Analyzing the Multi-Faceted Process of 3-link Robot Arm Simulation

Arya Paliwal

Summary

The simulation of robot arms are an integral component of robotics engineering as they allow us to test and alter conditions of the system and troubleshoot errors. This paper outlines the process of constructing a 3-link robot simulation, from the basic concepts to the main algorithm. The different shapes of graphs produced by varying different parameters (i.e. number of obstacles, location, coefficients)are also analyzed to determine the main contenders influencing an algorithm's success. The efficiency and drawbacks of given parameters and algorithms are also analyzed to create a paper that provides a comprehensive view of the simulation process and potential keys for exploration. Individuals seeking an improved understanding of robot arms and their algorithms and potential applications shall benefit from this report.

1. Coordinate Transformations

Coordinate Transformations are a foundational concept in robotics and engineering. Given x, y coordinates, a rotation of the regular plane by angle θ will result in new coordinates expressed below. This concept shall be useful in the remainder of this report.

$$x' = x\cos\theta + y\sin\theta$$
$$y' = -x\sin\theta + y\cos\theta$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

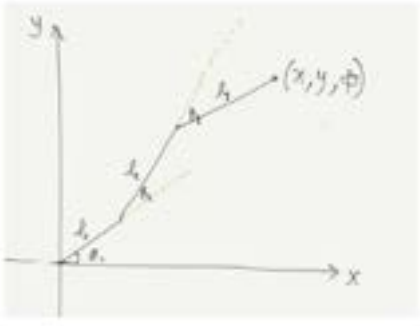
Note: Blue lines indicate the tilted x,y plane.

2. Kinematics

2.1 Forward and Inverse Kinematics

The forward kinematics provides the position and directional vector for the end effector of the robot arm. Below is the diagram for the end effector, along with the x and y coordinates. Inverse kinematics determines the configurations required to reach a goal. In some cases, inverse kinematics results in two solutions. The next part of the algorithm checks if the angles as determined by inverse kinematics are within the range - π and π inclusive;

2.2 Jacobian Matrix Derivation for a 3-link robot arm



$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) + l_3 sin(\theta_1 + \theta_2 + \theta_3) \\ l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) + l_3 cos(\theta_1 + \theta_2 + \theta_3) \end{bmatrix}$$

A robot's observed motion is the result of infinitesimal steps of the end effector and joints in a direction. Taking the time derivative results in the linear and angular velocity of the

joints and the end effector. The Jacobian determines the linear and angular motions of the end effector via solely the motions of its joints.

Given the end effector coordinates:

$$x_{\text{tip}} = l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) + l_3 sin(\theta_1 + \theta_2 + \theta_3)$$
$$y_{\text{tip}} = l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) + l_3 cos(\theta_1 + \theta_2 + \theta_3)$$

Taking derivative of end effector coordinates with respect to each joint angle finds the linear velocity of each coordinate:

$$\frac{\partial x}{\partial \theta_1} = -l_1 cos(\theta_1) - l_2 cos(\theta_1 + \theta_2) - l_3 cos(\theta_1 + \theta_2 + \theta_3)$$
$$\frac{\partial x}{\partial \theta_2} = -l_2 cos(\theta_1 + \theta_2) - l_3 cos(\theta_1 + \theta_2 + \theta_3)$$
$$\frac{\partial x}{\partial \theta_3} = -l_3 cos(\theta_1 + \theta_2 + \theta_3)$$
$$\frac{\partial y}{\partial \theta_1} = l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) + l_3 sin(\theta_1 + \theta_2 + \theta_3)$$
$$\frac{\partial y}{\partial \theta_2} = l_2 sin(\theta_1 + \theta_2) - l_3 sin(\theta_1 + \theta_2 + \theta_3)$$
$$\frac{\partial y}{\partial \theta_3} = l_3 sin(\theta_1 + \theta_2 + \theta_3)$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

2.2 Inverse Jacobian Algorithm

The inverse Jacobian determines the joint velocities required to generate a particular velocity of the end effector. Note that the inverse Jacobian is the reciprocal of the Jacobian Matrix.

3. RRT Algorithm

3.1 Algorithms outline

The algorithm starts when the ‘tree’ is initialized by adding a start node as its root. Nodes are randomly generated within the boundaries, in this case -1.5 to 1.5. They are tested to determine if the node has reached a region within the radius of the threshold rather than testing if the node exactly matches the goal.

Algorithm 1 Inverse Jacobian Algorithm

```
Initialize q with random angles
current = forwardKinematics(q)
dx = goal - current
while ‖ dx ‖≥ threshold do
    q = q + αJ^{-1}dx
    current = forwardKinematics(q)
    dx = goal - current
end while
Return q
```

Algorithm 2 RRT Algorithm

```
SET i = 0
while i < numberIterations do
    if RANDOM() < β then
        SET qTarget to qGoal
    else
        qTarget ← q.RANDOM(LowLim, HighLim)
    end if
    XYDistance ← MIN(LIST(qNode − qTarget))
    qNew = qNear + α(qTarget − qNear)
    XYDistance = forwardKinematics(qGoal) - forwardKinematics(qNew)
    if XYDistance < threshold then
        PathFound = true
        if PathFound then
            while parent ! = qStart do
                add parent to PATH list
                PATH.reverse()
            end while

            return PATH, i
```

The nearest node is extended in the direction of the target. After the generation of a new node, the program checks if it lies within an obstacle, and returns false if that is the case.

3.2 Choosing a random configuration vs goal configuration

Generating new nodes is influenced by coefficients α and β

- $0 < \alpha <= 1$
- Determine the step size of each iteration;
- $\theta_{new} = \theta_{old} + \alpha(\theta_{new} - \theta_{old})$
- $0 < \beta <= 1$
- determines the probability of the existence of a random configuration vs a goal configuration.

The length of each iteration is dependent on a value between 0 and 1. Small alpha values increase precision at the cost of time(iterations). Smaller steps are taken, increasing the likelihood of a solution configuration for the end effector to reach the goal. Larger alpha values may reduce the time, but in the case of large obstacles, will either be unable to find a solution, or take a path that is inefficient.

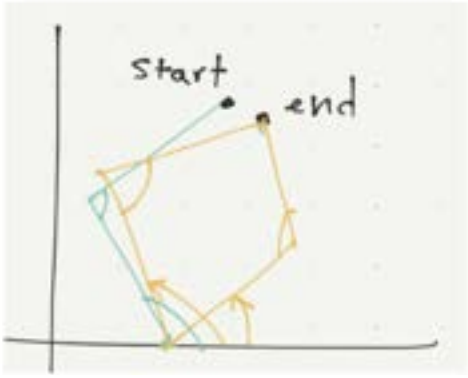
The probability of a sample being in the direction of the goal or a random sample depends on beta. Similar to alpha, a high value of beta might increase efficiency, so long as there are not many obstacles. A low value of beta might cost time, but result in a valid configuration.

It can be observed that the success of the algorithm given the beta value depends on 3 factors:

- Spatial Arms of blocks
- Spatial Area of robot arms
- Alpha value

3.3 Choose the distance comparison as the angles instead of x-y coordinates

One has two choices, use x,y coordinate distances to find the length taken by each joint to reach the desired configuration, or use angles to determine the angle changes required to reach a particular configuration. Angles are more convenient to determine the ‘energy’ or time required to form a configuration. For example, the end effector wishes to move a small distance x in a direction, say x-direction; however, the joint angles change drastically (illustrated in figure below), providing a clearer understanding of the energy required to move from one formation to another.



Original configuration (blue) and two new possible configurations(yellow)

3.4 Creating the tree data structure

Dictionaries are used to organize the parent and child points and angles. For instance the key (10, 20, 30) would reference (10, 20, 30): (20, 40, 60) in the dictionary. Note that the joint angles with their corresponding x,y coordinate of the end effector is stored in a list. As the algorithm chooses a point(regardless of its bias towards the goal or random sample), it first creates another key corresponding to the selected point; then, it stores the coordinate and the corresponding joint angles as the 'value' in the key of the parent point. This continues until the final sample is within the threshold of the goal.

3.5 Finding the final path from tree data structure

Find the path from the end goal to the start by tracing back the coordinates. For instance, if the end goal was met at the joint configuration(10, 20, 30), then one would search for the key containing (10, 20, 30)(i.e. parent node), this parent node is then a 'value' for another key (or parent node) in the dictionary. Each time a parent node is recovered, it is added to a path list. The algorithm while the parent node is not equal to the start node. The path list generated contains the parent node from the end to the start.The list is reversed to generate the path from the start node to the end node.

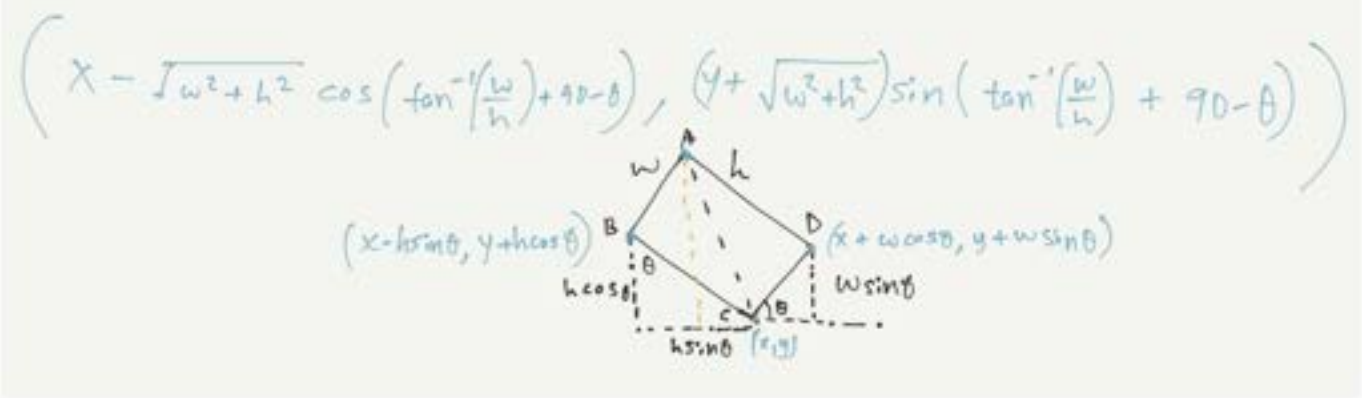
3.6 Collision Check

We used rectangles of different parameters to check for collision.

3.6.1. Geometry

Instead of storing each (x, y) coordinate for each vertex of the obstacle, we expressed other coordinates in terms of one corner. For instance, if there is a rectangle with corner (x,y), its adjacent vertex will be $x + wcos(\theta), y + wsin(\theta)$; the derivations for the rest are shown in the image. Note: to find the corner opposite to (x, y), construct the triangle

in yellow, $tan(\angle ACB) = w/h$, so $\angle ACB = arctan(w/h)$. $\sqrt{w^2 + h^2}$ sin(∠ACB + ∠C) and $\sqrt{w^2 + h^2}$ cos(∠ACB + ∠C) are the lengths of the yellow triangle.

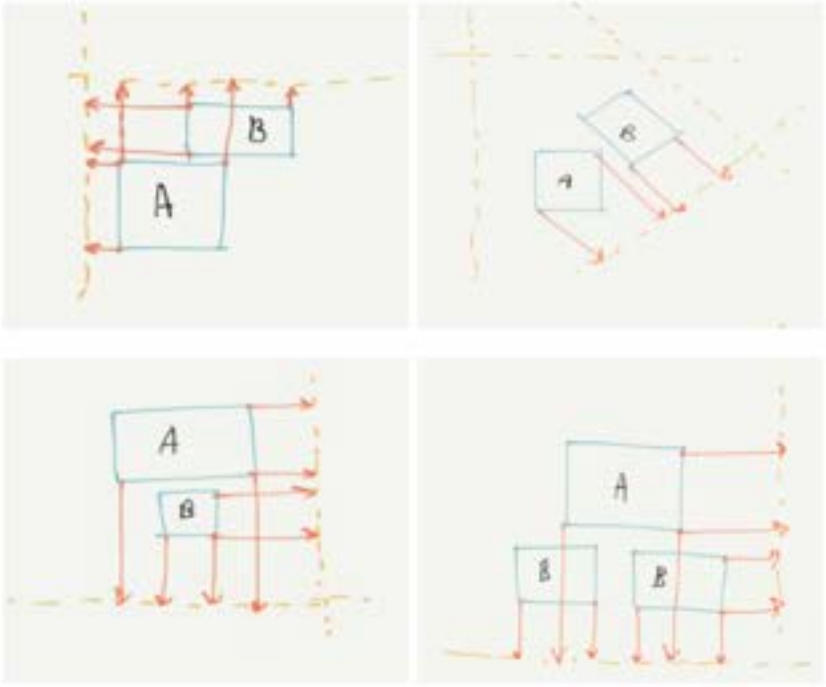


3.6.2. Cases

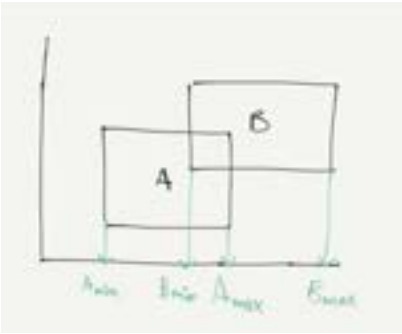
The algorithm used for this simulation uses convex shapes, namely quadrilaterals; however, this concept can be extended to n sided objects. To create the collision checking program, each coordinate is assigned a vector. Since two sides of a rectangle are parallel, only at most 4 lines need to be investigated(2 perpendicular lines on each rectangle). Below are the 4 cases to be considered (note that all of them are superimposed on a Cartesian coordinate system).

Note that the code uses lists to store robot arm links and objects. This process systematically loops through all elements of the robot arm to check if there is any collision. Loop through the link-list, and check if the link at the given index is within another obstacle. This is accomplished by creating a ‘baseline’ that tilts the x plane to be parallel to the side of the obstacle we are looking at. All the points of the object and link are projected onto the ‘baseline’ via a dot product between each vertex and the ‘Baseline’ and the cases above are checked for.

Check for a gap between the two obstacles. There is at least one viewing angle where the obstacles do not collide.



Non-collision scenarios



Collision of two 4-sided objects

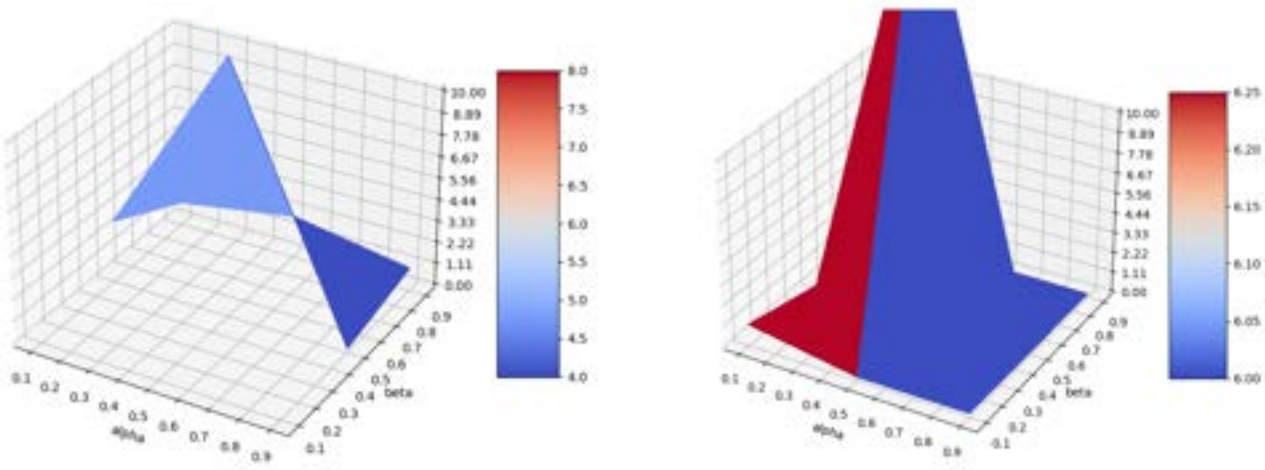
Overlap Scenarios:

$$A_{min} < B_{min}, A_{max} > B_{min}$$
$$A_{min} < B_{max}, A_{max} > B_{min}$$
$$A_{min} > B_{min}, A_{max} < B_{max}$$
$$A_{min} < B_{min}, A_{max} > B_{max}$$

Note: Once the overlap conditions are satisfied, the program terminates, increasing efficiency.

4. Experiments

Investigate the relationship between alpha and beta values on efficiency.



X axis indicates alpha values, Y axis indicates beta values and values on the spectrum indicate the overall cost of moving to the required position. Graph on the left is the robot-arm's efficiency without any obstacles, and on the right, with a 0.2 by 0.2 square obstacle in the plane. It can be noted that graph 1, regardless of alpha and beta values, maintains a lower cost than in graph 2. In graph 2, at lower values of alpha and beta, efficiency is the lowest.

5. Conclusion

This report outlined the processes behind the simulation of a 3-link planar robot arm, from the mathematics to the algorithm. The benefits and drawbacks of used constants and the influence of obstacle positioning on the path taken were evaluated. The advantages of using angular measurements rather than linear measurements for robot link positioning and obstacle positioning were also investigated. Common to engineering, there was a trade off between efficiency and cost with the alpha and beta values. It was then essential to find a specific combination resulting in minimized cost and maximized efficiency. A further investigation into this would entail the creation of a 3-D model to find the optimal location for alpha and beta values. The results accrued from this investigation and higher n-link robot arms could contribute to a novel algorithm to find optimal solutions for path planning.

6. References

[1]Analyzing a 3-joint planar robot arm. (n.d.). Robot Academy. https://robotacademy.net.au/lesson/analyzing-a-3-joint-planar-robot-arm/

[2]Assignment 3 - Path Planning. (n.d.). https://cs.brown.edu/courses/cs148/documents/asgn3_planning/btcohen/index.html

[3]Choset, H., & Kuffner, J. (n.d.). Robotic Motion Planning: RRT's [Slide show]. Carnegie Melon University. https://www.cs.cmu.edu/~motionplanning/lecture/lec20.pdf

[4]Inverse kinematics using the Jacobian inverse, part 2. (2017, December 11). https://nrsyed.com/2017/12/10/inverse-kinematics-using-the-jacobian-inverse-part-2/

[5]Jacobians. (n.d.). http://ltcconline.net/greenl/courses/202/multipleintegration/jacobians.html

[6]Robot control part 2: Jacobians, velocity, and force. (2017, March 18). Studywolf. https://studywolf.wordpress.com/2013/09/02/robot-control-jacobians-velocity-and-force/

[7]Robotics Explained | Robot Course. (n.d.). https://robotics-explained.com/jacobian/

[8]Svegliato, J. (2021, December 13). How does a robot plan a path using RRT? - Towards Data Science. Medium. https://towardsdatascience.com/how-does-a-robot-plan-a-path-in-its-environment-b8e9519c738b

[9]Velocity of 3-Joint Planar Robot Arm. (n.d.). Robot Academy. https://robotacademy.net.au/lesson/velocity-of-3-joint-planar-robot-arm/