Jarvis - Object Manipulation for Industrial and Household Applications

Lohitaksh Gupta
Electrical and Computer Engineering
College of Engineering
University of Illinois
Urbana - Champaign, IL 61820, USA
lg6@illinois.edu

Miguel Jimenez Aparicio
Electrical and Computer Engineering
College of Engineering
University of Illinois
Urbana - Champaign, IL 61820, USA
miguelj2@illinois.edu

Kehan Long
Electrical and Computer Engineering
College of Engineering
University of Illinois
Urbana - Champaign, IL 61820, USA
kehan2@illinois.edu

***Abstract* —We are going to use V-REP simulator to show how our robotic arm, UR-3, will suck a cube and a glass full of balls from their initial pose in a table. The first one will be reallocated in a conveyor belt and the balls will be dropped in a basin. Concepts such as Forward Kinematics, Inverse Kinematics and Motion Planning will be used in order to implement the sorting of objects.**

Keywords — Forward Kinematics, Inverse Kinematics, Motion Planning, Suction Cup, Robotic Arm, UR-3 and V-REP.

I. Introduction

The minimum requirement for this project was given in this following statement:

"Created a dynamic simulation (i.e., a simulation of real physics) in which at least one robot - with at least one arm that has at least six joints - moves at least one object (e.g., by pushing or grasping) from a random initial pose to a given final pose."

Going beyond that, we decided to attach a suction gripper to our robot and pick two objects - a cube and a cup full of balls - from their initial position and, in the case of the cube, release it in a conveyer belt. For the glass, the robot takes it and moves until it is located above the basin. Then, it rotates the last joint so all the balls fall from the glass. After that, the robot leaves the empty glass in its initial place.

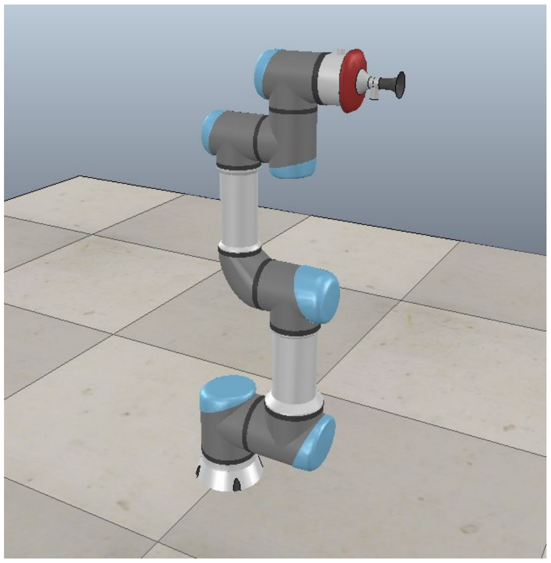
The final goal of our project is to show the capability of the robot to perform tasks both in industrial and household taks, such as reallocation and manipulation of objects and automated search of a path without collision with itself and the environment

For this project, we have used the following items:

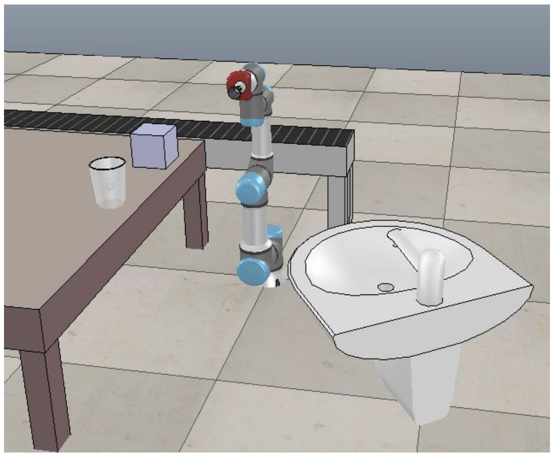
- Robot: UR-3
- Simulator: V-REP
- Programming Language: Python

In the first picture, our UR-3 robot is shown. It is possible to notice the Baxter suction cup attached as its end effector. In the second picture, the scene with conveyor belt, the table where the objects are located initially and the basin can be appreciated.

Picture of robot in a simulator:



Picture of the V-REP scene:



Link to GitHub Repo: https://github.com/lohitakshgupta/ece470_object-sorting

Summarize of the solution approach:

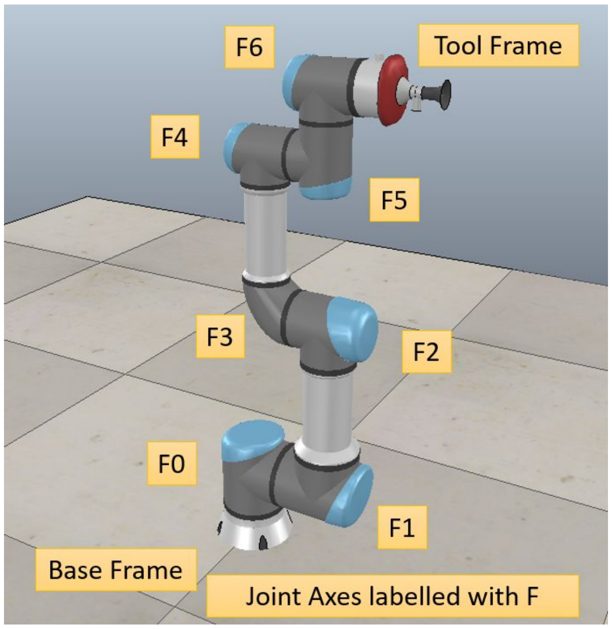
The first goal is to enable the suction cup on the robot arm, making it able to pick up some object in the scene. Then, the most important approach we use is inverse kinematics, we calculate the joint angles with the given pose each time, i.e. where to release the cube on the conveyor belt, where to pour the cup to the basin. The inverse kinematics give us the joint angles so that we can move the robot to the given pose.

Summarize of the results:

1. The robot arm picks up the cube on the table and move it over the conveyor belt, then the robot arm releases the cube on the conveyor belt.
2. The robot arm picks up a cup filled with small balls on the table, then it moves the cup above a basin. Next, by rotating the last joint of the robot arm, we poured the balls inside the cup into the basin. Finally, we put the cup back to its initial pose and orientation.

II. Forward Kinematics

A. *Schematic of the robot in the zero configuration*



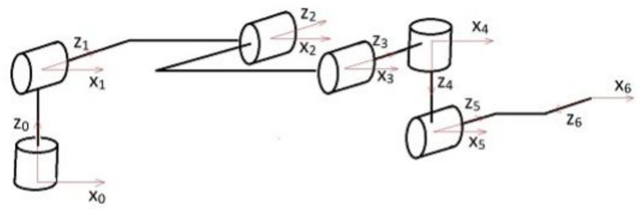
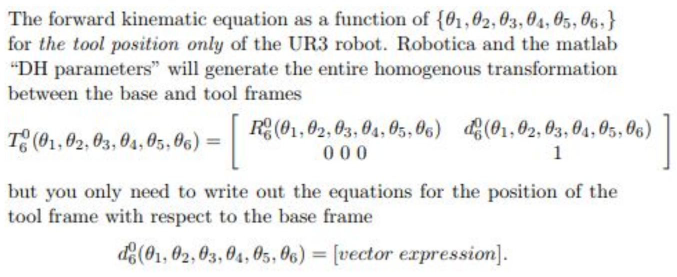


Figure 3.1: UR3 robot with DH frames assigned. Suction Cup Gripper Attached.

B. *How to compute the pose of the tool frame*

To calculate the forward kinematics of an open chain using the space form of the PoE formula, we need the following elements:
(a) the end-effector configuration M ∈ SE(3) when the robot is at its home position.
(b) the screw axes S1, . . . , Sn expressed in the fixed base frame, corresponding to the joint motions when the robot is at its home position. For this, we used the dimensions that we measured in one lab session.
(c) the joint variables θ1, . . . , θn.
$T(\theta) = e^{([B1]\theta1)}\, e^{([B2]\theta2)} \cdots e^{([B6]\theta6)}M$

C. *The derivation of the data that implement the computation*



The forward kinematic equation as a function of $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6,\}$ for *the tool position only* of the UR3 robot. Robotica and the matlab "DH parameters" will generate the entire homogenous transformation between the base and tool frames

$$T_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = \begin{bmatrix} R_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) & d_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) \\ 0\,0\,0 & 1 \end{bmatrix}$$

but you only need to write out the equations for the position of the tool frame with respect to the base frame

$$d_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = [\mathit{vector\ expression}].$$

III. Inverse Kinematics

A. *Computation of joint variables (Algorithms)*

It is an iterative code that starts with an initial set of thetas and eventually converges in the the set of joint angles that result in the desired pose.

Once the first pose is calculated, we calculate then the jacobian matrix. With that and with the required twist between the actual pose and the final pose, it is possible to compute the angular velocities.

The set of thetas is updated adding these velocities multiplied by a small constant.

The algorithm stops when the norm of the body twist is below 0.01 (no more velocities are needed).

B. *Discussion*

- Tool poses for which no solution exists

Some tool poses can't be achieved because the calculated pose is out of the joint range of each joint. For example, we used UR-3 robot arm to simulate our project; during the simulation, we found that the robot arm can't achieve for the set of points which $z<0$. And, the set of points can't be achieved if the distance between the points and the initial center is greater than the sum of all arms' lengths.

- Tool poses for which more than one solution exists

In most cases, if the tool pose can be achieved, there are always more than one solution exist. For example, for most of the poses inside the maximum range of the robot arm, since all the six joints of the robot arm are revolute, a given configuration can be achieved by at least two ways.

- Tools poses for which the only solutions are singular configurations

There are a few poses that the inverse kinematics can only give solutions that are singular configurations. For example, the highest pose that the robot arm can achieve always has only one singular configurations.

But, in our algorithm, our code always return one solution for each tool pose even though there exists more than one solution.

IV. Motion Planning

A. *How to decide if a given set of joint variables places a robot in collision*

There are several spheres (dummies) located across the robot. When the robot moves according to a given set of joint angles, the position of each sphere is recalculated.

In order to know if two spheres are in collision, it is checked if the norm of the distance of the vectors is smaller or larger than the sum of the radius.

B. *How to compute a collision-free path*

First, the Inverse Kinematics function is used to find the set of thetas of the final pose.

After that, from a initial set of thetas (the first node of the tree), a new configuration is chosen and then it is checked if there is collision between both sets checking it for 1000 set of thetas in between.

Only in the case that there is no collision, this new configuration becomes the next node of the tree. This process is repeated until the last set of thetas matches the final set.

Finally, the UR3 follows all the set thetas until it reaches the final position.

C. *Discussion*

- Collision detector returns a false positive or false negative.

It sometimes happens.

The reason of a false positive is that the spheres that cover the joints are too big. A better way to make the collision detection would be using more and smaller spheres to cover the robot once collision is detection and check it again.

A false negative is produced when the collision happens in a zone not covered by any sphere. It can be solved covering the robot with more spheres.

- Choices that no collision-free path exists between given start and goal pose.

If the final pose is inside one of the obstacles the algorithm is not going to find a solution. Currently it would just run forever but a counter of iterations could be set and stop the algorithm if it reaches a number.

Also, as the obstacles are also dummies, it could be easy to check if the final pose is inside the volume of the obstacle.

- How much time is required to plan a collision-free path

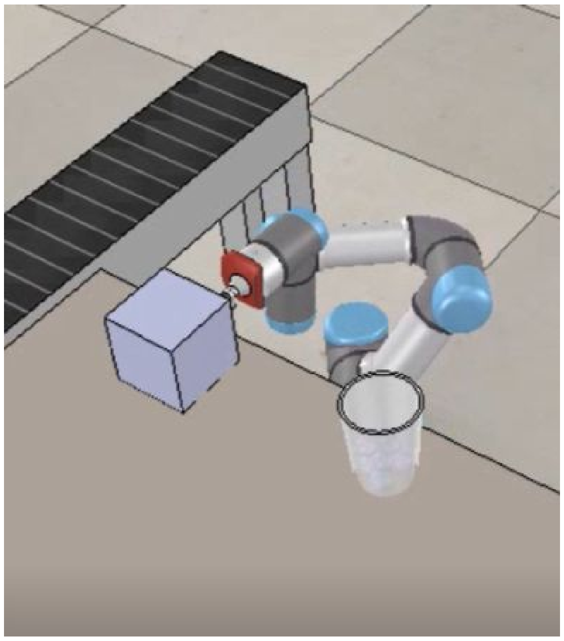
Usually, a collision-free path can be calculated in few seconds. But sometimes we can't get results with bad initial guess.

V. Results

A. *Results of any experiments during simulation*

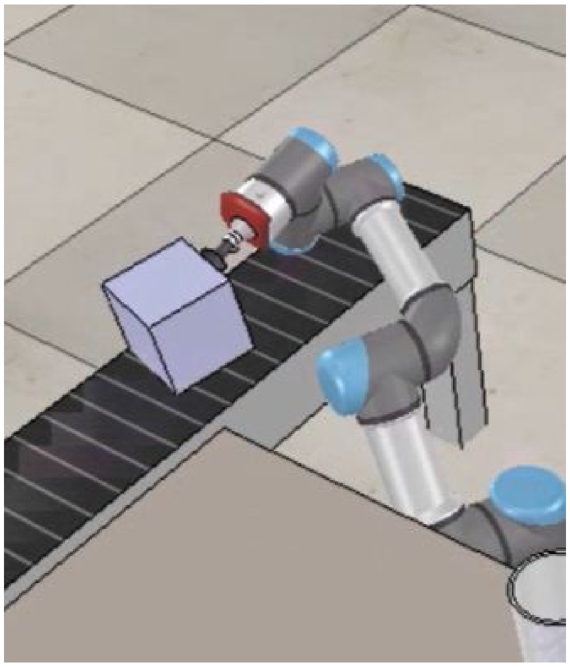
1. Picking up the cube

We set the point that we need to reach to be (0, 0.22, 0.4) and the corresponding transformation matrix T = [[0,1,0,0] [0,0,1,0.22] [1,0,0,0.4] [0,0,0,1]]. Then, we called the inverse_kinematics function we defined to calculate the six joint angles for the six joints. Finally, we called vrep to move our robot arm to the given pose and set the suction cup on to pick up the cuboid in the scene.



2. Release the cube on the conveyor belt

Similarly, we set the point we need to reach to be (0.4, 0, 0.35) and called the inverse_kinematics to find joint angles again. Besides, this time we also called collision_check function to avoid collisions when finding the path to the conveyor belt. Finally, we dropped the cuboid by setting the signal to 0 and then we moved the robot arm back to its home position.

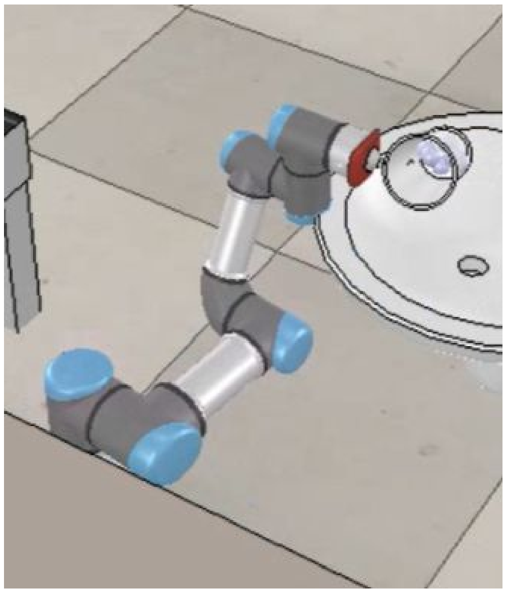


3. Picking up the cup

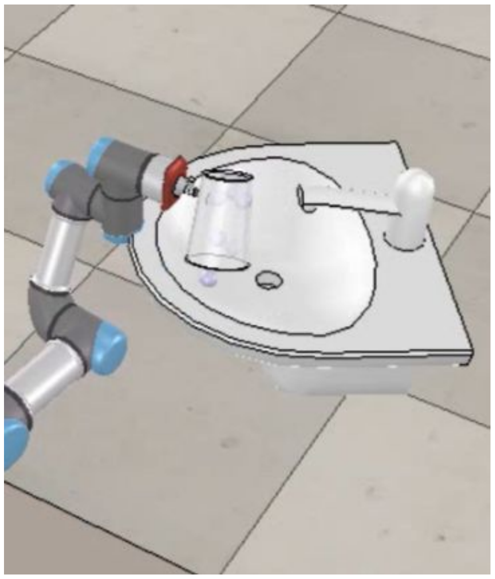
Again, we set the point we need to reach to be (-0.3, 0.24, 0.4) and called the inverse_kinematics to find the joint angles to reach the cup. Next, we turned the suction cup on and picked up the cup on the table.

4. Rotate the cup

There are several ways to pour the small balls in the cup to the basin. Our approach is about changing the sixth joint angle of the robot arm, and this is the easiest way to pour out the cup. By adding pi/2 to the sixth joint angle, the cup became parallel with the floor and several small balls came out. Then, we added another pi/2 to the sixth joint angle and all the small balls would be dropped into the basin.



Note: In the above picture, after picking the cup from table, just one instance of rotation of the cup is shown.



Note: In the above picture, the robotic arm emptied the contents inside the cup by rotating the cup upside-down.

5. Place the cup back on the table

Now, we set the point we need to reach to be (-0.3, 0.24, 0.4), where exactly was the place we picked up the cup. Then, we called the inverse_kinematics function to find the joint angles for each joint and then called the robot arm to reach each joint angle. Next, we turned the suction cup off and dropped the cup on the table. By the way, we also called collision_check function to find a path to reach the destination without any collisions. Finally, we moved our robot back to its home position and turned off the vrep simulator.

B. How successful we were in our simulation

1. Basically, we completed the minimum requirement that asked us to move an object from one place to another given place. We did something more than it by checking the collisions along the path and find the collision free path when using inverse_kinematics.

2. Another interesting thing is about rotating the sixth joint to pour out the cup. There are some tasks seems complicated but we are able to solve them in an easy way, like changing the value of one joint angle.

VI. Future Work

A. The improvements we can make

1. We can improve the suction process by adding one more point before picking up the object; the added point should be a little bit higher than the actual suction point. Hence, our robot arm first goes to the added point to get prepared for suction, then, we turn on the suction cup slowly move the robot arm to reach the object. Finally, we picked up the object.

2. In relation to releasing the cube in the conveyor belt, there are two problems: we don’t exactly align the two objects and the point where we drop the cube is so high. Both issues result in a abrupt and misaligned delivery of the cube. This could be solved by getting the Euler angles of the conveyor belt and set a final pose that match those angles. In order to release the object from a smaller height, we have to set a lower point. This increases the possibility of collision between the conveyor belt and the cube or the arm. Then, it is necessary to improve the accuracy of the method that we use for collision checking.

3. We can better improve our simulation by adding some sensors on our suction cup to sort objects. For example, we can use proximity sensor to distinguish different sizes of objects; we can use vision sensor to distinguish different color of objects in same shape. Besides, if we set a proximity sensor on the suction cup, we can delete the step of setting the suction point because the proximity sensor can give us the distance between the suction cup and the object.

4. Another important improvement is about using motion_planning function to help the robot find the collision free path to reach a given point. Motion planning function is able to give us a more accurate

way of finding the joint angles for a collision free path.

5. Improve our collision checking method by using a small number of big dummies that cover the robot arm. In case of collision, use a larger number of smaller smaller dummies and check again. This improves the calculation time only checking a larger number of collision when necessary. The mayor challenges of this approach is to set the centres of the dummies in the robot in an intelligent way, which implies a high knowledge of the geometry and dynamics of the robot. Also, estimating the right size and number of dummies is a process that has to be set iteratively to find the best combinations.

Acknowledgment

The team worked with ECE 470: *Introduction to Robotics* faculty. Especially, Professor Tim Bretl and Teaching Assistants: Hang Cui, Xinke Deng, Siwei Tang and Jifei Xu, helped the team in dealing with the technicalities of python and V-REP.

In addition to that, Piazza posts helped the team in getting a better sense of the different parts of the project. V-REP Forum posts provides us a lot of useful information about how to use the Python API for V-REP.

References

- Modern Robotics: Mechanics, Planning, and Control by Kevin M. Lynch and Frank C. Park.
  Link: http://hades.mech.northwestern.edu/images/7/7f/MR.pdf

- ECE 470 Lecture Slides and Supplemental Notes by Professor Tim Bretl.

- ECE 470 Lab Manual.
  Link: http://coecsl.ece.illinois.edu/ece470/lab.pdf

- Remote API functions (Python) for V-REP.
  Link: http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm

- V-REP Forum.
  Link: http://www.forum.coppeliarobotics.com