Kinematic Simulation of UR-3 in V-REP

Liao Jiaqi
Department of Mechanical Science and Engineering
University of Illinois at Urbana-Champaign
Champaign, USA
jiaqi5@illinois.edu

Zhou Hongyi
Department of Mechanical Science and Engineering
University of Illinois at Urbana-Champaign
Champaign, USA
hzhou39@illinois.edu

Jin Lukai
Department of Mechanical Science and Engineering
University of Illinois at Urbana-Champaign
Champaign, USA
ljin6@illinois.edu

***Abstract*—In this report, we demonstrated the basic kinematic control of UR-3 robot. We derived the solution of forward and inverse kinematics problem based on screw theory. A path-planning algorithm was developed by random sampling and tree structure. An outline for future work and the final robot simulation in v-rep was shown at the end.**

Keywords— forward kinematics, inverse kinematics, path planning, UR-3 robot, V-rep

I. Introduction

Kinematics, as the foundation of robot control, studies the relationship between the dimensions and connectivity of kinematic chains and the position, velocity of each links in the robotic system. Forward kinematics uses the kinematic equations of a robot to compute the position of the end-effector from given joint angle parameters, while inverse kinematics does the reverse process. In addition for end-effector to reach desired position, it's also important to make sure no collision with either the robot itself or other obstacles during the course of its movement.

V-rep, virtual robot experimentation platform is an efficient robot simulator with comprehensive physic engine and easy to use remote api function. Algorithms of forward, inverse kinematics, and path planing are implemented with UR-3 robot using Matlab.

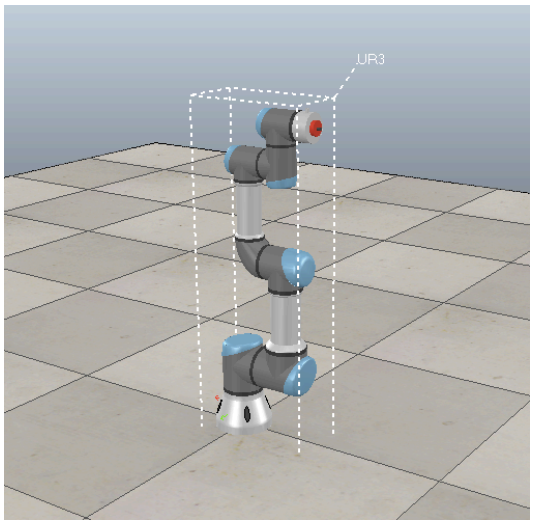

Fig. 1. UR-3 robot in V-rep

II. Forward Kinematics

Kinematics, including forward kinematics and inverse kinematics are the basis of robot control. Forward kinematics computes the position of the end-effector from given joint angle parameters. One common way to solve the forward kinematics problem is Denavit-Hartenberg (D-H) method, which utilizes coordinate transform matrix. In this report, we adopt screw theory to solve the problem.

A. Schematic of the UR-3 Robot

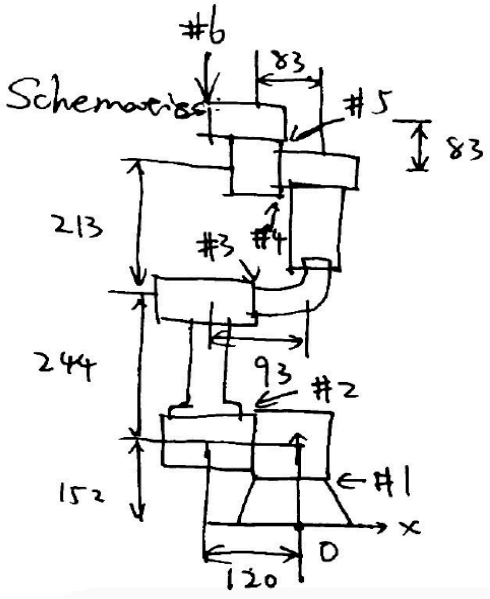

Fig. 2. Schematic of UR-3 robot

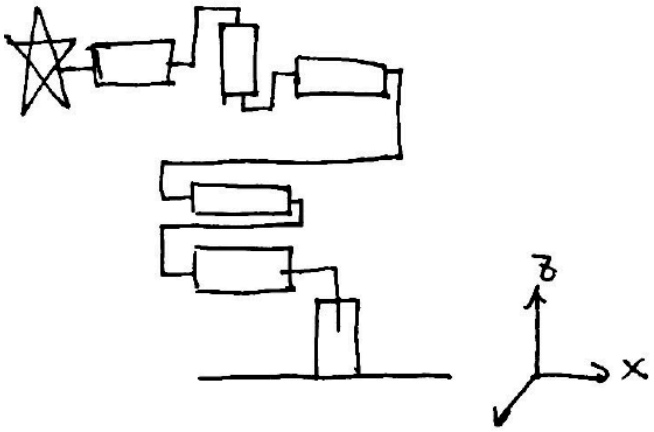

Fig. 1. Screw axis represenation of UR-3 robot

B. Computation of the Pose of Tool Frame

For a revolute joint, its spatial twistis given by

$$\gamma_s = \begin{bmatrix} a \\ -[a]q \end{bmatrix}\theta$$

where a is the joint orientation, q is the position of the joint in the base frame.

Since $\dot{T_1^0}(T_1^0)^{-1} = [\gamma_{0,1}^0]$, we have $T_1^0(t) = e^{[S]\theta}T_1^0(0)$.

Let t be 1 second, the equation becomes $T_1^0 = e^{[S]\theta}M$, which is the kinematics of a single joint.

For the UR-3 robot which has 6 joints, its forward kinematics equation is

$$T_1^0 = e^{[s1]\theta 1} * e^{[s2]\theta 2} * e^{[s3]\theta 3} * e^{[s4]\theta 4} * e^{[s5]\theta 5} * e^{[s6]\theta 6} * M$$

where M is the initial pose, s represents scres axis, and θ represents joint angles [1].

C. Data of UR-3 Robot

In order to implement the algorithm, we need data of screw axis of the robot, and its initial pose. Data of screw axis is acquired by collecting the position of each screw axix and its orientation from the v-rep simulator; whereas initial pose is obtained by writing the rotation matrix and relative position of tool frame in terms of base frame with data from v-rep.

```
   a1 = [0;0;1]; p1 = [0;0;0.152]
a2 = [-1;0;0]; p2 = [-0.12;0;0.152]
a3 = [-1;0;0]; p3 = [-0.12;0;0.396]
a4 = [-1;0;0]; p4 = [-0.027;0;0.609]
 a5 = [0;0;1]; p5 = [-0.11;0;0.692]
a6 = [-1;0;0]; p6 = [-0.11;0;0.6941]
```

$$M = \begin{bmatrix} 0 & 0 & -1 & -0.1941 \\ 0 & 1 & 0 & 0.025 \\ 1 & 0 & 0 & 0.6941 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

III. Inverse Kinematics

Inverse Kinematics deals with the 'reverse' problem from Forward Kinematics, i.e. given the target and initial pose of the robot, find the corresponding joint angles which would move the robot to the target position.

A. Algorithm

Consider a robot with only one joint (n=1). From the forward kinematics, we have

$$T_1^0 = e^{[S]\theta}M,$$

where M is the initial pose of the system, i.e. $M = T_1^0(t_0)$.

Also, we have

$$\dot{T_1^0}(T_1^0)^{-1} = [V_{0,1}^0],$$

where $V_{0,1}^0$ is called the spatial twist of the robot.

Since $(T_{0,1}^0)^{-1} = M^{-1}(e^{[S]\theta})^{-1}$,

$$\dot{T_1^0} = [S]e^{[S]\theta}\dot{\theta}M,$$

we have

$$[V_{0,1}^0] = \dot{T_1^0}(T_1^0)^{-1} = [S]\dot{\theta},$$
$$V_{0,1}^0 = S\dot{\theta} = J\dot{\theta}.$$

When the robot has more two joints, we have

$$T_1^0 = e^{[S_1]\theta_1}e^{[S_2]\theta_2}M,$$
$$\dot{T_1^0} = \left([S_1]e^{[S_1]\theta_1}\dot{\theta_1}e^{[S_2]\theta_2} + e^{[S_1]\theta_1}[S_2]e^{[S_2]\theta_2}\dot{\theta_2}\right)M,$$
$$(T_1^0)^{-1} = M^{-1}(e^{[S_2]\theta_2})^{-1}(e^{[S_1]\theta_1})^{-1},$$
$$[V_{0,1}^0] = \dot{T_1^0}(T_1^0)^{-1} = [S_1]\dot{\theta_1} + \left[[Ad_{e^{[S_1]\theta_1}}]S_2\right]\dot{\theta}_2.$$

From the equations above, we have

$$V_{0,1}^0 = \begin{bmatrix} S_1 & [Ad_{e^{[S_1]\theta_1}}]S_2 \end{bmatrix}\begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} = J\dot{\theta},$$

where J is denoted as space Jacobian of the robot and adjoint operator $[Ad_T]$ is defined as

$$[Ad_T] = \begin{bmatrix} R & 0_{3\times 3} \\ [p]R & R \end{bmatrix}$$

for

$$T = \begin{bmatrix} R & p \\ 0_{1\times 3} & 1 \end{bmatrix}$$

Similarly, for the robot with n joints, if we denote space Jacobian as

$$J = [J_1\ J_2\ \cdots\ J_n],$$

then for k=1, 2, …, n

$$J_k = \left[Ad_{e^{[S_1]\theta_1}e^{[S_2]\theta_2}\ldots e^{[S_{k-1}]\theta_{k-1}}}\right]S_k.$$

Suppose the robot has 6 joints and denote T_1^0 as current pose, T_2^0 as target pose. We assume that the robot moves with spatial twist $V_{0,1}^0$ for one second, given S, M, T_2^0, we choose an initial guess of joint angles θ. While the current value of θ is not close enough to the target pose T_2^0, i.e. $\|V\| > \epsilon$, iterate the following steps:

- find the current pose T_1^0
- find the spatial twist which could move the robot from the current pose T_1^0 to the target pose T_2^0

$$[V] = \log(T_2^0(T_1^0)^{-1})$$

- find the space Jacobian

$$J = [J_1\ J_2\ \cdots\ J_6]$$

- find the joint velocity

$$\dot{\theta} = J^{-1}V$$

- update the joint angles

$$\theta = \theta + \dot{\theta}$$

B. Some Discussion

1) Are there tool poses for which no solution exists? What does your algorithm do in this case?

There exists tool poses where no solution exists. The target poses could be outside the range where the robot could reach. To solve this problem, the loop structure described above would have a maximum limit of times it looped. If the times it looped exceeded the maximum limit, the algorithm would terminate itself and display 'No solution found'.

2) Are there tool poses for which more than one solution exists? What does your algorithm do in this case?

There always exists more than one solution to a target pose since the UR3 has 6 revolute joints whose axis are not aligned to each other. The algorithm would randomly choose a solution and it would only give out one solution every time.

3) Are there tool poses for which the only solutions are singular configurations? What does your algorithm do in this case?

Yes. Singularities usually occur when the robot's links are lined up straight and/or when a joint approaches zero

degree. This scenario occurs when the robot tried to reach a position as far as it could reach. The best way to solve this case is never make the target frame too far from the base. Another way to solve this is make the axis of the tool always vertical.

IV. Motion Planning

Motion Planning makes it possible to move a robot without colliding with itself and/or the obstacles. The implement of Motion Planning involves two parts, collision detection with given joint variables and searching a collision-free path from start to the destination.

A. Collision Detection Algorithm

1) Collision detection at a given configuration

Suppose

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix},$$

then denote

$$\underline{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}.$$

From Forward Kinematics for robot with n joints, we know

$$T_1^0 = e^{[S_1]\theta_1} e^{[S_2]\theta_2} \dots e^{[S_n]\theta_n} M.$$

It's also true for a point p. Then we have

$$\underline{q} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} \dots e^{[S_n]\theta_n} \underline{p},$$

where $\underline{p}$ is the initial position of p.

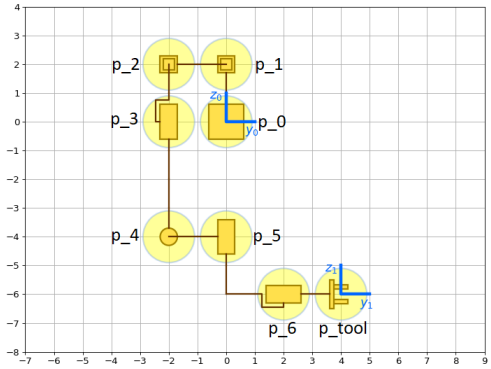

Fig. 4. Notation for e.

Then, the target position for each joint would be

$$\underline{q_0} = \underline{p_0},$$
$$\underline{q_1} = \underline{p_1},$$
$$\underline{q_2} = e^{[S_1]\theta_1} \underline{p_2},$$
$$\underline{q_3} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} \underline{p_3},$$
$$\underline{q_4} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} \underline{p_4},$$
$$\underline{q_5} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} \underline{p_5},$$
$$\underline{q_6} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} e^{[S_5]\theta_5} \underline{p_6},$$
$$\underline{q_{tool}} = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} e^{[S_5]\theta_5} e^{[S_6]\theta_6} \underline{p_{tool}}.$$

The general idea to detect collision between several objects is checking the distance between the objects. For two spheres of radius r_i and r_j have centers at points q_i and q_j respectively, we have

$$c = \|q_i - q_j\| - (r_i + r_j).$$

Then the two spheres are in collision if $c \leq 0$.

Note that the robot would collide not only with itself, but also the given obstacles.

2) Collision detection along a given path

Suppose we have θ_{start} and θ_{goal}, for arbitrary position θ along the path between θ_{start} and θ_{goal}, we have

$$\theta = (1 - s)\theta_{start} + s\theta_{goal}, s \in [0, 1].$$

Then we can check collision at the given θ. Furthermore, s should be set as a discrete set of values with a fixed gap, e.g.

```
s = 0:0.01:1
```

B. Path Planning Algorithm

The Path Planning Algorithm describe below is based on the idea of random sampling. Initialize the forward tree and backward tree before the iteration first, i.e.

- add θ_{start} to the forward tree $T_{forward}$ with parent 0
- add θ_{goal} to the backward tree $T_{backward}$ with parent 0;

Repeat the following steps n times, then return an error message:

- generate a random set of joint variables θ that is not in collision
- find the closest node $\theta_{forward}$ in $T_{forward}$ to θ
- if it is collision free along the path between $\theta_{forward}$ and θ, add θ to $T_{forward}$ with parent $\theta_{forward}$
- find the closest node $\theta_{backward}$ in $T_{backward}$ to θ
- if it is collision free along the path between $\theta_{backward}$ and θ, add θ to $T_{backward}$ with parent $\theta_{backward}$
- if $\theta \in T_{forward}$ and $\theta \in T_{backward}$, return $T_{forward}$ and $T_{backward}$

We could get $T_{forward}$ and $T_{backward}$ from the algorithm above. Then we can get the matrix q contains sets of joint variables in sequence since we have the parent of each point.

C. Some Discussion

1) Are there configurations at which your collision detector returns a false positive (i.e., says there is a collision when there is not) or a false negative (i.e., says there is not a collision when there is)? Why? How could you improve your collision detector?

There is never observed with a false positive in my collision detector. However, a false negative occurred quite often, especially in V-REP simulation. The main reasons are listed as following.

- The amount of sample points along the path between θ_{start} and θ_{goal} is not enough. It could be solved by making the gap of s smaller, i.e. change s from

```
s = 0:0.1:1
```

to

```
s = 0:0.01:1
```

- The robot in V-REP doesn't move every joint in the same time. The joint velocity is not controlled by the algorithm when moving the robot arm, so it is possible that some of all the joints would reach the goal joint angles before the other joints do. So, the equation we used to sample the points along the path, i.e.

$$\theta = (1 - s)\theta_{start} + s\theta_{goal}, s \in [0, 1],$$

could be different from the joint angles occurred along the path in V-REP. This is the problem, which could hardly be fixed without controlling the joint velocity every time when moving the robot.

2) Are there choices of start and goal between which no collision-free path exists? What does your planner do in this case?

The scenario exists when the robot is surrounded by obstacles. The path planner would only take limited attempts and return an error message with 'No path found'.

3) How much time is required to plan a collision-free path? How does planning time vary, both from one run to the next for a given start and goal configuration, and between different start and goal configurations?

Typically, it takes less than two seconds to plan a collision-free path. The planning time depends on the density of s in collision detection along a path, the number of joints, the number of obstacles and the distance between obstacles and initial/goal position.

V. Future Work

A. Statement of Goal

We hope to make a simulation of UR-3 robot picking up random placed blocks and stacking them in the fashion of a Jenga tower.

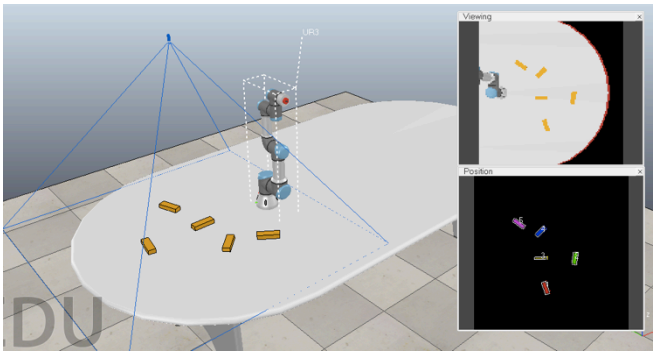

Fig. 5 Final Project Scene

B. Plan and Goal Breakdown

1) Scene components: Large table, 24 cuboid blocks, UR-3 robot arm with suction cup, camera (vision sensor)

2) Workflow:

a) Camera (vision sensor) sensing the orientation and x,y coordinates of each block on the table and return a matrix of position and orientation.

b) Commanding UR-3 Robot to move to the position of each block (inverse kinematics), turn on suction cup, and move the block to the target position with desired orientation.

c) Blocks are stacked in a way that 3 parallel blocks form a layer, and the orientation of each layer is perpendicular to the next layer. The position of each block will be calculated. We'll use path planing algorithm to determine the path.

d) If time allows, we will attempt to add another UR-3 robot with end effector of a stick to actually play Jenga. User can select a block from the tower by clicking on the surface of the block. The robot arm will then push out the block that the user selected. This would involve the use of inverse kinematics and path planing.

C. Challenges

1) Figure out the position of each block relative to the world frame from vision sensor, i.e., translate from the camer frame to world frame.

2) Calculate the position and orientation of each block in the stacked tower.

3) Move the block to the desired location with efficient path planing algorithm

Acknowledgment

We thank Prof. Bretl for assistance with manipulating vision sensor data that greatly improved the project.

References

[1] Lynch K.M and Park F.C, "MODERN ROBOTICS: MECHANICS, PLANNING, AND CONTROL" Cambridge University Press, 2017. pp. 140-148.