Robot Manipulation through Inverse Kinematics

Here the comparison of two widely used methods, namely, **inverse kinematics** and **Jacobian inverse methods**, for robot manipulation is done.

Find Added notes here on:

1) Matlab Code of Inverse Kinematics of six DOF Freedom Robot (here: KUKA KR5) as used in the paper.
2) Matlab Code for the Jacobian and Manipulability Index of the manipulator.
3) RoboAnalyzer Inverse Kinematics Module

"Robot Manipulation through Inverse Kinematics"

Website Link:

For Matlab Code texts refer page 8:

Roboanalyzer based forward and inverse kinematics of KUKA KR5
[Note: The Roboanalyzer software can be downloaded from the link given below:
http://www.roboanalyzer.com/downloads.html. RoboAnalyzer V7 (July, 2015), is used here.

Robot Manipulation through Inverse Kinematics

Abdullah Aamir Hayat
Department of Mechanical Engineering
Indian Institute of Technology Delhi
New Delhi, India
aamir_hayat@rediffmail.com

Ratan Sadanand O. M
Department of Mechanical Engineering
Indian Institute of Technology Delhi
New Delhi, India
ratan.sadan@gmail.com

Subir. K. Saha
Department of Mechanical Engineering
Indian Institute of Technology Delhi
New Delhi, India
saha@mech.iitd.ac.in

ABSTRACT

Inverse kinematics of a robot is very essential to find the joint variables that satisfy the desired pose of the robot during its manipulation. This is used in controlling the robot position, animation of the robot, etc. In this paper, step-by-step explanation and comparison of two widely used methods, namely, inverse kinematics and Jacobian inverse methods, for robot manipulation are presented. For this purpose a six degrees-of-freedom wrist-partitioned industrial robot KUKA KR5 Arc was used to illustrate the methods. A novel approach has been proposed for selecting the appropriate set of joint angles among the several inverse kinematic solutions. It is based on weight of each link and manipulability. The comparison of these approaches for linear and circular trajectory is presented. Their advantages, limitations, applications, and computations involved are also highlighted.

Categories and Subject Descriptors

I.2.9 Robotics

General Terms

Theory, Verification.

Keywords

Inverse kinematics, Jacobian, Manipulability, Robot manipulation

1. INTRODUCTION

The goal of a robot controller is to generate an acceptable motion of the end-effector by precisely actuating its joints for a specified task. To study the geometry of motion without considering its cause comes under the subject of kinematics. Robot kinematics is divided into forward and inverse kinematics which are depicted in Figure 1. Forward kinematics (FK) utilizes kinematic equations to find the pose, i.e., position and orientation of the end-effector (EE), given the joint angles, while the inverse kinematics (IK) computes the joint angles for a desired pose of the EE. Joint angles are then input to the actuators attached to move the links of the robot according to the specified trajectory. So there is a requirement of accurate joint angle values for precise robot manipulation and/or kinematic animation.

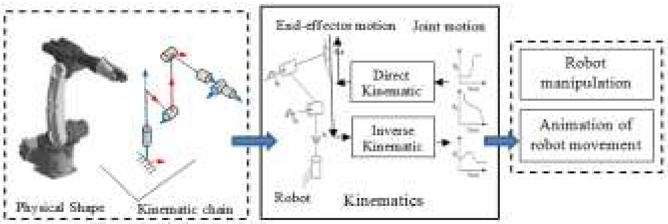


Figure 1. Robot kinematics and its utility

Table 1. Number of inverse kinematic solutions

DOF	Number of solutions
6 (6R, 5RP)	8
6R (intersecting wrist)	16
Greater than 6 (Redundant manipulator)	∞

For serial robots, FK is straightforward, which has unique solution, while the inverse kinematics has multiple solutions satisfying a particular pose. One approach to the inverse kinematics problem is to find a closed-form solution using algebraic or geometric method. Another approach is to find a numerical solution by a successive approximation algorithm. Although the former approach is generally more desirable in applying the solution to real-time control of robots, it is not always possible to obtain the closed-form solutions for the manipulators with arbitrary architectures [1]. There exist several methods of modeling and solving IK of a robot. The kinematic modeling of serial-chain robots are commonly done using the Denavit-Hartenberg (DH) parameters [2]. The analytical method for solving inverse kinematics for six degrees of freedom (DOF) robot manipulator with three consecutive axes either intersecting or parallel was reported in [3]. Such architecture have eight inverse kinematic solutions. A general six DOF robots, have sixteen solutions [4]. The number of analytical solutions for different degrees of freedom robots are listed in Table 1. Hence, for continuous motion of a robot, an accurate set of solutions among several inverse kinematic solutions is required. Another method for solving IK problem is by the use of Jacobian inverse [5, 6]. The numerical method for solving the inverse of Jacobian matrix was presented in [7, 8]. The analytical solution to the inverse kinematics problem for six-DOF robot manipulator is presented in [9]. The advantage of analytical expression is that it gives formulas for relationship between joint angle and link parameters. These relationship can be directly embedded in the robotics controller.

AIR '15, July 02 - 04, 2015, Goa, India
© 2015 ACM. ISBN 978-1-4503-3356-6/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2783449.2783497

Serial robots with wrist-partitioned feature, i.e., the last three link axes intersecting, are widely used in industries [2]. An algebraic method for solving the IK of a six-DOF robot with wrist-partitioned structure with all revolute joints, abbreviated here as *6R-WP-robot,* presented in [1] is used in this paper. Further, step-by-step procedure for implementing the IK for continuous robot manipulation is explained.

The strategy for resolving the conflict in selection of a set of joint angles from the eight inverse kinematics solutions of an industrial robot KUKA KR5 Arc is proposed in this paper. These strategies can also be followed for robots with multiple inverse kinematic solutions for manipulation and control. The analytical and inverse Jacobian methods are compared by tracing the given linear and circular trajectories. Such comparisons are not easily traceable in open literature, even though the industrial robot manufacturers may use them. Hence, this paper will help those researchers who would like to take forward the proposed concepts and implement them.

This paper is divided into three sections. In Section 2, the general inverse kinematic solutions and Jacobian inverse methods are presented for 6R-WP robot. In Section 3, the trajectories traced by the robot for a given task using the two algorithms are shown. Finally the conclusions are given in Section 5.

2. METHODOLOGY

This section explains in brief the formulation used behind the inverse kinematics of a 6R-WP robot. Two methods, namely, analytical method and the Jacobian inverse method are discussed.

2.1 Inverse Kinematics

Geometric model of the serial robot with wrist-partitioned feature i.e., the DH parameters is given in Table 2. Kinematic diagram of the corresponding robot is shown in the Figure 2(a). The DH parameters are taken from its identified value presented in [10]. The net transformation matrix to obtain the position and orientation of the EE can be given as,

$$\mathbf{T} = \prod_{i=1}^{6} \mathbf{T}_i = \underbrace{(\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3)}_{\text{Arm}} \overbrace{\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6}^{\text{Wrist}} \tag{1}$$

where the 4×4 matrix $\mathbf{T}_i$ is given by

$$\mathbf{T}_i \equiv \begin{bmatrix} \mathbf{Q}_i & \mathbf{p}_i \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & b_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The position pEE and orientation QEE of the EE for six-DOF robot can be explicitly expressed as:

$$FK(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = (X, Y, Z, \phi, \theta, \psi) \tag{3}$$

$$\mathbf{P}_{EE} = \sum_{i=1}^{6} \boldsymbol{a}_i \text{ and } \mathbf{Q}_{EE} = \prod_{i=1}^{6} \mathbf{Q}_i \tag{4}$$

where **p** is the position vector, **Q** is the 3 × 3 rotation or orientation matrix, $\mathbf{0} \equiv [0, 0, 0]^T$ is the three-dimensional vector of zeros, whereas the 4 × 4 matrix, **T**, is called the homogeneous transformation matrix (HTM) of the end-effector. Based on Equation (1) – (4), once can define the IK solutions as :

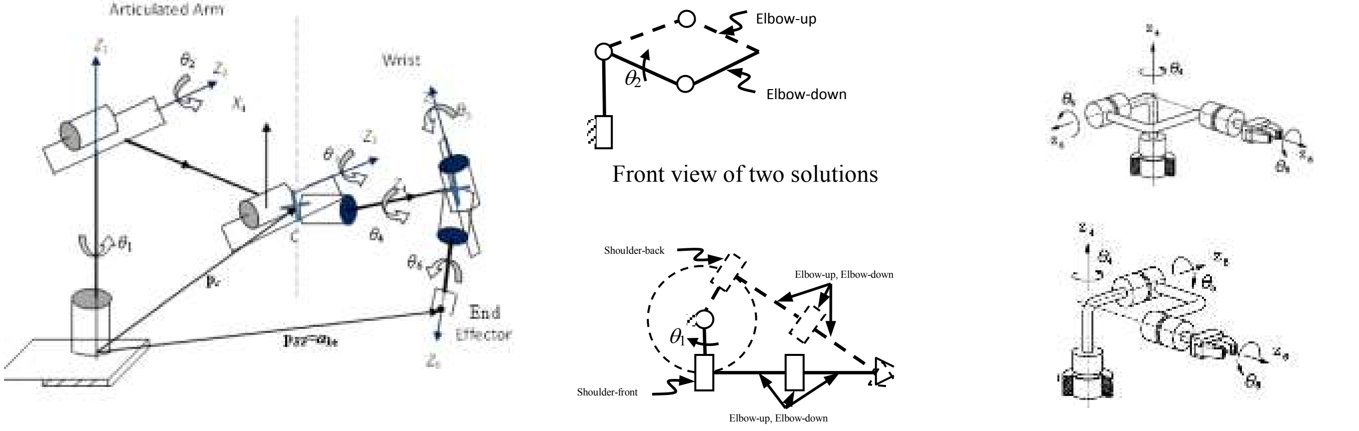
$$IK(X, Y, Z, \phi, \theta, \psi) = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) \tag{5}$$

The detailed derivation to find the joint angles has been discussed in [11],[1]. Note that the point *C* in Figure 2(a) shows the intersection of axes 4, 5, and 6.Hence it is independent of those joints and function only the first three joints. First, joints angles 1, 2, and 3 are solved before obtaining the joint angles 4, 5, and 6. The motion of the robot end-effector can be obtained by dividing the path into a number of intermediate points and moving the end effector from point-to-point.

Table 2. DH parameters of a 6R-WP robot KUKA KR5 Arc [9].

No.	b_i (mm)	θ_i (°)	Minimum joint limit (°)	Maximum joint limit (°)	a_i (mm)	α_i (°)
1	400	θ_1	-155	155	180	π/2
2	0	θ_2	-65	180	600	π/2
3	0	θ_3	-68	105	120	0
4	620	θ_4	-350	350	0	π/2
5	0	θ_5	-130	130	0	π/2
6	0	θ_6	-350	350	0	0

In the inverse kinematics solution, the required joint angles corresponding to the subsequent points are calculated. This yield



Front view of two solutions

a) Kinematic model of 6R robot with last three links intersecting

b) Top view of other two solutions [1]

c) Two solutions for the wrist [10]

Figure 2. Kinematic diagram of a six-DOF wrist-partitioned robot with its inverse kinematic solutions

multiple solutions. Figure 2(b) and Figure 2(c) gives four sets of θ_1, θ_2, and θ_3 by solving a quartic equation and two sets for θ_4, θ_5, and θ_6 respectively . Hence, a 6R-WP robot can have up to eight possible configurations for a given pose of the EE. They are indicated in Figure 3.

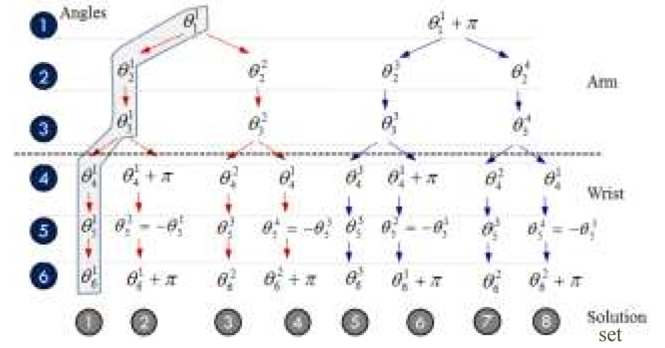


Figure 3. Eight inverse kinematics solutions

Each link joint has some mechanical limit which was mentioned in Table 2. The IK solutions for an EE pose specified by the user, needs to be checked for the joint limits first. If it exceeds the given value of joint limits of the robot, those values get discarded.

The wrapping of angles is widely used in the robots where the angles are obtained from the encoder values. Wrapping of the IK solutions in the range of (-π, π) is necessary for the continuous and smooth motion of the robot. The algorithm for wrapping the angles in the interval is given below.

With several IK solutions, the appropriate set of solution among the eight solutions, as shown in Figure 3, must be selected in order to have a smooth motion of the robot. This is very important to avoid sudden change in robot configuration while moving from one point to another. Four methods are listed for selecting one of the eight possible solutions.

```
Algorithm #1: Wrap angles in interval [-π,π]
Theta [i][j]
  <comment> 'i' is number of IK solutions, i.e., 8
   <comment> 'j' is number of joint angles, i.e., 6 here
for i=1to8
 for j=1to6
Theta [i][j]=  Theta [i][j] % (2π)   <comment> '%' is modulo operator
     if Theta[i][j] < pi
        Theta[i][j]= Theta [i][j] + (2π)
     elseif  Theta[i][j] > pi
         Theta[i][j]= Theta [i][j] - (2π)
return Theta[i][j]
```

2.1.1 *Minimum deviation of all joint angles*

Minimum deviation in joint angles is desired to avoid any sudden change in the configuration of the robot while in motion. This index utilizes the previous joint states to select the next set of solution among the eight IK solutions.

$$\eta \;=\min\left(\sum_{i=1}^{6}\left\|\boldsymbol{\theta}_{i,current}-\boldsymbol{\theta}_{i,next}\right\|^{2}\right) \tag{6}$$

where $\boldsymbol{\theta}_{i,current}$ is the column vector of joint angles corresponding to the current configuration of end-effector, and $\boldsymbol{\theta}_{i,next}$ is the i^{th} joint angle for the next configuration. Minimum norm set of angles are selected among the eight IK solutions of the next step.

2.1.2 *Minimum deviation in first three joints*

The first three joints of a six axis WP robot is generally used for positioning of the end-effector of a robot. For most of the industrial robots, first three joints supply the maximum torque for the motion of the end-effector. Hence it is preferable to minimize the motion of the first three joints while moving from the current to the next point, without compromising on the accuracy. Using this rationale, the next joint configuration is chosen out of the possible multiple inverse kinematics. The solution that has minimum variation is selected as the joint values for the next point in the path. In this way, elbow-up and elbow-down configurations are insured.

$$\eta \;=\min\left(\sum_{i=1}^{3}\left\|\boldsymbol{\theta}_{i,current}-\boldsymbol{\theta}_{i,next}\right\|^{2}\right) \tag{7}$$

2.1.3 *Weighted deviation of joints*

As discussed in Section 2.1.2, the first three joints have to supply a relatively higher joint torques for robot motion. It is natural that while going from one configuration to the next, the motion of those joints must be minimized. In order to account for the same while choosing the next solution, the following index is utilized, by putting some weights to the first three joints, i.e.

$$\eta \;=\min\left(\sum_{i=1}^{3}\left\|\left(\boldsymbol{\theta}_{i,current}-\boldsymbol{\theta}_{i,next}\right)w_i\right\|^{2}\right) \tag{8}$$

where w_i is the weight which was choosen proportionaly to the inverse of the mass of each link.

2.1.4 *Manipulability*

The manipulability of a robot [12] can be used as a criterion to select an appropriate solution from the multiple inverse kinematics solutions. Applying the maximum manipulability criterion in each step of the trajectory may result in flip configuration in between. Hence, it is applied once for selecting the pose for the starting position only.

$$\eta=\max\left(\sqrt{\det(\mathbf{J}(\theta_i)\mathbf{J}^{T}(\theta_i))}\right) \tag{9}$$

2.2 Jacobian inverse method

In robotics, the Jacobian can be interpreted as the time derivative of the kinematic equations which relates the velocity of the end-effector to the joint rates. The expression for the Jacobian is given as follows

$$\boldsymbol{\omega}_e=\mathbf{J}_{\omega}\dot{\theta} \quad\text{and}\quad \mathbf{v}_e=\mathbf{J}_{v}\dot{\theta} \tag{10}$$

where $\mathbf{J}_{\omega}$ and $\mathbf{J}_{v}$ are the 3×6 matrices relating the contribution of the joint velocities or rates $\dot{\theta}$ to the end-effector angular velocity $\boldsymbol{\omega}_e$ and velocity $\mathbf{v}_e$ respectively. Also $\mathbf{J}_\omega$ and $\mathbf{J}_v$ are the function of $\boldsymbol{\theta}$. Equation (10) can be re-written as

$$\Delta\mathbf{x}_e = \mathbf{J}\,\Delta\boldsymbol{\theta}$$

where $[\Delta\phi\,\Delta\theta\,\Delta\psi\,\Delta x\,\Delta y\,\Delta z]^T = \mathbf{J}[\Delta\theta_1\,\Delta\theta_2\,\Delta\theta_3\,\Delta\theta_4\,\Delta\theta_5\,\Delta\theta_6]^T$ (11)

The term $\Delta\mathbf{x}_e$ is the change in the pose of the end-effector corresponding to the change in joint angles $\Delta\boldsymbol{\theta}$. The Jacobian **J** is the 6×6 matrix for a 6-DOF robot, which is obtained by the recursive relation of the angular velocity and linear velocities for 6R manipulator. It is is given as

$$\mathbf{J} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_6 \\ \mathbf{e}_1 \times \mathbf{a}_{1e} & \mathbf{e}_2 \times \mathbf{a}_{2e} & \cdots & \times\mathbf{a}_{6e} \end{bmatrix} \quad (12)$$

in which, $\mathbf{a}_{1,e}$ is the vector as shown in Figure 2 (a) and $\mathbf{e}_1$ is the joint axis direction of joint 1. Similarly $\mathbf{e}_2$ is the joint axis direction and $\mathbf{a}_{2,e}$ is the position of the EE from joint 2, and so on. Each column of the Jacobian consists of the differential rotation and translation vector corresponding to the differential change in the joint rates. In order to obtain the joint variation for the desired increment at the k^{th} point pose is given by the Jacobian inverse as below,

$$\Delta\boldsymbol{\theta}_k = \mathbf{J}_k^{-1}\Delta\mathbf{x}_e \quad (13)$$

where $\Delta\mathbf{x}_e$ is the desired increment in the pose of the EE. The efficient way to decide the increment was given in [13]. The joint angles required to reach the k^{th} point, which are similar to the first order Taylor series expansion of the joint angle evaluated as

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \mathbf{J}_k^{-1}\Delta\mathbf{x}_e \quad (14)$$

In general, $\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \mathbf{J}^{\dagger}\Delta\mathbf{x} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\,\Delta\boldsymbol{\varphi}$ (15)

where $\mathbf{J}^{\dagger} = \mathbf{J}^{\mathrm{T}}(\mathbf{J}\mathbf{J}^{\mathrm{T}})^{-1}$ is pseudo inverse of **J** matrix and $(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})$ is the null space of **J** with $\Delta\boldsymbol{\varphi}$ as objective function. Here for manipulator with DOF=6, (14) is used as the Jacobian matrix is square and it can be inverted.

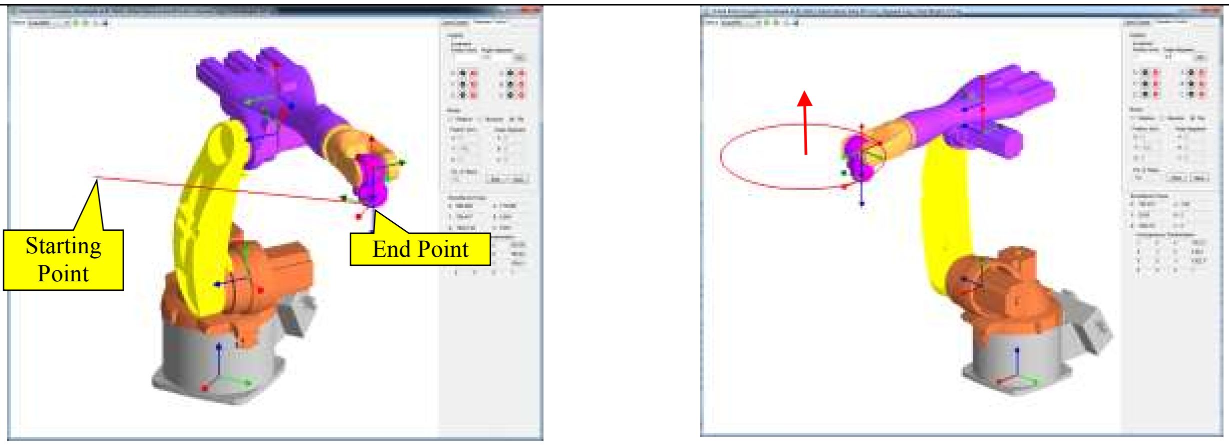
3. RESULTS AND DISCUSSIONS

The angular positions obtained from the IK are updated by the controller of the robot. The control frequency of a typical industrial robot generally varies between 10 to 50 Hz that determines the time instance in which the joint position values are updated and commanded by the controller.

The inverse kinematics and the Jacobian inverse method explained in Sections 2.1 and 2.2 respectively, are implemented in MATLAB environment for a linear and circular trajectory. For animation of the robot motion, the joint angles obtained using IK were provided to Virtual Robot Module (VRM) [14] of RoboAnalyzer software. They are shown Figure 4. The joint angles calculated were given as input to the VRM for the animation and visualization purpose. These algorithms were tested on Intel® Core, i7-3770 CPU and 3.40 GHz with 8 GB RAM, 64 bit Windows 7 OS. The comparisons of the approaches given in Section 2 are now discussed in following sections.

3.1 Straight line motion

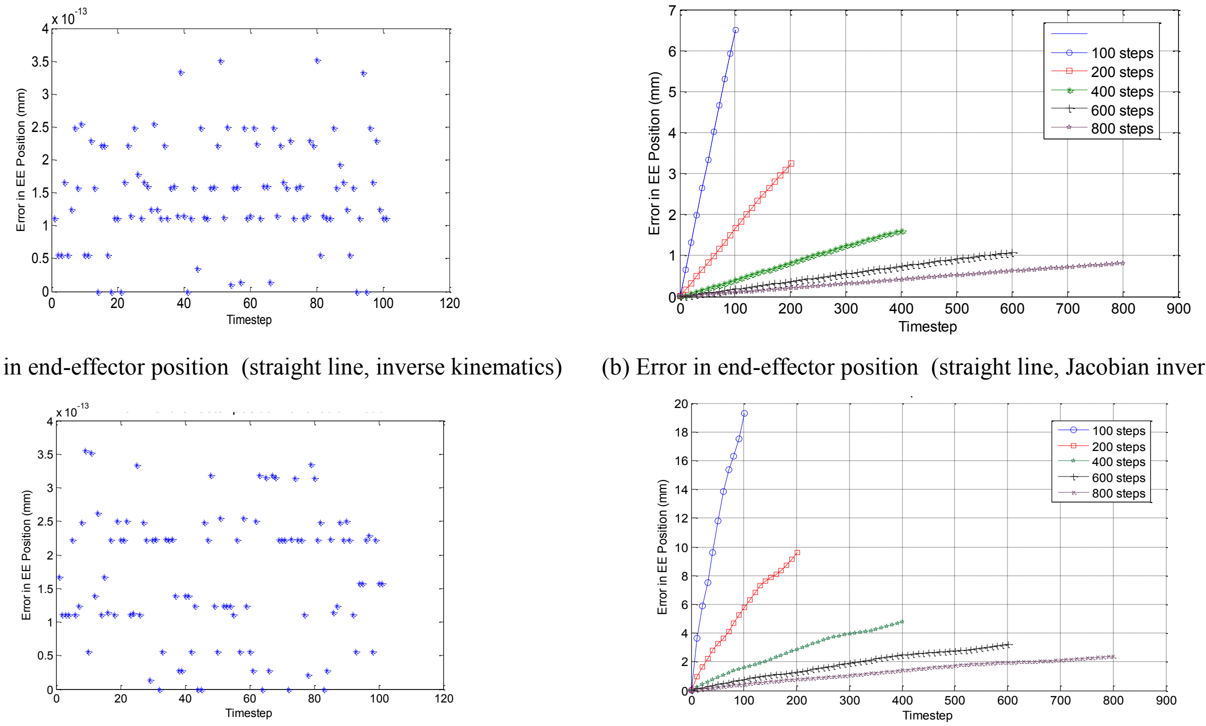
Generally, a robot programmer specifies the initial and final points, as shown in Figure 4(a). The values of $[X, Y, Z, \phi, \theta, \psi]$ at the starting and at the end positions are [0.8, -0.4, 1 0, 0, π] and [0.6, 0.4, 1, 0, 0, π] respectively. The desired motion between the two points is a straight line. The values of *X, Y* and *Z* are in meters and the orientations $\phi, \theta,$ and ψ are in radians.



a) Straight line motion

b) Circular motion

Figure 4. The desired path to be traced by the robot.



(a) Error in end-effector position (straight line, inverse kinematics) (b) Error in end-effector position (straight line, Jacobian inverse)

(c) Error in end-effector position (circular, inverse kinematics) (d) Error in end-effector position (circular, Jacobian inverse)

Figure 5. Comparison of two methods for straight line and circular trajectories.

Figure 4(a) shows the straight line motion traced by the robot. The inverse kinematic method gave positioning errors of the order of 10^{-13} mm between commanded and desired position of the end-effector for straight line motion as shown in Figure 5 (a). Table 4 lists the time taken using different selection criteria proposed in Section 2.1.4 for tracing the straight line.

Table 3. Results for linear trajectory using Jacobian inverse

S.No.	No. of steps	Straight line motion		Circular motion	
		Error (mm)	Time (s)	Error (mm)	Time (s)
1	100	6.49	0.508	19.42	0.534
2	200	3.25	0.941	9.87	0.957
3	400	1.86	1.770	4.73	1.821
4	600	1.01	2.606	3.47	2.663
5	800	0.89	3.465	2.43	3.502

The Jacobian inverse method was also tested. The difference between the desired EE position and the position reached using the Jacobian inverse method for 100 steps is 6.49 mm. Note that it is an approximate method. Hence, if the number of steps in increased it is expected to reduce the errors. This is evident from Table 3 and Figure 5(b). When the number of steps is 800, the error is only 0.89mm. However, the computation time increases. Table 3 also lists the time taken to generate the results, i.e., the joint angles to achieve the desired path (in this case, the straight line).

3.2 Circular motion

Here the robot programmer needs to specify the center, radius and normal of the circle which was required to be traced by the robot. The center *C* of the circle was taken as (0.8, 0, 0.9) m and the radius of the circle specified as 0.3 m.

The axis of the circle was kept parallel to the joint axis 1 which is shown in Figure 4(b). The end effector was at a constant orientation. Using the inverse kinematics method an error of the order 10^{-13} mm was observed. Table 4 lists the time taken.

Here the robot programmer needs to specify the center, radius and normal of the circle which was required to be traced by the robot. The center *C* of the circle was taken as (0.8, 0, 0.9) m and the radius of the circle specified as 0.3 m. The axis of the circle was kept parallel to the joint axis 1 which is shown in Figure 4(b). The end effector was at a constant orientation. Using the inverse kinematics method an error of the order 10^{-13} mm was obtained as shown in Figure 5(c). Table 4 lists the time taken with different step size using different selection criteria of IK solutions.

The Jacobian inverse method resulted in higher positioning errors (19.42 mm) compared to inverse kinematics method, where the positioning errors were almost negligible. Figure 5(d) illustrates that by increasing the number of intermediate points or reducing the step size the error of EE positioning reduces.

It was found that the Jacobian inverse solution has greater error while tracing a circular trajectory than the straight line motion. This is quite obvious as the method is based on straight line approximations. Hence, more the curvature of a path, more error is expected. The inverse kinematic method on the other hand is not affected by the step size as the calculations are exact. The step size for calculating the intermediate configurations here depends on the control frequency and the resolution of the robot.

Table 4. Time taken in inverse kinematics method (100 steps)

Criterion for inverse kinematics	Straight line (in sec)	Circular motion (in sec)
Minimum deviation of all joint angles	4.338	4.468
Minimum deviation in first three joints	4.319	4.341
Weighted deviation of joints	4.338	4.423
Manipulability	4.412	4.452

4. CONCLUSIONS

This work has reported the comparison between two widely used methods for manipulation and animation of a typical industrial robot, namely the inverse kinematic and the Jacobian inverse methods. Linear and circular trajectories were selected for comparing the two approaches. The step by step method to obtain IK solutions and selecting criteria of one solution among eight solutions was given. Among the four criteria for selecting solutions, least computation time is for checking minimum deviation in first three joint angles, but the user can select other criteria depending on the requirement and task. The wrist partitioned six-DOF robot results in attaining far better accuracy in positioning the robot by commanding them calculated IK joint space than the numerical method, i.e., Jacobain inverse technique. Though increasing the step size in numerical methods reduces the error at the cost of increased computation. In animation environment these algorithms were successfully implemented and compared. In future adaptive learning techniques for IK will be interesting to implement and compare.

5. ACKNOWLEDGMENTS

The financial support to the first two authors is from the sponsored project entitled “Adaptive Force Control of an Industrial Robot (KUKA KR5) Equipped with Force/Torque Sensor” by BRNS/BARC Mumbai under the setting up of “Programme for Autonomous Robotics Lab” at IIT Delhi is sincerely acknowledged.

6. REFERENCES

[1] Saha, S. K., 2014. Introduction to Robotics, 2nd ed.: Tata McGraw-Hill Education.

[2] Siciliano, B., and Khatib, O., 2008. Springer Handbook of Robotics: Springer.

[3] Peiper, D. L., 1968. The kinematics of manipulators under computer control, DTIC Document.

[4] Manseur, R., and Doty, K.L., 1989. A robot manipulator with 16 real inverse kinematic solution sets, *The International Journal of Robotics Research*, vol. 8, no. 5, pp. 75-79.

[5] Goldenberg, A. A., Benhabib, B., and Fenton, R. G., 1985. A complete generalized solution to the inverse kinematics of robots, Robotics and Automation, IEEE Journal of, vol. 1, no. 1, pp. 14-20.

[6] Wampler, C. W., 1986. Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods, *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16, no. 1, pp. 93-101.

[7] Lapresté, J.T., Jurie, F. M., Dhome, M., and Chaumette, F., 2004. An efficient method to compute the inverse jacobian matrix in visual servoing. *IEEE International Conference on Robotics and Automation*, pp. 727-732.

[8] Fijany, A., and Bejczy, A.K., 1988 Efficient Jacobian inversion for the control of simple robot manipulators. *IEEE International Conference on Robotics and Automation*, vol.2, pp. 999-1007.

[9] Vasilyev, I.A. and Lyashin, A.M., 2010. Analytical solution to inverse kinematic problem for 6-DOF robot-manipulator, Automation and Remote Control, vol. 71, no. 10, pp. 2195-2199

[10] Hayat, A.A., Chittawadigi, R.G., Udai, A.D., and Saha, S.K., 2013. Identification of Denavit-Hartenberg Parameters of an Industrial Robot. pp. 1-6.

[11] Angeles, J., 1997. *Fundamentals of robotic mechanical systems: theory, methods, and algorithms*, Springer, 1997.

[12] Yoshikawa, T., 1985. Manipulability of robotic mechanisms, *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3-9.

[13] Castellet, A., and Thomas, F., 1997. Towards an efficient interval method for solving inverse kinematic problems, *IEEE International Conference on Robotics and Automation*, pp. 3615-3620.

[14] Rajeevlochana,C.G., and Saha,S.K., 2011. RoboAnalyzer: 3D model based robotic learning software. *International Conference on Multibody dynamics*, pp 3-13.

Matlab Codes for Inverse Kinematics:

```
%% Block 1
% IIT Delhi
% Programme by A. A. Hayat || aamir_hayat@rediffmail.com
% INVERSE KInematics of SIX degree of freesom with intersecting
Wrist
% Input will be DH and orientation and positions of EE

clear all
clc
%% Block 2
%Input DH and orientaiton
% DH parameters for KUKA KR5
% Link Length in METERS
a1=.180; a2=.600; a3=.120; a4=0; a5=0; a6=0;

% Joint Offset in METERS
% b6 for kuka =0.115 in RA
b1=.400; b2=0.135; b3=0.135; b4=0.620; b5=0; b6=0.115;

% Twist Angle input in degree and then for calculation rad is used
alpha1=degtorad(90); alpha2=degtorad(180) ; alpha3=degtorad(-90) ;
alpha4=degtorad(90) ; alpha5=degtorad(-90) ; alpha6=degtorad(0) ;

%% Block 3
%Input Position and Orientation of End-effector
% Orientaion Matrix 'Q'

        Ikin_sol_wcell=cell(1,3);
        sol=1;
        Q=[ -0.8086     0       0.5883;
            0          1       0
            -0.5883    0       -0.8086];

%% Block 4
% Uncomment Block 3 and uncomment block 4 to enter rotaiton about
Z, Y, X
% And comment block 4
% The rotation matrix can be found after after multiplying
% Order say, ZYX EULER ANGLES
% Q =Rotation about Z * Rotation about Y * Rotiatoin about X
% Input rotation about x, y and z

%         alpha = degtorad(90);
%         beta = degtorad(0);
%         gamma = degtorad(30);
%
%         Q_x = [cos(alpha) -sin(alpha) 0; sin(alpha) cos(alpha)
0; 0 0 1];
%         Q_y = [cos(beta) 0 sin(beta); 0 1 0; -sin(beta) 0
cos(beta)];
%         Q_z = [1 0 0;0 cos(gamma) -sin(gamma) ; 0 sin(gamma)
cos(gamma)];
%         Q = Q_x*Q_y*Q_z;

%%  Block 5
% Position End-effector
% KUKA RA
xe=0.08;
ye=.1;
ze=1.2;

Ikin_sol_wcell=cell(1,3);
sol=1;
%% Calculation of Wrist position from above

% POSITION OF 'C' ,i.e. Wrist
xc = xe - (Q(1,1)*a6 +
Q(1,2)*b6*sin(alpha6)+Q(1,3)*b6*cos(alpha6));
yc = ye - (Q(2,1)*a6 +
Q(2,2)*b6*sin(alpha6)+Q(2,3)*b6*cos(alpha6));
zc = ze - (Q(3,1)*a6 +
Q(3,2)*b6*sin(alpha6)+Q(3,3)*b6*cos(alpha6));

%% CONSTANTS Defined on pg 189 S. K Saha Second Edition

A=2*a1*xc;
B=2*a1*yc;
C=2*a2*a3-2*b2*b4*sin(alpha2)*sin(alpha3);
D=2*a3*b2*sin(alpha2)+2*a2*b4*sin(alpha3);
E=a2^2+a3^2+b2^2+b3^2+b4^2-a1^2-xc^2-yc^2-(zc-
b1)^2+2*b2*b3*cos(alpha2)+2*b2*b4*cos(alpha2)*cos(alpha3)+2*b3*b4*
cos(alpha3);
F=yc*sin(alpha1);
G=-xc*sin(alpha1);
H=-b4*sin(alpha2)*sin(alpha3);
I=a3*sin(alpha2);
J=b2+b3*cos(alpha2)+b4*cos(alpha2)*cos(alpha3)-(zc-
b1)*cos(alpha1);          %(6.46g)  p.188 book Prof. Saha
“Introduction to Robotics”


K=4*a1^2*H^2+(sin(alpha1))^2*C^2;
L=4*a1^2*I^2+(sin(alpha1))^2*D^2;
M=2*(4*a1^2*H*I+(sin(alpha1))^2*C*D);
N=2*(4*a1^2*J*I+(sin(alpha1))^2*E*D);
P=2*(4*a1^2*J*I+(sin(alpha1))^2*E*D);
```

```
QQ=4*a1^2*J^2+(sin(alpha1))^2*E^2-
4*a1^2*(sin(alpha1))^2*(xc^2+yc^2);  % Since Q used for
Orientation

R=4*a1^2*(J-H)^2+(sin(alpha1))^2*(E-C)^2-
4*(xc^2+yc^2)*a1^2*(sin(alpha1))^2;
S=4*(4*a1^2*I*(J-H)+(sin(alpha1))^2*D*(E-C));
T=2*(4*a1^2*(J^2-H^2+2*I^2)+(sin(alpha1))^2*(E^2-C^2+2*D^2)-
4*(xc^2+yc^2)*a1^2*(sin(alpha1))^2);
U=4*(4*a1^2*I*(J+H)+(sin(alpha1))^2*D*(E+C));
V=4*a1^2*(J+H)^2+(sin(alpha1))^2*(E+C)^2-
4*(xc^2+yc^2)*a1^2*(sin(alpha1))^2;

%% Quantic Equation for getting four solution of theta3
syms zeta
% m=R*gama3^4+S*gama3^3+T*gama3^2+U*gama3+V;
Z=eval(solve(R*zeta^4+S*zeta^3+T*zeta^2+U*zeta+V,zeta));  %
(6.48a)
a=imag(Z);
temp=1;
for i=1:4
    if a(i)==0
        theta3=2*atan2(Z(i),1);
        theta3_d=radtodeg(theta3);
        THETA3(temp)=theta3_d;
        del1=-2*a1*sin(alpha1)*(xc^2+yc^2);
        A11=a2+a3*cos(theta3)+b4*sin(alpha3)*sin(theta3);
        A12=-
a3*cos(alpha2)*sin(theta3)+b3*sin(alpha2)+b4*cos(alpha2)*sin(alpha
3)*cos(theta3)+b4*sin(alpha2)*cos(alpha3);
        del2=A11^2+A12^2;

        if abs(del1)>0.0001
            c1=(1/del1)*(-G*(C*cos(theta3)+D*sin(theta3)+E)+
B*(H*cos(theta3)+I*sin(theta3)+J));
            s1=(1/del1)*(F*(C*cos(theta3)+D*sin(theta3)+E)- A*(
H*cos(theta3)+I*sin(theta3)+J));
            theta1=atan2(s1,c1);
            theta1_d=radtodeg(theta1);
            THETA1(temp)=theta1_d;

            if abs(del2)>0.0001
                c2=(1/del2)*(A11*(xc*cos(theta1)+yc*sin(theta1)-
a1)-A12*(-
xc*cos(alpha1)*sin(theta1)+yc*cos(alpha1)*cos(theta1)+(zc-
b1)*sin(alpha1)));
                s2=(1/del2)*(A12*(xc*cos(theta1)+yc*sin(theta1)-
a1)+A11*(-
xc*cos(alpha1)*sin(theta1)+yc*cos(alpha1)*cos(theta1)+(zc-
b1)*sin(alpha1)));
                theta2=atan2(s2,c2);
                theta2_d=radtodeg(theta2);
                THETA2(temp)=theta2_d;

            end

        end

    end
    temp=temp+1;
end

Theta_123= [THETA1' THETA2' THETA3'];
%
Theta123=[Theta_123(4,:);Theta_123(3,:);Theta_123(2,:);Theta_123(1
,:)];
%
%% In RA it is Displayed as following set
%  disp('solution1')
sol1_degree = [Theta_123(4,1) Theta_123(4,2) Theta_123(4,3)];
sol1=degtorad([Theta_123(4,1) Theta_123(4,2) Theta_123(4,3)]);
%
%  disp('solution2')
sol2_degree = [Theta_123(4,1) Theta_123(4,2) Theta_123(4,3)];
sol2 =   degtorad([Theta_123(4,1) Theta_123(4,2) Theta_123(4,3)]);
%
%  disp('solution3')
sol3_degree =   [Theta_123(3,1) Theta_123(3,2) Theta_123(3,3)];
sol3 = degtorad([Theta_123(3,1) Theta_123(3,2) Theta_123(3,3)]);
%
%  disp('solution4')
sol4 =   [Theta_123(3,1) Theta_123(3,2) Theta_123(3,3)];
%
%  disp('solution5')
sol5_degree =   [Theta_123(2,1) Theta_123(2,2) Theta_123(2,3)];
sol5 =deg2rad([Theta_123(2,1) Theta_123(2,2) Theta_123(2,3)]);
%
%   disp('solution6')
sol6_degree =   [Theta_123(2,1) Theta_123(2,2) Theta_123(2,3)];
sol6 = degtorad([Theta_123(2,1) Theta_123(2,2) Theta_123(2,3)]);
%
%   disp('solution7')
sol7_degree =  [Theta_123(1,1) Theta_123(1,2) Theta_123(1,3)];
sol7 = degtorad([Theta_123(1,1) Theta_123(1,2) Theta_123(1,3)]);
%
%   disp('solution8')
sol8_degree =  [Theta_123(1,1) Theta_123(1,2) Theta_123(1,3)];
sol8 =degtorad([Theta_123(1,1) Theta_123(1,2) Theta_123(1,3)]);


%% Wrist partition inverse kinmeatic pg. 184 Prof. S. K. Saha
```

```
% For solution Set 1 angle 4 5 6
R36_1 =
((RZ(sol1(3))*RX(alpha3))'*(RZ(sol1(2))*RX(alpha2))'*(RZ(sol1(1))*
RX(alpha1))')*Q ;  % RZ and RX are rotation functions
Theta_456_1 = Theta456(R36_1) ;     %Theta456 is a function

% For solution set 3 angle 4, 5, 6
%For solution set 3 and 4
R36_3 =
((RZ(sol3(3))*RX(alpha3))'*(RZ(sol3(2))*RX(alpha2))'*(RZ(sol3(1))*
RX(alpha1))')*Q  ; % RZ and RX are rotation functions
Theta_456_3 = Theta456(R36_3) ;     %Theta456 is a function
%
% For solution set 3 angle 4, 5, 6
% For solution set 5 and 6
R36_5 =
((RZ(sol5(3))*RX(alpha3))'*(RZ(sol5(2))*RX(alpha2))'*(RZ(sol5(1))*
RX(alpha1))')*Q  ; % RZ and RX are rotation functions
Theta_456_5 = Theta456(R36_5) ;     %Theta456 is a function

% For solution set 3 angle 4, 5, 6
% For solution set 7 and 8
R36_8 =
((RZ(sol8(3))*RX(alpha3))'*(RZ(sol8(2))*RX(alpha2))'*(RZ(sol8(1))*
RX(alpha1))')*Q ;  % RZ and RX are rotation functions
Theta_456_7 = Theta456(R36_8);      %Theta456 is a function


%% Solutions set in the form of RoboAnalyzer
sol1_degrees = [Theta_123(4,1) Theta_123(4,2) Theta_123(4,3)
Theta_456_1(2,1) Theta_456_1(1,2) Theta_456_1(1,3)];
sol2_degrees = [Theta_123(4,1) Theta_123(4,2) Theta_123(4,3)
Theta_456_1(1,1) Theta_456_1(2,2) Theta_456_1(2,3)];
sol3_degrees = [Theta_123(3,1) Theta_123(3,2) Theta_123(3,3)
Theta_456_3(2,1) Theta_456_3(1,2) Theta_456_3(1,3)];
sol4_degrees = [Theta_123(3,1) Theta_123(3,2) Theta_123(3,3)
Theta_456_3(1,1) Theta_456_3(2,2) Theta_456_3(2,3)];
sol5_degrees = [Theta_123(2,1) Theta_123(2,2) Theta_123(2,3)
Theta_456_5(2,1) Theta_456_5(1,2) Theta_456_5(1,3)];
sol6_degrees = [Theta_123(2,1) Theta_123(2,2) Theta_123(2,3)
Theta_456_5(1,1) Theta_456_5(2,2) Theta_456_5(2,3)];
sol7_degrees = [Theta_123(1,1) Theta_123(1,2) Theta_123(1,3)
Theta_456_7(2,1) Theta_456_7(1,2) Theta_456_7(1,3)];
sol8_degrees = [Theta_123(1,1) Theta_123(1,2) Theta_123(1,3)
Theta_456_7(1,1) Theta_456_7(2,2) Theta_456_7(2,3)];

%% Next
Ikin_sol=[sol1_degrees ; sol2_degrees; sol3_degrees; sol4_degrees;
sol5_degrees; sol6_degrees; sol7_degrees; sol8_degrees];



%% Wrapping angle in the range of -180 to 180 degrees
[nrows ncol]= size(Ikin_sol);
for r = 1:nrows
    for c = 1:ncol
Ikin_sol_w(r,c)= mod(Ikin_sol(r,c),360);
         if (Ikin_sol_w(r,c)) < -180
         Ikin_sol_w(r,c) = Ikin_sol_w(r,c)+360;
        elseif (Ikin_sol_w(r,c)) > 180
            Ikin_sol_w(r,c) = Ikin_sol_w(r,c)-360;
        end
%
    end
end
Ikin_sol_w;
format bank
Ikin_sol_wcell{sol}= Ikin_sol_w;

%% Display the matrix in command window
printmat(Ikin_sol_w, 'Eight Inverse Kinematics Solution of Wrist
Partitioned Robot',...
       'Sol1 Sol2 Sol3 Sol4 sol5 Sol6 Sol7 Sol8',...
       'Theta_1 Theta_2 Theta_3 Theta_4 Theta_5 Theta_6');


%% Writing in the file
filename = 'IKin_eight_solutions.csv';
fid = fopen(filename, 'w');
fprintf(fid, 'Theta_1 Theta_2 Theta_3 Theta_4 Theta_5 Theta_6\n');
fclose(fid)
dlmwrite(filename, Ikin_sol_w, '-append', 'precision', '%.6f',
'delimiter', '\t');
```

%% Functions Files

1. For Rotation about X

```
% this function returns a rotation matrix corresponding to
% X "1" rotation through angle
function Rotmat = RX(angle)
   c = cos(angle); s = sin(angle);
   Rotmat = [ 1 0 0;
              0 c -s;
              0 s c ];
```

2. For Rotation about Y

```
function Rotmat = RY(angle)

   c = cos(angle); s = sin(angle);
   Rotmat = [ c 0 s;
              0 1 0;
              -s 0 c];
```

3. For Rotation about Z

```
function Rotmat = RZ(angle)
   c = cos(angle); s = sin(angle);
   Rotmat = [ c -s 0;
              s c 0;
              0 0 1];
```

4. For Wrist angle partition solutions

```
% A WRIST pg. 184 of Intro. to robotics by S. K. Saha

% Input to the function is the rotation ,matrix obtained by ...
% R_30(Transpose)* R*60 = R_63
% R_63 is the roation matrix of the EE w.r.t wrist
% Input Q of the function is R_63 gives I_Kin angles of joint 4,
5, 6.

function Theta_456 = Theta456(Q)

%  FIRST CONFIGURATION-angles in radian
 theta_41=atan2(Q(2,3),Q(1,3));                               %
6.43a
 theta_51=atan2(sqrt(Q(1,3)*Q(1,3)+Q(2,3)*Q(2,3)),Q(3,3));  %
6.43b
 theta_61=atan2(-Q(3,2),Q(3,1));                              %
6.43c

%  SECOND CONFIGURATION-angles in radian
theta_42=theta_41+pi;
theta_52=-theta_51;
theta_62=theta_61+pi;

% CONVERSION IN DEGREE
% result:angles in degree
r2d=180/pi; %r2d means radian to degree conversion factor

th41d=theta_41*r2d;
th51d=theta_51*r2d;
th61d=theta_61*r2d;

th42d=theta_42*r2d;
th52d=theta_52*r2d;
th62d=theta_62*r2d;

% disp('Two sets of solutions are arranged row wise');
Theta_456= [th41d th51d th61d;
            th42d th52d  th62d];
```

Note: Keep all the files in one folder along with the function files with there names as mentioned in function.

(For example `Rotmat = `**`RX`**`(angle)]` is saved as **RX.m**

Find the files on the web link:

http://web.iitd.ernet.in/~mez118119/Conference_Proceedings.html

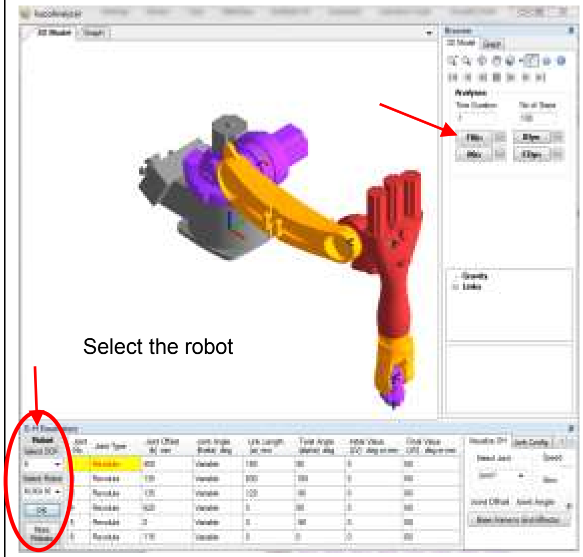
Matlab code for Jacobian

```
%% IIT Delhi
% A. A. Hayat :: aamir_hayat@rediffmail.com
% Function for calculating Jacobian of six DOF,
% On the basis of number of links it can be modified

% The angle values are the input to the Jacobian, i.e., the joint
%values in DEGREES (here in this code) for a given pose of the
%robot
%angle value are [theta1, theta2, theta3, tehta4, theta5, theta6]

function [kr5_jacobian] = kuka_jacobian( angle_value)

syms a1 a2 a3 a4 a5 a6 b1 b2 b3 b4 b5 b6 al1 al2 al3 al4 al5 al6
th1 th2 th3 th4 th5 th6
% % Numerical Values of the DH
% a1=.180; a2=.600; a3=.120; a4=0; a5=0; a6=0;
%
% % Joint Offset in METERS
% % b6 for kuka =0.115 in RA
% b1=.400; b2=0.135; b3=0.135; b4=0.620; b5=0; b6=0.115;
%
% % Twist Angle input in degree and then for calculation rad is
used
% A1=degtorad(90); A2=degtorad(180) ; A3=degtorad(-90) ;
% A4=degtorad(90) ; A5=degtorad(-90) ; A6=degtorad(0) ;

b1 = 0.400;     a1 = 0.180 ;   al1 = 90 ; sal1 = sind(al1)  ;
cal1 = cosd(al1);
b2 = 0.135;    a2 = 0.600;    al2 =  180 ;  sal2 = sind(al2)  ;
cal2 = cosd(al2);
b3 = 0.135  ;  a3 = 0.120;    al3 = -90 ;  sal3 = sind(al3) ; cal3
= cosd(al3);
b4 = 0.620;    a4 = 0   ;     al4 = 90 ; sal4 = sind(al4)  ; cal4
= cosd(al4);
b5 = 0  ;      a5 = 0   ;     al5 = -90 ; sal5 = sind(al5) ; cal5
= cosd(al5);
b6 = 0.115;    a6 = 0   ;     al6 = 0  ; sal6 = sind(al6)   ; cal6
= cosd(al6);

% th1 = theta_1 ; th2 = theta_2 ;  th3 = theta_3; th4 = theta_4 ;
th5 = theta_5; th6 = theta_6;
th1 = angle_value(1); th2 = angle_value(2); th3 = angle_value(3);
th4 = angle_value(4); th5 = angle_value(5); th6 = angle_value(6);

%% DH TRANSFORMATION MATRIX CALCULATION

t1m =  [ cosd(th1) ,-sind(th1)*cal1 ,  sind(th1)*sal1
,cosd(th1)*a1 ;
sind(th1) , cosd(th1)*cal1 , -cosd(th1)*sal1 ,sind(th1)*a1  ;
0  ,  sal1      ,     cal1   , b1     ;
0,       0     ,      0     ,  1     ; ];

t2m =  [ cosd(th2) ,-sind(th2)*cal2 ,  sind(th2)*sal2
,cosd(th2)*a2 ;
sind(th2) , cosd(th2)*cal2 , -cosd(th2)*sal2 ,sind(th2)*a2  ;
0  ,  sal2      ,     cal2   , b2     ;
0,       0     ,      0     ,  1     ; ];
t3m =  [ cosd(th3) ,-sind(th3)*cal3 ,  sind(th3)*sal3
,cosd(th3)*a3 ;
sind(th3) , cosd(th3)*cal3 , -cosd(th3)*sal3 ,sind(th3)*a3  ;
0  ,  sal3      ,     cal3   , b3     ;
0,       0     ,      0     ,  1     ; ];
t4m =  [ cosd(th4) ,-sind(th4)*cal4 ,  sind(th4)*sal4
,cosd(th4)*a4 ;
sind(th4) , cosd(th4)*cal4 , -cosd(th4)*sal4 ,sind(th4)*a4  ;
0  ,  sal4      ,     cal4   , b4     ;
0,       0     ,      0     ,  1     ; ];
t5m =  [ cosd(th5) ,-sind(th5)*cal5 ,  sind(th5)*sal5
,cosd(th5)*a5 ;
sind(th5) , cosd(th5)*cal5 , -cosd(th5)*sal5 ,sind(th5)*a5  ;
0  ,  sal5      ,     cal5   , b5     ;
0,       0     ,      0     ,  1     ; ];
t6m =  [ cosd(th6) ,-sind(th6)*cal6 ,  sind(th6)*sal6
,cosd(th6)*a6 ;
sind(th6) , cosd(th6)*cal6 , -cosd(th6)*sal6 ,sind(th6)*a6  ;
0  ,  sal6      ,     cal6   , b6     ;
0,       0     ,      0     ,  1     ; ];

% NEW ADDITION


dh = t1m * t2m * t3m * t4m * t5m * t6m;

% dh(1:3,1:3)  = initial_orientation(1:3,1:3);



%% EXTRACTION OF  ai VECTORS FROM FRAMES

t56m = t5m * t6m  ;
t46m = t4m * t56m ;
t36m = t3m * t46m ;
t26m = t2m * t36m ;
t16m = t1m * t26m ;

% disp( t16m - dh); % to check if the transofrmation has been
correct;
```

```
% extracting a_i vectors
a6_vec  = t6m(1:3 ,4) ;
a56_vec = t56m(1:3 ,4);
a46_vec = t46m(1:3 ,4);
a36_vec = t36m(1:3 ,4);
a26_vec = t26m(1:3 ,4);
a16_vec = t16m(1:3 ,4);

%%  EXTRACTION OF e_i VECTORS FROM FRAMES

%  We need to first extract the rotation matrices.

rot_1m =          t1m(1:3,1:3);
rot_2m = rot_1m * t2m(1:3,1:3);
rot_3m = rot_2m * t3m(1:3,1:3);
rot_4m = rot_3m * t4m(1:3,1:3);
rot_5m = rot_4m * t5m(1:3,1:3);

e1_vec = transpose([ 0 0 1]);
e2_vec = rot_1m(1:3,3);
e3_vec = rot_2m(1:3,3);
e4_vec = rot_3m(1:3,3);
e5_vec = rot_4m(1:3,3);
e6_vec = rot_5m(1:3,3);

e1_cross_a1_vec = cross(e1_vec, a16_vec ) ;
e2_cross_a2_vec = cross(e2_vec, a26_vec ) ;
e3_cross_a3_vec = cross(e3_vec, a36_vec ) ;
e4_cross_a4_vec = cross(e4_vec, a46_vec ) ;
e5_cross_a5_vec = cross(e5_vec, a56_vec ) ;
e6_cross_a6_vec = cross(e6_vec, a6_vec ) ;

%% WRITING THE JACOBIAN MATRIX
% kr5_jacobian =  [ e1_cross_a1_vec e2_cross_a2_vec
e3_cross_a3_vec e4_cross_a4_vec e5_cross_a5_vec e6_cross_a6_vec ]
;
i=1;
kr5_jacobian = [ e1_vec   e2_vec   e3_vec   e4_vec   e5_vec
e6_vec ;
                 e1_cross_a1_vec e2_cross_a2_vec e3_cross_a3_vec
e4_cross_a4_vec e5_cross_a5_vec e6_cross_a6_vec ] ;

end
```

For performing Forward Kinematics in ROBOANALYZER SOFTWARE:

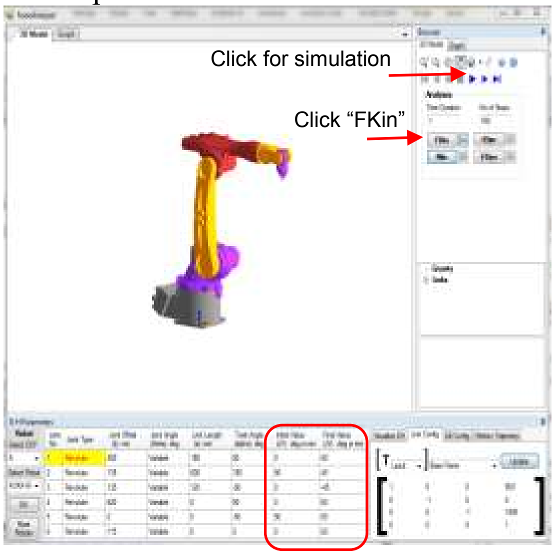
Step1: Select the Robot model



Select the robot

Initial configuration of the KUKA KR5 model when it appears on the screen for first time is shown.

Step2: Provide initial and final configuration of Joint Space

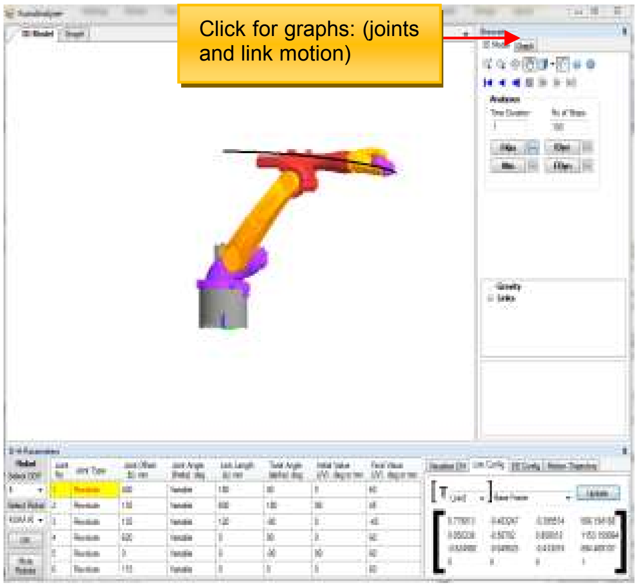


Click for simulation

Click “FKin”

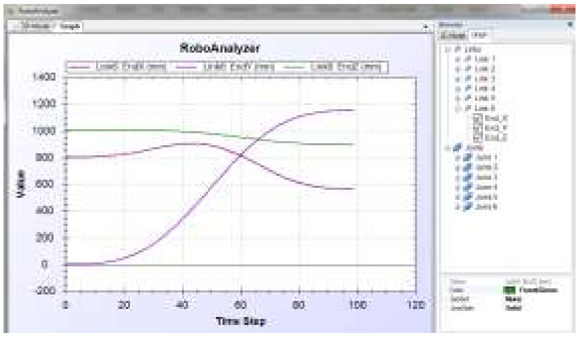
Change the configuration to the desired initial and final position of joint values. Here, for joint 1 to 6 initial values are (0, 90, 0,0 ,90, 0) and final values are (60, 45, -45, 60, 60, 60).

Step 3: Simulation plot of the end-effector



Click for graphs: (joints and link motion)

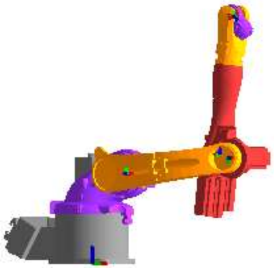
Step4: Analyze the plots of each joint and the links accordingly by clicking on the plotting options.

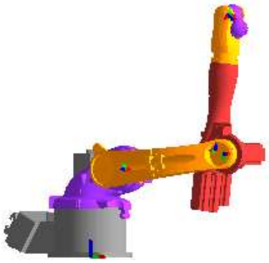


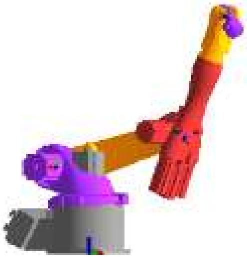
For example the above plot is forthe joint 6 positions (X, Y, and Z)

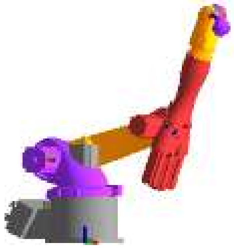
[Note: One can export the data points by right clicking on the graph and then “Export data as CSV.” This will save the data for future analysis]

Click “IKIN”, it will open a new window. Click “Show”

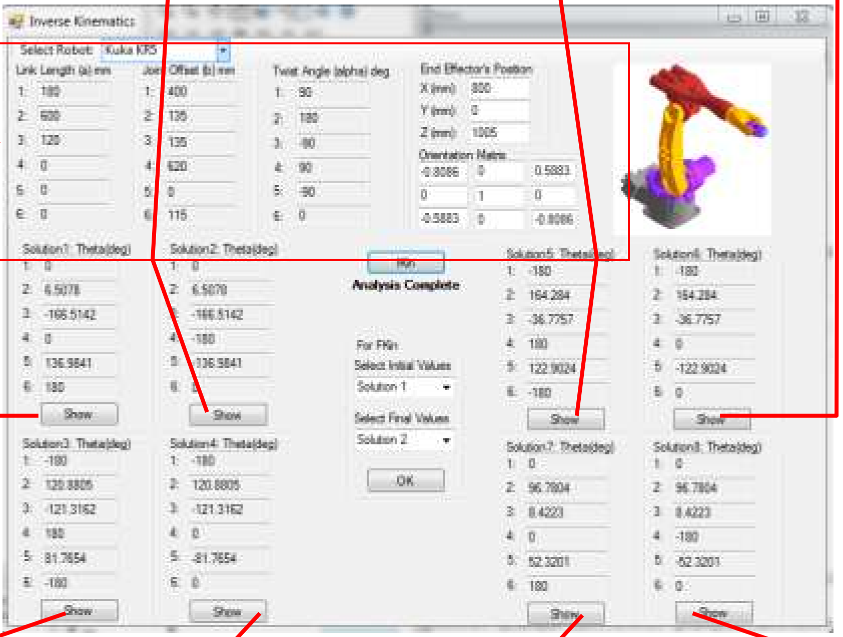








Initial Configuration



Select Robot: Kuka KR5

	Link Length (a) mm	Joint Offset (b) mm	Twist Angle (alpha) deg
1:	180	400	90
2:	600	135	180
3:	120	135	-90
4:	0	620	90
5:	0	0	-90
6:	0	115	0

End Effector's Position

X (mm): 800

Y (mm): 0

Z (mm): 1005

Orientation Matrix

-0.8086	0	0.5883
0	1	0
-0.5883	0	-0.8086

Analysis Complete

For FKin

Select Initial Values: Solution 1

Select Final Values: Solution 2

	Solution1: Theta(deg)	Solution2: Theta(deg)	Solution5: Theta(deg)	Solution6: Theta(deg)
1:	0	0	-180	-180
2:	6.5078	6.5078	164.284	164.284
3:	-166.5142	-166.5142	-36.7757	-36.7757
4:	0	-180	180	0
5:	136.9841	-136.9841	122.9024	-122.9024
6:	180	0	-180	0

	Solution3: Theta(deg)	Solution4: Theta(deg)	Solution7: Theta(deg)	Solution8: Theta(deg)
1:	-180	-180	0	0
2:	120.8805	120.8805	96.7804	96.7804
3:	-121.3162	-121.3162	8.4223	8.4223
4:	180	0	0	-180
5:	81.7654	-81.7654	52.3201	-52.3201
6:	-180	0	180	0