Block Stacking Using The Rexarm Robot With A Vacuum Gripper

Haonan Chang, Tor Shepherd, Siyuan Feng
{harveych, tors, siyuanf}@umich.edu

***Abstract*—Pick and place tasks such as block stacking are one of the fundamental abilities for robots. However, building the whole system from scratch for a specific pick and place task is not trivial. Many different functions of robot, such as perception and planning, have to be implemented for different stages in the task. In this project, we are required to build a robot system to finish a block stacking task. We need to pick and stack eight blocks in a specific color order. To complete this task, we propose a system including block detection, kinematics, trajectory planner and task planner using the rexarm robot with a vacuum gripper. Different tests has been made to validate the capacity of each part in the system. It turns out our proposed system can meet the most requirements for each component of the system and successfully finish the task I, block stacking task.**

I. Introduction

Serial-link manipulators, or robot arms, are chains of rigid bodies coupled by cylindrical or prismatic joints which can be actuated or passive [1]. By applying input from actuated joints, the end effector can be positioned and/or rotated in space in a desired way. The simple construction and precise positioning of robot arms has led to extensive use in a variety of industries, especially manufacturing.

Automation applications in industry commonly require moving material from one position and orientation to another, a task termed "pick-and-place". The simplest solution to the pick-and-place problem is:

1) Assign joint values that position the robot gripper at the starting pose[1] of the object and close the gripper[2]
2) Assign joint values that position the robot gripper at the ending pose of the object and open the gripper

This procedure makes an assumption that the starting location and ending location of the object are known, but some tasks require the robot to first locate the object. A camera above the workspace can be used to sense the pose of the object.

[1]defined as the location and orientation

[2]Steps 1) and 3) require a conversion from world coordinates to joint configuration, which is discussed in section IV.

The proposed solution to this pick-and-place task is implemented with the Rexarm robot arm, Microsoft Kinect camera, and wooden blocks as the object to manipulate.

The Rexarm is a 5-degree-of-freedom (DOF) robot arm developed by April Lab at the University of Michigan which has the option to append a gripper to the end effector in order to accomplish pick-and-place tasks[3].

To minimize inertia while meeting constraints on approach angles, gripping force, and workspace dimensions, a vacuum gripper is proposed and the design is analyzed.

As for the whole system, we implemented the trajectory planner, kinematics, block detection and task planner. The efficacy of the 6DOF arm with vacuum gripper is demonstrated by the ability to manipulate the objects from start positions and orientations to desired end positions and orientations.

The performance of the robotic system is evaluated by the measured error in positioning, undesirable vibrations in the end effector observed from motor encoder values, and error in camera perception.

II. Gripper Design

A. Problem definition

The ability of a gripper to pick up and maintain grip on a desired object is an important factor in robot arm performance. The Rexarm is provided without a standard gripper, so the product development process is used to generate an original design. The following constraints are proposed:

1) Fasten to the motor output of joint 5 using OLLO rivets in the bolt pattern specified
2) Grip cubes of dimension 35mm from flat or from an angled platform with arbitrary rotation.
3) Have a radial reach (specified from the center of the base) of $[60mm - 300mm]$

The Rexarm is a 5-degree-of-freedom arm; to satisfy constraint 2 and reach arbitrary orientations, a sixth motor can be appended along the axis of the final link,

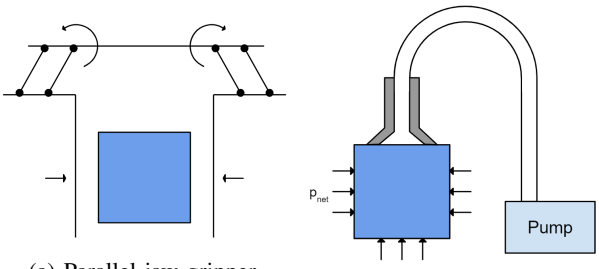
[3]To ensure that the robot has full dexterity in 6DOF, a sixth motor is appended to the Rexarm, discussed in section II

completing a standard "spherical wrist" configuration [1]. The extent to which the robot arm can reach blocks when gripping from the top face is limited by the length of the links of the Rexarm. Adding the sixth degree-of-freedom means that blocks can be gripped from side faces as well, satisfying constraint 3.

B. Parallel-jaw design

A common configuration of robot arm grippers is the parallel-jaw gripper. This gripper is useful for precise gripping of objects with parallel surfaces, but requires a large mechanical construction and an additional servo motor.

C. Vacuum design



(a) Parallel-jaw gripper.

(b) Vacuum gripper.

Fig. 1: Diagrams of two gripper designs.

An alternate design involves the use of a vacuum pump to generate lower pressure inside a suction cup than the surrounding environment. The difference in pressure, p_{net}, acts opposite gravity and holds the block as long as the boundary between the suction cup and outside is sealed.

TABLE I: Partial Pugh matrix for two grippers.

	Weight	Parallel-jaw	Vacuum
Inertia	2	2	1
Rigidity	2	2	1
Total		5	4

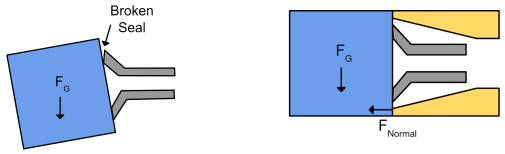
To evaluate designs quantitatively, we use a weighted Pugh matrix, shown in table I. Higher values denote undesirable characteristics.

The vacuum gripper is selected because the small end effector has a lower inertia and offers increased rigidity over a bulky traditional gripper.

D. Iteration

Testing shows a strong seal when lifting blocks from the top face, but lifting from the side causes the seal to break as shown in fig. 2a.

The seal breaks due to tension in the top of the suction cup caused by the moment arm due to the force of gravity; to prevent tension, a balancing moment is needed. The proposed solution is to mount a rigid support that applies normal force on the block.



(a) Demonstration of seal breaking due to gravity.

(b) Vacuum gripper with support.

Fig. 2: Vacuum gripper holding block sideways.

The final gripper design rendering is shown in fig. 3. The gripper is offset to allow room for the pneumatic tube to pass through.

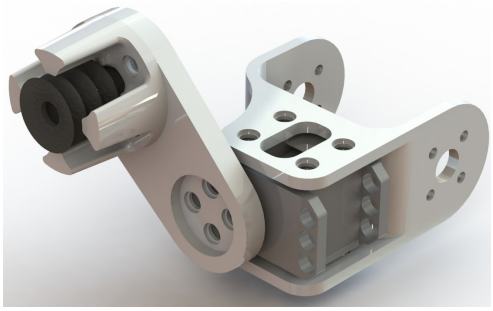


Fig. 3: 3D CAD rendering of vacuum gripper design.

E. Gripper Integration

The system shown in fig. 12 is implemented to control the vacuum gripper.

1) Arduino: If the Arduino microcontroller receives a 1 over serial, it writes the pin connected to the motor driver's PWM port HIGH.

2) Motor Driver: The motor driver commands the motor to turn at the rate specified by the PWM port (HIGH or LOW, from Arduino).

3) Vacuum Pump: The brushed motor vacuum pump nozzle is press-fit into the pneumatic tube that runs up the arm to the end effector, allowing the pump to be located outside of the workspace entirely.

4) Power Supply: The power supply provides power only to the motor driver/motor.

The Arduino only changes state if the serial command does not match the current state. If the Arduino receives a nonsense byte, it does not change state.

III. Teach & Repeat State Machine

A. Straight-line linear time trajectories

The robot arm can be moved between two position specified by joint angles using straight-line interpolation. The joint angle $q_i(t)$ is given by

$$q_i(t) = a_{0,i} + a_{1,i}t \tag{1}$$

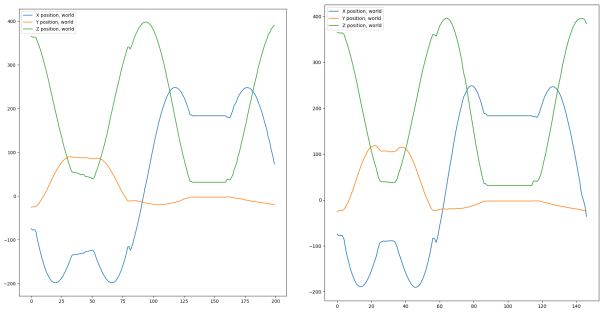
where a_0 and a_1 are constants determined by the start and end positions by

$$a_{0,i} = q_{0,i}, \quad a_{1,i} = \frac{q_{1,i} - q_{0,i}}{T} \tag{2}$$

assuming that the trajectory starts at time 0 and finishes at time T. The time T is constrained by the maximum specified motor velocity by

$$T = \frac{max(q_{1,i} - q_{0,i})}{\alpha_{max}} \tag{3}$$

The gripper x,y, and z coordinates in world space for an example pick-and-place task using manual teach-and-repeat are shown in fig 4a and 4b.



(a) 2.5 rad/s (b) 4 rad/s

Fig. 4: Gripper world coordinates over time without trajectory smoothing

B. Cubic trajectories

The linear time paths provide a good enough solution for some cases, but the endpoints of the segments have an instantaneous change in direction and magnitude of motor velocity, which can cause the arm to jerk and behave undesirably, even dropping the object. To prevent this, a polynomial time-scaled function with the added boundary condition of zero initial and final velocity is implemented.

$$\dot{q}_{0,i}, \quad \dot{q}_{1,i} = 0 \tag{4}$$

$$q_{0,i} = \theta_{initial,i}, \quad q_{1,i} = \theta_{final,i} \tag{5}$$

Four boundary conditions implies four unknowns for a fully-defined system, so we construct

$$q_i(t) = a_{0,i} + a_{1,i}t + a_{2,i}t^2 + a_{3,i}t^3 \tag{6}$$

The solution to this system is provided [1] as

$$a_{0,i} = q_{0,i}, \quad a_{1,i} = 0 \tag{7}$$

$$a_{2,i} = 3(q_{1,i} - q_{0,i}), \quad a_{3,i} = -2(q_{1,i} - q_{0,i}) \tag{8}$$

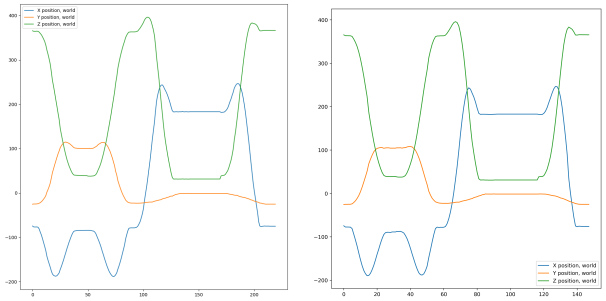
C. Lookahead and speed limiting

The cubic trajectory works well in theory, has practical issues in execution on the Rexarm. The servo motors track position using a proportional controller, which results in slower motor speed very close to the setpoint. Moving in a cubic trajectory by only assigning setpoints therefore results in jerky motion that stops after each single interpolated point.

The solution proposed is twofold:

1) Limit the motor speed: By assigning the speed calculated from derivative of the cubic trajectory, the motors should not move quicker than desired or arrive at the setpoint early.

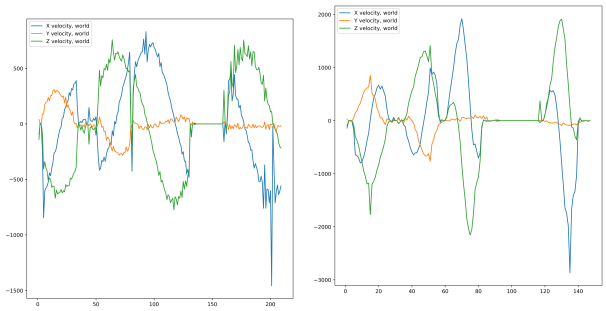
2) Assign a lookahead setpoint: By setting the motor's setpoint to a point sufficiently further down in the trajectory, the motor's proportional controller will saturate at the speed limit and behave as expected.



(a) 2.5 rad/s (b) 4 rad/s

Fig. 5: Gripper world coordinates over time with cubic lookahead trajectory smoothing

The resultant motion of the robot through the same set of waypoints is smoother in transition points between trajectories as shown in 5a (2.5 rad/s) and 5b (4 rad/s). Figures 6a and 6b show the end effector velocity with and without cubic lookahead smoothing. Without smoothing, the end effector velocity shows high frequency oscillations consistent with observations stemming from the internal controllers of the servo motors. With smoothing, the velocity has minized disturbance.



(a) Linear-time trajectory.

(b) Cubic lookahead smoothing.

Fig. 6: Gripper world velocities over time with and without cubic lookahead trajectory smoothing

IV. Kinematics

By kinematics, we need to implement forward kinematics first, which will return the end effector pose given the joints angles. Forward Kinematics can be done by using the DH table. Second, inverse kinematics is required in most tasks, which return the joint angles given the end-effector pose.

1) Rotation matrix: Rotation matrix is the most widely used representation of rotation for its importance by matrix calculation, and its usage in Lie groups $SE(3)$ and $SO(3)$. Though containing nine entries, rotations in $SO(3)$ only take three degrees of freedom. That property constraint the rotation matrix to be orthogonal with determinant as one.

2) Euler angle: Euler angles, however, have a very intuitive representation for rotation by using three rotations on each axis, namely row, pitch, and yaw, where it will be widely used in this project as a direct indication for rotations.

As illustrated in the previous paragraph, 3D rotation matrices R constitute a special orthogonal group $SO(3)$. With the 3D translation vector t, together they constitute the special Euclidean group $SE(3)$, written in Eq. 15

In our project, we implement the $SE(3)$ class `T` in `kinematics_convert.py`. Class `T` is instantiated by a $SO(3)$ 3×3 `R` and a `(3,)` shaped `t`, both as ndarray. Class `T` methods implement inverse calculation, multiplication, Euler angle generation, along side with other supports.

A. DH Tables

DH table is one of the many methods to generate $SE(3)$ matrices. Four parameters need to be specified in DH: joint offset, joint angle, link length, and link twist. From the four parameters of the DH table, the T matrix can be easily obtained by using Eq. 16.

1) DH Table For Inverse Kinematics: To have a frameset this is plausible for inverse kinematics, the DH joint coordinates are designed as follows.

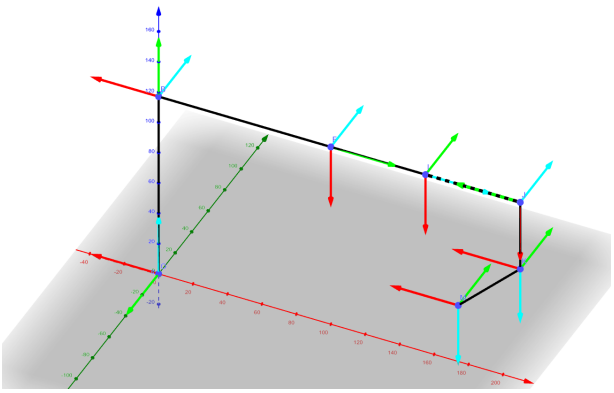


Fig. 7: Bent DH Frames Set

The base, shoulder, elbow, wrist1, wrist2, and gripper is represented by B, F, I, J, K and M. For each frame, the XYZ axis is shown in RGB order. It is worth noticing that the grey plane has the same view of perspective as the board in the lab. Also by constraints of the DH table, the wrist1 is located in point F and the wrist2 in point J. These two points are shown in the joints' physical positions for the sake of plotting.

TABLE II: Bent DH Table

	d	θ	a	α
base	l_{01}	θ_0	0	$\pi/2$
shoulder	0	$\pi + \theta_1$	l_2	0
elbow	0	$\pi/2 + \theta_1$	0	$\pi/2$
wrist1	l_{34}	θ_3	0	$-\pi/2$
wrist2	0	θ_4	0	$\pi/2$
gripper	d_{56}	θ_5	$-a_6$	0

The lengths between different joints are OB, BF, FJ, JK, KM, denoted as $l_{01}, l_2, l_{34}, l_5, l_6$.

2) DH Table For Zero Joints: The reason why we did not directly use the upright position, where most joints are zeros from motor reading, as the default DH table, was to avoid the singularity positions (Gimbal Lock) for the inverse kinematics. By using an upright pose as the assigned frames, the zero joints angles would correspond to the zero motor readings, which is easy for debugging and understanding. However, due to the singularity issue with the Euler angle, starting the inverse kinematics analysis from an upright manner turns out to be troublesome. To counter this problem, it is very necessary to introduce another set of frames as in Fig. 13. And its DH table parameters are in chapter VI.

B. Forward Kinematics

From a chain of transformation matrices from frame 0 to frame n, the T matrix between 0 and n is obtained by

$$T_n^0 = T_1^0 T_2^1 T_3^2 \dots T_n^{n-1} \tag{9}$$

FK is implemented is `kinematics.py`. The `FK_dh` function is the main function that returns the end-effector pose matrix as a class `T` instance by passing in the joint angles and the frame number requested.

First, the DH table parameters are determined by joint angles. Second a transformation matrix generator returns the class `T` matrix between the ith and $i+1$th frames by passing the corresponding DH parameters. This generator is recursively called to return a series of transformation matrices. These matrices are multiplied to yield the end-effector pose.

C. Inverse Kinematics

Inverse kinematics reads the end-effector pose and returns the joints angles that can realize the given pose. The method for IK varies, most of which are optimization-based. However, due to the constraints of 6-DOF robots, we can have an explicit analytical solution. IK is implemented is `kinematics.py` in function `IK`.

The 6-DOF robots can be viewed as a three-link to a two-link robot, separated from wrist1. Because both wrist1 and gripper frame is spherical wrist, their rotation would not affect the positions of the frame origins. This enables us to obtain the position of the origin of the wrist1, denoted as $o_c = (x_c, y_c, z_c)$ (line 144-156 in `kinematics.py`).

With the o_c, we can calculate the joint angle for the first three links. Because the DH table used here is the bent one, that its asymmetric shape in x-axis leads to different solutions. So there are three branches need to be addressed, which are $x_c < 0$, $x_c > 0$ and $x_c = 0$. This part is implemented inline 166-185 in `kinematics.py`

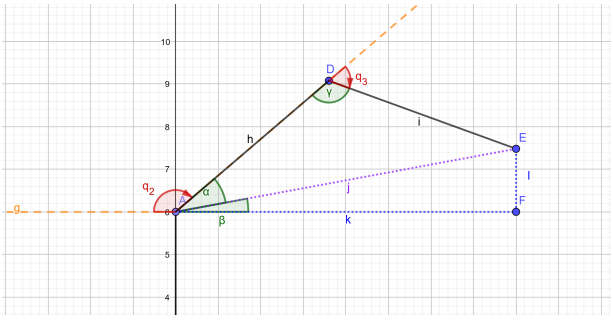


Fig. 8: Inverse Kinematics on Positive x_c

1) Positive x_c: In Fig. 8, the o_c is point E. From o_c and the DH tables, it is convenient to calculate the length of line g, k, i.

First, the joint angle for base can be easily calculated, denoted as q_1. The direction of base origin to o_c in xy plane is the base joint angle (line 158-164 in `kinematics.py`). When $x_c < 0$, the $q_1 = \arctan(yc, xc)$ when implemented in numpy.

Second, as shown in Fig. 8, the second joint angle is q_2, where dashed line v shows the default zero position for link 2 in the default DH table. The arrow on q_2 shows the positive direction on the angle. Obviously, the absolute of $\angle q_2 = \pi - \angle\alpha - \angle\beta$. Here we define $\angle\alpha = \angle DAE$, and $\angle\beta = \angle EAF$. $\angle\alpha$ is calculated by cosine formula within triangle DAE, thus it must be non-negative. $\angle\beta$ is calculated by arcsin of edge l and j, so it can be both positive and negative.

Third, the third joint angle q_3 is the supplementary angle of $\angle\gamma$. γ is calculated by cosine formula within triangle DAE.

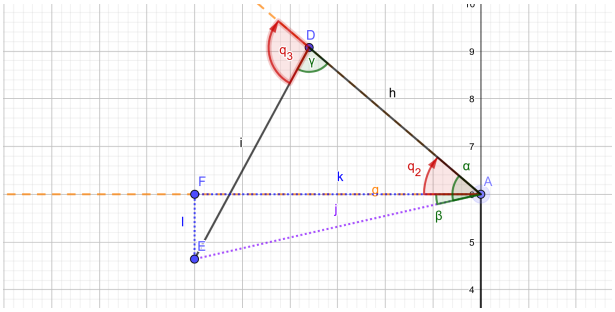


Fig. 9: Inverse Kinematics on Negative x_c

2) Negative x_c: Similar to the positive x_c, the angles are defined the same. The difference lies in how to calculate q_2 and q_3. From line 181-186 in `kinematics.py`, the q_2 is the opposite of the supplementary angle, while the q_3 is merely the opposite to their counterparts under positive x_c.

3) Singularity: The singularity problem occurs when $x_c = 0$. It can be simply solved by assigning $q_1 = \pi$ when $x_c = 0$. Due to numerical stability problems, the threshold is set to $x_c^2 + y_c^2 < 1e^{-14}$

4) The Rest Joint Angles: The rest of the joint angles is calculated by the SO(3) matrix R of the 3rd frame to the 6th as shown in 17. The r are the entries in matrix R

5) Convert To Upright DH Table: The conversion from the default DH table to the upright DH table is simple. They are mere changes of static numbers in joint angles. The output of the previous joint angles need to add $0, -\frac{1}{2}\pi, 0, 0, \frac{\pi}{4}, \frac{\pi}{4}$.

D. Accuracy

The kinematics is validated by teaching and learn.

1) Forward Kinematics: First, grab the motor to a static position predefined on the board. Record the transformation output from the FK. Extract the translation term t from the T matrix, and compared them to the predefined positions.

TABLE III: Kinematics Error

Quad	(x, y)	$(\bar{x}, \bar{y})$	dist	error
1	(129, 75)	(126, 72)	4.24	0.70%
2	(-110, 73)	(-110, 73)	0.00	0.00%
3	(125, -80)	(126, -82)	2.24	0.37%
4	(-115, -77)	(-115, -77)	3.00	0.49%

where the error is the ratio of dist to the length of the board surface (608mm). All other columns are measured in mm.

2) Inverse Kinematics: Direct cal FK and call IK with the FK output, we can compare the initial input of joint angles and the output angles. The difference is smaller than 10^{-5}, which can be seen as zero.

V. Camera Calibration

The camera calibration transform the image plane coordinates into world coordinates. The pixels should be transformed into camera frame by intrinsic matrix, and then into world frame by extrinsic matrix that is manually determined.

A. Intrinsic

The intrinsic matrix is generated in `data/camera_cal.py`. The calibration can be automatically conducted by using OpenCV with `cv2.calibrateCamera` with a chessboard detected by `cv2.findChessboardCorners`.

B. Depth Calibration

Depth Calibration required at least three pairs of corresponding coordinates from the same place on the camera image and depth image. The affine transformation from one set of points to another set of points can be solved by using least square method. In this project, five pairs of points are needed to make the affine transformation more robust to human error. Our custom depth to world coordinates is

$$world_z = -0.0009z^2 - 0.3393z + 712.76 \tag{10}$$

C. Extrinsic

By assigning the clicked points on the board in world coordinates, the extrinsic matrix is another affine transformation to move the xy-plane coordinates from camera frame to world frame. This affine transformation is actually in SE(2) as we only need to adjust the xy-plane.

D. Verification

In fact, we can organize the calibration errors into four categories:

1) the human error by clicking at the corresponding point sets on the RGB and depth image
2) the pixel-wise error by clicking
3) the camera distortion
4) the board surface was not actually flat nor normal to the Kinect focus

The four points used in chapter IV-D are being examined in the chart below. The first three columns are measured in millimeter, and the error equates $dist/608mm$, which is the ratio to the board size.

TABLE IV: Calibration Error

Quad	(x, y)	$(\bar{x}, \bar{y})$	dist	error
1	(129, 75)	(134, 76)	5.83	0.96%
2	(-110, 73)	(-110, 75)	2.83	0.47%
3	(125, -80)	(129, -83)	5.00	0.82%
4	(-115, -77)	(-115, -80)	3.00	0.49%

VI. Block Detection

In the block detection part, we are required to get the color and position of a block from a BGR image and a depth image.

A. Block Detection

Blocks have a discrete depth height in depth image. Assume the height of each layer of the block in depth image is d. We can use the following filter to give an image mask for the following steps:

$$D(i,j) = \begin{cases} [\frac{D(i,j)}{d}] & |\frac{D(i,j)}{d} - [\frac{D(i,j)}{d}]| \leq \epsilon \\ 0 & |\frac{D(i,j)}{d} - [\frac{D(i,j)}{d}]| > \epsilon \end{cases} \tag{11}$$

Here i, j are the index of pixel in image coordinates. ϵ is a threshold deciding if a pixel belongs to block or not.

After getting a mask for blocks from depth, we need to use the second filter is used to distinguish those blocks in a different color. Due to the change of light, it is hard to distinguish colors directly in BGR color space. Thus, we use the dominating color in one pixel to represent the color there. For example, if the r channel of one pixel is

TABLE V: Accuracy and recall of block detection and accuracy of color classification

Detect Accuracy	Detect Recall	Color Accuracy
97.6%	0.0%	96.7%

the largest, it will be regarded as red. And if both r and g channels are large, it is yellow.

Once we classified all colors, we can get pixels that belong to one color and is of the same height. Then, we first use the contour detection algorithm provided by $cv2.findContours()$. After finding those contours, a bounding rectangle estimation algorithm is implemented to those contours through $cv2.minAreaRect$. These bounding rectangles are our approximation of blocks. Their centers and oriented angles are our estimations for the poses of blocks.

B. Strategy for removing false positive

During the whole process of block detection, there can be many false positive detections. Three strategies are implemented to remove them.

1) Robot masking: A major problem for the depth detection step is that blocks are not the only items having height on the table. Our Rexarm also has height in the depth image and can be detected as a block in our pipeline. To solve this problem, we make the Rexarm vertical to the table every time we are making detection and put an image mask on the center of the table to ignore the robot for the block detection.

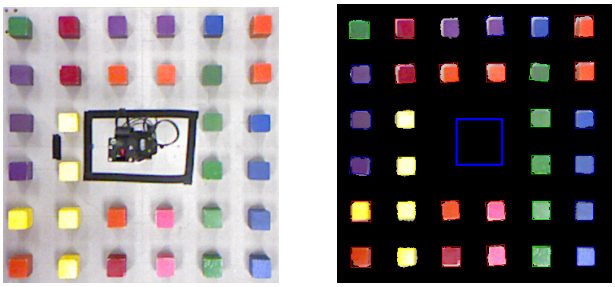
2) Rectangle filter: Because we are only required to detect the blocks in this problem, we can use the information of these blocks to remove that false-positive detection. The area and ratio of the length of width are checked for every candidate rectangle. Rectangles that are too small or too different with a square shape are removed.

C. Evaluation for block detection

To evaluate the performance of block detection, we built a scene as shown in Fig. 10a. We made five calibrations in this scene. Then we performed a block detection after each calibration. Accuracy and recall of block detection and accuracy of color classification are shown in Table. V.

As shown in the table, we can find the accuracy of our detection is 97.6% and recall of detection is 0%, which means the system can remove those false positives. The color classification is 96.7%.

As for the accuracy of the estimated position, the position error for each block estimation in the image coordinate is logged. The change of uncertainty of



(a) Scene for block detection

(b) Result of block detection

Fig. 10: Scene and result for block detection evaluation

estimation with the position on the board is shown in Fig. 11.

As shown in the figure, we can find the estimation error is high when the block is at the boundary of the board and decreases when it is at the center of the board. I think this is because we did not make a 3D calibration, and there is a drift between the bottom and top of the block when the block is at the boundary of the board.

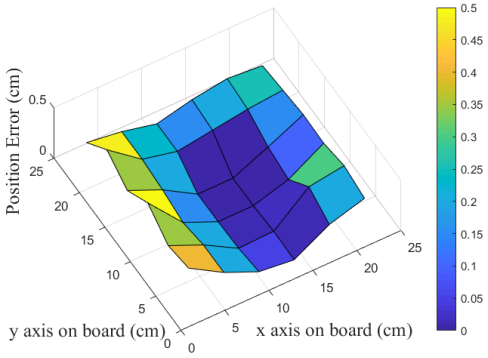


Fig. 11: Uncertainty vs location on board

VII. Algorithm

In this section, the problem formulation for stacking task and task planning for the pick and stack task will be covered.

A. Problem formulation

Stacking task is a task where we need to use our robot arm to pick up blocks and stack them in a pre-defined place in the required color order. In this process, we are only given the rgb image I_r and the depth image I_d, and a pre-defined stack location $p_s = (x_s, y_s)$. The output is the motion plan for our robot.

B. Task planning

The whole system is running iteratively. In each iteration, we first use the depth image I_d to get the depth

in p_s. Through Eq. 12, we can get the current block layer we are stocking.

$$N_{stack} = [\frac{I_d(\pi(x_p, y_p))}{H_b}] \tag{12}$$

Here H_b is the height of a block. π is the mapping from world coordinate to image coordinate.

After knowing the layers of block we are stacking, the block detection system will be run to find the locations of current available blocks. The nearest block from the stacking place of the right color will be chosen to pick.

Once we know the location of the block, we will generate a set of waypoints for trajectory planner. To clearly represent the waypoints, a few definition are made here: $p_h = (x_h, y_h, z_h, \psi_h, \theta_h, \psi_h)$ is the 6D pose of end effector at the pre-defined holding position. $p_g = (x_g, y_g, z_g, \psi_g, \theta_g, \psi_g)$ is the 6D pose of end effector at the gripping position. $p_p = (x_p, y_p, z_p, \psi_p, \theta_p, \psi_p)$ is the 6D pose of end effector at the placing position.

Then the waypoints for the whole picking process W_p will be:

$$W_p = \begin{bmatrix} x_h & y_h & z_h & \psi_h & \theta_h & \psi_h & 0 \\ x_g & y_g & z_g + h_l & \psi_g & \theta_g & \psi_g & 0 \\ x_g & y_g & z_g & \psi_g & \theta_g & \psi_g & 1 \\ x_h & y_h & z_h & \psi_h & \theta_h & \psi_h & 1 \end{bmatrix} \tag{13}$$

Here h_l describes the lift up distance before picking. And the 7th value for each way point is the gripper control value. 1 indicates to open the gripper and 0 indicates to close the gripper.

Then the waypoints will be fed into the trajectory planner to get the full trajectory for picking.

Once the picking stage has been finished, the system continues on placing. Because of the limitation of robot work space, only the first three layers of blocks can placed from top to bottom. For those layers higher than three, the blocks can only be placed from sides. Thus, we are not going to use a lift distance in the placing process, but use a pre-defined pre-place pose p_r instead.

The waypoints for placing will be:

$$W_p = \begin{bmatrix} x_h & y_h & z_h & \psi_h & \theta_h & \psi_h & 1 \\ x_r & y_r & z_r & \psi_r & \theta_r & \psi_r & 1 \\ x_p & y_p & z_p & \psi_p & \theta_p & \psi_p & 0 \\ x_r & y_r & z_r & \psi_r & \theta_r & \psi_r & 0 \\ x_h & y_h & z_h & \psi_h & \theta_h & \psi_h & 0 \end{bmatrix} \tag{14}$$

After these waypoints are fed into the trajectory planner, and actuated by motors. The whole iteration of pick and place is finished.

The whole task planning system will keep running until the required layers for stacking are placed.

VIII. Evaluation & Outlook

A. Gripper evaluation

The vacuum gripper demonstrates sufficient holding pressure to pick up and transport a block, but has a substantial fall time since the vacuum in the suction cup equalizes through small inlet leaks in the design. To reduce this time and complete the task more quickly, we recommend adding a solenoid in-line with the suction cup to quickly open an inlet valve to equalize the pressure.

B. Result evaluation for competition

We successfully finished the task 1 in the competition but failed on the other tasks. An illustration for task 1 is shown in Fig.14 in Appendix. We used about 60 seconds in this task. However, we failed on the other tasks. The main reason for this is because our workspace is quite limited.

C. Further improvement

There are several things left for us to do in the future. The first one is a better mechanical design for the gripper. Although the current compact design makes our gripper very light and smooth in motion, the workspace of our gripper is quite limited. The current arm can not reach places far from the center of the board, which is the main reason why we failed on most tasks in competition.

The second improvement is in the color classification in the block detection system. The dominating color algorithm can not successfully distinguish some similar colors such as pink and red. In the future, we can try to include the lightness of the target pixel to make a better color classification.

IX. Conclusion

We proposed a robot system including block detection, kinematics, trajectory planner and task planner using the Rexarm robot with a vacuum gripper. Instead of using a traditional gripper design, we choose a vacuum gripper and build the corresponding gripper policies for it. This compact design decreases the inertia of the gripper and makes the motion smoother.

We explored the possibility of using a vacuum gripper for the block stacking task and succeeded in task I, blocking stacking task in the final competition. Although we failed some other tasks of the competition, our encountered problems can also guide future students who wants to continue in this direction.

References

[1] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ

[2]

X. Appendix

A. Gripper Design

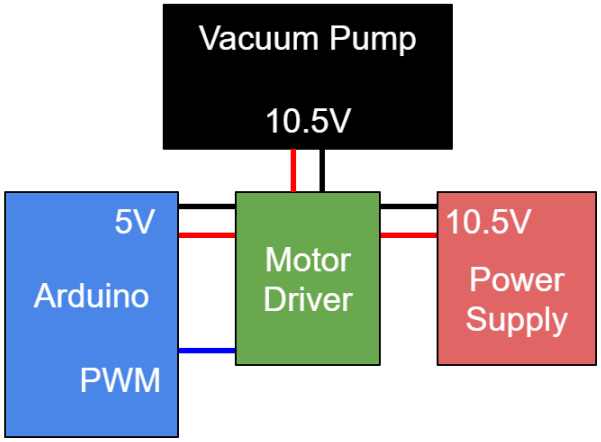


Fig. 12: Circuit schematic of gripper control system.

B. Teach & Repeat State Machine

C. Kinematics

$$SE(3) = \left\{ T = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in SO(3), t \in \mathbb{R}^3 \right\}$$

where

$$SO(3) = \mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}\mathbf{R}^T = I, \det(\mathbf{R}) = 1 \tag{15}$$

$$\begin{aligned} T &= Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \\ &= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{16}$$

TABLE VI: Upright DH Table

	d	θ	a	α
base	l_{01}	θ_0	0	$\pi/2$
shoulder	0	$\pi/2+\theta_1$	l_2	0
elbow	0	$\pi/2+\theta_1$	0	$\pi/2$
wrist1	l_{34}	θ_3	0	$-\pi/2$
wrist2	0	$\pi/4+\theta_4$	0	$\pi/2$
gripper	d_{56}	$\pi/4+\theta_5$	$-a_6$	0

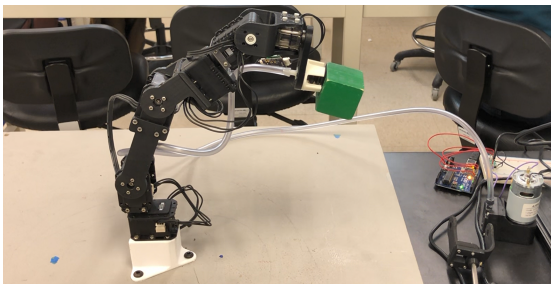


Fig. 14: Our robot succeed in stacking task

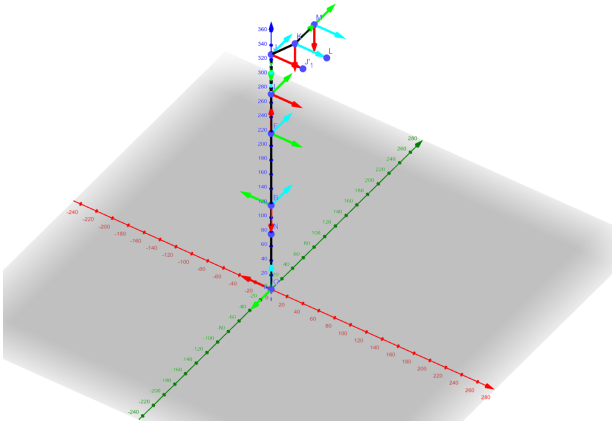


Fig. 13: Upright DH with Zero Joints

$$\begin{aligned} \theta_4 &= \arctan(r_{23}, r_{13}) \\ \theta_5 &= \arctan\left(\pm\sqrt{1-r_{33}^2}, r_{33} \right) \\ \theta_6 &= \arctan(r_{32}, -r_{31}) \end{aligned} \tag{17}$$

D. Camera Calibration

E. Block Detection

F. Algorithm

G. Evaluation & Outlook