The control of a five-axis robotic arm and its experimental application

Ozcan Cetinkaya¹, Hilmi Kuscu², Kenan Kilicaslan³
¹Kesan Vocational School, Trakya University, Edirne, Turkey
²Faculty of Engineering, Department of Mechanical Engineering, Trakya University, Edirne, Turkey
³Giresun Governorship, Giresun, Turkey
¹Corresponding author
E-mail: ¹ozcancetinkaya@trakya.edu.tr, ²hilmi@trakya.edu.tr, ³kenan@kilicaslan.net.tr

Received 10 September 2020; received in revised form 1 October 2020; accepted 9 October 2020

Copyright © 2020 Ozcan Cetinkaya, et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract. The study discusses the control of a five-axis robotic arm, the orbit planning involved, and experimental applications that make use of the arm. A robotic arm unit located at the Trakya University has been used for the study; LabVIEW 2010 (student edition) and MATLAB 2010a (student edition) applications have been used for the orbit planning programs and the arm has been successfully operated. Using the program developed, the coordinates of the starting and ending positions for the robotic arm have been specified on a three-dimensional Cartesian coordinate system and inverse kinematics computations have been performed, resulting in the movement of the arm.

Keywords: robotic arm, orbit planning, inverse kinematics computation.

1. Introduction

In this study, the control of a five-axis robotic arm, including computation of its orbit, has been achieved. The MATLAB 2010a (student edition) and LabVIEW 2010 (student edition) applications have been used to perform the computations. A PC expansion card, manufactured by JS Automation (part number AIO-3320) and having LabVIEW support, has been used to communicate with the robotic arm through the PCI port. The values obtained through the inverse kinematics computations performed using the LabVIEW and MATLAB applications have been relayed to the robotic arm over the AIO-3320 expansion card, using LabVIEW [1].

2. Problem statement

The analytical calculation method has been chosen for use when performing forward and inverses kinematics computations. The robotic arm is rotated using servo motors. LabVIEW, a graphical programming environment, has been used as the programming language. The LabVIEW system was chosen for its capability for parallel computation. It has been possible to simultaneously operate more than one servo motor with this parallel operation capability. Additionally, LabVIEW provides support to run MATLAB scripts and functions. (This requires MATLAB to be installed on the PC along with LabVIEW.)

MATLAB has been used for analytical calculations, for matrix operations (e.g. matrix multiplication, finding inverses), and for solving non-linear equations.

2.1. Forward kinematics computations for the robotic arm

The Denavit-Hartenberg method has been used to obtain the forward kinematics models. As shown in Fig. 2, joint variables and constants are defined using the Denavit-Hartenberg method and the coordinate systems are placed over the joints. [2]
The Denavit-Hartenberg variables are shown in Table 1. The parameters which do not change as a result of the motion of the robot are the length of the arm $a_{i-1}$, and the angles of the axis $\alpha_{i-1}$. The parameters that vary are the joint angle $\theta_i$ (in case of a rotating joint), and the joint offset $d_i$ (in case of a prismatic joint) [3].

The homogenous conversion matrix for the arm is shown below, where $i$ is the axis number.

$$s_i = \sin \theta_i \quad \text{and} \quad c_i = \cos \theta_i \quad \theta_i \quad \text{is the rotation angle and:}$$

$$^0T_5 = \begin{bmatrix} ^0R_5 & ^0p_5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$  \hspace{1cm} (1)

$^0R_5$ is the rotation matrix and $^0p_5$ is the position vector. The transformation matrices for the first and the second joint are:

$$^0T_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad ^1T_2 = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$  \hspace{1cm} (2)
The transformation matrix for the third, fourth, and fifth joints is:

\[
\begin{bmatrix}
    c_i & -s_i & 0 & l & \\
    s_i & c_i & 0 & 0 & \\
    0 & 0 & 1 & 0 & \\
    0 & 0 & 0 & 1 & 
\end{bmatrix}
\]

(3)

The five transformation matrices that have been computed are multiplied to obtain the rotational matrix for the arm:

\[
^0T_5 = ^0T_1^{-1} ^1T_2 ^2T_3 ^3T_4 ^4T_5.
\]

(4)

The transformation matrices obtained through Eq. (1) and (4) are equivalent.

3. Result and discussion

We will use forward kinematics Eqs. (1) and (4) for computations. Eq. (1) and (4) are equal to each other. By multiplying each side of the Eq. (4) with the inverse of \(^0T_1\), we obtain the Eq. (5). Since \(^0T_1^{-1} ^0T_5 = ^0T_1^{-1} ^0T_1 ^1T_2 ^2T_3 ^3T_4 ^4T_5\) and \(^i^{-1}T_i^{-1} \cdot ^i^{-1}T_i = I\), it follows that:

\[
^0T_1^{-1} ^0T_5 = ^1T_2 ^2T_3 ^3T_4 ^4T_5.
\]

(5)

Let’s find the inverse matrices for all transformational matrices in Eq. (2) and Eq. (3), using the inv() function in MATLAB [4]:

\[
^i^{-1}T_i^{-1} = inv(^i^{-1}T_i).
\]

(6)

The position vectors located within Eq. (5) are mutually equalized and the following equations are then obtained:

\[
\frac{p_x c_1 + p_y s_1}{l} = (c_2 c_3 - s_2 s_3) + c_2 + (c_4 (c_2 c_3 - s_2 s_3) - s_4 (c_2 s_3 + c_3 s_2)),
\]

(7)

\[
\theta_1 = \text{atan} \left( \frac{p_y}{p_x} \right),
\]

(8)

\[
\frac{p_x - h}{l} = (c_2 s_3 + c_3 s_2) + s_2 + (c_4 (c_2 s_3 + c_3 s_2) + s_4 (c_2 c_3 - s_2 s_3)).
\]

(9)

Let’s label the left-hand side of Eq. (7) as \(A\):

\[
\frac{p_x c_1 + p_y s_1}{l} = A.
\]

(10)

Similarly, let’s label the left-hand side of Eq. (9) as \(B\):

\[
\frac{p_x - h}{l} = B.
\]

(11)

Each side of Eq. (5) is multiplied with \(^i^{-1}T_i^{-1}\):

\[
^1T_2^{-1} \left( ^0T_1^{-1} ^0T_5 \right) = ^3T_2^{-1} \left( ^1T_2 ^2T_3 ^3T_4 ^4T_5 \right),
\]

\[
^1T_2^{-1} ^0T_1^{-1} ^0T_5 = ^2T_3 ^3T_4 ^4T_5.
\]

(12)

In Eq. (12), the position vectors are equalized and the equations below are obtained:
\[ s_2B + c_2A = 1 + (c_3c_4 - s_3s_4) + c_3, \quad (13) \]
\[ c_2B - s_2A = (c_3s_4 + c_4s_3) + s_3. \quad (14) \]

Each side of Eq. (12) is multiplied with \( T_{3}^{-1} \):
\[
\begin{align*}
&2T_{3}^{-1} \begin{pmatrix} T_{2}^{-1} & 0 & T_{1}^{-1} & 0 \end{pmatrix} T_{5} = 2T_{3}^{-1} 2T_{3} 3T_{4} 4T_{5}, \\
&2T_{3}^{-1} \left( \begin{pmatrix} T_{2}^{-1} & 0 & T_{1}^{-1} & 0 \end{pmatrix} T_{5} \right) = 3T_{4} 4T_{5}. \quad (15)
\end{align*}
\]

The position vectors in Eq. (15) are equalized, and the following equations are obtained:
\[
\begin{align*}
& (c_2c_3 + c_3s_2)B - c_3 + (c_2c_3 - s_2s_3)A = 1 + c_4, \quad (16) \\
& (c_2c_3 - s_2s_3)B + s_3 - (c_2s_3 + c_3s_2)A = s_4. \quad (17)
\end{align*}
\]

In the inverse kinematics solution, the angle \( \theta_1 \) is solved from Eq. (8). The angles \( \theta_2, \theta_3 \) and \( \theta_4 \), on the other hand, are obtained through iterations. For the three unknowns, Eqs. (13, 14, 17) have been selected. The \( \theta_4 \) the angle indicates the open and close states of the end effector. This angle does not affect the position vector but affects only the rotation matrix.

3.1. Using Matlab to calculate the inverse kinematics

The fsolve() function of MATLAB was employed for the solution. This function solves nonlinear equations through iteration. We had solved for angle \( \theta_1 \) using Eq. (8). To find the angles \( \theta_2, \theta_3 \) and \( \theta_4 \), we need three equations, for which Eqs. (13, 14, 17) have been chosen. The starting matrix for the iteration is \( x_0 = [0 \ 0 \ 0] \).

In the MATLAB function shown below, \( x(1), x(2), \) and \( x(3) \) designate the unknown angles, corresponding to \( \theta_2, \theta_3 \) and \( \theta_4 \), respectively. The equations have been implemented as the MATLAB function below:
\[
\begin{align*}
[x, fval, exitflag] &= \text{fsolve}(@(x) \begin{pmatrix} \cos(x(1)) \sin(x(1)) \cos(x(2)) \cos(x(3)) + \sin(x(2)) \sin(x(3)) - \cos(x(2)) - \sin(x(2)) \end{pmatrix}, x0) ;
\end{align*}
\]

The function above, \( x \) designates the unknown vector, the FIDELITY VALUE FACTOR ETF (Fval) is an error, and the exit flag indicates whether or not a result has been obtained [5, 6].

3.2. Programming code

The student edition of LabVIEW has been used for programming. LabVIEW is a graphical programming environment. The program code below shows the inverse kinematics computations for orbit planning.

3.2.1. Orbit planning using LabVIEW

The code moves the robotic arm from its starting position coordinate of \( (X_1, Y_1, Z_1) \) to its ending position coordinate of \( (X_2, Y_2, Z_2) \). Fig. 3 shows the corresponding user interface.

3.2.1.1. Outline for the orbit computation algorithm

1) Using the inverse-kinematic.vi function, compute the angles necessary for the starting position coordinate (Fig. 4).
2) Using inverse-kinematic3.vi, correct the computed angles.
3) Using the inverse-kinematic.vi function, compute the angles necessary for the ending position coordinate.
4) Using inverse-kinematic3.vi, correct the computed angles.
5) If steps 1, 2, 3 and 4 produce a solution and if the “Move arm along the orbit” box has been clicked:
   a) Find the difference between t1A and t1B, convert it into the digital value required by the robotic arm, and bind the value to the “Difference of t1A and t1B” indicator.
   b) Bind the state where t1A is larger than t1B to the “t1A > t1B” indicator.
   c) Repeat steps a and b for remaining angles.
6) Illustrate a for loop running from 0 to 16384.
7) Within the for loop:
   a) Use i, the loop variable, as the numeric value for the angle of the robotic arm.
   b) Determine angle value as i_say = i_say + “rate of change of the angle”.
   c) If the “t1A > t1B” indicator value is “true”, increment the numerical value of the t1A angle by the value of i_say; if “false”, decrement it by the value of i_say.
   d) Convert the value in step c into degrees and bind it to the t1Y control.
   e) Repeat steps a, b, c, d for remaining angles.
   f) When the “t1A > t1B” value and the corresponding values for the remaining angles are less than the value of i_say, or when the “arm is moving” indicator light is off, exit the loop.
   g) Move the arm to the position indicated by t1Y and the remaining angles.

| Name of function          | Symbol | Description                                                                 |
|---------------------------|--------|----------------------------------------------------------------------------|
| T01.vi                    | ![T01.vi](image) | Creates transformation matrix using height (-h), arm length (-L), and angle values. |
| forward-kinematic.vi     | ![forward-kinematic.vi](image) | Creates transformation matrices from the start to the end positions, and vectors PX, PY and PZ, based on the transformation matrix. |
| inverse-kinematic.vi     | ![inverse-kinematic.vi](image) | Computes the angles t2, t3 and t4 as well as other items from coordinates X, Y, and Z. |
| inverse-kinematic3.vi    | ![inverse-kinematic3.vi](image) | Moves angles computed by the inverse-kinematic.vi function into arm boundaries. |
| aio3320_signal_transformation.vi | ![aio3320_signal_transformation.vi](image) | Converts an angle value in degrees to analog/digital signals or converts analog/digital signals into an angle value in degrees. |

Fig. 3. The user interface showing the “Orbit” tab
3.2.1.2. LabVIEW diagrams for the “Orbit” tab

Fig. 5 shows the calculation of the starting and ending angles using the “inversekinematic.vi” function. The output contains indicators t1A, t1B, t2A1, t2B1, t3A1, t3B1, t4A1, and t4B1.

Fig. 4. The LabVIEW code for “inversekinematic.vi” making use of MATLAB

Fig. 5. Calculation of the starting and ending angles using the inversekinematic.vi function

Fig. 6. Correction of the starting and ending angles
Figs. 6 show corrections applied to the angles computed in Fig. 5, using the inversekinematic3.vi function. Angle $t_2A_1$ is transformed to $t_2A$, angle $t_3A_1$ to $t_3A$, and angle $t_4A_1$ to $t_4A$; the same applies to $t_2B$, $t_3B$, and $t_4B$. For the computation of the orbit, the angle $\theta_1$ of the robotic arm will change from $t_1A$ to $t_1B$, the angle $\theta_2$ from $t_2A$ to $t_2B$, the angle $\theta_3$ from $t_3A$ to $t_3B$, and angle $\theta_4$ from $t_4A$ to $t_4B$. The change of the angles from A to B is accomplished using a for loop shown in Fig. 7.

The for-loop shown in Fig. 7 is located in a case structure having the conditions shown in Fig. 8. To move the arm, the angles of the positions A and B must be calculated, the “No Solution” indicator light must be off, and the “Move arm along the orbit” button must be clicked.
Fig. 9 shows the differences between the angles for position A and the angles for position B having been converted into the digital values required by the arm. Additionally, for each angle, a boolean indicator has been used to indicate whether the movement from A to B will be in increasing or decreasing terms. If angle A is larger than angle B, this indicator is set to true; otherwise, the indicator is set to false.

3.2.1.2.1. Diagrams within the for loop

The transition of the arm from position A to position B is accomplished using a for loop. For the angles to get closer to position B from position A, on each iteration through the loop, the angles are adjusted by the value contained in i_say. The for loop variable i is at value zero at the start of the for a loop. For each iteration of the loop, its value is incremented by 1. It runs if the condition of the case structure shown in Fig. 10 is “true”. It sets the i_say indicator value to zero. In Fig. 10, the value contained within “The angle rate of change” control is added to the value of i_say, and its output is assigned to i_say.

![Diagram 10](image)

**Fig. 10.** Definition of angle increment and decrement values within the for loop

In the program code shown in Fig. 11, the value of the i_say indicator is continuously incremented. When the value of i_say is larger than the difference of the angles for position A and position B, the loop is exited. This means that the arm has arrived at position B.

We will now describe the block diagrams shown in Fig. 12. The condition for the case structure shown in (1) is the “t1A > t1B” indicator. If this value is “true”, the approach from the t1A value to t1B value is in increments of i_say; otherwise, if the value is “false”, the approach is in decrements. The values are converted to degrees using mathematical processing at the output the of case (1). This value goes to t1Y value until it arrives at t1B value. When it reaches t1B, it goes to the value of t1Y. Fig. 12(a) and 12(b), and Fig. 12(c) and 12(d) are sequential. A Similar method is applied for angles t2, t3, and t4.

![Diagram 11](image)

**Fig. 11.** Conditions for an early exit from the for loop
4. Conclusions

Forward and inverse kinematics computations have been performed using LabVIEW and MATLAB. Transformation matrices and their multiplication have been employed in the computations. Using the software developed, the required end coordinate \((X, Y, Z)\) for the robotic arm is provided. The study has been implemented on a robotic arm, and it has been observed that the arm has moved to the position indicated by the \(x, y, \) and \(z\) values entered. Following the entry of two coordinates into the system, the arm has first located to the starting position and then moved to the ending position. Throughout the movement of the arm along the orbit, the current coordinate computed by the software is displayed by the user interface.

The study has met its objective. Using the software, the movement of the end effector between two positions has been accomplished. With minor modifications in the software, the arm can be made to visit more than one position along the orbit.

During orbit planning, it has been assumed that no obstacles exist which would affect the orbit.

References

[1] JS Automation, ftp://automation.com.tw/download/Manual_Driver/AioCard/AIO3320_1A/manual/aio3320ae.pdf.
[2] Rocha C. R., Tonetto C. P., Dias A. A comparison between the Denavit–Hartenberg and the screw-based methods used in kinematic modeling of robot manipulators, Robotics and Computer-Integrated Manufacturing, 2010.
[3] Çetinkaya Ö. Design of Robot Arm with Four Rotary Joints Tracing Movement of One Arm and Experimental Research. Trakya University, Edirne, 2009, (in Turkish).
[4] Matlab, https://www.mathworks.com/help/matlab/ref/inv.html;jsessionid=1472af4d759e6358471f61a0b797.
[5] Matlab, https://www.mathworks.com/help/optim/ug/fsolve.html.
[6] Kilicaslan K. Orbit planning of 5 axis robotic arm and experimental application (in Turkish), Trakya University PhD Seminar, p.50-52, 60-66 2011
[7] LabVIEW, http://zone.ni.com/reference/en-XX/help/371361G-01/lvconcepts/for_loop_and_while_loop_structures/.
[8] LabVIEW, http://zone.ni.com/reference/en-XX/help/371361G-01/lvconcepts/case_and_sequence_structures/.
Özcan Çetinkaya received Ph.D. degree in Institute of Science from Trakya University, Edirne, Turkey, in 2017. Now he works as a Asst. Dr. at Keşan Vocational School, Trakya University, Edirne.

Hilmi Kuşçu received Ph.D. degree in Institute of Science from Trakya University, Edirne, Turkey, in 2002. Now he works as a Prof. Dr. at Faculty of Engineering, Department of Mechanical Engineering, Trakya University, Edirne.

Kenan Kiliçaslan is currently continuing his Ph.D. education. He is currently working as an engineer in the provincial special administration of Giresun governorship.