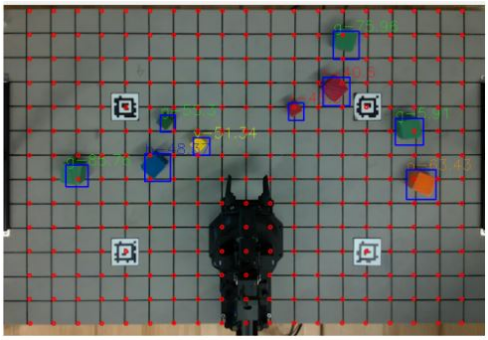

Fig. 7. Image of grid point projection.

An accurately calibrated system allowed us to determine the position of the joints in the global frame. This was performed using DH tables and garnered the results seen in figure 1.1 of the methodology section. We verified the accuracy of these results by manually driving our arm positions to known coordinates in the workspace and comparing the global coordinates read out by the GUI to what we knew to be confirmed with the system. The results of this test are shown below.

Desired Pos. (mm)	(300,-75, 43.1)	(-300,325,12)
Actual Pos. (mm)	(297, -60, 36.3)	(-302,303, 14)
Error (%)	(12.23%)	(7.30%)

The average error observed in the results is approximately 10.2%, with an increasing trend towards the edges of the workspace. It is noteworthy that the highest average error is observed in the z-axis, which can be attributed to image warping as the robot moves further towards the workspace edges. To address this issue, an offset can be introduced at the center of the workspace, which mitigates the error accumulation towards the edges, ultimately resulting in a minor overall error. Inverse kinematics was successfully applied to control the movement of a 5-degree-of-freedom robotic arm. The proposed algorithm employed geometric methods to determine the joint angles required to achieve a desired end-effector position and orientation. To evaluate the accuracy of the algorithm, a variety of end-effector poses were tested, and the resulting joint angles were found to be accurate to within several millimeters of the desired position. The algorithm was also tested in a workspace to grasp objects at different locations. The robotic arm was able to accurately determine the joint angles required to achieve the desired end-effector position and orientation, resulting in successfully grasping the objects. The algorithm's performance was tested in simulation and on the actual robotic arm, and the results show that it is a reliable and accurate method for determining joint angles for the robotic arm. Now that the position of the joints of the robotic arm can be determined in the workspace, we created and implemented a block detector that could distinguish blocks by their shape, size, color, and orientation. The OpenCV-based block detector was implemented and evaluated to detect blocks in real time. The proposed approach utilized image processing techniques, including thresholding, morphological operations, and contour detection, to segment the blocks from the background. The results of the block detector are expressed in the figure below.

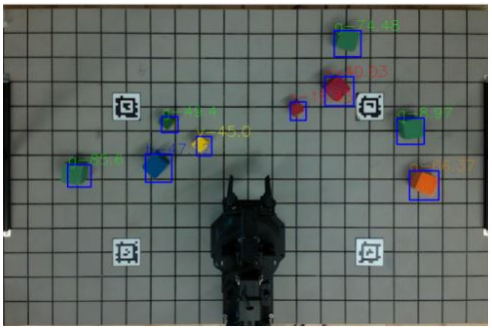

Fig. 8. Block detector.

After implementing the block detector, we integrated all the components to prepare for the competition. During the competition, we encountered challenges with our arm's precision when grabbing blocks. The arm had difficulty adjusting for different block orientations, resulting in imprecise block placement. To resolve this, we attempted to modify our inverse kinematics algorithm to enable the arm to pick up blocks from the edges of the workspace. However, due to limitations with our system, we restricted the arm's range to a perpendicular position for block picking. We also encountered issues with our depth sensor, preventing us from accurately detecting stacked blocks. Although we attempted to adjust the thresholding, we faced challenges obtaining consistent readings for the z-axis on the outer edges of the workspace. If we had resolved this issue, we would have implemented a depth function to detect stacked blocks by analyzing the thresholded z-axis values. The function would have unstacked any blocks that exceeded the threshold and placed them in the workspace, enabling us to differentiate accurately based on size and color. Overall, our robotic arm struggled to perform past levels 1 and 2 for the individual events due to these challenges, resulting in a lower overall score. We acknowledge that further refinement is required to overcome these challenges and achieve better performance.

IV. Discussion

A. Camera Calibration

Sources of error in the intrinsic matrix can include inaccuracies in measuring these parameters or assumptions made in the calibration process. The extrinsic matrix, on the other hand, contains parameters related to the position and orientation of the camera relative to the world. Sources of error in the extrinsic matrix can include errors in estimating the camera's position and orientation or errors in the assumptions made about the coordinate system used to represent the camera and the world. In this case, the source of error was measuring the model by hand. Hand-measured extrinsic matrices can be helpful as an initial estimate for camera position and orientation. Still, human errors, such as imprecise measurement or difficulty in ensuring precise camera alignment, make it insufficient for use in our system. To better calibrate our RGB-D camera and relate the image points with the world points, we created a homographic transform. Each element in the homography matrix represents a linear combination of the x- and y-coordinates of the points in the first image, with coefficients that are determined by the position, orientation, and intrinsic parameters of the camera that captured the images. The first row of the matrix determines the scaling, rotation, and translation of the x-coordinate in the second image. Likewise, the second row defines the y-axis. The third row determines the depth of the points from the first image to the second image; they determine the perspective distortion of the image.

B. Kinematics

Kinematics is a critical aspect of robotic arm control and positioning, as it provides a mathematical framework for accurately controlling the arm's movement and position. By describing the relationships between the world coordinate and end-effector, kinematics enables precise control of the arm's motion. However, it is important to ensure that the accuracy of the robotic arm is consistent across all axes. Through careful measurement and inspection, it was discovered that the z-axis of the robotic arm was suffering from significant error. This error is likely due to lens distortion, which can be rectified in the future by providing a radial distortion model and adjusting the intrinsic and extrinsic matrices to improve precision. The inverse kinematics process is relatively simple in nature, involving the analytical method of solving. While it is less computationally expensive than other methods, it is still widely used due to its ability to be easily employed on simple systems. However, the simplicity of the analytical method limits its applicability to areas with simplistic arm designs.

C. Block Detection

To implement our block detector, we utilized OpenCV to perform various tasks in conjunction with our depth function to not only locate the blocks on a 2D plane but also precisely find, grasp, and manipulate them in a 3D plane. To enhance the robustness of our detector, we employed multiple methods for block detection, including color thresholding and feature detection. Both techniques can help reduce false positives and increase overall detector accuracy. Additionally, we utilized a combination of filters, such as size, shape, and aspect ratio, to eliminate false positives and reduce noise in detected contours or features. As a future improvement to the detector, we plan to implement machine learning to detect the blocks. However, this approach is computationally expensive, but it has the potential to significantly improve detector accuracy by leveraging a larger dataset. Furthermore, a machine-learning approach could make the block detector impervious to lighting variations in the workspace.

D. Motion Plan

Although the motion plan seemed straightforward, it revealed several shortcomings in our kinematics and block-grabbing systems. Achieving precision proved to

be a major challenge, resulting from inaccuracies with the z-axis and our inverse kinematics methodology. Our current approach only yielded an elbow-up method for grasping blocks, keeping the arm at a fixed 90-degree angle during pick-up and placement. Unfortunately, this severely limited the arm's range of motion within the workspace and made it challenging to adapt the end effector for different orientations. To address these limitations in the future, we intend to refine our inverse kinematics by extending the arm's reach with an elbow-down configuration, which will be incorporated as a function in the kinematics code. This approach will first check if the arm can reach the block with an elbow-up configuration, and if not, it will attempt the elbow-down method to expand the workspace. We also encountered issues with our depth sensor, which we failed to address due to the aforementioned z-axis problem. Specifically, we lacked the ability to detect stacked blocks in the workspace. To overcome this limitation, we plan to implement a depth function that analyzes the thresholded z-axis values to detect stacked blocks. The function will then unstack any blocks exceeding the threshold and place them within the workspace, enabling us to differentiate them accurately based on size and color. Overall, by refining our kinematics and addressing the depth sensor issue, we will be able to improve the system's performance and achieve greater accuracy and flexibility in grasping and manipulating blocks within the workspace.

E. Competition

During the competition, our arm performed as expected based on the motion plan. However, we did experience some challenges with block grabbing offset, and our end effector consistently picked up blocks in the same orientation, leading to an accumulation of errors as more blocks were stacked or placed. Our current inverse kinematics solution also had limited workspace range, lacking an elbow-down arm movement solution. As a result, our arm was unable to reach several predetermined block positions, which significantly hindered our progress in individual events. To address these limitations, we intend to implement the improvements outlined in the motion plan, including refining our inverse kinematics solution to incorporate an elbow-down movement solution. Additionally, we believe that integrating a Convolutional Neural Network (CNN) would significantly enhance our system's capabilities. A CNN would improve block recognition and localization, facilitating more precise picking and placing of blocks. Moreover, it could assist in overcoming the depth issue and aid in detecting stacked blocks within the workspace. In summary, by incorporating a CNN and implementing the necessary improvements to our kinematics solution, we anticipate achieving more accurate and efficient performance in future competitions.

V. Conclusion

In conclusion, we have successfully developed a 5-DOF robotic arm that uses a combination of kinematics and computer vision to stack blocks autonomously of different sizes, shapes, and colors. The arm is capable of detecting and locating blocks in a 2D plane and then precisely finding and manipulating them in a 3D plane using inverse kinematics. Our block detection algorithm employs multiple methods, including color thresholding and feature detection, to reduce false positives and increase overall detector accuracy. Additionally, we implemented various filters, such as size, shape, and aspect ratio, to eliminate false positives and reduce noise in detected contours or features. Our robotic arm can handle various block sizes, shapes, and colors with ease, and its autonomous capabilities make it an ideal tool for industrial applications where repetitive tasks need to be performed. The combination of kinematics and computer vision has proven to be a reliable and accurate method for performing such tasks. In the future, we plan to further improve our algorithm by integrating machine learning techniques to improve detection accuracy, as well as to expand the capability of the arm to handle more complex tasks. Overall, our project showcases the power and versatility of robotics and computer vision technologies, and we believe that this project will pave the way for further advancements in the field of automation and robotics.

References

[1] P. Gaskell. (2023). ROB550W23_Lectures. [PDF] Available: https://drive.google.com/drive/u/0/folders/transform

[2] "Intel realsense Lidar Camera L515," Intel RealSense Depth and Tracking Cameras, 06-Apr-2022. [Online]. Available: https://www.intelrealsense.com/lidar-camera-l515/. [Accessed: 21-Feb-2023].

[3] "*Drawings*", trossenrobotics.com https://www.trossenrobotics.com/reactorx-200-robot-arm.aspx

[4] E. Olson, "AprilTag," AprilTag, 2010. [Online].Available: https://april.eecs.umich.edu/software/apriltag. [Accessed: 21-Feb-2023]

[5] D. Torres, "camera_calibration," ros.org, 19-Oct-2020. [Online]. Available: htp://wiki.ros.org/camera_calibration. [Accessed: 21-Feb-2023]2.

[6] A. Huaman, "More morphology transformations," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d3/dbe/tutorial_opening_closing_hats.html. [Accessed: 20-Feb-2023].

[7] "Contour approximation method," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html. [Accessed: 20-Feb-2023].

[8] "Contour features," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/dd/d49/tutorial_py_contours_features.html. [Accessed: 20-Feb-2023].

[9] "Color space conversions," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/d8/d01/group_imgproc_color_color_conversions.html [Accessed: 20-Feb-2023].