ROB550-F19 ArmLab: Group 11 Report

Linyi Jin, Xi Lin, Nalin Bendapudi
{jinlinyi, bexilin, bnalin}@umich.edu

***Abstract*—This report chronicles our work on a robotic arm called *rexarm* and the methodology to make it perform various pick and place tasks autonomously. A gripper mechanism was designed and 3D-printed so as to give an additional degree of freedom to the the originally 5DOF rexarm and to make it grasp 40mm×40mm×40mm cubes efficiently. Perception was achieved using a block detection algorithm with sensory inputs from a RGB camera and a Kinect Depth sensor. The detected block poses were fed into an inverse kinematics module which calculated the initial and final joint angles and passed them on to the trajectory generation module for path smoothing. A state machine was used to govern the logic flow.**

***Index Terms*—DH parameters, inverse kinematics, camera calibration, object detection, state machine**

I. Introduction

In this project, our group worked on a system with a 6 degree of freedom robot arm and a kinect RGB-D camera to perform tasks such as block detection and placement. In the following sections, we will explain the various parts of our work.

II. Gripper Design

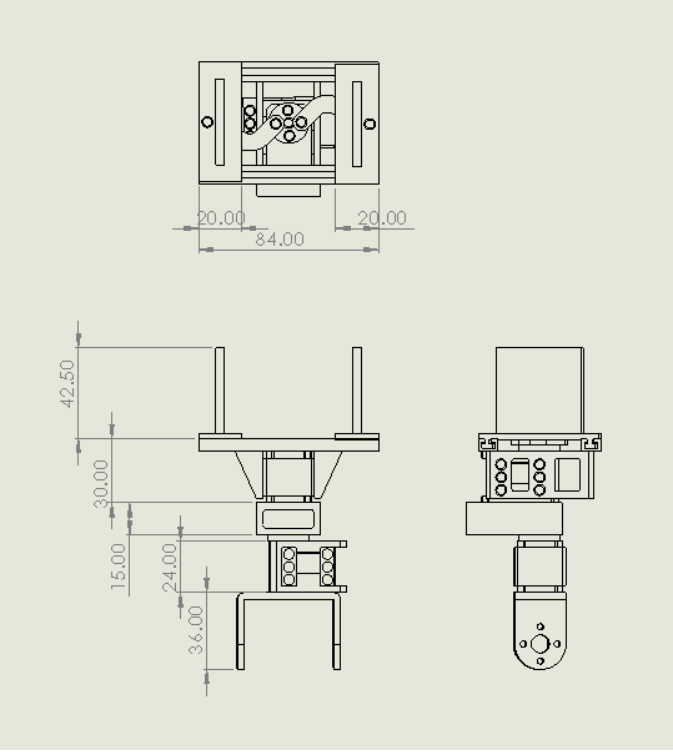

Fig. 1. Gripper Assembly drawing

A. General Description

The three orthographic views of our gripper assembly are shown in Fig. 1, which are composed of several part(from low to high, ignoring motors): a wrist 3 motor base, a gripper motor base, a gripper base, two links and two gripper boards. The Gripping and releasing motions are achieved by gripper boards sliding towards and away from each other, which are controlled by the gripper motor through two links. The maximum design interval between two gripper boards is 60 mm (The state shown in Fig. 1), and the minimum is 36 mm (when gripper motor performs a 180 degree counterclockwise rotation).

B. Design Process

We chose the structure of our gripper to be a double side slider with two considerations: (1) it has a relatively large moving range while having a relatively small size and light weight. (2) In the case that one board is fixed, when the gripper release a block, the block adjoins the fixed gripper board, then its position could be easily affected by the movement of the fixed board when it leaves. To determine the moving range, we considered the blocks' diagonal and side length, being 53.2 mm and 37.6 mm respectively. Since the distance between opposite holes on the motor panel is 12 mm, the maximum moving range is 24 mm thus is enough to include block dimensions. Besides, it's possible that we need to attach something to the gripper boards for increasing friction or adjusting distance between them, we reserve 2 mm width for both side, and decide the moving range to be 36 mm to 60 mm. Then, we made the size of gripper board to be a litter bit larger than block surface, 40mm $\times$ 40 mm, and the interval between slider to be also 40 mm. Dimensions of other parts of gripper are determined according to the above dimensions and motor size.

C. Actual Problems and Revision

We found that the actual range of gripper motor is 0 to 150 degree, but since we've reserved space for adjustment in design, we could just attach a card board the inner surface of one of the gripper boards to solve the problem. The second issue is that the clearance between slider and slideway is designed as 0.5 mm, but in 3D printing, some extra supporting materials in the slide way take up the space. Thus we enlarged the clearance to 1 mm and polished the slider. The third problem is that in the first version of design, the contact surface between slideway and gripper base is too small so that they detached when a small force is applied. To make the gripper

base stronger, we added some extended triangular ribs between slideway and gripper base.

III. Trajectory Generation and Path Smoothing

In this part we generate a cubic polynomial trajectory for the robot arm to move from one configuration to another.

A. Computation

The total time of trajectory T is estimated using the maximum angle changes among all joints, $\Delta\theta_{max}$, and maximum angle speed limit v_{max}. Because speed curve is parabolic, equation (1) we have a upper bound estimation $T_{upper} = 2\Delta\theta/v_{max}$.

$$\Delta\theta_{max} = \int_0^T v dt > \frac{1}{2} v_{max} T \tag{1}$$

Then we decide a coefficient α so that when $T = \alpha T_{upper} = 2\alpha\Delta\theta/v_{max}$, the actual maximum value of the speed curve is quite close to v_{max} for v_{max} varies within a reasonable range.

For every joint, its angle in the start and end configuration, respectively q_0 and q_f, are given, and we need to have zero velocity at both points, thus $v_0 = v_f = 0$. We've known that $t_0 = 0, t_f = T$, Then by solving equations (2) we get the position and speed curve.

$$\begin{cases} q_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 \\ v_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2 \\ q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \\ v_f = a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{cases} \tag{2}$$

To make robot arm move according the curves, we compute the sequence of position and speed with interval $dt = 0.05s$ for every joint, then use function $rexarm.set\_potisions()$ and $rexarm.set\_speeds()$ to give instructions to the robot arm.

B. Performance evaluation

The comparisons between performance of trajectories with and without planning are shown in Fig.2-4. The low and high speed are respectively $0.3m/s$ and $0.8m/s$, which are the maximum linear velocity limits, v_{max}, of all joints. In Fig. 2, the trajectories with planning are almost the same between different speeds, but that without planning are quite distinct. Besides, the velocity profiles in high speed also show clear difference.

We plot angle and linear velocity variation of each joint at high speed case in Fig. 3 and 4. It's clear that compared to case with planning, velocity of all joints in that without planning increase to about v_{max} and drop to 0 with more steep slope, requiring larger acceleration in both start and end section. Thus robot arm has a larger jittering in no planning case, which could affect its exact trajectory. In 3, joints reach target angles at different time in no planning case, while for planning case all joints reach their targets simultaneously, hence their trajectories are different.

IV. Kinematics

A. Forward Kinematics

Seven frames (six for each joint and one for the end-effector) have been identified and designated as per the DH convention. The orientation of the frames are shown in Fig.5. The values of the marked lengths in the figure were measured and are summarized below:

$$l_0 = 113.69 \pm 0.01mm$$
$$l_1 = 101.4 \pm 0.01mm$$
$$l_2 = 112.85 \pm 0.01mm$$
$$l_3 = 119.17 \pm 0.01mm$$

The DH parameters according the frames marked in Fig.5 are given in the table below:

TABLE I
DH Table

Joint No (i)	θ_i	d_i	a_i	α_i
1	θ_1	l_0	0	$-\pi/2$
2	$\theta_2 - \pi/2$	0	l_1	0
3	$\theta_3 + \pi/2$	0	0	$\pi/2$
4	θ_4	l_2	0	$-\pi/2$
5	θ_5	0	0	$\pi/2$
6	θ_6	l_3	0	0

The homogeneous transformation matrix for joint i, that is the transformation from frame $i-1$ to frame i is given by:

$$T_{i-1}^i = \begin{pmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Hence, the final pose of the end effector can be calculated in the ground frame by recursively combining the transformations of adjacent frames according to the following formula:

$$T_0^i = T_0^{i-1} T_{i-1}^i \tag{4}$$

T_0^i in eqn.4 will give the end-effector pose as a function of joint variables $[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$

B. Inverse Kinematics

Given a desired end effector pose, Inverse Kinematics tries to find all possible configurations or return failure if it's not reachable. The problem could be decoupled into two parts: (1) find the solutions for the first three joints given wrist center position, (2) find the solutions for the last three joints given the first solution and wrist orientation. The required computations are demonstrated in the following parts.

1) Wrist Centre Position: The wrist-center (P_W) is the point where the last three axes meet. It can be calculated as follows:

$$P_W = P - b_6 R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

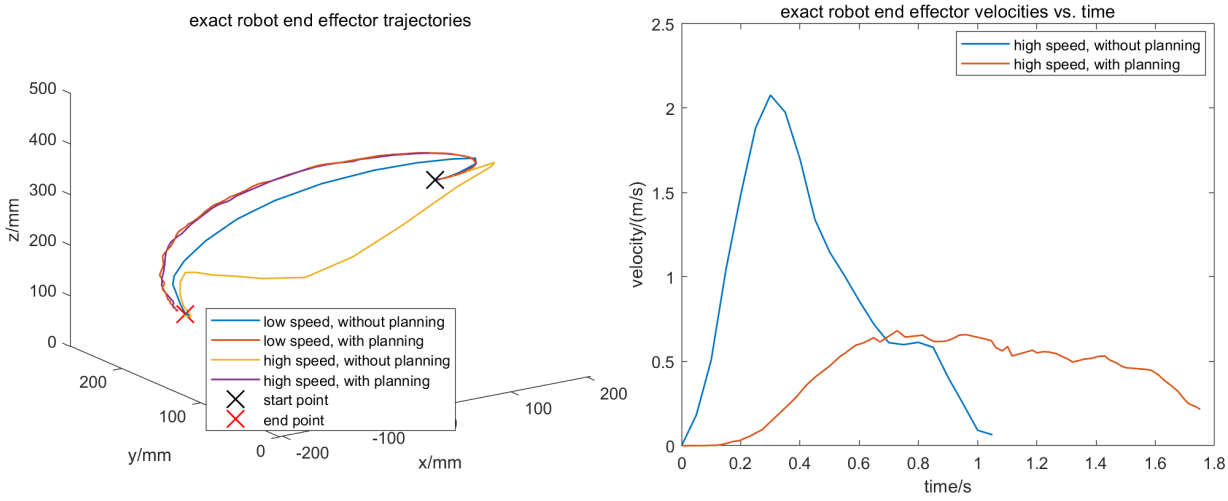

Fig. 2. Robot arm end effector trajectories and velocities

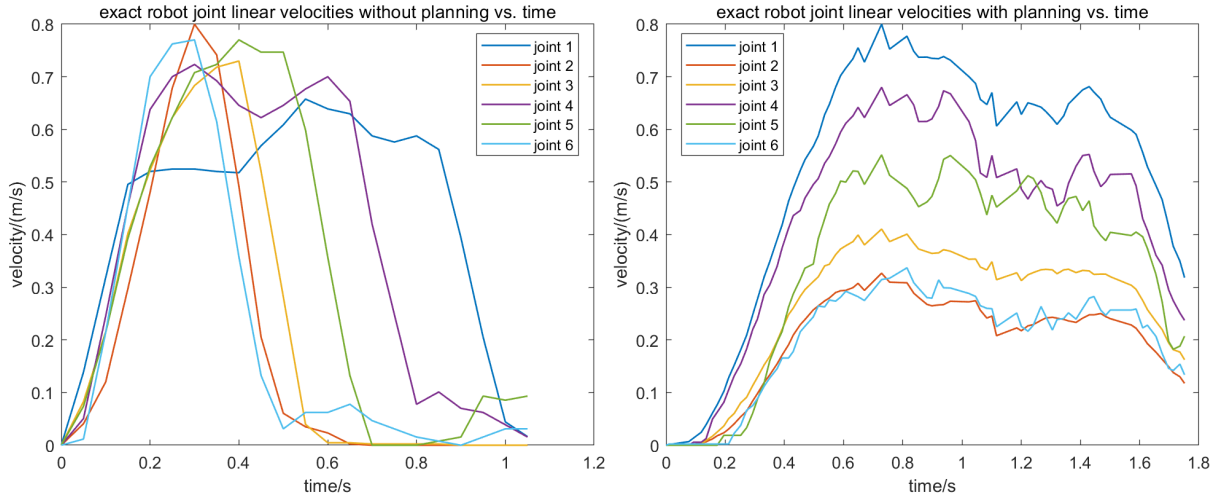

Fig. 3. Robot arm joints' linear velocity variation with time

where P and R are the translation and rotation part of desired end effector pose, and b_6 is the joint offset of the 6th link. Equation (5) implementation: kinematics.py, line 90-92.

2) Articulated 3R Joint Angles: Because Wrist center position is also the end effector position of first three links, we could get the solutions for the first three joints through geometry.

There are usually four solutions in this part(except for some singular positions), which are shown in Fig. 6. Let $P_W = [x_w \ y_w \ z_w]^T$. For the first joint angle:

$$\theta_1 = atan2(y_w, x_w) \ or \ atan2(y_w, x_w) + \pi \tag{6}$$

Where the first solution corresponds to the left two cases and the second one to the right two. Equation (6) implementation: kinematics.py, line 98-100. For the third joint:

$$\begin{cases} r = \sqrt{x_w^2 + y_w^2} \\ s = z_w - l_0 \\ \theta_3 = \pm \cos^{-1}\left(\frac{r^2+s^2-l_1^2-l_2^2}{2l_1l_2}\right) \end{cases} \tag{7}$$

In equation (7), we need to first verify that the value in the cos^{-1} is within $[-1, 1]$, otherwise no solution. The two solutions of θ_3 correspond to higher and lower two cases in Fig. 6 respectively. Equation (7) implementation: kinematics.py, line 95-96, 104-111. Then, for the second joint, its solution depends on θ_1 and θ_1:

$$\begin{cases} \alpha = atan2(s, r) \\ \gamma = atan2(l_2 sin\theta_3, l_1 + l_2 cos\theta_3) \\ \theta_2 = \frac{\pi}{2} - \gamma - \alpha, if \ \theta_1 = atan2(y_w, x_w) \\ \quad\quad or \\ \theta_2 = -\frac{\pi}{2} - \gamma + \alpha, if \ \theta_1 = atan2(y_w, x_w) + \pi \end{cases} \tag{8}$$

Equation (8) implementation: kinematics.py, line 114-119.

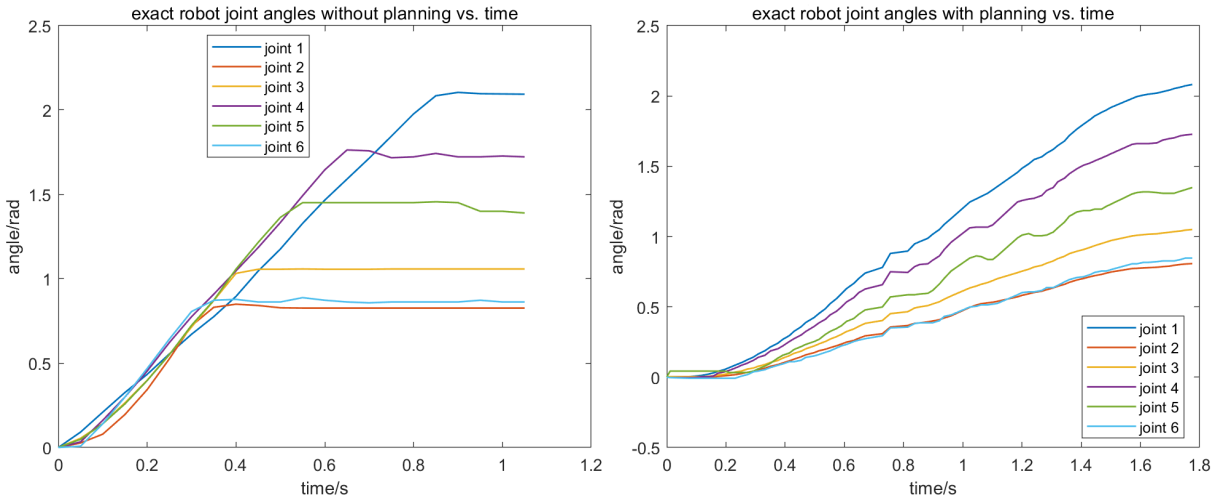

Fig. 4. Robot arm joints' angle variation with time

3) Wrist Joint Angles: For a certain solution $(\theta_1, \theta_2, \theta_3)$ gotten from above, we can calculate the end effector pose of the first three links R_3^0:

$$R_3^0 = R_{z,\theta_1} R_{x,\theta_2} R_{x,\theta_3} \tag{9}$$

Equation (9) implementation: kinematics.py, line 121 - 126. Then the orientation of the last three joints could be computed:

$$R_6^3 = (R_3^0)^{-1} R \tag{10}$$

Equation (10) implementation: kinematics.py, line 129. Since R_6^3 can also directly written as a combination of last three joint angles $(\theta_4, \theta_5, \theta_6)$:

$$R_6^3 = \begin{pmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & -c_4s_5 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5c_6 + c_4c_6 & -s_4s_5 \\ s_5c_6 & -s_5s_6 & c_5 \end{pmatrix} \tag{11}$$

In the above equation, $c_i = cos(\theta_i)$, $s_i = sin(\theta_i)$. In this part, there are two solutions exist, using equations (12) or (13):

$$\begin{cases} \theta_4 = atan2(R_6^3(2,3), R_6^3(1,3)) \\ \theta_6 = atan2(R_6^3(3,2), -R_6^3(3,1)) \\ \theta_5 = atan2(R_6^3(1,3)/cos(\theta_4), R_6^3(3,3)) \end{cases} \tag{12}$$

$$\begin{cases} \theta_4 = atan2(R_6^3(2,3), R_6^3(1,3)) + \pi \\ \theta_6 = atan2(R_6^3(3,2), -R_6^3(3,1)) + \pi \\ \theta_5 = atan2(R_6^3(1,3)/cos(\theta_4), R_6^3(3,3)) \end{cases} \tag{13}$$

Equation (12) and Equation (13) implementation: kinematics.py, line 131 - 153.

4) Joint limit verification: The computed solution does not necessarily conform to the exact joint angle limits, thus we need to have a check and remove those that don't satisfy the joint limits.

C. Evaluation

To exam the accuracy of our inverse kinematics, we select some test positions in workspace coordinate to be end effector positions (the center point of gripper). As for the rotation part of end effector, we want the gripper to pointing exact down for the convenience of measurement, thus rotating around workspace x axis for π. Then the i-th test pose is:

$$T_i = \begin{pmatrix} R_{x,\pi} & P_i \\ 0 & 1 \end{pmatrix} \tag{14}$$

We run the inverse kinematics function to move robot arm to these positions with workspace coordinate (x_d,y_d,z_d), measure the exact workspace coordinate of the gripper center (x_a,y_a,z_a), and compute the relative translation error with respect to the designated positions. The results are summarized in table II. From the table, we know the average relative translation error between expected positions and actual positions is 4.74%.

TABLE II
IK EVALUATION, UNIT [MM]

index	x_d	y_d	z_d	x_a	y_a	z_a	error(%)
1	130	130	100	125.78	140.04	95.36	5.66
2	-80	145	80	-83.68	147.77	77.35	2.89
3	-150	-60	30	-140.53	-59.57	25.29	6.44
4	60	-180	50	74.45	-173.91	43.79	8.60
5	145	-80	40	142.81	-75.55	37.62	3.23
6	170	100	60	164.58	107.45	57.39	4.64
7	-100	20	120	- 99.29	15.89	117.46	3.10
8	-150	-120	70	- 151.47	-125.27	65.96	3.33

V. CAMERA CALIBRATION AND BLOCK DETECTION

A. Camera Calibration

1) Camera Intrinsic: The camera intrinsic is obtained using Chessboard multiplane calibration method implemented in

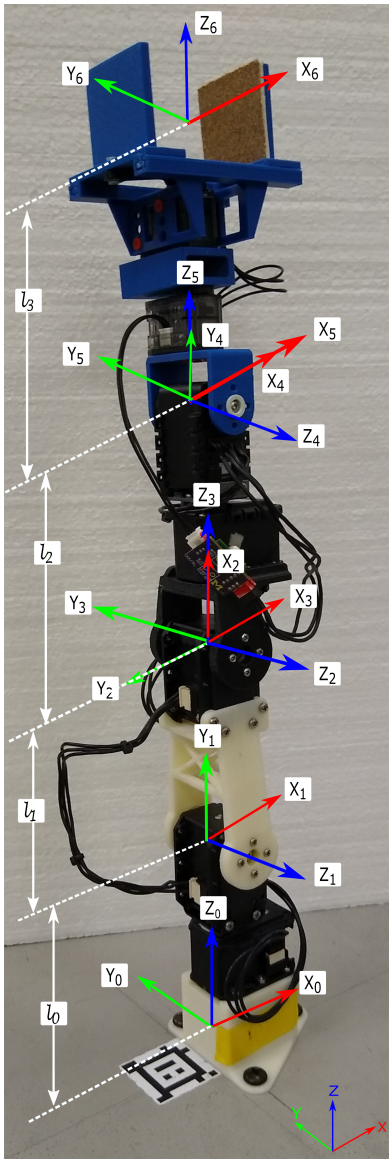

Fig. 5. Rexarm Schematic Diagram showing Joint Frames

OpenCV. The code adopted from the starter code. The intrinsic is shown in Eqn. 15.

$$K = \begin{pmatrix} 526 & 0 & 300 \\ 0 & 527 & 274 \\ 0 & 0 & 1 \end{pmatrix} \tag{15}$$

2) Depth Calibration: The depth value is calculated using Eqn. 16.

$$Z = 0.1236 \times \text{np.tan}(d/2842.5 + 1.1863) \tag{16}$$

The depth image is calibrated to align with the RGB image by using mouse clicks to locate the four corners of the workstation. Assume the transformation between the depth image and the RGB image is an Affine Transformation, satisfying Eqn. 17, which is a linear system with six unknowns. Each match gives two linearly independent equations. Therefore, at least

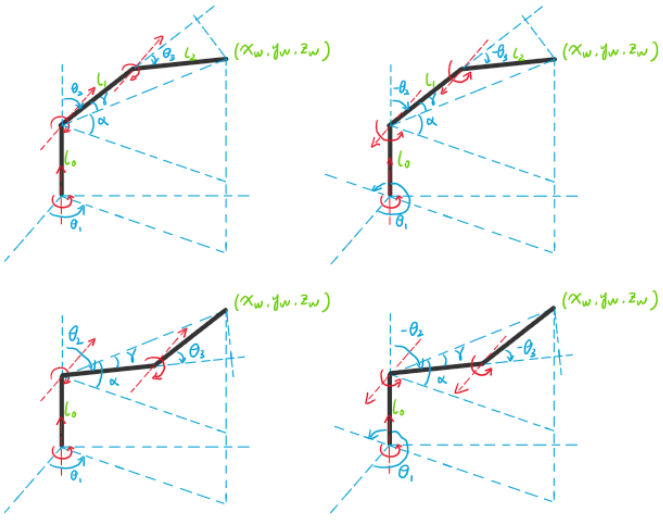

Fig. 6. Four solutions for first three joints in non-singular position

three points are required to solve the equation. In our case, we use four points to reduce the error. The transformation equation can be written as

$$A \cdot T = b.$$

We solve T by

$$T = (A^T A)^{-1} A^T b$$

$$\begin{pmatrix} & & \Lambda & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \Lambda & & & \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} \Lambda \\ x_i' \\ y_i' \\ \Lambda \end{pmatrix} \tag{17}$$

3) RGB to World Frame Calibration: In order to convert from pixel space to world space, we must calculate the transformation from pixel space to world space. To do so, we take click on the four corners of the grid in world space, which correspond to known locations in the real world. Similar to the aligning depth frame to RGB frame, using Eqn. 17, we align the RGB frame to the world coordinate where the origin lies on the base joint of the arm.

4) Evaluation of Calibration: We verified the calibration by measuring the difference between the world frame coordinate calculated from RGB-D image and the actual world frame coordinate measured by a ruler, Fig. 7 shows how we do the evaluation. The results are shown in Table III and Fig. 8. Column 1-3 is the XYZ location in the world frame measured by the ruler, with the origin centered at the arm base joint, x points to the right, y points to the top. Column 4-6 is the XYZ location of world frame coordinate calculated from RGB-D image. The last column shows the L2 error. From the result, we can see if we fix the x and y location and change z, the L2 error does not change significantly. Therefore, the depth calibration is good. However, as we see the L2 error increases when the xy location is far away from the origin, the intrinsic

calibration still has certain amount of error. But overall, the error is within 1.5 cm and most of the errors are within 1cm. We think the calibration is good enough to complete our tasks.

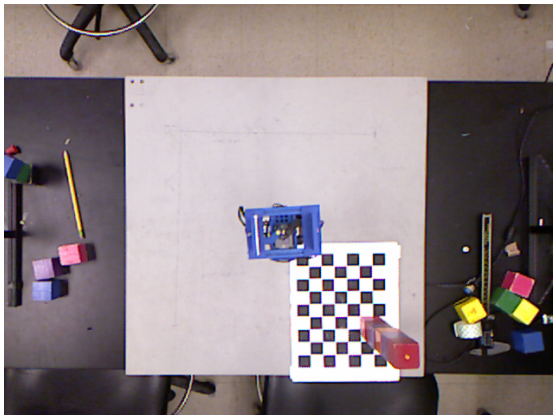

Fig. 7. Evaluation of calibration. We measure the location of the yellow dot at the center of the cube and calculate the error.

TABLE III
EVALUATION OF CALIBRATION, UNIT [MM]

x_{gt}	y_{gt}	z_{gt}	x_{cam}	y_{cam}	z_{cam}	L2 Error
84	-102.66	43.47	88	-106	47	6.3
84	-102.66	82.07	86	-106	85	4.9
84	-102.66	120.67	90	-108	123	8.4
84	-102.66	159.27	86	-100	164	5.8
84	-102.66	197.87	87	-105	201	4.9
84	-102.66	236.47	86	-104	242	6.0
135	-153.96	43.47	143	-155	42	8.2
135	-153.96	82.07	141	-152	85	7.0
135	-153.96	120.67	144	-153	120	9.1
135	-153.96	159.27	142	-154	162	7.5
135	-153.96	197.87	144	-153	198	9.1
135	-153.96	236.47	141	-155	237	6.1
186	-205.26	43.47	194	-208	40	9.1
186	-205.26	82.07	193	-207	80	7.5
186	-205.26	120.67	195	-210	117	10.8
186	-205.26	159.27	198	-208	157	12.5
186	-205.26	197.87	198	-206	198	12.0
186	-205.26	236.47	197	-211	236	12.4

B. Block Detection

Block detection includes detecting the XYZ coordinates of the blocks (Block localization) and classify the color of blocks into 8 classes: *Black, Red, Orange, Yellow, Green, Blue, Violet, and Pink*.

1) Block localization: We regard blocks as several blobs and use Laplacian of Gaussian filter to detect blobs on calibrated depth image. The whole process is shown in Fig 9. The magnitude of the Laplacian response will achieve a maximum at the center of the blob, provided the scale of the Laplacian is "matched" to the scale of the blob. The radius of the blob we are detecting is related to the standard deviation of the gaussian filter.

$$r = \sqrt{2}\sigma$$

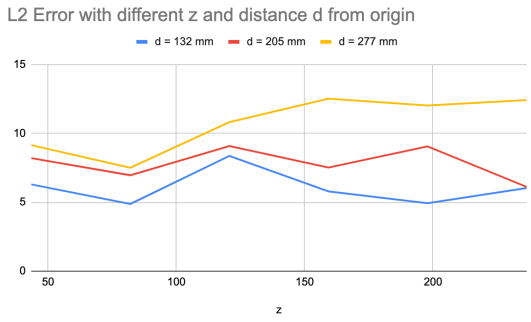

Fig. 8. L2 Error of the real position of the yellow dot measured by the rule and the position calculated by the system. The horizontal axis is changing in z direction. Three colors of lines represent the distance to the origin in the xy-plane.

The filter Laplacian of Gaussian filter is calculated as

$$\mathrm{LoG} = \nabla^2_{\mathrm{norm}} g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

The gaussian filter g will smooth the image and the laplacian filter will give the gradient of the image. The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases. Since the Laplacian is the second Gaussian derivative, so it is multiplied by σ^2. Our detailed implementation is listed as follows:

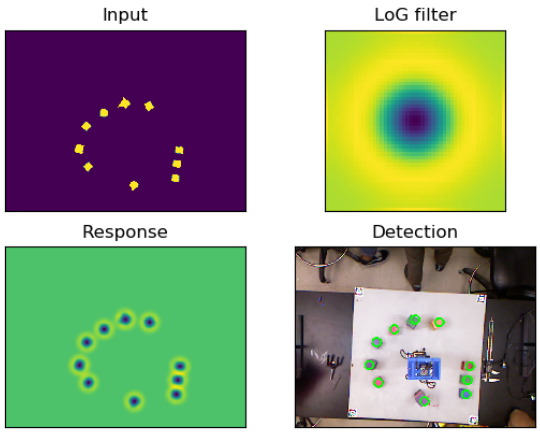

Fig. 9. Detection of blobs with fixed radius on a testing image, circled in green in the bottom right figure. Since the blocks we detect are of the same size, using LoG is helpful in filtering out noises. Our detector generates less false positive outputs compared to clustering methods.

1) Preprocess the depth image to remove noises and set threshold to remove negative height and too large height to exclude false positives.
2) Measure the radius r of the desired blob size and calculate the standard deviation σ of the Gaussian filter.
3) Use *scipy.ndimage.filters.gaussian_filter* to generate a 2D gaussian kernel of size $(6\sigma + 1, 6\sigma + 1)$.

4) Use *cv2.Laplacian* to generate the Laplacian of Gaussian kernel.
5) Use *cv2.filter2D* to get the response of the kernel.
6) Set a threshold and find location of all peaks in the response.

2) Color Classification: To classify the color of the blocks, we convert the color space to HSV by *cv2.cvtColor* and pick HSV value of detected block in the previous section. We set HSV ranges, shown in Table IV for each color and use this range to determine the color of each block. If the HSV value of detected block is not in any of the HSV ranges, we consider it to be the background.

TABLE IV
THRESHOLDS IN HSV SPACE FOR DIFFERENT COLORS.

Color	H_{min}	S_{min}	V_{min}	H_{max}	S_{max}	V_{max}
red	110	146	118	134	210	200
green	33	37	100	66	105	218
blue	0	111	130	14	158	240
yellow	0	0	214	100	200	294
orange	108	87	206	122	201	294
black	129	0	41	160	61	166
pink	122	79	202	142	188	294
violet	146	70	93	170	123	228

3) Evaluation of Block Detection: To verify the accuracy of the block detector, we place 16 blocks on the board and measure the classification error as well as the block localization error. The blocks are placed on the board in the configuration shown in Fig. 10. We record ten consecutive frames for the configuration and run block detector on each frame. We then calculate the fraction of wrong colors and error of location. The localization error with respect to the block 2D location is shown in Fig. 11. The maximum error is less than 15 mm and the mean is 9 mm. Among 10 frames recorded, the color classification accuracy is shown in Table V. Most of the color classification has 100% accuracy while Black and Orange blocks are a little bit unstable. To make the system robust to these errors. Since there is really low false positive in the detection, the system check for several frames and consider a detection positive so long as any frame has a positive detection.

TABLE V
COLOR CLASSIFICATION RESULTS AMONG 10 FRAMES WHERE EVERY COLOR APPEARS TWICE IN EACH FRAME.

	False Negative	False Positive	Accuracy
Black	0.3	0	0.7
Red	0	0	1
Orange	0.3	0	0.7
Yellow	0	0	1
Green	0	0	1
Blue	0	0	1
Violet	0	0	1
Pink	0	0	1

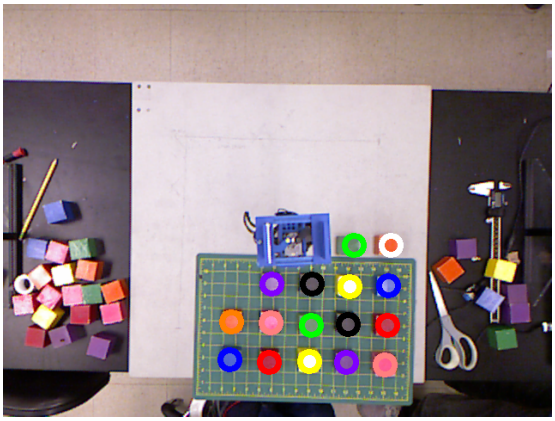

Fig. 10. Evaluation of block detection. In this frame, all the block are detected, however the color of one orange block is wrong.

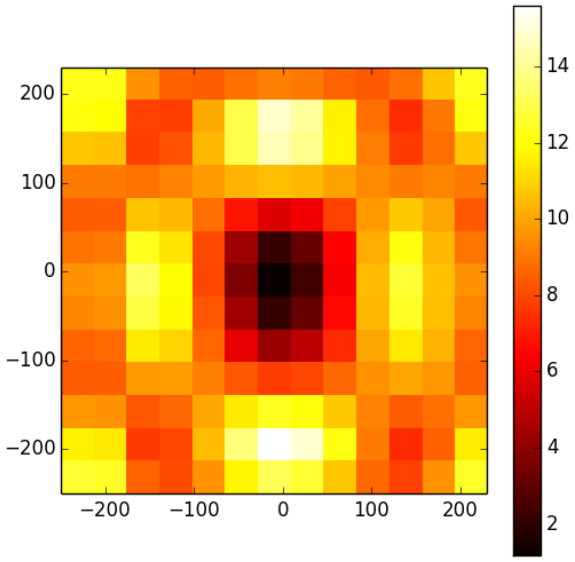

Fig. 11. Block localization uncertainty vs. location on the board. Unit mm.

VI. COMPETITION

A. Gripper Performance

After printing the first version of gripper, we found some problem in design and revise them as section II-C mentions. The revised gripper is satisfactory and we never change any part of it since then. As for improvement, the size of gripper board could be larger so that gripping could be more stable.

B. Motion Planning Algorithm

The basic motion for robot arm is to pick a block from one position and put it to another, we view it as a continuous process and plan for it with Algorithm. 1(Function names here are not the same as those in the code). We define five poses as way points of the whole motion sequence.

1) Origin: The pose where all joint angles are 0, robot arm go back to this pose when finishing pick and put motion. It's denoted by C_0.

2) Pick Pose: The pose where gripper can pick the block, denoted by C_1.

3) Pick Preparation Pose: The pose that is higher than C_1, robot arm need to reach this pose first, then slowly descent to C_1. In this way, we can ensure that robot arm won't hit the block, It's denoted by C_2

4) Put Pose: The pose where gripper can put the block, denoted by C_3.

5) Put Preparation Pose: Similar to C_2, It's a pose higher than C_3, aiming at guaranteeing the safety and stability of put motion. It's denoted by C_4.

Apart from the position of these poses, we also need to specify orientation for them. We divide workspace x-y plane into four parts, each part corresponds to a pose frame that is gotten by rotate workspace frame about z axis for $-\pi/2$, 0, $\pi/2$, π. Then we can decide rotation about z axis for a pick position. $grip\_angle$ In Algorithm. 1 are the candidate angles for computing rotation about x axis, which control how much the gripper point down. With Algorithm. 3, we could got the orientation for poses. Note that the grip angle for pick pose and put pose need to be the same, otherwise the block could roll when putting. Then we loop through all candidate grip angles for all poses, once an available motion sequence is found, we plan a trajectory and execute it, otherwise return failure.

Algorithm 1 MotionPlanning

function MotionPlanning($pick\_position$,$put\_position$)
  for all $grip\_angle$ **do**
    $angle$ = $grip\_angle$
    C_1 = GetAngles($angle$, $pick\_position$)
    if C_1 is **None then continue**
    for all $grip\_angle$ **do**
      $pick\_prepare$ = $pick\_position$+$height\_margin$
      C_2 = GetAngles($grip\_angle$, $pick\_prepare$)
      if C_2 is **None then continue**
      C_3 = GetAngles($angle$, $put\_position$)
      if C_3 is **None then continue**
      for all $grip\_angle$ **do**
        $put\_prepare$ = $put\_position$+$height\_margin$
        C_4 = GetAngles($grip\_angle$, $put\_prepare$)
        if C_4 is **None then continue**
        $sequence$ = [C_2,C_1,C_2,C_0,C_4,C_3,C_4,C_0]
        return PlanTrajectory($sequence$)
  return None

C. Performance for Each Task

We have tried the first three task during competition. The first task is simply stacking three blocks, we finished it normally.

For the second task, we met a problem that our motion planning algorithm failed to find solutions for some blocks, so did not do well in this task. The reason is probably that we require the pick and put pose to have the same gripping angle, and try to plan trajectory that directly go from pick position and put position. Then for some cases, like when the pick position is far from robot arm and the put position is close to it, it would be difficult to find a solution while having the grip angle constraints.

For the third task, it's lucky that we didn't encounter problem in task 2 and succeed to plan trajectory for all blocks. Since our motion planning algorithm is designed to pick and put blocks stably, we managed to stack 7 blocks.

Algorithm 2 GetAngles

function GetAngles($angle$,$position$)
  $orientation$ = ComputeOrientation($angle$,$position$)
  $pose$ = GetTransform($orientation$,$position$)
  $joint\_anlges$ = InverseKinematics($pose$)
  if no solution for $joint\_angles$ **then return None**
  else return $joint\_anlges$

Algorithm 3 ComputeOrientation

function ComputeOrientation($angle$,$position$)
  if $-position.x > |position.y|$ **then** $rotate = -\pi/2$
  else if $position.y > |position.x|$ **then** $rotate = \pi$
  else if $position.x > |position.y|$ **then** $rotate = \pi/2$
  else then $rotate = 0$
  return $R_{z,rotate}R_{x,angle}$

D. Possible Improvements

Currently there are still some problems with our motion planning algorithms.

Firstly, we does not consider the orientation of blocks, and assume that workspace coordinate axes are orthogonal or parallel to there surface. To solve it, we need to include block orientation in Algorithm 3 and replace $R_{z,rotate}$ with it. Note that the gripper can access the block from four orthogonal directions, so there are also four solutions for the new Algorithm 3.

Besides, as it's mentioned above in task 2, We met some cases when there are solutions for pick and put motion separately but have no solutions when combining them together. Thus if no solution could be found for going directly from pick to put position, robot arm should try to pick the block and put it to some intermediate positions, then go from there to the put position.

VII. Conclusion

We finished all the parts in the check list and the first three tasks in the final competition, and we gained practical experiences in kinematics and computer vision through this project.