An Off-Line Robot Simulation Toolbox

SERDAR KUCUK,1 ZAFER BINGUL2

1Department of Electronics and Computer Education, University of Kocaeli, Umuttepe Campus, Kocaeli 41380, Turkey
2Department of Mechatronics Engineering, University of Kocaeli, Umuttepe Campus, Kocaeli, Turkey

Received 14 September 2007; accepted 25 December 2007

ABSTRACT: Robotics has gained popularity in education so many engineering schools are offering robotics courses. In this study, a novel robot toolbox, “ROBOLAB” is developed for the educational users to improve the understanding of robotics fundamentals through interactive real-time simulation. To understand manipulator movement in 3-D space with increasing number of joints is very difficult for engineering students; because the mathematical model between joint space and physical space becomes more complex. In order to overcome this complexity, ROBOLAB based on MATLAB Graphical User Interface (GUI) includes a library for the 16 different 6 degree of freedom (6-DOF) fundamental serial robot manipulators. The user has option to view an animation of the robot manipulators with choosing one of them or to create own GUIs based on robot project. ROBOLAB provides valuable analysis tools to students and engineering professionals in which they can compute rotation and transformation matrix, forward and inverse kinematics, and trajectory planning. Additionally, it allows user to use powerful MATLAB features such as controlling over data and formatting. In order to illustrate the features of ROBOLAB, RS cylindrical robot manipulator is given here as an example. Usage of real industrial robots in laboratory may be a very expensive way to teach robotics courses. ROBOLAB may help students to accomplish the courses and projects involving robots with minimum cost and to see 3-D space robot applications more effectively © 2009 Wiley Periodicals, Inc. Comput Appl Eng Educ 18: 41–52, 2010; Published online in Wiley InterScience (www.interscience.wiley.com); DOI 10.1002/cae.20236

Keywords: GUI, fundamental robot manipulators, off-line robot programming, robot toolbox

INTRODUCTION

Robot programming is a very critical task in robotics education since it is a bridge between robotics theory and physical implementation. There are basically two types of robot programming, namely, on-line, and off-line. Some programmers prefer on-line programming because their system is appropriate to the programming interface. Others prefer off-line programming to see the results before manipulating the robot. Off-line programming is that the mechanical robot and other equipment are not occupied during the programming; everything takes place on a host computer. The main
advantage of off-line programming is that a production cell can be designed, programmed, and its operations may be simulated before the cell is actually built. One can determine which type of robot that should be used, or how the equipment in the cell should be arranged according to the simulation results.

Recently, rather then using tedious keyboard-entered commands, users desire to point the mouse to some part of a graphical representation of an application in order to invoke the events easily. GUI has become a standard language for engineering computation, analysis, and visualization. It involves universal idea of icons, buttons, sliders, etc., which means for the “front-end” of a software application [1]. Major features of a powerful GUI are automating a function for using many times and creating an interactive demonstration. The main advantage of GUI is that one can communicate with the computer without using any programming command.

Until now, several robot software tools have been developed with different capabilities and properties. Das et al. [2] made a software package called Robot Computer Aided Analysis and Design (RCAAD) which assists designers of robot arms for planetary rovers and landers to analyze and optimize their designs. Although it has a very powerful GUI, it is not programmed for educational purposes and is not covered theoretical background of robotics. The Robotics Toolbox for Matlab [3] provides many useful subroutines written for robotics education and addresses topics such as kinematics, dynamics and trajectory generation. Although serial-link manipulator can be created by the user, a limited number of robot examples are given such as the PUMA and Stanford arm. All inputs are given by command line and its GUI is not satisfactory because of limited number of facilities provided. Nethery and Spong [4] presented Robotica based on Mathematica Software. This program encapsulates over 30 functions for computing kinematics and dynamics, and animating robots. Although the front end of the software is completely independent from the actual Mathematica source code, it is necessary to set up Mathematica before loading Robotica. Robotica does not permit interactive GUI facilities for the users. Hill [5] developed a stand alone PC-based integrated robot modeling and analysis package which is called Rapid Analysis Manipulator Program (RAMP). This program is not also suitable for students. Nayar [6] developed Windows-based graphical user interface robot package (Robotect) to model and analyze manipulator designs. Although it is stated that it can be used as educational tool for robot manipulator designs, it does not have a library including fundamental robot manipulators and has limited educational functions. Zlajpah [7] presented a toolbox for dynamic simulation of redundant planar manipulators only. This tool is not an educational tool and does not provide GUI. ROBOOP [8] is a C++ object oriented programming robot toolbox for synthesis and simulation of robotic manipulator models. ROBOOP has similar features of Corke’s toolbox [3]. Bingul et al. [9] developed windows-based robot simulation tool called RoboSim for modeling, visualization and performance analysis of serial-link manipulators. A built in LISP interpreter based on common LISP allows a user to model and manipulate the robot while the results is displayed on a three dimensional graphical display. In order to animate the robot manipulators, knowing common LISP programming language is needed. It is a big disadvantage of this simulation program. Turnell et al. [10] made a robot simulation program called SimBot for helping the development and teaching of autonomous robots. The SimBot tool allows students to experiment with robot programming and to get involved with the development of the simulator’s components. The purpose of SimBot is to create scenarios containing multiple autonomous robots rather then teaching basic robot fundamentals. Vollmann [11] presented KUKA’s new range of simulation tools based on visual components. This simulation tool does not support GUI facilities and interactive learning. Alfs et al. [12] presented a simulation tool for designing and testing control devices for advanced robots with an arbitrary number of redundant joints only. Although the simulation tool supports GUI, it is not an educational tool. Cakir and Butun [13] developed an educational tool for 6-DOF industrial robots with quaternion algebra. This program is written for serial robots with revolute joints only and it has limited GUI operations. In order to compare the robot programs mentioned above, Table I summarizes some important properties of these simulation programs.

In this paper, an off-line robot simulation toolbox [14] is developed using powerful GUI facilities and incorporating MATLAB functions. The main advantages of ROBOLAB over the other robot toolboxes mentioned above are given as follows.

1. ROBOLAB helps students to visualize the fundamental robotic theory such as the forward and inverse kinematics, and trajectory planning in more effective way.
2. The toolbox has a broad robot library including the 16 different 6-DOF robot manipulators with Euler wrist (well known robot manipulators
such as Prismatic, Scara, Puma, and Stanford). Animating these robot manipulators allows students to learn their operation limits without occupying the real robot and other equipments.

3. The toolbox illustrates real configurations with solid models of robot manipulators. Students have opportunity to change the related robot parameters by either sliders or edit boxes and to see interactively the animation of the solid model.

4. ROBOLAB serves students as a powerful platform to improve their ability of understanding the orientation and positioning in three dimensional spaces.

Based on our 10-year robotic teaching experience, the following statements about the toolbox can be reached. ROBOLAB improved student’s problem solving skills in 3-D space and made them to cope with much more sophisticated robotics problems themselves. It also enhanced student’s motivation to robotics courses and projects. Their interests to graduate study in the field of robotics increased. The interactive environment of ROBOLAB provided students to establish relationship between the theoretical knowledge and physical robot applications, and to save a lot of time analyzing robotic problems. It decreased the student’s frustration about robotics problems. Ultimately, it gave the students greater insight about robotics without knowing complicated mathematical background.

The paper is organized as the following manner. In Robot Configurations Section, the library of the robot configurations are presented. The programming background is described in Background Section. In Overview of the Robot Toolbox Section, the features of the ROBOLAB are explained with an example. Finally, conclusions of this study are presented.

### ROBOT CONFIGURATIONS

Huang and Milenkovic [15] used a two-letter code to classify 3-DOF robot configurations. The first letter characterizes the first joint and the first joint’s relationship to the second joint. The second letter identifies the third joint and third joint’s association to the second joint. The code letters and their meanings are: S is slider, C is rotary parallel to slider, N is rotary perpendicular to rotary and R is rotary perpendicular to rotary or rotary parallel to rotary. The combination of these rotary and prismatic joints compose the 16 robot configurations which are named as SS, SC, SN, CS, CC, CR, NS, NN, NR, RC, RN, RR, RS, SR,
CN, and NC. In order to have 6-DOF robot manipulators, each robot structure is equipped with Euler wrist.

**BACKGROUND**

In this section, forward and inverse kinematics, three different rotation representations and trajectory planning are explained in detail to have fundamental knowledge on robotics.

### Forward Kinematics

Calculating the position and orientation of the end-effector in terms of the joint variables is called forward kinematics. To obtain the forward kinematics of a robot manipulator, one should define the homogeneous transformation matrix for each joint. Using D-H [16] parameters, the homogeneous transformation matrix for a single joint is expressed as,

\[
i^{-1}T = \begin{bmatrix}
c_{\theta_i} & -s_{\theta_i} c_{\alpha_{i-1}} & 0 & a_{i-1} \\
s_{\theta_i} c_{\alpha_{i-1}} & c_{\theta_i} c_{\alpha_{i-1}} & -s_{\alpha_{i-1}} & -s_{\alpha_{i-1}} d_i \\
s_{\theta_i} s_{\alpha_{i-1}} & c_{\theta_i} s_{\alpha_{i-1}} & c_{\alpha_{i-1}} & c_{\alpha_{i-1}} d_i \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

where \(a_{i-1}, s_{i-1}, d_i, 0_i, c_0, s_0\) and \(s_\theta\) are the link length, link twist, link offset, joint angle, \(c_\theta\) and \(s_\theta\), respectively. In this way, the forward kinematics of the end-effector with respect to the base frame is obtained by multiplying all of the \(i^{-1}T\) matrices.

\[
\text{end-effector}^{\text{base}} T = 0^T_1 T_2 \ldots n^{-1} T
\]

An alternative representation of \(\text{end-effector}^{\text{base}} T\) can be written as,

\[
\text{end-effector}^{\text{base}} T = \begin{bmatrix}
\vec{n} & \vec{s} & \vec{a} & \vec{p} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

where, \(\vec{n}, \vec{s}, \vec{a}\) and \(\vec{p}\) are the normal, sliding, approaching and position vectors, respectively. Using Equations (2) or (3), one can determine the position and orientation of the end-effector in terms of the joint variables.

\[
\text{end-effector}^{\text{base}} T = \begin{bmatrix}
r_{11} & r_{12} & r_{13} & p_x \\
r_{21} & r_{22} & r_{23} & p_y \\
r_{31} & r_{32} & r_{33} & p_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

where \(r_{kj}\)'s represent the rotational elements of transformation matrix \((k \text{ and } j \text{ } = 1, 2, \text{ and } 3)\). \(p_x, p_y, \text{ and } p_z\) are the elements of position vector. For a six jointed robot manipulator, the position and orientation of the end-effector with respect to the base is given by,

\[
0^T_6 = 0^T(q_1)1^2 T(q_2)3^4 T(q_3)4^5 T(q_5)6^5 T(q_6)
\]

where \(q_i\) is the joint variable (revolute joint or prismatic joint) for joint \(i (i = 1, 2, \ldots, 6)\).

### Inverse Kinematics

Forward kinematics problem is straightforward and there is no complexity deriving the equations. Hence, there is always a forward kinematics solution for different robot manipulators with different structures. Whereas actuators work in joint space, tasks to be performed by a robot manipulator are in Cartesian space. Cartesian space includes orientation matrix and position vector but joint space is represented by joint angles. The conversion of the position and orientation of a robot end-effector from Cartesian space to joint space is called as inverse kinematics problem. In order to find the inverse kinematics solutions as a function of the known elements of \(\text{base}^{\text{end-effector}} T\), the inverses of link transformation matrix are premultiplied as follows.

\[
[0^T(q_1)]^{-10} T_6 = [0^T(q_1)]^{-10} T_1 2 T_2 3 T(q_2) 4 T(q_3) 5 T(q_5) 6 T(q_6)
\]

where \([0^T(q_1)]^{-10} T(q_1) = I, I\) is identity matrix. Then Equation (6) can be simplified as follows.

\[
[0^T(q_1)]^{-10} T_6 = 2^T(q_2) 3^T(q_3) 4^T(q_4) 5^T(q_5) 6^T(q_6)
\]

If necessary, the following equations are also obtained in a similar manner.

\[
[0^T(q_1)]^4 T(q_2)]^{-10} T_6 = 3^T(q_3) 4^T(q_4) 5^T(q_5) 6^T(q_6)
\]

\[
[0^T(q_1)]^2 T(q_2) 3^T(q_3)]^{-10} T_6 = 4^T(q_4) 5^T(q_5) 6^T(q_6)
\]

\[
[0^T(q_1)]^2 T(q_2) 3^T(q_3) 4^T(q_4)]^{-10} T_6 = 5^T(q_5) 6^T(q_6)
\]

\[
[0^T(q_1)]^2 T(q_2) 3^T(q_3) 4^T(q_4) 5^T(q_5)]^{-10} T_6 = 6^T(q_6)
\]

Twelve simultaneous set of nonlinear equations needs to be solved for inverse kinematics. The only unknown on the left hand side of Equation (7) is \(q_1\). The 12 nonlinear matrix elements of right hand side are either zero, constant or functions of \(q_2\) through \(q_6\).
If the elements on the left hand side are equated with elements on the right hand side then the joint variable $q_1$ can be solved as functions of $r_{11}, r_{12}, \ldots r_{33}, p_\alpha, p_\beta, p_\gamma$ and fixed link parameters. Once $q_1$ is found, then other joint variables are solved by the same procedure as before. There is no necessity that first equation will produce $q_1$ and second $q_2$ etc. To find suitable equation to solve the inverse kinematics problem, any equation defined above (Eqs. 7–11) can be used arbitrarily. A set of transcendental equations [17] are also used for extracting the joint angles.

### Rotation Representations

There are three common ways of performing orientation, namely, roll-pitch-yaw angles, Euler angles, and the equivalent angle-axis representation.

#### 1. Roll-Pitch-Yaw Angles

Roll, pitch, yaw are nautical terms used to describe the motion of a boat with respect to its three axes. There are twelve roll-pitch-yaw angle sets that may be used for specifying orientation. Each of these requires performing three rotations about principal axes of the fixed reference frame. In this simulation program, orientation. Equating $R_{XYZ}$ with a rotation matrix giving desired orientation, $\alpha, \beta$, and $\gamma$ angles can be found as follows.

$$R_{XYZ}(\gamma, \beta, \alpha) = R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

(13)

where $R$ is the $3 \times 3$ rotation matrix. The fixed angles are determined using inverse kinematics equations.

$$\beta = A \tan 2 \left( -r_{31}, \sqrt{r_{11}^2 + r_{21}^2} \right)$$

(14)

$$\alpha = A \tan 2 \left( \frac{r_{21}}{c\beta}, c\beta \right), \beta \neq \pm 90^\circ$$

(15)

$$\gamma = A \tan 2 \left( \frac{r_{32}}{c\beta}, c\beta \right), \beta \neq \pm 90^\circ$$

(16)

#### 2. Euler Angles

Another way of describing the orientation is to use Euler angle sets. With Euler angles, each rotation is performed about an axis of the moving frame. There are 12 possible permutations of orientations as in fixed angle sets. In this toolbox, ZYX-Euler angle set given by Equation (17) is used for performing the orientation.

$$R_{Z-Y-X}(\alpha, \beta, \gamma) = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}$$

(17)

XYZ-fixed angle set given by Equation (12) is used for performing the orientation.

$$R_{XYZ}(\gamma, \beta, \alpha) = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}$$

(12)

where $c\alpha$ and $s\alpha$ are the abbreviation for cos $\alpha$ and sin $\alpha$, respectively. Extracting $\alpha, \beta$ and $\gamma$ fixed angles from a rotation matrix is necessary for having suitable Comparing the Equation (17) with XYZ-fixed angle set, it can be seen that the results are identical. Hence,
the Euler angles are determined as the way the fixed angles are found.

3. Equivalent Angle-Axis

A general orientation of a coordinate frame relative to another one can be defined $R_K(\theta)$ that is named as equivalent angle-axis representation. $K$ is a vector whose length is taken to be one and $\theta$ represents the amount of rotation. Given an arbitrary rotation $R$ that is equated to $R_K(\theta)$ to solve for $K$ and $\theta$ as a function of the elements of $R$.

$$R = R_K(\theta)$$

where $k_x, k_y, k_z$ are the elements of vector $K$ and $\nu \theta = (1 - c \theta)$. $K$ and $\theta$ are found by using inverse kinematics, as follows.

$$\theta = A \tan 2 \left( \sqrt{(r_{32} - r_{23})^2 + (r_{13} - r_{31})^2 + (r_{21} - r_{12})^2} \right)$$

$$K = \frac{1}{2s \theta} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} (\theta \neq 0 \text{ or } 180^\circ)$$

**Trajectory Planning**

Rough and jerky motions can cause vibration in the manipulator. Therefore, smooth motion is very essential for robot manipulators. Joint and Cartesian trajectories are two common ways to generate smooth motion.

In joint trajectory, initial and final positions of the end-effector are converted into joint angles by using inverse kinematics equations. A time dependent smooth function is computed for each joint. All of the robot joints pass through initial and final points at the same time. Several smooth functions can be obtained from interpolating the joint values. In this toolbox, a 7th order polynomial is used with boundary conditions for position, velocity, acceleration, and jerk. The 7th order polynomial is given as follows

$$y = s_0 + s_1t + s_2t^2 + s_3t^3 + s_4t^4 + s_5t^5 + s_6t^6 + s_7t^7$$

The desired velocity, acceleration and jerk are calculated as

$$y' = s_1 + 2s_2t + 3s_3t^2 + 4s_4t^3 + 5s_5t^4 + 6s_6t^5 + 7s_7t^6$$

$$y'' = 2s_2 + 6s_3t + 12s_4t^2 + 20s_5t^3 + 30s_6t^4 + 42s_7t^5$$

$$y''' = 6s_3 + 24s_4t + 60s_5t^2 + 120s_6t^3 + 210s_7t^4$$

Combining the Equations (21–24) with eight constrains yields eight equations in eight unknowns.

$$q_0 = s_0 + s_1t_0 + s_2t_0^2 + s_3t_0^3 + s_4t_0^4 + s_5t_0^5 + s_6t_0^6 + s_7t_0^7$$

$$q_1 = s_0 + s_1t_0 + s_2t_0^2 + s_3t_0^3 + s_4t_0^4 + s_5t_0^5 + s_6t_0^6 + s_7t_0^7$$

$$q_0' = s_1 + 2s_2t_0 + 3s_3t_0^2 + 4s_4t_0^3 + 5s_5t_0^4 + 6s_6t_0^5 + 7s_7t_0^6$$

$$q_1' = s_1 + 2s_2t_0 + 3s_3t_0^2 + 4s_4t_0^3 + 5s_5t_0^4 + 6s_6t_0^5 + 7s_7t_0^6$$

$$q_0'' = 2s_2 + 6s_3t_0 + 12s_4t_0^2 + 20s_5t_0^3 + 30s_6t_0^4 + 42s_7t_0^5$$

$$q_1'' = 2s_2 + 6s_3t_0 + 12s_4t_0^2 + 20s_5t_0^3 + 30s_6t_0^4 + 42s_7t_0^5$$

**Figure 1** Linear segments with parabolic blends.
The Equations of the Position, Velocity, and Acceleration

\[ q''_1 = 2s_2 + 6s_3 t_t + 12s_4 t_t^2 + 20s_5 t_t^3 + 30s_6 t_t^4 + 42s_7 t_t^5 \]

(30)

\[ q'^{0}_0 = 6s_3 + 24s_4 t_0 + 60s_5 t_0^2 + 120s_6 t_0^3 + 210s_7 t_0^4 \]

(31)

\[ q''^{0}_1 = 6s_3 + 24s_4 t_1 + 60s_5 t_1^2 + 120s_6 t_1^3 + 210s_7 t_1^4 \]

(32)

These eight equations can be combined into a single matrix equation as follows

\[
\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 & t_0^6 & t_0^7 & s_0 \\
1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 & t_f^6 & t_f^7 & s_f \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 & 6t_0^5 & 7t_0^6 & s_t \\
0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 & 6t_f^5 & 7t_f^6 & s_t \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 & 30t_0^4 & 42t_0^5 & 60t_0^6 & 120t_0^7 & s_1 \\
0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 & 30t_f^4 & 42t_f^5 & 60t_f^6 & 120t_f^7 & s_7 \\
0 & 0 & 0 & 6 & 24t_0 & 60t_0^2 & 120t_0^3 & 210t_0^4 & 360t_0^5 & 540t_0^6 & 720t_0^7 \\
0 & 0 & 0 & 6 & 24t_f & 60t_f^2 & 120t_f^3 & 210t_f^4 & 360t_f^5 & 540t_f^6 & 720t_f^7 \\
\end{bmatrix}
\begin{bmatrix}
s_0 \\
s_f \\
s_t \\
s_t \\
s_1 \\
s_7 \\
s_2 \\
s_6 \\
s_3 \\
s_5 \\
\end{bmatrix}
= 
\begin{bmatrix}
s_0 \\
s_f \\
s_t \\
s_t \\
s_1 \\
s_7 \\
s_2 \\
s_6 \\
s_3 \\
s_5 \\
\end{bmatrix}
\]

(33)

The coefficients \((s_0, s_1, ..., s_7)\) of the polynomial is determined by solving this matrix equality.

In Cartesian trajectory, the end-effector is constrained to move along a prescribed path (typically a straight line) between via points. Inverse kinematics is required in real-time at each update cycle. Therefore, Cartesian motion is much more computationally intensive. To make smooth path with continuous position and velocity, two parabolic blends are added to the beginning and end of the motion. This path is also called linear segments with parabolic blends. Figure 1 illustrates a constructed path with a linear segment at the medium and two parabolic blends at the beginning and end. In Figure 1, \(a, v, t_0\), denote acceleration, velocity, and blend time, respectively.

Table II illustrates the position, velocity, and acceleration equations of the path shown in Figure 1 [18].

The assumptions of \(t_0 = 0, t_f > 0\) and \(t_b > 0\), and the constraints of \(\theta_f - \theta_0) / v < t_b \leq 2(\theta_f - \theta_0) / v, v^2 \leq a(\theta_f - 0)\) can be satisfied to have a smooth path.

To illustrate the features of the ROBOLAB, the RS robot manipulator in Figure 4 is presented as example. It can be chosen from the robot library which will appear on the screen after clicking RMEW button.

**Forward Kinematics-I**

ROBOLAB can be run in forward kinematics-I mode by clicking “F.Kin-I” button. In this mode, each angle of the robot joints is controlled by a slider. The numbers on the sliders show the joint limitations. The user moves the indicator bar on the slider to specify a desired value within the allowable joint ranges. Let revolute joints move with the angles, for example, \(T_1 = 123.84°, T_2 = -48.06°, T_4 = 77.04°, T_5 = 64.8°\) and \(T_6 = 0\), prismatic joint moves with certain unit length value of \(d_3 = 10.144\), shown in Figure 5.

ROBOLAB has two different trajectory planning methods, namely Cartesian “Cart. Trj.” and joint “Joint Trj.” trajectories shown in Figure 5.
The position, velocity and acceleration profiles for each joint can be displayed by clicking the “Trajec.” button as shown in Figure 5. The corresponding position, velocity and acceleration profiles for the first and fourth joints are displayed in Figure 6. In the figure, the first joint is actuated in joint trajectory and the fourth joint is actuated in the Cartesian trajectory.

Moreover, the following specifications can be handled by choosing “Trajec.” button: (1) The position, velocity and acceleration profiles for the end-effector (Fig. 7), (2) transformation matrix
and, (3) three different angle sets (Fig. 8). As shown in Figure 7, the joints can also be actuated in the Cartesian trajectory, and the end-effector follows a desired trajectory.

**Forward Kinematics-II**

Unlike forward kinematics-I, all of the manipulator joints are actuated at the same time in the forward kinematics-II. User can activate the robot manipulator
by entering the joint values in the corresponding editable text boxes as shown in Figure 9. Thus the joint motions can be examined in more detail. The position and orientation limitations are affixed below the editable textboxes.

The desired orientation can be obtained from forward kinematics-II operations by entering 4th, 5th, and 6th joint angles into the appropriate editable boxes. Then, three different angle sets can be achieved by simply clicking “Trajec.” button.

**Inverse Kinematics**

ROBOLAB in inverse kinematics mode computes the set of joint angles, given the desired position and orientation of the end-effector relative to the base frame. The position and orientation of the end-effector...
in Cartesian space can be obtained by specifying \( p_x, p_y, p_z \) and one of the RPY-fixed, Euler, and RK6 angle sets.

“The Kin” button provides user several inverse kinematics options as shown in Figure 10. User can execute the inverse kinematics by using one of the three angle sets. An example for the inverse kinematics can be seen in Figure 10 where the position values are \( p_x = 10.05, p_y = 39.044, p_z = 7.856 \) and the orientation values are \( \gamma = 180^\circ, \beta = -64.803^\circ, \alpha = 1.259^\circ \) in Euler mode. It can be noticed that, the position and orientation (based on the data in Figure 8) of the robot manipulator in Figure 10 is the same as those in Figure 5, as expected.

Continuous Trajectory-I

Continuous trajectory-I mode of operation can be selected simply clicking the “C.Traj.-I” button in Figure 10. More than one forward kinematics angles can be run successively in this mode. The user may select either “Read table” option to execute maximum of 10 forward kinematics joint angle lines shown in Figure 11, or “Read file” option to run any number of forward kinematics joint angle lines.

Figure 11 shows the “Read table” option which provides six editable text boxes for each line. It is not necessary to enter data each line in this mode. User can run the robot manipulator successively by entering the “Loop number” and “Line number”. For instance, the first two lines (Line number = 2) are run five times (Loop number = 5) as shown in Figure 11.

When “Read file” option is selected, the related data stored to a text file to be specified by the user can be loaded into the program memory as shown in Figure 12a. The confirmation of the name and data length of the text file will also appear on the screen as given in Figure 12b. An example of data file is shown in Figure 13. By specifying “Start line = 1,” “Finish line = 11,” and “Loop number = 1” for the data file as shown in Figure 12b, the workspace of the RS robot manipulator can be displayed as shown in Figure 14. Several trajectories such as linear, circular or parabolic can also be defined by processing data files.

Continuous Trajectory-II

Continuous trajectory-II mode of operation provides more than one inverse kinematics angle lines. The simulation procedure is the same as the continuous trajectory-I mode.

CONCLUSIONS

In this paper, a novel robot toolbox “ROBOLAB” is developed for the educational users to improve the understanding of robotics fundamentals through interactive real-time simulation. ROBOLAB based
on MATLAB GUI includes a library for the 16 different 6-DOF solid models of fundamental serial robot manipulators. It provides valuable analysis tools to students and engineering professionals who can compute rotation and transformation matrix, forward and inverse kinematics, and trajectory planning, and can solve robotics problems with less time and effort. The toolbox allows the students to focus on the programming instead of wasting time in the large amount of calculations involved.

REFERENCES


BIographies

Serdar Kucuk received the degree in electronics and computer education from the University of Marmara, Istanbul, Turkey, in 1995, the MSc degree in electronics and computer education from the University of Marmara, Istanbul, Turkey, in 1998, and the PhD degree in electrical education from the University of Kocaeli, Kocaeli, Turkey, in 2004. Currently, he is working as an Assistant Professor at the University of Kocaeli, in Kocaeli, Turkey.

Zafer Bingul received the BA degree from Istanbul Technical University, Istanbul, Turkey and the MS and PhD degrees from Vanderbilt University, Nashville, TN, in 1992, 1996, and 2000, respectively, all in electrical engineering. He is currently an Associate Professor of Mechatronics Engineering, School of Engineering, Kocaeli University, Kocaeli, Turkey. His research interests are robotics and welding automation, optimization, evolutionary algorithms, and intelligent control.