Inverse Kinematics (IK) Solution of a Robotic Manipulator using PYTHON

R. Venkata Neeraj Kumar and R. Sreenivasulu

Received: 20-07-2019
Revised: 06-08-2019
Accepted: 22-08-2019

Corresponding Author:
Dr. R. Sreenivasulu
R.V.R.&J.C. College of Engineering (Autonomous), Chowdavaram, Guntur Andhra Pradesh, India
Email: rslu1431@gmail.com

Abstract: Present global engineering professionals feels that, robotics is a somewhat young field with extremely ruthless target, the crucial one being the making of machinery/equipment that can perform and feel like human beings. Robot kinematics deals with the study of motion of linkages which includes displacement, velocities and accelerations of a robot manipulator analytically. Deriving the proper kinematic models for an open chain mechanism of a robot is essential for analyzing the performance of industrial robotic manipulators. In this study, first scrutinize a popular class of two and three degrees of freedom open chain mechanism whose inverse kinematics admits a closed-form analytic solution. A simple coding was developed in python in an easy way. In this connection a two link planer manipulator was considered to get inverse kinematic solution developed in python environment. For this task, we present a solution for obtaining the joint variables of linkages to reach the position in a work space with the corresponding input values such as link lengths and position of end effector.

Keywords: Robotic Manipulator, Inverse Kinematics, Joint Variables, PYTHON

Introduction

Robot kinematics deals with the study of motion of linkages which includes displacement, velocities and accelerations of a robot manipulator analytically. Deriving the proper kinematic models for an open chain mechanism of a robot is essential for analyzing the performance of industrial robotic manipulators. The task related to robot trajectory path planning control can be split into two types, one is the coordination of the links of kinematics chain to produce desired motions of the robot and the other is dynamic control i.e., linkage driving mechanism using actuators technology by providing position and velocity sensors. In the elementary level, the robot manipulator design concentrate on physical arrangement of linkages and mechanisms, includes development of forward and inverse kinematic equations with standard existing methods.

Literature Review

In the field of robotic engineering, researchers such as Kircanski and Vukobratovic (1985; 1986), Morris (1987), Hussain and Nobie (1985) and Tsai and Chiou (1989) were used MACSYMA, REDUCE, SMP and SEGM methods, these methods requires typical mathematical concepts to achieve forward and inverse solutions. Khatib (1987) worked in this area and gave solution to prevent singular positions obtained during the control of path planning within the work volume. Lloyd and Hayward (1993) developed a new design of multi-RCCCL motion generator which is operated by a programming with ‘C’ language in a UNIX environment. Mandava and Vundavilli (2016) proposed a closed form solution for 18 DOF biped robot based on inverse kinematics and concept of Zero Moment Point (ZMP) method. Sreenivasulu (2012) attempted an inverse solution for a two degree freedom robotic manipulator using geometric approach. Nugroho et al. (2014) designed and implemented to develop a NAO humanoid robot using kinematic approach. Sadiq et al. (2017) applied to obtain exact solutions to found optimal path using particle swarm optimization (PSO) algorithm in Cartesian space map for two degrees of freedom robotic manipulator. Chaitanya and Reddy (2016) developed a model for optimization of path planning of two degree of freedom robotic manipulator.


Literature depicts that previous investigators focused on various aspects of methods involved in design and development of inverse kinematics solutions for a different configurations of robotic manipulators. Compared to geometric approaches, programmable studies on inverse kinematic solutions have been found to be a limited extent. Also found that nobody applied PYTHON software to get inverse solution of robotic linkages with multi degrees of freedom problems especially in robotic field.

Inverse Kinematics

The robot inverse kinematics task is concerned with the recognition of the whole feasible and proper sets of joint variables that would understand the solution to find out the positions and orientations of the end effector. In the inverse kinematics problem would not specify constantly a unique solution compared with forward solution i.e., number of solutions to be obtained for one end effector position to reach the specified position and orientation.

Case I. Two Link Planar Manipulator Manipulator

Consider a two link planer manipulator having link lengths $l_1$, $l_2$ and joint angles $\theta_1$ and $\theta_2$ to reach a desired position $(P_x, P_y)$ as shown in Fig. 1. For this, inverse solution is derived from geometric approach as per the diagram shown in Fig. 2 is as follows:

$$P_x = l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2)$$

(1)
Fig. 2: Geometric model for a 2 link manipulator

\[ p_x = l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) \]  
\[ \theta_2 = \arctan \left( \frac{\pm \sin \theta_2 \cos \theta_2}{\cos \theta_2 + \sin \theta_2} \right) \]  
Finally, two possible solutions for \( \theta_2 \) can be written as:

\[ \cos \theta_2 p_x = l_1 \cos^2 \theta_2 + l_2 \cos \theta_2 \cos \theta_1 - l_2 \cos \theta_1 \sin \theta_1 \sin \theta_2 \]
\[ \sin \theta_2 p_y = l_1 \sin \theta_1 \cos \theta_1 \sin \theta_2 \]

The simplified equation obtained as follows:

\[ \cos \theta_2 p_x + \sin \theta_2 p_y = l_1 \cos \theta_1 \cos \theta_2 + l_2 \sin \theta_1 \cos \theta_1 \sin \theta_2 \]
\[ \sin \theta_2 p_y = l_2 \sin \theta_1 \cos \theta_1 \sin \theta_2 \]
Equation (IK) of a two link RR planar robotic manipulator in the PYTHON environment.

\[
\begin{align*}
\cos \theta_1 & = \frac{p_x (l_1 + l_2 \cos \theta_2) + p_y l_2 \sin \theta_2}{p_x^2 + p_y^2}, \\
\sin \theta_1 & = \pm \sqrt{1 - \left(\frac{p_x (l_1 + l_2 \cos \theta_2) + p_y l_2 \sin \theta_2}{p_x^2 + p_y^2}\right)^2}
\end{align*}
\]

As a result, two possible solutions for \( \theta_1 \) can be written:

\[
\theta_1 = \arctan 2 \left\{ \pm \sqrt{1 - \left(\frac{p_x (l_1 + l_2 \cos \theta_2) + p_y l_2 \sin \theta_2}{p_x^2 + p_y^2}\right)^2}, \right. \}
\]

Using Equations 5 and 10 joint variables are found by substituting the given input values. In this method laborious calculations involved and it increases by increasing the series of linkages. So it is necessary to develop some programming codes. Till now the researchers utilized Artificial Neural Networks (ANN), MATLAB, C&C++, Fuzzy Logics and some statistical techniques. In this study a programmable code developed for Inverse Kinematics (IK) of a robotic manipulator in simple manner to understand by the young people. Python is an efficient and easy to learn programming language. One of the most popular applications of Python is in numerical analysis. Solving problems related to numerical analysis using programming languages makes the task easier. It takes less effort to code using Python compared to other programming languages like C/C++/Java. Within a shorter period of time, one can do more with relatively less code. Also, unlike C, Python has a large library of built-in functions and the programming syntax is much simpler. It could save a great deal of time since it allows one to focus on actual research rather than tool being used.

**Results and Discussion**

In the inverse kinematics, geometric approach contains lot of mathematical expressions. These are increased by increasing number of linkages causes more complexity to solve manually but it is little bit easy in MATLAB. In this study a simplified code generated in PYTHON environment in easy way with fast generation of output results. In the present discussion two and three link planer manipulators were selected and the same procedure can also be applied to complex robotic linkages. For this case selected a linkages of length 10 cm and 5 cm to reach a position (12.99 cm, 2.5 cm) as x, y positions of end effector or tip of the manipulator. To found the joint variables of manipulator linkages, Equation 5 and 10 are applied and obtained the values. Then these values are compared with PYTHON outputs.

The following is the code developed for Inverse Kinematics (IK) of a two link RR planar robotic manipulator in the PYTHON environment.

```python
import math

class Kine2:
    def inputCalc(self):
        px = float(input("px:"))
        py = float(input("py:"))
        l1 = float(input("l1:"))
        l2 = float(input("l2:"))
        ctheta1 = (px * (l1 + l2 * ctheta2) + py * l2 * ctheta2) / (px**2 + py**2)
        ctheta2 = (px**2 + py**2 - l1**2 - l2**2) / (2 * l1 * l2)
        stheta2 = math.sqrt(1 - ctheta2**2)
        ctheta1 = px * (1 + ctheta2 + py * l2) / (px**2 + py**2)
        theta1a, theta1b = self.calcAngle(ctheta1)
        theta2a, theta2b = self.calcAngle(ctheta2)
        print("theta1: {} and {}").format(math.degrees(theta1a), math.degrees(theta1b))
        print("theta2: {} and {}").format(math.degrees(theta2a), math.degrees(theta2b))
        kine2 = Kine2()
        kine2.inputCalc()

    def calcAngle(self, reqcos):
        res1 = math.atan2(math.sqrt(1 - reqcos**2), reqcos)
        res2 = math.atan2(math.sqrt(1 - reqcos**2), reqcos)*(-1)
        return (res1, res2)

    def inputCalc(self):
        px = float(input("px:"))
        py = float(input("py:"))
        l1 = float(input("l1:"))
        l2 = float(input("l2:"))
        ctheta1 = (px * (l1 + l2 * ctheta2) + py * l2 * ctheta2) / (px**2 + py**2)
        ctheta2 = (px**2 + py**2 - l1**2 - l2**2) / (2 * l1 * l2)
        stheta2 = math.sqrt(1 - ctheta2**2)
        ctheta1 = px * (1 + ctheta2 + py * l2) / (px**2 + py**2)
        theta1a, theta1b = self.calcAngle(ctheta1)
        theta2a, theta2b = self.calcAngle(ctheta2)
        print("theta1: {} and {}").format(math.degrees(theta1a), math.degrees(theta1b))
        print("theta2: {} and {}").format(math.degrees(theta2a), math.degrees(theta2b))

    def inputCalc(self):
        px = float(input("px:"))
        py = float(input("py:"))
        l1 = float(input("l1:"))
        l2 = float(input("l2:"))
        ctheta1 = (px * (l1 + l2 * ctheta2) + py * l2 * ctheta2) / (px**2 + py**2)
        ctheta2 = (px**2 + py**2 - l1**2 - l2**2) / (2 * l1 * l2)
        stheta2 = math.sqrt(1 - ctheta2**2)
        ctheta1 = px * (1 + ctheta2 + py * l2) / (px**2 + py**2)
        theta1a, theta1b = self.calcAngle(ctheta1)
        theta2a, theta2b = self.calcAngle(ctheta2)
        print("theta1: {} and {}").format(math.degrees(theta1a), math.degrees(theta1b))
        print("theta2: {} and {}").format(math.degrees(theta2a), math.degrees(theta2b))

Inputs and Outputs:
px: 12.99 cm
py: 2.5 cm
l1:10cm
l2:5cm
theta1:8.21477006006 and -60.0065495712
theta2:60.0065495712 and -60.0065495712
```

Now, multiply each side of Equation 6 by \( p_x \) and Equation 7 by \( p_y \), and add the resulting equations in order to obtain \( \cos \theta_1 \):

\[
\cos \theta_1 p_x^2 + \sin \theta_1 p_x p_y = p_x (l_1 + l_2 \cos \theta_2)
\]

\[
-\sin \theta_1 p_x p_y + \cos \theta_1 p_y^2 = p_y l_2 \sin \theta_2
\]

\[
\cos \theta_1 (p_x^2 + p_y^2) = p_x (l_1 + l_2 \cos \theta_2) + p_y l_2 \sin \theta_2
\]

Therefore:

\[
\cos \theta_1 = \frac{p_x (l_1 + l_2 \cos \theta_2) + p_y l_2 \sin \theta_2}{p_x^2 + p_y^2}
\]

\[
\sin \theta_1 = \pm \sqrt{1 - \left(\frac{p_x (l_1 + l_2 \cos \theta_2) + p_y l_2 \sin \theta_2}{p_x^2 + p_y^2} \right)^2}
\]
Fig. 3: Elbow up position and orientation of 2 link manipulator for first solution

Fig. 4: Elbow down position and orientation of 2 link manipulator for first solution
The following are the different configurations executed by feeding the input values in developed programme and depicted in the following Fig. 3 to 6.

**Case II. Three Link RR Planar Manipulator**

Consider a three link RR planar manipulator along xy plane. For this mechanism also, similar geometric approach attempted for obtaining inverse solution. But the results in this method revealed that when number of links are increased then the solution procedure becomes complex. So for serial planer manipulators geometric approach gives complex solution. In this connection, PYTHON code provides easy steps to achieve solution and it helps to reduce the time and also gives accurate performance while running the programme comparing with MATLAB. The programme code and elbow down and elbow up configurations shown in the following Fig. 7 to 9.
Fig. 7: Geometric model diagram of 3 link manipulator

Fig. 8: Elbow down position and orientation of 3 link manipulator for first solution
Fig. 9: Elbow up position and orientation of 3 link manipulator for first solution

The following is the code developed for Inverse Kinematics (IK) of a three link RR planar robotic manipulator in the PYTHON environment.

```python
import math
class Kine3:
    def cosCalc(self, a, b, c):
        angle = math.acos((a*a + b*b - c*c)/(2*a*b))
        return angle

    def inputCalc(self):
        xe = float(input("xe: "))
        ye = float(input("ye: "))
        phie = math.radians(float(input("phie in degrees: ")))
        l1 = float(input("l1: "))
        l2 = float(input("l2: "))
        l3 = float(input("l3: "))

        xw = xe - l3 * math.cos(phie)
        yw = ye - l3 * math.sin(phie)
        r = math.sqrt(xw**2 + yw**2)
        gamma = self.cosCalc(r, l1, l2)

        theta2 = math.pi - self.cosCalc(l1, l2, r)
        theta1 = math.atan2(yw, xw) - gamma
        theta3 = phie - theta1 - theta2

        print("theta1: {} and ".format(math.degrees(theta1), math.degrees(theta1 + 2 * gamma)))
        print("theta2: {} and ".format(math.degrees(theta2), math.degrees(theta2 - 1)))
        print("theta3: {} and ".format(math.degrees(theta3), math.degrees(theta3 + 2 * (theta2 - gamma))))

kine3 = Kine3()
kine3.inputCalc()
```

**Inputs and Outputs**

<table>
<thead>
<tr>
<th>Input</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xe</td>
<td>12.99 cm</td>
</tr>
<tr>
<td>ye</td>
<td>2.5 cm</td>
</tr>
<tr>
<td>phie in degrees</td>
<td>130</td>
</tr>
<tr>
<td>l1</td>
<td>10 cm</td>
</tr>
<tr>
<td>l2</td>
<td>5 cm</td>
</tr>
<tr>
<td>l3</td>
<td>2 cm</td>
</tr>
</tbody>
</table>

theta1: -8.297738622523704 and 16.05539094694738
theta2: 37.12800501988501 and -37.12800501988501
theta3: 101.1697336026387 and 151.07261407293765

The following configurations drawn for every elbow up and elbow down positions of manipulator links to reach a desired position.
Conclusion

The proposed approach has been demonstrated for a two and three link planar manipulators, it should be examined for a few more intricate arrangements of robotic linkages such as greater than two degrees of freedom manipulators and parallel linkages. This can be possibly implemented for different model cases in which the solution for the mathematical expression of the inverse kinematics is using PYTHON software code.

Acknowledgement

The authors are thankful to management, principal and department of mechanical engineering and computer science engineering staff for providing advices while preparing this paper.

Funding Information

There is no funding agency to support for preparation of this manuscript.

Author’s Contributions

R. Sreenivasulu: Developed the theoretical frame work and supervised the research.

R. Venkata Neeraj Kumar: Developed a programme in PYTHON and obtained the results.

Ethics

This article is an original research paper. There are no ethical issues that may arise after the publication of this manuscript.

References


