ROB 550 Robot Kinematics and Control

Yi Shen, Ke Liu, Ashokkumar Thirumalaesh
{shenrsc, kliubiyk, thiruchl}@umich.edu

***Abstract*—This work aims to control a 5 DOF robot arm to complete multiple tasks. In particular, the arm should pick and drop multiple blocks in specific locations and orientations. The work is separated into two parts: Modelling of the robot arm, Control it with the model.**

I. Introduction

The robot arm used in this work is the ReactorX-200. It has 5 revolute joints and is driven by 5 servo motors. Therefore, it is equipped with 5 degrees of freedom(DOF).[1]

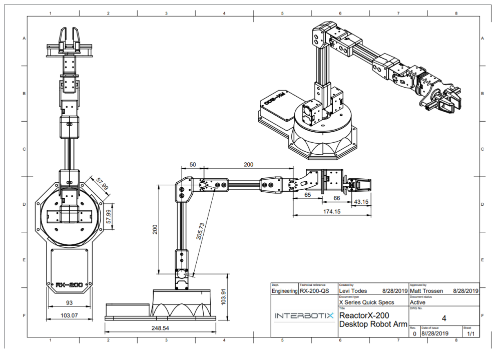

Fig. 1. Technical Drawings of ReactorX-200 robot arm

To control the robot arm, the requirement is to understand how joints' values in the configuration space determine the position and pose of the arm. Then, a forward kinematics model is necessary to describe this process. Conversely, when position and orientation of blocks are known, corresponding joints' values need to be identified to catch the blocks solidly. This is done by a model of the inverse kinematics. Besides, the arm should also have a well defined path from it's current position and pose to a new one such that there are no collisions during the traversal of the path. A path planning model is developed for the same. Finally, we discuss the strategies on accomplishing multiple tasks under the state machine section.

II. Methodology

A. *Forward kinematic solution*

Based on the technical design of RX-200 shown in Fig1, we can derive the schematic of the arm with DH frames. We can then derive the DH-tableI.

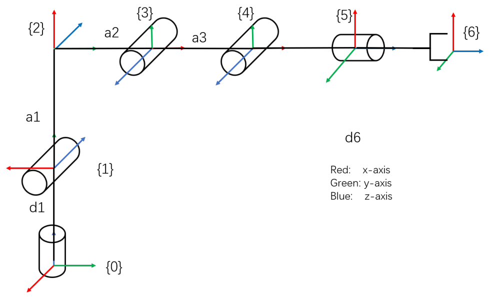

Fig. 2. schematic of the arm with relevant DH parameters

TABLE I
DH-table for the RX-200 robot arm

param	1	2	3	4	5	6
a	0	200	50	200	0	0
α	$\frac{\pi}{2}$	0	π	0	$\frac{\pi}{2}$	0
d	103.91	0	0	0	0	172
θ	$\theta_1 - \frac{\pi}{2}$	$\theta_2 + \frac{\pi}{2}$	$\frac{\pi}{2}$	θ_3	$\theta_4 + \frac{\pi}{2}$	θ_5

We draw the schematic in this specific way following the required conventions. The robot arm's initialized position and the Z-axis of each frame is along the positive direction of a joint's servo. We also draw the 2^{th} frame as a fixed pseudo-frame to simplify further calculations. With the help of this DH-table, we get a series of homogeneous matrices. The matrix in terms of the i^{th} link can be acquired by following equation. In the equation, each angle θ, α and the corresponding offsets are taken from the DH-table.

$$T_i^{i-1} = \begin{bmatrix} c_{\theta i} & s_{\theta i} \cdot c_{\alpha i} & s_{\theta i} \cdot s_{\alpha i} & a_i \cdot c_{\theta i} \\ s_{\theta i} & c_{\theta i} \cdot c_{\alpha i} & -c_{\theta i} \cdot s_{\alpha i} & a_i \cdot s_{\theta i} \\ 0 & s_{\alpha i} & c_{\alpha i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

After we derive this series of matrices, we can multiply them together to get the position and pose of the end effector from the final transformation matrix.

While we built the DH-table based on the blue print, it resulted in errors. And thus, an adjustment to the table was carried out based on the error returned through few test cases. We performed a teach-and-repeat to check the errors during a mission to swap blocks at locations (-100, 225) and (100, 225) through an intermediate location

(250,75). We recorded joints and through those joints and derived the positions of our end effector through FK. They are visualized below: From the figures, it is evident

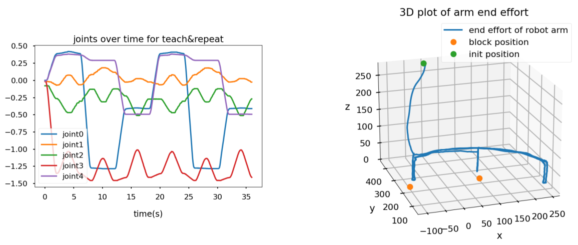

Fig. 3. joints over time for teach&repeat and 3D plot of end effector position

that the end effector is not close enough to the blocks. After comparison, a modification to d_6 from 174.15 to 172 was made so that the end effort can robustly grasp blocks at the middle. In addition, after modified, we can swap for more than 10 times without fail.

B. Inverse kinematic solution

Inverse kinematics(IK) is an essential part of our task, as this model determines the joint values of the servo when we want to catch something in world frame.

In most cases of possible IK, a suitable point along the structure can be found, whose position can be expressed both as a function of: given end-effector position and orientation, reduced number of joint variables.This implies we can articulate the inverse kinematics problem as two subproblems.[2](page 94). For a manipulator with a spherical wrist, the natural choice is to locate a point W at the intersection of the three terminal revolute axes. Although our arm does not have such a construction, we can regard the last two links as a spherical wrist with a fixed first revolute joint. Once the end-effector position and orientation are specified in terms of $\boldsymbol{p}_e$ and $\boldsymbol{R}_e = [\boldsymbol{n}_e \boldsymbol{s}_e \boldsymbol{a}_e]$, the wrist position can be found as[2](page 94,last line)(kinematics.py line206-217)

$$\boldsymbol{p}_w = \boldsymbol{p}_e - d_6 \boldsymbol{a}_e \quad (2)$$

If so, all links before $\boldsymbol{P}_W$ can be regarded as an anthropomorphic Arm. We begin by solving inverse kinematics for them, followed by solving the kinematics for the spherical wrist. Note that, although we have a pseudo-frame, the links before them are also an anthropomorphic arm. Thus, all we need to do is solve for the same and give it the necessary offset based on our specific configuration. If so, we can assume our a_2 is $50\sqrt{17}$ and a_3 is 200 and that the initialized position is parallel to the ground. It follows[2] (page 97,(2.95)-(2.97)) :

$$p_{wx} = c_1(a_2c_2 + a_3c_{23}) \quad (3)$$

$$p_{wy} = s_1(a_2c_2 + a_3c_{23}) \quad (4)$$

$$p_{wz} = a_2s_2 + a_3s_23 + d_1 \quad (5)$$

Then we proceed to computing θ_1,that gives rise to two solutions[2](page 98,(2.109,2.110)(kinematics.py line221-227):

$$\theta_{1,1} = Atan2(p_{wy}, p_{wx}) \quad (6)$$

$$\theta_{1,2} = Atan2(-p_{wy}, -p_{wx}) \quad (7)$$

Further, we have an equation for θ_3[2] (page 97,(2.98))((kinematics.py line234)):

$$c_3 = \frac{p_{wx}^2 + p_{wy}^2 + (p_{wz} - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \quad (8)$$

$-1 <= c_3 <= 1$ is a necessary condition, else the wrist point is found to be outside the reachable workspace. Thus, we can get two solutions of θ_3 with c_3. And once we acquire θ_3, the θ_2 becomes certain and we can get it through the ratios of a triangle.(.py line247-252)

$$tan\alpha = \frac{a_3sin\theta_3}{a_2 + a_3cos\theta_3} \quad (9)$$

$$\alpha = atan2(a_3sin\theta_3, a_2 + a_3cos\theta_3) \quad (10)$$

$$\theta_2 = atan2(p_{wz} - d_1, \sqrt{p_{wy}^2 + p_{wx}^2}) - \alpha \quad (11)$$

In the end, we input our offsets and acquire our final solutions:(kinematics.py line256-278)

$$\theta_{2final} = atan(4) - \theta_2 \quad (12)$$

$$\theta_{3final} = atan(4) + \theta_3 \quad (13)$$

With θ_1,θ_2,θ_3, we can compute the rotation matrix R_4^0.Further, we have $R_6^4 = R_4^{0^T} R_6^0$. As we only has 2 unknown angles, the rotation matrix has the following equation:(kinematics.py line299-312)

$$R_6^4 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_4c_5 & -c_4s_5 & s_4 \\ s_5 & c_5 & 0 \\ -s_4c_5 & s_4s_5 & c_5 \end{bmatrix} \quad (14)$$

The constant matrix is used as a rotation offset to rotate our frame4 to the same pose of frame6 at the initialized position. Then the ZYZ Euler angles degenerate to YZ, and form the equations shown. We shall term the matrix formed by YZ angles as R_{yz}. Hence, we can solve for θ_4 and θ_5

$$\theta_4 = atan2(R_{yz}[1,3], R_{yz}[3,3]) \quad (15)$$

$$\theta_5 = atan2(R_{yz}[2,1], R_{yz}[2,2]) \quad (16)$$

1) Verifying the accuracy:

We have four solutions for the IK since we have two separate solutions for θ_1 and θ_3. However, if it gives rise to singularity, degenerate poses or unreachable areas, we need an mechanism to account for the same. Firstly,our naive solution is to choose the best option from 4 solutions. As servos are constricted on their angles of rotation, $\theta_{1,2} = Atan2(-p_{wy}, -p_{wx})$ is not a good approach as it is difficult for RX200 to rotate towards it's backside. Similarly, we want to keep θ_{2final} less than 0, and so we prefer a better choice for $theta_3$. Now, we have only one solution to verify. If any position can not be reached, that is, $-1 <= c_3 <= 1$ can not be satisfied, we return an error. However, if the position is reachable but the pose is not, we will still have a R_{yz} matrix, but it is considered invalid. During these situations, we verify after receiving a potential solution. We put joint angles generated by our IK into FK to get a transform matrix T_g. We then use the position and pose of the end effector given, to get a transformation matrix T_c. Given the veracity of our solution, we should have following euqation:

$$det(T_g - T_c) < \epsilon \tag{17}$$

ϵ is a threshold close to zero. However, real world scenarios have constraints like servo ranges and self-collision poses. Therefore, in such cases our legal IK solution may not be applicable. This is a topic beyond theoretical IK and as such will be discussed in path planning.

C. Path planning

1) Collision check: Collisions can be classified into two cases: manipulator-environment collision and manipulator-manipulator collision.[3] Collision checks are essential when our IK solutions are legal but are not necessarily applicable, due to the possibility of a collision. In our case, all obstacles are formed by blocks. Thus based on theory surrounding bounding boxes [3], we choose cylinders to simplify both obstacles and links.

Distance check between two cylinders is shown below:

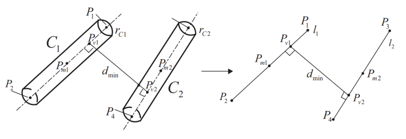

Fig. 4. Collision detection of cylinder-cylinder

First, we compute the shortest distance d_min between 2 lines

$$l_1 : P_{l1} = P_1 + \lambda_1 \boldsymbol{s_1}; l_2 : P_{l2} = P_3 + \lambda_2 \boldsymbol{s_s}$$

$$\boldsymbol{s_1} = P_2 - P_1, \boldsymbol{s_2} = P_4 - P_3, 0 <= \lambda_1, \lambda_2 <= 1$$

Computing d_min equals solving a minimisation-optimization problem with the constraint:

$$\begin{aligned} min f(\lambda_1, \lambda_2) = ||(P_1 + \lambda_1 \boldsymbol{s_1}) - (P_3 + \lambda_2 \boldsymbol{s_2})||^2 \\ s.t. 0 <= \lambda_1, \lambda_2 <= 1 \end{aligned} \tag{18}$$

Then we have $\frac{\partial f}{\partial \lambda_1} = 0, \frac{\partial f}{\partial \lambda_2} = 0$:

$$\begin{aligned} \lambda_1 &= \frac{(\boldsymbol{s_1 s_2})[(P_1 - P_3) \cdot \boldsymbol{s_2}] - ||\boldsymbol{s_2}||^2[(P_1 - P_3) \cdot \boldsymbol{s_1}]}{||\boldsymbol{s_1}||^2||\boldsymbol{s_2}||^2 - (\boldsymbol{s_1} \cdot \boldsymbol{s_2})^2} \\ \lambda_2 &= \frac{(\boldsymbol{s_1 s_2})[(P_1 - P_3) \cdot \boldsymbol{s_1}] - ||\boldsymbol{s_1}||^2[(P_1 - P_3) \cdot \boldsymbol{s_2}]}{||\boldsymbol{s_1}||^2||\boldsymbol{s_2}||^2 - (\boldsymbol{s_1} \cdot \boldsymbol{s_2})^2} \end{aligned} \tag{19}$$

if $0 <= \lambda_1, \lambda_2 <= 1$, then $d_{min} = f(\lambda_1, \lambda_2)$. Otherwise, we shall check the projection of the points to the lines. For example, if we wanted to check the projection from P3 to line1:

$$\lambda_1 = \frac{[(P_3 - P_1) \cdot \boldsymbol{s_1}]}{||\boldsymbol{s_1}||^2} \tag{20}$$

if$0 <= \lambda_1 <= 1$ then $d_{min} = f(\lambda_1)$, and So do other points and lines. Else,$d_{min} = \{||P_1P_3||, ||P_1P_4||, ||P_2P_3||, ||P_2P_4||\}_{MIN}$ if$d_{min} > r_{C1} + r_{C2} + d_s$, then this two cylinders are collision free. d_s is a safe offset.

In our work, obstacles are blocks on a board. As we can detect the height of the blocks, we assume their height as their length and the diagonal length as the radius for the cylinders. For links, we measure their lengths and radii and get their position by FK. We then achieve a map for the robot arm, to do path planning. However, there is an exception. For adjacent links, the method of using two cylinders cannot apply, as the method will consider them to be colliding objects. For those links, we acquire a safe range from the website of the robot arm and check whether joint angles are within safe range.

2) RRT algorithm: For this robotic arm path planning task, we use Rapidly-exploring Random Tree (RRT) algorithm and thus has to be implemented for 5DOF.

The pseudo code for the algorithm is shown below[4]:

Algorithm 1 Pseudocode for RRT Algorithm

Input: Start config P, Goal config Q

Output: A path formed by list of 5DOF joints for the arm to reach to the goal config from start config

```
RRT_Connect($q_{init}$)
    T.init($q_{init}$)
  for k = 1 to K do
    $q_{rand}$ = RANDOM_CONFIG
    EXTEND_TREE(T, $q_{rand}$)
  end for
```

This algorithm helps generate a random tree that extends from the start configuration to goal configuration. To optimize the RRT algorithm, we tune the randomness of creating a random configuration, the step size, and the bias value measuring the distance between current position and final goal.

(a) random_config()
This helps in generation of random points of joint configurations(in 5DOF), denoted as $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$. For each iteration, we try to extend the tree towards this random point. So we randomly generate a point within the safe range(the

range limit of each joint in robot arm), with 10% possibility of generating the goal configuration.

(b) nearest_neighbor()
After generating the next set of joints, $q = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$, we shall find the node in the tree, that is the nearest to connect the same with q.

(c) extend_tree()
By using $step\_size = 0.1$, we will extend the nearest node $p = (x_1, x_2, x_3, x_4, x_5)$ towards the next configuration $q = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$. For each step, we take size=0.1 to reach the q. The extended point is $q_{new} = (\alpha_1 - x_1, \alpha_2 - x_2, \alpha_3 - x_3, \alpha_4 - x_4, \alpha_5 - x_5)/d * step$, where $d = np.linalg.norm(p - q)$ represent the Euclidean distance between p and q. Here, we will have three different status:"Reached","Advanced", and"Trapped". "Reached" implies that by taking this step, the extended point q_{new} is within the $bias = 0.1$ distance to the goal configuration. "Advanced" implies that the q_{new} has not reached the goal configuration yet. "Trapped" denotes that the newly found extended point q_{new} collides with its links or obstacles.
Thus, if the status is "Trapped", we will record all nodes that we extended previously for the tree, and start another round for extending path towards new random configuration q. If the status is "Advanced", we can continuously extend the nearest node p towards the random configuration q step by step, until it either reaches the goal configuration (status = "Reached"), or is trapped through collision check (status = "Trapped").

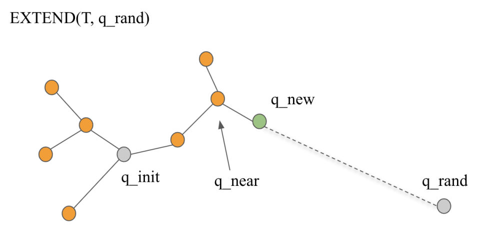

Fig. 5. RRT algorithm for extending tree towards random config q [4]

(d) path_smooth()
The $path\_smooth()$ function utilize the Short-cut Smoothing method to shorten the path in Algorithm2.

After generating a long path for robot arm to reach the goal from start, we may shorten the path by picking the shortest path between multiple points. We set the $Maxiter$ to 50, so it will allows us to generate random indices p_1, p_2 for 50 times, randomly within the original path length. And we will also try to shorten the path using $extend\_tree()$ to check if connecting p_1, p_2 is possible. If the connecting step is "Advanced", it will continuously try extending the tree towards p_2, until it has either "Reached"(success) or "Trapped". We have tuned the $Maxiter$ to 50 as this maximum iteration will narrow down the original path to within about 5 joint positions, which saves enough time for robot arm to execute through all joint positions to reach the goal efficiently.

Algorithm 2 Short-cut Smoothing[4]

```
for i in range(Maxiter) do
    p1 ← random(0, $path\_len - 1$)
    p2 ← random(0, $path\_len - 1$)
    if p1 < p2 then
        if SHORTEN(p1, p2) is success then
            p1 is p2's parent node
        end if
    else if p1 > p2 then
        if SHORTEN(p2, p1) is success then
            p2 is p1's parent node
        end if
    else
        continue
    end if
end for
```

D. State machine

Although we deal with different tasks, a certainty is that we shall pick blocks from a position and place it at another location with high precision and accuracy. Therefore, the mechanism used to pick and place the blocks are essential.

1) Grasp technique: As our robot arm has only 5 DOF, it can not approach blocks at arbitrary poses. If a block is close enough to our arm, the arm can approach from above it and grasp it in the desired manner. Else, the arm will pick the block, but may not be so through a desired pose. These scenarios are depicted below.

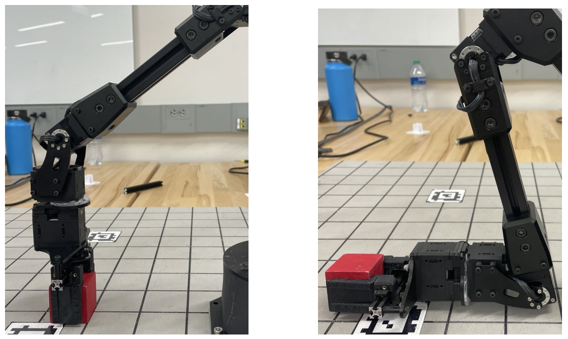

Fig. 6. grasp in good manner and secondary manner

Therefore, our control logic is that, we will try the 'secondary manner' as the approach mechanism only if we our 'desired manner' is not possible. In order to not disturb blocks before we actually hold it, we make the gripper to be positioned right above the block before translating it orthogonal to the board and picking the block. After this step, we will move the arm straight upwards, to avoid scrapping the ground with the block or the gripper.

At the same time, the block detections can have some errors about the position and pose of the blocks. As a result, our gripper may not grasp the blocks at the center but on one side. To solve this, we catch blocks in 2 steps and are termed as *align step*. Firstly, we pick the block. We release the gripper, go up and rotate the gripper by 90 degrees. We then catch the block again. This will reduce the error along the horizontal axis and vertical axis separately.

2) Place technique: Place technique is almost the same as grasp. We place first and then apply the *align step*. Besides, when we place, the position is just considered as a rough reference. Our arm will gradually go down, and when the torque in the third servo is large(200 more than that when it started to go down), we assume to have reached the right position.

3) Task1: In task one, we are given random blocks placed in the positive half plane, and our mission is to drop small blocks to the left, and big ones to the right of the arm, in the negative half plane. Our logic is as shown below:

Algorithm 3 logic for task1

```
set place in negative half sphere to place blocks
while time is not over do
    detect blocks
    sort blocks in order by distance
    for each block do
        if block is in negative half sphere then
            continue
        else
            catch it and place it to the right place
            renew the place to place blocks
        end if
    end for
end while
```

4) Task2: In task two, we are given random blocks in the positive half plane and our mission is to stack them up vertically. We repeat steps of task1, except that we place all blocks on the right side. Then, we pick them and stack them up in order. The location to drop blocks will be revised each time.

5) Task3 and 4: Task3 and Task4 are almost the same, except that we shall stack in the order of a rainbow for 4, but sort in 3. Our logic is as shown below:

Algorithm 4 logic for task3 and 4

```
do exactly what task1 do
set place to stack/line blocks
while time is not over or task is not finished do
    detect blocks
    sort blocks in order by color and size
    for each block do
        if block is in stacked/lined then
            continue
        else
            catch the desired block and stack/line it to the
            assigned place
            renew the place to stack/line blocks by CV
            break
        end if
    end for
end while
```

6) Task5: In task5, we want to stack big blocks to the maximum possible height. We do so by stacking a small tower and a high tower separately and then stacking the small one onto the high one later.

III. Results and Discussion

We have completed Task1 to Task5 efficiently during the competition, with the exception of task2 moving into overtime, due to errors arising in difficult block orientation and locations. Repeated tries by the arm to catch the block, led to the overtime. However, as a high point, we stacked 15 blocks in Task5.

Although we have done tasks successfully, there were some potential trivial problems that could be improved.

Firstly, when we carry a block, the position of our end effector will be lowered by gravity. In general, we set an offset to the third servo to overcome this but a fixed offset may not be good enough always. Perhaps a better offset mechanism should be implemented, as a function of the pose of the robot arm.

In addition, we set the accelerate-time and moving-time for robot arm manually. It would be efficient to assign them based on the length of the path and distribution of the obstacles.

References

[1] T. Robotics, "Reactorx-200," http://support.interbotix.com/html/specifications/rx200.html.

[2] *Kinematics*. London: Springer London, 2009, pp. 39–103. [Online]. Available: https://doi.org/10.1007/978-1-84628-642-1_2

[3] Y. Shen, Q. Jia, G. Chen, Y. Wang, and H. Sun, "Study of rapid collision detection algorithm for manipulator," in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, 2015, pp. 934–938.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

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