Armlab Computer Vision Report

Yulun Zhuang, Sibo Wang, Jinze Liu
{yulunz, sibo, jzliu}@umich.edu

***Abstract*—This report focuses on the computer vision aspect of the Armlab project in ROB 550. With the given LiDAR camera, we implemented a robust vision system which is capable of autonomous hand-eye calibration with Interbotix ReactorX-200 5-DOF Robot Arm and perform block detection and classification to identify blocks' locations (2D and 3D coordinates), colors (rainbow colors from red to purple), sizes (large and small) and orientations in real time (25 FPS) with 99.5% color classification accuracy.**

I. Introduction

In the Armlab, a 5-DOF robotic arm fully autonomously arranges blocks of different sizes, colors and positions into the desired arrangement. Analytical inverse kinematics is used to determine the appropriate waypoints for our desired end-effector position. A heuristic motion planning method was developed to generate feasible waypoints. An overhead Intel RealSense LiDAR Camera is utilized to identify blocks on the board. Homogeneous transformations are used to relate pixel and depth coordinates to real-world coordinates. For reliability, the extrinsic matrix is calibrated using four AprilTags with known positions.

Hardwares:

- Interbotix ReactorX-200 5-DOF Robot Arm
- Intel RealSense LiDAR Camera L515
- ASUS ROG Laptop with Intel i7-10750H CPU

II. Camera Calibration

Given an object detected in an image, we use the pinhole camera model to find its location in workspace coordinates. In this model, a scene view is formed by projecting 3D points into the image plane using a perspective transformation.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Intrinsic}} \underbrace{\begin{bmatrix} \mathbf{R} \mid \mathbf{t} \end{bmatrix}}_{\text{Extrinsic}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (1)$$

where (X_w, Y_w, Z_w) are the coordinates of a 3D point in the worldspace coordinate, (u, v) are the coordinates of the projection point in pixels, (c_x, c_y) is a principal point at the image center and (f_x, f_y) are the focal lengths expressed in pixel units. Note that the equal sign here means "equal up to scale".

The intrinsic matrix (camera matrix) does not depend on the scene viewed. So, once estimated, it can be reused as long as the focal length is fixed. The joint rotation-translation matrix $[\mathbf{R}|\mathbf{t}]$ is called the extrinsic parameters. It is used to describe the rigid motion of an object in front of a still camera.

A. Intrinsic Calibration

We utilize the ROS *camera_calibration* package to perform easy calibration using a checkerboard calibration target[1].

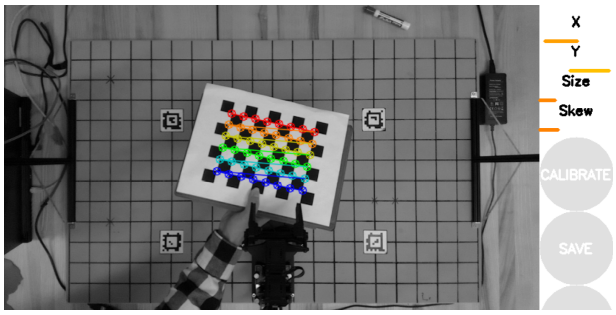


Fig. 1. Intrinsic calibration process using a checkerboard calibration target

TABLE I
Intrinsic Matrix Comparison

Method	Camera Matrix
Manual Calibration	$\begin{bmatrix} 938.634 & 0 & 662.617 \\ 0 & 938.462 & 364.877 \\ 0 & 0 & 1 \end{bmatrix}$
Factory Calibration	$\begin{bmatrix} 908.355 & 0 & 642.593 \\ 0 & 908.404 & 353.127 \\ 0 & 0 & 1 \end{bmatrix}$

We demonstrate the average intrinsic matrix after performing the checkerboard calibration 3 times and compare it with the factory intrinsic matrix in Table I. The focal length parameters in factory calibration is shorter than those in manual calibration by 20 mm, and

the image center parameters are also inaccurate. This deviation may caused by accidental collision, aging of lenses and mechanical vibration during use.

B. Extrinsic Calibration

Rough Calibration

We first use physical measurements of camera position (distance in X, Y, Z-axes between camera and workspace center) to form a rough extrinsic matrix with following assumptions.

(a) Image sensor plane and board plane are parallel.
(b) X, Y, Z-axes of the world frame are parallel to u,v,d axes of the sensor.
(c) Camera frame is at front surface of the sensor.

TABLE II
EXTRINSIC MATRIX COMPARISON

Method	Extrinsic Matrix
Manual Calibration	$\begin{bmatrix} 1 & 0 & 0 & 20 \\ 0 & -1 & 0 & 211 \\ 0 & 0 & -1 & 974 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
PnP Calibration	$\begin{bmatrix} 1 & -0.003 & 0 & -9.865 \\ -0.003 & -1 & 0 & 176.745 \\ 0 & 0.0124 & -1 & 972.815 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Pose Computation Calibration

Next we use the Perspective-n-Point (PnP) pose computation method[2] to solve for the rotation and translation that minimizes the reprojection error of 4 ApirlTags from 3D-2D point correspondences. The detection and 3D pose estimation of 4 given AprilTags[3] are done by the ROS $apriltag\_ros$ package. Given the 3D coordinates of tags in camera frame returned by the $AprilTagDetectionArray$ message, we compute the corresponding 2D coordinates of 4 tags by multiplying it with the intrinsic matrix and then undo the scaling by $1/Z_c$ (subscript $\{C\}$ denotes the camera frame). Note that the intrinsic matrix we use here is the factory intrinsic matrix, because the $apriltag\_ros$ package can only access the intrinsic information published under $/camera\_info$ topic.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \quad (2)$$

As we know the 3D coordinates of 4 tags in world frame are $(-250, -25, 0)$, $(250, -25, 0)$, $(250, 275, 0)$, $(-250, 275, 0)$, we can solve the rotation and translation vector between world frame and camera frame using $cv2.solvePnP()$ function.

As shown in TableII, the result of manual calibration is valid only under strong assumptions, and can not handle undesired camera rotation (-0.124 rad) and translation (more than 30 mm errors).

Depth Frame Calibration

We need a depth calibration because the depth frame and RGB color frame are not perfectly aligned[4], with 3-7 mm deviations, especially at the edge of frame. This can damage block color classifier if we perform block detection using depth segmentation and then retrieve block color in that segmented mask.

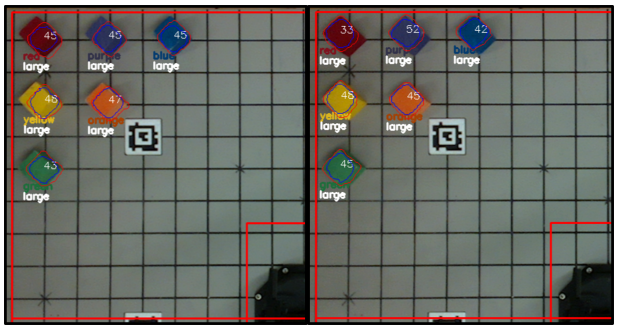


Fig. 2. Block detection in depth frame. Left: before depth calibration; Right: after depth calibration.

TABLE III
DEPTH TRANSFORMATION COMPARISON

Method	Transformation Matrix
Homography Transformation	$\begin{bmatrix} 1.027 & 0.007 & -18.962 \\ 0 & 1.038 & -12.647 \\ 0 & 0 & 1 \end{bmatrix}$
Affine Transformation	$\begin{bmatrix} 1.029 & 0.005 & -19.301 \\ 0 & 1.032 & -12.136 \end{bmatrix}$

Instead of using affine transformation, we use homography[5] to warp perspective transform of depth frame and then register depth and color frames together. The reason is that affine transform can consider only 3 correspondences points from depth and color frames, while homography can take more than 4 point pairs so that gives a more robust estimation. In practice, since the rotation and translation between two frames are small, the result homography matrix (3x3) is degenerated to be a affine transform matrix (2x3) with the third row fixed. Table III compares the estimation results, where

the homography is computed using workspace corner point pairs and affine transform is computed using three of them.

III. Workspace Reconstruction

We reconstruct the workspace (with bounding boxed) from pixel coordinates by utilizing the inversion of the pinhole model. The extrinsic parameters are calibrated autonomously using Apriltags visible in the workspace (with ids 1-4 shown in Figure 4), and the depth calibration is done once but the homography matrix is reused, since homography is valid against planner rotation about its origin (as long as no translation).

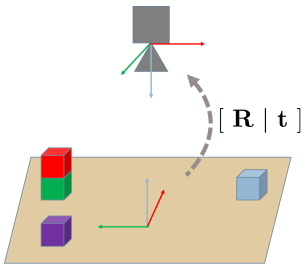


Fig. 3. Workspace reconstruction (image taken from lecture slides). Define the boundary of workspace and perform hand-eye calibration to transform pixel coordinates to world coordinates.

A. Pixel Coordinate to World Coordinate

Given the intrinsic and the extrinsic matrix, we are able to project a pixel coordinate to a object point in world coordinate using the inversion of the full camera model.

We first project pixel frame to camera frame, and undo the scaling with depth information.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z_c \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (3)$$

Then change it to homogeneous coordinates and multiply inverse extrinsic matrix.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (4)$$

We verify our calibration results by hovering mouse alone each horizontal and vertical lines on the workspace board to check the deviation of values in y-axis when hovering alone x-axis and vice versa. The resulting deviations are quiet small with an magnitude of 1-3 mm. However, we can achieve a more accurate extrinsic calibration by detecting the grid points on the board and find the transformation using $cv2.solvePnPRANSAC()$, which is more robust against outlier point correspondents.

B. Bounding Box for Workspace

The image frames are masked by two specific bounding boxed. The larger one define the edge of the workspace, and the small one masks out the robot arm. The distance between each corners and the origin of robot arm is no more than 550 mm, which is the maximum reach distance for arm. The positive and negative half planes of the workspace are separated by a horizontal line to allow for quick switching of detection ranges during the competition.

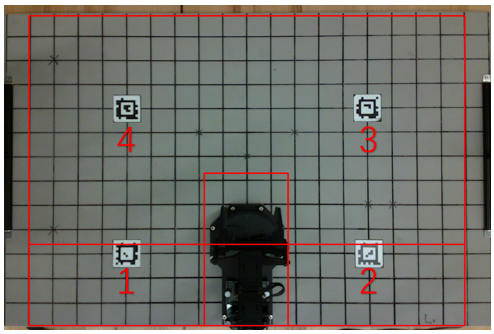


Fig. 4. Boundary of workspace. The board has a 50 mm grid. Apriltags with ids 1-4 are placed on the board at the indicated locations 1-4.

IV. Block Detection

The block detector we implemented is capable of robustly detecting blocks' location (both 2D and 3D coordinates), color (rainbow colors from red to purple), size (large and small) and orientation (angle with y-axis) in real time (25 FPS) with 99.5% color classification accuracy.

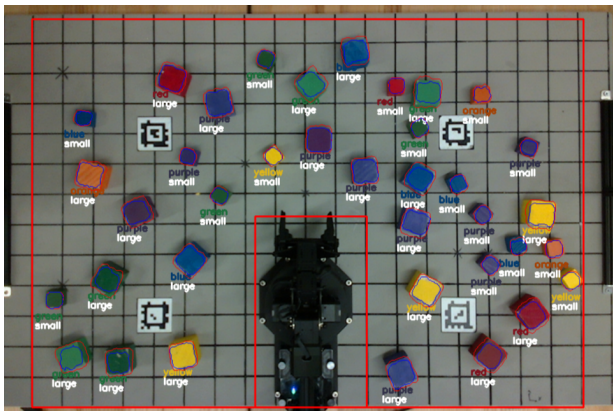


Fig. 5. Block detection system demonstration for detecting blocks with different colors, sizes, orientations and under different illumination.

A. Block Detection through Depth Segmentation

The general procedure of detecting blocks' (or block stacks) location in depth frame is following.

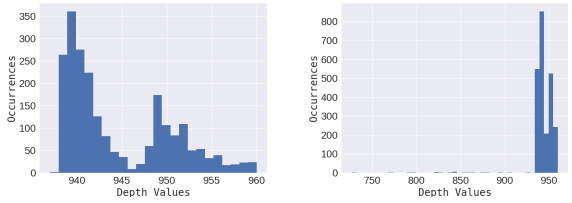
(a) Remove depth noise using median filter.
(b) Mask out the area outside the workspace boundary.
(c) Segment the depth frame between least half block high (12.5 mm) to least 7 block high (260 mm).
(d) Find all contours in depth segmentation.

For each contour we find in above procedure, we next perform dedicated thresholding and classification to determine its center coordinates of top surface, size and orientations.

Top Surface Segmentation

Since blocks can be stacked, we should be able to locate the top surface of each block/stack for picking and placing. One simplest idea is directly retrieve the minimum depth value (distance between camera and stack top) and use it as the height of the top surface. This approach works well without depth calibration, but fails after affine transformation.

The reason is that affine trainsformation is not value invariant. Although the location of each block is perfect aligned after calibration, the height values of the same top surface have huge deviations which can not be ignored.



(a) Before calibration (b) After calibration

Fig. 6. Histograms of depth distribution of one big block and one small block on the board. Outliers range from 600 to 900 occur after depth calibration.

Therefore, we use the interquartile range (IQR) to remove these outliers and result in reliable top surface segmentation[6].

$$IQR = Q_3 - Q_1 \tag{5}$$

where Q_3 is the third quartile point and Q_1 is the first quartile point. The lower bound of normal data range is defined as $Q_1 - 1.5 \times IQR$.

Given the accurate height value, we perform $cv2.inRange()$ thresholding again to segment block top in each contour. This may return multiple new contours, and we treat the largest contour as the valid top surfaces segmentation by comparing $cv2.contourArea()$. Finally, image moments[7] for each contour are calculated to find its centroid coordinates in pixel frame by $cv2.moments()$. The workspace coordinates of each

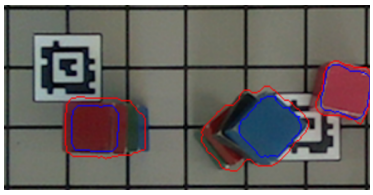


Fig. 7. Top surface segmentation. The red contours show the block/stack segmentation and the blue contours show the accurate top surfaces segmentation.

block is transformed by the function described in section III-A.

$$M_{ij} = \Sigma_x \Sigma_y x^i y^j I(x,y) \tag{6}$$

$$c_x = \frac{M_{10}}{M_{00}}, c_y = \frac{M_{01}}{M_{00}} \tag{7}$$

where $I(x,y)$ is pixel intensities at (x,y).

Note that at each segmentation step, contours with first moment of area (M_{00}) smaller than 200 or larger than 7000 will be rejected as false positive detection.

Block Size and Orientation

The block size classification is a bit tricky, since the largest area of small block (6 block high) is grater than the smallest area of large block (1 block high), while our classification algorithm only take contour areas into account. As a result, we take the smallest area of large block (850) as the threshold to make sure all large blocks can be classified correctly, because the consequences of mistaking a large block as small are more severe than the other way around.

However, the size of blocks will still be correctly classified in executing stage, because the distance between gripper fingers can be read after the block is grasped.

The orientation of each block is calculated by $cv2.minAreaRect()$. Note that the return angle of that rectangle box ranges from -90 to 0 degrees , but since OpenCV version 4.5.1, the value of angle becomes positive from 0 to 90.

B. Color Classification using Color Space Features

The idea of block color classification is to retrieve a color feature from each segmentation mask in color frame and classify them using vector range, vector distance or a Machine Learning classifier.

We propose a robust illumination invariant (RII) 9-dimensional color feature utilizing RGB, Lab and HSV color space. For each color space, we compute a 3-dimensional feature from the average of the block segmentation of 3 channels respectively.

Minimum Vector Distance Classification

As our baseline classification method, we manually record the average values of 3 channels of 6 rainbow colors in each color space, and concatenate them as the classification criteria matrix.

Procedure for a given segmentation mask:

(a) Compute the RII feature.
(b) Compute the Euclidean distance between each row of the criteria matrix and the incoming feature.
(c) Take the index of the minimum distance as the classification result.

This method has a classification accuracy of about 95% and may misclassify green as purple under certain lighting conditions.

Support Vector Classification

Support vector machines (SVMs)[8] are supervised machine learning method widely use in classification problems. SVM maps training samples to points in hyperplane so as to maximise the width of the gap between categories[9].

To maximize the color classification accuracy, we deploy the SVMs with radial basis function (RBF) kernel implemented by Scikit-learn library[10].

The training dataset and labels are simple to prepare, since they can be obtained directly from the baseline prediction method, and then the incorrectly classified data are manually removed.

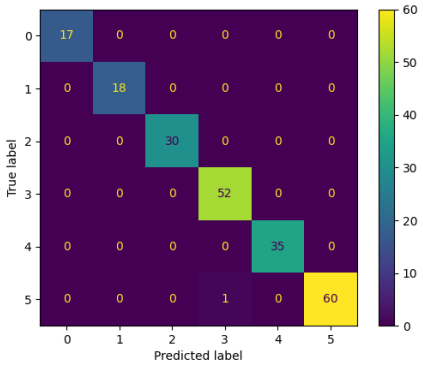


Fig. 8. Confusion matrix for trained SVM classifier trained over 212 data points.

After trained over 212 data points, the prediction accuracy of SVMs is 99.5%.

C. Blocks Class Design and Composition

We describe those detected blocks as a individual class define in Figure 9, and integrated it with $Camera$ class through composition[11]. In the detection stage, we treat blocks' attributes as list structures ($reset()$), and append new values for each valid block. After detection, we transfer blocks' attributes to numpy arrays ($update()$), and sort the order of each attributes ($sort()$) according to the sort keys (rainbow color order or distance order). The accessibility of detection results is controlled by $detected\_num$, and can be easily interfaced with $StateMachine$.

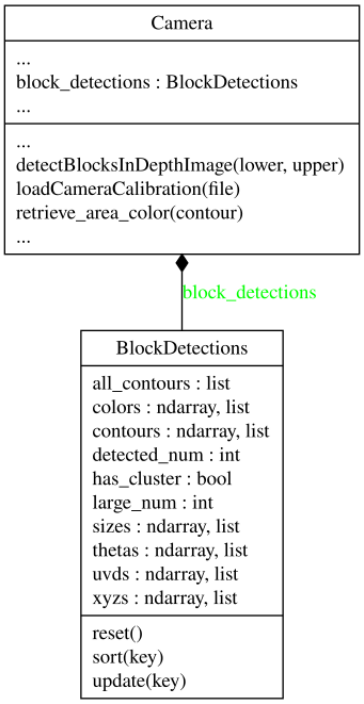


Fig. 9. $BlockDetections$ class design and composition with $Camera$ class.

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] E. Marchand, H. Uchiyama, and F. Spindler, "Pose estimation for augmented reality: a hands-on survey," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 12, pp. 2633–2651, 2015.
[3] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 3400–3407.
[4] A. Breitbarth, C. Hake, and G. Notni, "Measurement accuracy and practical assessment of the lidar camera intel realsense l515," in *Optical Measurement Systems for Industrial Inspection XII*, vol. 11782. SPIE, 2021, pp. 218–229.
[5] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.
[6] X. Wan, W. Wang, J. Liu, and T. Tong, "Estimating the sample mean and standard deviation from the sample size, median, range and/or interquartile range," *BMC medical research methodology*, vol. 14, no. 1, pp. 1–13, 2014.
[7] M. R. Teague, "Image analysis via the general theory of moments," *Josa*, vol. 70, no. 8, pp. 920–930, 1980.
[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[9] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.
[10] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.
[11] M. Lutz, *Learning python: Powerful object-oriented programming*. " O'Reilly Media, Inc.", 2013.