Team 5 ROB 550 Armlab Report

Haoyang Li, Weilun Peng, Tianyu Zhang

***Abstract*—Robotic arms are widely used in harsh environment doing repetitive and complicated tasks in replace of human hands. As a result, more advanced design of robotic arm system is required. The main goal of this project is to design a 6 degree of freedom robotic arm system to detect, grasp and place the blocks. This goal is achieved by integration of gripper design, object perception, reasoning and motion planning. The gripper is designed to perform the gripping motion. The objection perception is implemented to location the blocks. Autonomous execution logic navigates the arm to designated locations. Additionally, the robotic arm successfully completed the task with high accuracy and efficiency. Finally, we proposed approaches to improve the performance of our robotic arm in the future.**

Keywords- Robotics arm, Kinematics, Perception

I. Introduction

ARTICULATED robotic arms have been extensively used in industrial operations and automations, which serve an important role for perceiving and manipulating objects. In this project, we built a 6 DOF robotic arm, which can achieve sensing, reasoning and acting. A gripper is designed as the end effector to perform gripping motion. Together with RGB camera and depth sensor, the robotic arm system can detect blocks with different colors on a fixed plane, locate the position in 3D and finish the tasks of grabbing and placing the blocks without collision. To successfully finish the tasks we described above, we need smart motion planning to avoid obstacle, smart gripper design to hold the blocks firmly, accurate forward and inverse kinematic to guide the Rexarm to desired location, accurate block detection and camera calibration to decide where our gripper should go to, and last but not the least smart autonomous execution logic to implement them together.

In this report, we first presented the methodology used in Section II and then explained results for each method in Section III, and finally made conclusions with discussion in Section IV.

II. Methodology

A. Teleoperation & Motion Planning

1) Teleoperation: After implementing $controls.py$ we got the graphical user interface (GUI), which allowed us to figure out the physical limitations for each joint by sliding the sliders, respectively. The limited range of joint angles are shown in Table I.

Joints	Min θ (°)	Max θ (°)
Base	-180	180
Shoulder	-121	127
Elbow	-121	127
Wrist	-121	107

TABLE I: The minimum and maximum angles in degrees for each individual joint.

2) Spline: To smoothly move the arm to desired location, we tried to implement quintic polynomial scheme to control the velocity profile for each joints. Using quintic polynomial trajectories, each joint can start at a user defined speed, acceleration and reach destination at the same amount of time. The quintic splines equations describing the relation between time t and arm's angles q are shown as:

$$q(t) = \sum_{i=0}^{5} a_i \times t^i \tag{1}$$

The start time t_0 we used was always zero and t_f was the time period after which our joints should all reach from initial angle q_0 with initial velocity v_o their final angle q_f with final velocity q_f. Besides, v_o the initial velocity and v_f the final velocity were set to zero to guarantee the smoothness of angle movement[1]. a_0 the initial acceleration and a_f the final acceleration were also set to zero for the smoothness of velocity. Having all these boundaries, the quintic equations coefficients a_0, a_1, a_2, a_3, a_4, a_5 can be obtained by solving linear algebra below:

$$\begin{bmatrix} q_0 \\ v_0 \\ a_0 \\ q_f \\ v_f \\ a_f \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{2}$$

To realize the implementation of quintic spline motion planning, 100ms timer was used to record the current time stamp. Desired final time tf was selected based on the maximum joints displacement divided by a set of angular speed. The angles and velocities of each joints at each time stamp were calculated every 100ms inside the timer separately using look ahead strategy which means the command we sent was the position and velocity at next time stamp. However, because the servos cannot directly change their speed to our command speed and it needs some time to implement its PID control to

achieve the received position and velocity, the desired positions at every time stamp couldnt be achieved. Thus, our program waited until the last desired positions were reached to give next time stamp command which resulted in knots at every command changing time stamp.

We also used a timer to implement the spline without smoothing. The timer continuously sent the same destination command with constant velocity in every 70ms if the condition below didn't achieve.

$$\sum_{i=0}^{5} |q_i^{feedback} - q_i^{desired}|^2 < 0.05 \tag{3}$$

Once the inequality above is satisfied which means the Rexarm arrives the desired location, the timer stop and wait for next location command to wake it up. The simulated quintic spline and the spline without smoothing (set velocities at constant values in command for every way point) are shown in Fig. 1 and 2.

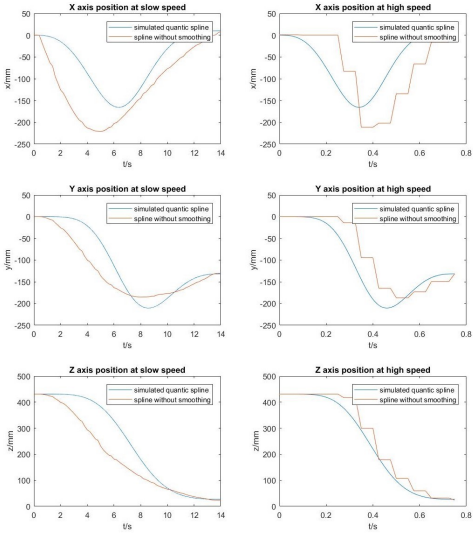

Fig. 1: Robot end-effector position in world coordinates at low speed(14 seconds move from $x = 0, y = 0, z = 420$ to $x = 10, y = -131, z = 27$) and high speed(0.75 seconds move from $x = 0, y = 0, z = 420$ to $x = 10, y = -131, z = 27$)

From the figure we know that the position spline without smoothing looks like our ideal quintic spline, which is definitely better than implementing quintic spline to motion planning command which introduces unsmooth knots into our arm movement. Although its velocity fluctuated a little, its total trend generally followed the quintic spline velocity profile no matter it is at slow or high speed. Therefore, we always set the velocity at a fixed constant value rather than using quintic spline for our tasks.

3) Obstacle Avoidance: Although directly sent position and velocity command to servos gave us a quint polynomial trajectories like motion, the arm still would

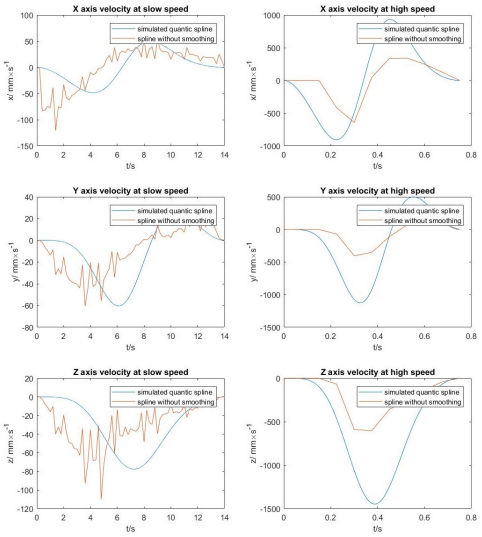

Fig. 2: Robot end-effector velocity in world coordinates at low speed(14 seconds move from $x = 0, y = 0, z = 420$ to $x = 10, y = -131, z = 27$) and high speed(0.75 seconds move from $x = 0, y = 0, z = 420$ to $x = 10, y = -131, z = 27$)

crush into obstacle if the obstacles were on the way of its given position. In order to avoid the obstacles, the strategy we used is to move the base, shoulder, elbow and wrist servos in different order and when some of them moved the others stay at its previous angles.

Because different tasks for the final events had different setting, we implement different moving order to our servos. Specifically, for event 1 and 2 [2], because the obstacles were not higher than 1 block (38mm), we can purely set the absolute value of shoulder servo angle less than 10 degree to avoid any potential collision. Thus, when the arm was going to grape the target block it first moved its base, elbow and wrist servos to its final position with shoulder servo stayed at 10 degree and then it lowered its shoulder to its desired degree and grape the block. When it was going to put the block, fist it moved up its shoulder to 10 degree from the graping shoulder angle and then moved its base, elbow and wrist to the desired angles at the same time. At last it lowered its shoulder to put the block.

For task 3, 4 and 5 because the obstacles might be higher, we needed to add extra way points when the arm was moving to its target point. First, when the arm was going to grape the block, its original position would alway be $x = 0, y = 0, z = 420$, and it would move its base servo to the target angle and then moved shoulder servo to the desired angles. After doing this, it would move elbow and wrist servos to desired values and then grape the block. Next, shoulder would move to 0 degree first and then the elbow and wrist would

move to 0 which grantee next grasp or dropping action will start at $x = 0, y = 0, z = 420$. By doing this, we could avoid any exiting obstacle on th board because we didn't sweep our arm around the board. We only change the base when the other servos are straight. The same procedure were implemented to the dropping motion to avoid potential collision. During all the movement, our speed command were set to base: 0.2, shoulder: 0.13, elbow: 0.13, wrist: 0.13, gripper serves: 0.7. All the maximum torque command are set to 1.

B. Grasp Execution

1) Gripper Design: The gripper is the end effector on the robotic arms. The main principle of designing the gripper was simplicity and reliability. To achieve the purpose that grasping and setting blocks in the work-plane, we designed a gripper with two clamps, one is fixed and another one can transform the rotation of the servo motor XL-320 into the gripping motion. To achieve better performance, we added another servo motor XL-320 to rotate the whole gripper in its own axial direction. The gripper is shown as Fig. 3.

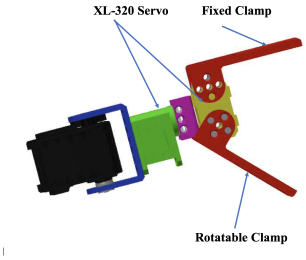

Fig. 3: : Gripper Design

The main advantage of our gripper is its simple structure. Without any teeth-gears or any sideways, connecting a clamp directly on motor servo ensure sufficient energy in gripping motion. Also, the simple structure allows us to assemble the gripper easily. And, our design was based on the competition: on the one hand the fixed clamp is able to narrow gaps between blocks in pyramid construction, on the other hand, the ability of rotating the whole gripper allows the block to be grasped in any orientation and placed without disturbing from motion of clamps in task3.

sHowever, the design still had some disadvantages. The main problem is its physical limitation in space. Clearly, the gripper cannot touch every position in the work-plane, when it grasps blocks in vertical and horizontal directions. Another disadvantage is its unstable joint. Compared with structure with sideways or gears, our simple design of the gripper has insufficient joint which leads to slight instability. To avoid this problem, the speed of gripper will be decreased as it picked up or places the blocks.

2) Forward Kinematics: Forward kinematics was used to get the workspace coordinates $X = \{x, y, z\}$ of the end factor based on joint angles $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$. Since the rotation and gripping motion of the gripper would not affect its end effector position and orientation relative to ground, those two joint angles were not included in the forward and inverse kinematics. Based on the limitation on joint angles, forward kinematics can avoid collision in workspace. In this project, we followed Devanit-Hartenberg convention to select the frames of reference. The global coordinate frame and local coordinate frames for each joint in rexarm are shown in Fig. 4. Four DH parameters θ, d, a and α for each joint are generated to form D-H table as Table II.

Joint Number	θ_i	$d_i(mm)$	$a_i(mm)$	α_i
1	θ_1	l_1	0	$\frac{\pi}{2}$
2	$\theta_2 + \frac{\pi}{2}$	0	l_2	0
3	θ_3	0	l_3	0
4	θ_1	0	l_4	0

TABLE II: DenavitHartenberg parameters specified the forward kinematics of the rexarm. The link lengths are measured as l_1 = 119.5, l_2 = 97.70, l_3 = 98.03, l_4 = 115.8

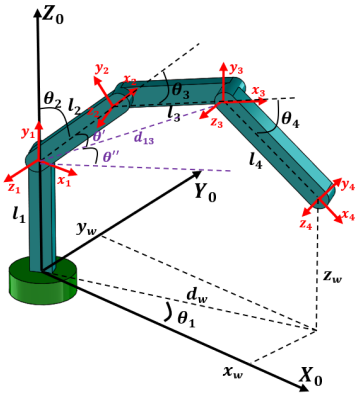

Fig. 4: Configuration for forward and inverse kinematics. X_0, Y_0 and Z_0 are the base frames. x_i, y_i and z_i (i = 1, 2, 3, 4) are local frames for each joint. All axises followed the right hand rule.

In the convention, we used four Denavit-Hartenberg parameters to generate each homogeneous transformation A_i given Eq. 4. Then we used four homogeneous transformations to form a transformation matrix from end effector coordinate system to global coordinate system shown in Eq.10 .

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \quad (4)$$

$$T^{world}_{endeffector} = A_1A_2A_3A_4 \tag{5}$$

3) Inverse Kinematics: Inverse Kinematics was used to transform workspace coordinates $X = x, y, z$ of the end factor to joint angles $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$. In addition, if the orientation of end effector ϕ shown in Fig. 4 is known, we can generate one solution of θ for the rexarm with following geometric analysis on the structure in Fig. 4.

We can firstly calculate θ_1 and d_w by only world coordinates x_w and y_w

$$\theta_1 = arctan2(y_w, x_w) \tag{6}$$

$$d_w = \sqrt{x_w^2 + y_w^2} \tag{7}$$

Then we can calculate d_{13} and thus find d_w by the law of cosines

$$d_{13} = \sqrt{(z_w + l_4cos(\phi) - l_1)^2 + (d - l_4sin(\phi))^2} \tag{8}$$

$$\theta_3 = arccos(-\frac{d_{13}^2 - l_2^2 - l_3^2}{2l_2l_3}) \tag{9}$$

Then we can calculate $\theta^{'}$ and $\theta^{''}$ to find θ_2 and θ_4.

$$\theta^{'} = arccos(\frac{l_3^2 - d_{13}^2 - l_2^2}{2l_2d_{13}}) \tag{10}$$

$$\theta^{''} = arctan2(z_w + l_4cos(\phi) - l_1, d - l_4sin(\phi)) \tag{11}$$

$$\theta_2 = \frac{\pi}{2} - \theta^{'} - \theta^{''} \tag{12}$$

$$\theta_4 = \pi - \theta_2 - \theta_3 - \phi \tag{13}$$

4) Flexible Workspace: In order to find the reachable workspace, we set ϕ as 0, $\frac{\pi}{2}$ and (0, $\frac{\pi}{2}$). In this project, first two configurations are preferred, since the blocks are cubed. It's more accurate to pick up and drop in these two configurations. If those two cannot be reached, $\phi \in (0, \frac{\pi}{2})$ will be used. The working space for $\phi = 0$ and $\frac{\pi}{2}$ shown in the Appendix B Fig. 6.

C. Object Perception

In order to locate the blocks in work-plane and classify the colors of blocks precisely, camera calibration, block detection and color classification are necessary for the tasks in this project.

1) Camera Calibration To connect block positions in images with real world coordinates, the camera calibration is implemented. We had access to two cameras, which can provide us with depth image and RGB image. And the first step of calibration is aligning the pixel values in RGB image with depth image by using affine transformation. By using $cv2.getAffineTransform()$[3] function, to get affine matrix and align the pixel value by the equation.

$$\begin{bmatrix} u_{depth} \\ v_{depth} \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{14}$$

Then, we can calculate the real-world depth by using (15),where $depth_{board}$ represents the depth value of work-plane. It relates the 2D pixel value with depth.

$$z = depth_{board} - 0.1236tan(d/2842.5 + 1.1863) \tag{15}$$

After getting the real-world depth, we can calculate the real-world x and y by combining Eq. (17) and Eq. (18). The intrinsic matrix in Eq. (17) was acquired by implementing $camera\_cal.py$. Also, through applying function $cv2.slovepnp()$, we computed the rotation and transformation matrix. In detail, three points were collected to compute the affine matrix. And five points were collected to relate the real world x y and depth with x y and z in RGB image. To reduce the error in calibration, representative points, four corners of the work-plane and a midpoint of one side, were selected, and both of their RGB coordinates and real world coordinate were processed as an input to $cv2.solvepnp$.The detail is in Fig.8 and Fig.7. The intrinsic matrix is shown as following:

$$\begin{bmatrix} 521.5763487 & 0 & 310.1226676 \\ 0 & 520.9781503 & 272.9414709 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & | & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{17}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} & \boldsymbol{R} & & \mathbf{T} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{18}$$

Then we defined world-to-RGB transformation matrix in Eq. (19), where T_1 is a 3-dimensional vector. Clearly we can figure out X_w, Y_w through Eq. (19), since we already know real-world depth Z_w and image pixel coordinates u and v, the derivation is shown in Eq. (20). To be more specific, subtracting Z_wT_3 allows us to get a 3×3 transformation matrix, which reduce the computational complexity.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{19}$$

$$(\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - Z_wT_3) \begin{bmatrix} T_1 & T_2 & T_4 \end{bmatrix}^{-1} = \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{20}$$

2) Block Detection In block detection, we found that we can set different depth thresholds to separate and detect the block based on the pixel values in depth image. In detail, owing to the top surfaces of blocks are parallel to the work-plane, the depth values of areas

with blocks are separable with the work-plane which is set as zero. And, the depth values of objects plied with different numbers of blocks are distinguishable. Also, to avoid the camera regard the arm as objects, we defined an acceptable area range for block detection. Additionally, to guarantee we can get precise location of block, we computed the blocks' orientations based on contour information. At last, to improve the accuracy, we set the work-plane as the detecting zone. The thresholds of depth values is shown in Table V and the pseudo algorithm is shown as follows:

Algorithm 1: Block Detection Algorithm

1: define area threshold ***R***, lower and upper boundaries
2:define detecting zone: work-plane
3: ***for i, j*** in lower boundaries and upper boundaries
4: find the areas with depth in [***i, j***]
5: compare the area with R
6: ***end for***
7: combine the areas by using bitwise or operation
8: figure out contour of the areas
9: compute rotation of rectangular from the contour

In tasks, the detection of orientations of blocks were not applied, since the detection of orientation was only required when the gripper grasped the blocks in vertical direction. To reduce computational complexity, this step was skipped.

3)Color Detection The color detection mainly depends on the difference of HSV values of different colors. Based on the former processing, the blocks can be precisely detected in work-plane. And the crucial operation is transforming the pixel values regarded as blocks into HSV values in RGB image. The thresholds of HSV is shown in Table VI

Algorithm 2: Color Classification Algorithm

1: define different colors' thresholds
2: transform the block pixels values into RGB pixels values
3: convert the RGB image to BGR
4: ***for i*** in detected blocks
5: find the the HSV values
6: compare the HSV values with thresholds
7: ***end for***

In real tasks, this algorithm has its limitations, since the HSV values is instable to the change of environment. Also, there exits overlap in HSV values between some colors, which may leads to the error in color detection.

D. Autonomous Execution

All our events functions were all connected individually to 60ms timers. Thus, they would check their process status and decided what command to send to the Rexarm every 60ms. When a event finished, the timer stopped immediately. The block detection result was saved in a queue including the detected blocks' colors, real world x, y, z coordinates. These information were used inside the events timers.The motion planning of grape and drop have been described in II-A.

Event1 and Event2 timer iterative process is shown in Appendix G Fig. 10. Their basic logic was the same except that the putting location of every block was different. The putting location of event1 is the mirrored location of the graping location of the block. The putting location in event2 is preallocated as $x = 0, y = 230$. z was one block height(38mm) times how many blocks it had put before.

Event3 timer iterative process is shown in Appendix G Fig. 11. The Rexarm would stop graping if all the 8 color blocks had been put correctly or no block was detected at the front which might result in failure if some unmoved blocked located in rear. We didn't put the block in any directional order. All the blocks had the same priority to be put firstly or lastly. The locations where we should put the blocks were allocated based on their color [2]. Specifically, each block's gap was 8mm and the center of the line was at $x = 100, y = 0$.

Event4 timer iterative process is shown in Appendix G Fig. 12. The Rexarm would stop graping if it reached 8 stack correctly or the desired color block couldn't be found after all the blocks in the board can be detected by our camera. The preallocated empty spots were located in the rear of the board separately. The block with right color would be put on $x = 230, y = 0$. z was predefined based on their color[2].

Event5 timer iterative process is shown in Appendix G Fig. 13. The Rexarm would stop graping if it reached Final block number 28(block numbers of 7 layers high pyramid) or no block was detected in the rear of the board. Our algorithm allowed us the put blocks continuously into the rear of board without detection failure. The form of the pyramid we built was shown in Fig 9. The arch angle between each block was 13.5 degree and their radius toward center of the board was 230mm.

III. Results

A. Motion Accuracy

The accuracy of the robotic arm motion was tested at several different positions in our range of workspace. We used caliper to measure those positions and used them as input of Inverse Kinematics. With the output joint angles, we used Forward Kinematics to find the calculated

positions. So we obtained the difference between the measured and calculated x, y and z to generate the motion accuracy. In addition, we commanded gripper to pick up the block at those positions and obtained the gripping performance. Each point was tested for ten times and results are shown in Table III.

World Coordinates	Error (mm)	Gripping Performance
(100, 0, 38)	(2.32, 3.19, 1.50)	success
(0, 100, 38)	(3.63, 3.93, 2.34)	success
(0, 150, 76)	(4.84, 5.75, 4.25)	success
(-150, 0, 76)	(4.53, 5.44, 3.73)	success
(0, 200, 114)	(5.12, 3.12, 2.55)	success
(-200, 0, 114)	(3.94, 5.46, 4.47)	success
(0, -250, 114)	(4.45, 3.45, 5.13)	success
(250, 0, 114)	(4.72, 5.82, 3.74)	success

TABLE III: With great result of motion accuracy, the gripping performed well too.

B. Grasp Execution

Once the block detected by camera system, the second servo will rotate according to the degree of orientation of the block. Then, the rotatable clamp of the gripper will rotate 20 degrees, and the fixed one is used to align with the block. Since there is a fixed clamp, we add an $6mm$ offset in the left side, which guarantee we can grasp the block with a tolerance of error in camera calibration. Similarly, we add a $19mm$ offset in z, to avoid the gripper crash on the work-plane or the blocks under the block that camera detect. After the gripper move to the blocks position, the clamp will rotate back, and owing to the fraction caused by torque from the servo, the block can be grasped. When it comes to placing blocks, the whole gripper will rotate firstly, then move to destination, and finally release the block by rotating the rotatable clamp.

C. Object Perception

At the very beginning of the project, only three point was collected to calibration, which probably led to unacceptable error in block detection. To improve the accuracy, we optimize our method by collecting five points, which improve the performance in block detection. The errors of detection are within $\pm 5mm$ which can be revised by the offset we set in our gripper.

In block detection, the thresholds we set allow the camera detect the block precisely, and avoid regard other objects, such as arm, and objects out of range of work-plane, as blocks by mistake. To verify the accuracy, we set 10 test points on the work-plane, and compared the real world coordinates we calculated by the algorithm with its real coordinates. The accuracy is 99%(99/100) with the acceptable error.

Similarly, in color detection, we also did experiments to verify our algorithm. And the accuracy our perception system is up to 98%(98/100). The thresholds of different colors are shown in appendix tableVI

D. Autonomous Execution

After many experiments before competition, including tuning parameters of rotation, velocity, and torque, the blocks in work-plane can be detected, grasped and placed with a success rate over 97%. Also, to improve the performance in competition, we optimized the path of our robotic arm through skipped some unnecessary way points. Consequently, both task1 and task2 can be completed within 20 seconds during the competition, and blocks were placed in correct location with high accuracy. In task three, the blocks were smoothly aligned in the correct sequence. Also, to narrow the gap between blocks, a special path, pushing pink blocks along with the direction of the line, was implemented. However, because of the slight error in block detection and camera calibration, the green block was not precisely placed in the line. In task4, the blocks can be piled in a correct color sequence with in 90 seconds. However, owing to the error in block detection and camera calibration, the blocks were not stacked in a line precisely. In the last task, a seven layers pyramid was constructed within 300 seconds. In this task, our fixed clamps ensured the acceptable gaps between the blocks, thus, our pyramid is stable. However, our pyramid had slight arcs, since as the limitation referred before, our gripper cannot touch every position in the work-plane, when it grasps blocks in vertical and horizontal directions. All of the details of pyramid are shown in the Fig. 9.

IV. Discussion

In conclusion, all the tasks were successfully completed in this project, including the gripper design, camera calibration, block detection, forward and inverse kinematics, state machine planning and all the five competition tasks. Owing to our great performance, we got the first place in the competition. The largest difficulties of our robotic arm system are its color detection, and limitation of our arm, which are clarified before.

There are several ways can be applied to improve the performance of this robotic arm. First on is add an extra light shelter in the perception system, since sometimes yellow blocks whose H value will be 0 under strong light environment cannot be detected. Also, to broaden the range of grasping and placing, a scalable elbow structure can be applied in this robotic arm.

References

[1] J. Z. Kolter and A. Y. Ng, "Task-space trajectories via cubic spline optimization," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 1675–1682.

[2] ROB550. (2018) Armlab description.
[3] R. Laganière, *OpenCV Computer Vision Application Programming Cookbook Second Edition*. Packt Publishing Ltd, 2014.
[4] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ

APPENDIX A

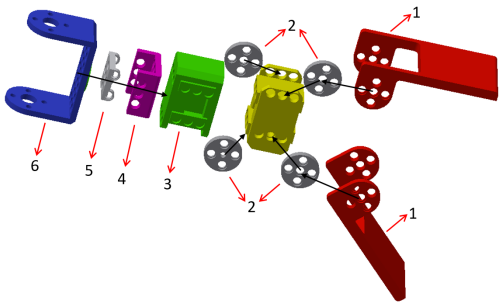

Fig. 5: Bill of material for gripper. Black arrow shows how to assemble the gripper

Material Name	Quantity	Label number in figure
3D-printing clamps	2	1
Wheel	5	2
XL320 Connector	1	4
XL320HubToOllo	1	5
XL320HubToXL320Base	1	6
MX-28 Servo motor	3	N/A
AX-12 Servo motor	1	N/A

TABLE IV: Bill of material for gripper.

APPENDIX B

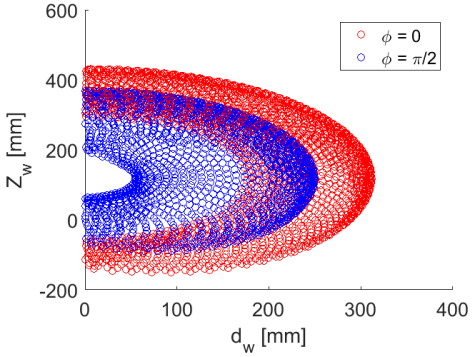

Fig. 6: Range of workspace for given orientation ϕ = 0 and $\frac{\pi}{2}$ rad, where d_w is the distance of end effector to the base in XY-plane and Z_w is the height of the end effector along the Z-axis

APPENDIX C

Number of Blocks	Lower Boundary	Upper Boundary
1	683	693
2	665	675
3	643	653
4	622	632
5	595	605
6	566	576
7	536	546
8	503	513

TABLE V: The thresholds for objects with different numbers of blocks are separable. And the units of thresholds is mm.

Color	H	S	V
Black	0–180	20–110	20–80
Red	169–180	120–255	120–210
Green	40–80	5–150	88–200
Blue	110–130	50–195	120–230
Yellow	10–30	0–255	210–255
Pink	165–173	90–215	210–255
Orange	0–10	135–255	170–255
Purple	140–160	70–150	80–210

TABLE VI: this is the range of HSV value of different color.

APPENDIX D

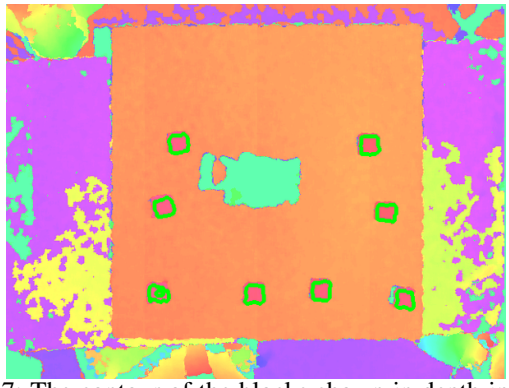

Fig. 7: The contour of the blocks shown in depth image.

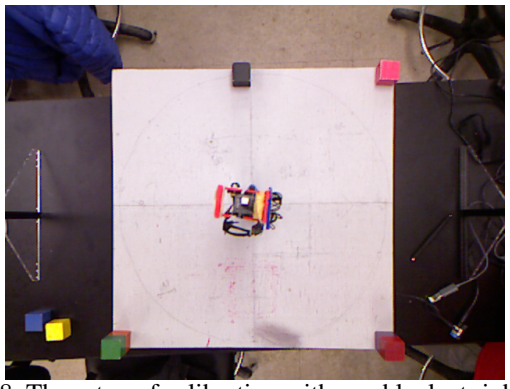

Fig. 8: The setup of calibration with one block at right-up corner and two blocks at bottom left and right corners. Also another one block at the top center.

APPENDIX E

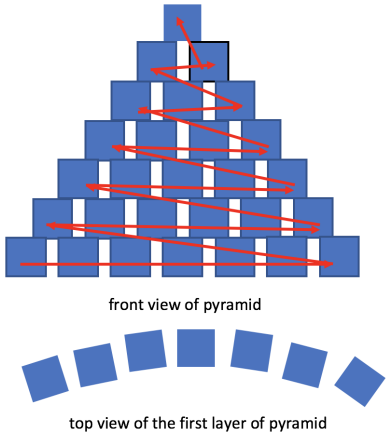

Fig. 9: The red arrows represents the path we design to stack the pyramid. and the arc caused by limitation of our robotic arm is demonstrated in the top view of the first layer of pyramid.

APPENDIX F

Event	timess	points	performance
1	17.9	97.5	good
2	16.9	97.5	good
3	1́1́9	140	green block is a little off
4	1́3́8	130	good
5	5´	350	stack to 7 level

TABLE VII: The performance of our robotic arm in competition

APPENDIX G

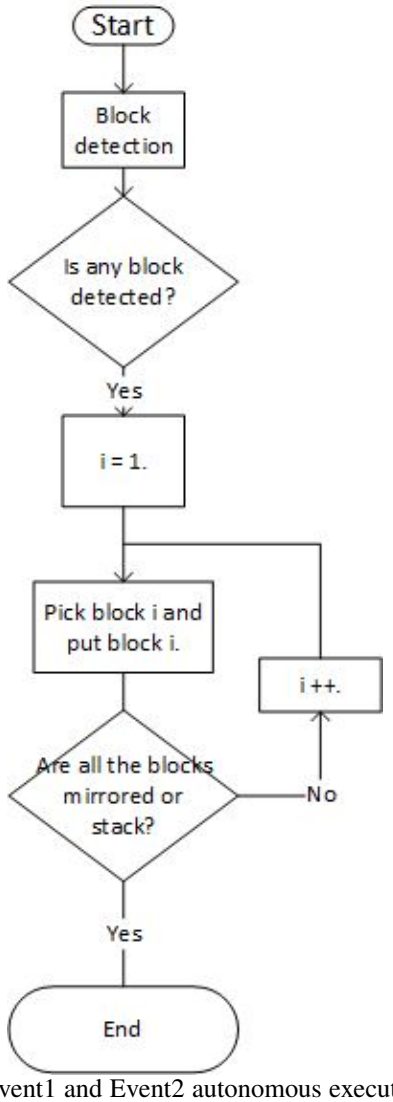

Fig. 10: Event1 and Event2 autonomous execution logic.

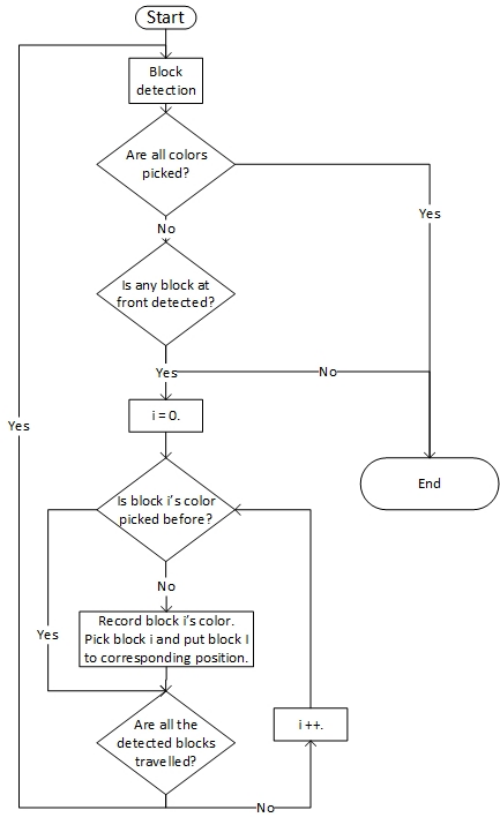

Fig. 11: Event3 autonomous execution logic.

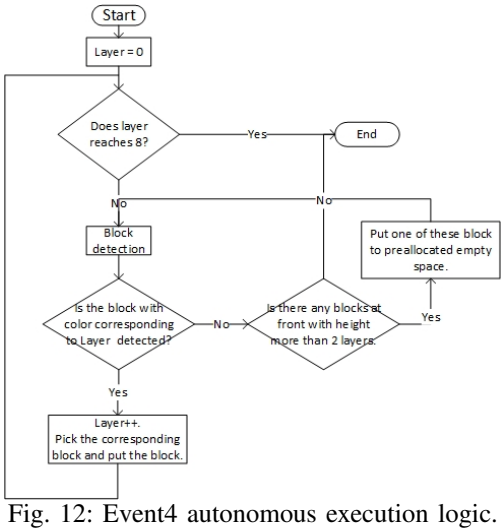

Fig. 12: Event4 autonomous execution logic.

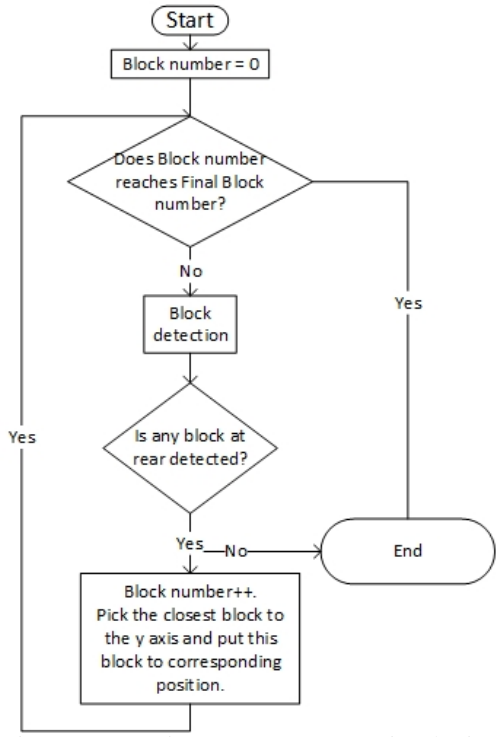

Fig. 13: Event5 autonomous execution logic.