Industrial Robotics for Spacecraft Rendezvous and Docking Simulation

Nicolae APOSTOLESCU*[,1], Tom SAVU[2], Dragos Daniel Ion GUTA[1], Achim IONITA[1]

*Corresponding author
[1]INCAS – National Institute for Aerospace Research “Elie Carafoli”,
B-dul Iuliu Maniu 220, Bucharest 061126, Romania,
apostolescu.nicolae@incas.ro*, guta.dragos@incas.ro, ionita.achim@incas.ro
[2]DOLSAT Consult srl,
Al. Valea lui Mihai 2, bl D2, sc 5, ap 48, sector 6, Bucharest 061756, Romania,
dolsat@dolsat.com

DOI: 10.13111/2066-8201.2019.11.4.3

Received: 21 october 2019/ Accepted: 05 November 2019/ Published: December 2019

Abstract: *It is to be expected that space robotics systems will develop more and more in the future spatial applications. One of the most challenging and risky missions for spacecraft is to perform autonomously Rendezvous and Docking (RvD) in space. This paper describes a hardware-in-the-loop RvD simulation facility which uses two ABB industrial robots to simulate the 6-DOF dynamic manoeuvring of the rendezvous process. Firstly, we also simulate the real robots and we deduce the kinematic and dynamic equations of space robotic system, then the trajectory is planned. The target motion is reduced only to the movement of the end-effector, but the chaser's motion is based on the plan which solves the inverse kinematic equations, considering the movement limitations of the joints. Finally, a 3D simulation system was developed to evaluate the proposed method using Matlab/ Simulink environment. Simulation results verified the corresponding method and algorithms.*

Key Words: *manipulator, robot modelling, simulation, forward and inverse kinematic*

1. INTRODUCTION

Industrial robots are mechanical manipulators that rely on user controls, or can make their own decisions based on sensors inputs. The decision making and actions of the robot completely depends on the program running on the robot's computing unit. In a robotic programming the input devices include robot sensors, teach pendants and touch screens, and the output devices include displays and actuators. Any of the programming languages can program robots, but C++, Python and Matlab/Simulink are the most commonly used.
Robot simulators are useful tools for developing robot behaviour and provide a fast and efficient means for testing real robots. Software such as Robotics Developer Studio built on the ABB Virtual Controller is a PC application for modelling, offline programming, and simulation of robot cells. RAPID is the programming language used in ABB industrial robots to automate robotic applications.
Hardware-in-the-loop (HIL) simulation is a technique for validation and control of the robotic programs by creating a virtual real-time environment that represents the real physical system.

HIL helps to test the behaviour of robots without physical prototypes. HIL simulation runs on an intended target controller.
The Robot Operating System (ROS) is another set_of software libraries and tools that helps building robot applications. ROS contains many open source implementations of common robotics functionality and algorithms [15-18], [21], [24-28].
Basically, ROS is a framework for communicating between two programs or processes. ROS processes are represented as nodes in a graph structure, connected by edges called topics over which nodes send and receive messages.

2. ROBOT MANIPULATORS IN JOINT SPACE

The mechanical system of a serial robot consists of a configuration of n+1 rigid body, called links or "body", linked to each other successively through the n rotating joints called “joint”. The links are assumed to be perfectly rigid and are numbered such that link 0 constitutes the base of the robot and link n is the terminal link. Thus, joint i connects link *i*-1 and link *i*. The relative positions of these elements determine the overall position of the mechanical arm.
This position is of interest in robot applications. Each link contains a single degree of freedom relative to the previous link so that the transformation relationships between items contain a single variable parameter.
This parameter is the angle of rotation in the case of a “revolute” joint. The crowd of these angles forms the joint positions or joint configurations.
Knowing the geometry of the robot and all the positions of its joint, it is possible perform its mathematics and determine the position and orientation of any point on the robot. Therefore, joint positions can be controlled to place the end effector (tool) of the robot in 3D space. This is known as **forward kinematics (FK)**.
Forward kinematics will determine where the tool of the robot manipulator will be if all joints are known. The FK objective is to determine the cumulative effect of the entire set of joint variables.
In many robotics applications, we want to calculate the joint angles needed such that the end effector reaches a specific position and orientation.
This is known as **inverse kinematics (IK).** Inverse Kinematics (IK) analysis determines the joint angles for desired position and orientation in Cartesian space for an end effector as a homogenous transformation.
The solution of inverse kinematic is more complex than direct kinematics and there is not any global analytical solution method.
Hence, forward kinematics is defined as transformation from joint space (joint configuration) to Cartesian space while inverse kinematics is defined as transformation from Cartesian space to joint space [12-14].
Modelling and simulation are two processes used for developing and testing robot behaviour inside his workspace.
The robot workspace is the total volume swept out by the end effector when the manipulator executes all possible motions. The position control problem consists in driving the manipulator end-effector (the joint variables, respectively) to the desired position, regardless of the initial posture.
This problem consists in solving, in one single step, the following sub-problems: path planning, trajectory generation and control design. Generally, the most important property in a control system, is its stability [8].

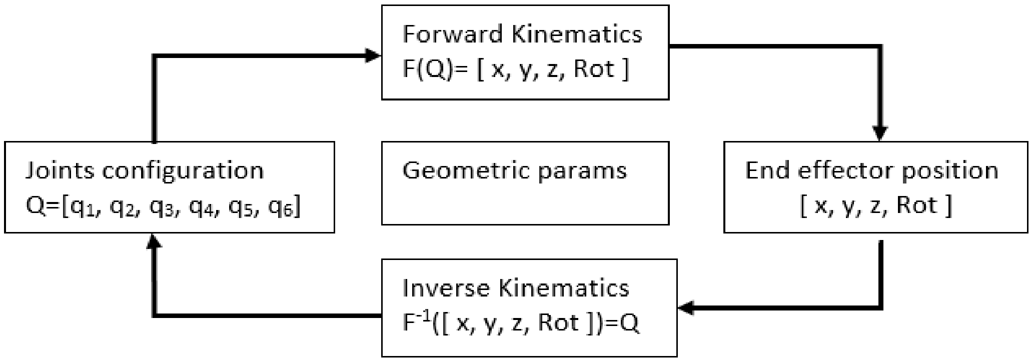


Fig. 1 The relationship between direct and inverse kinematics problem

3. KINEMATIC MODELLING

The kinematics model deals with the spatial position of links and joints without considering the forces or moments that cause the motion.

The forward kinematics studies the effect of the entire set of joint variables on the position and orientation of the tool.

Denavit and Hartenberg [1] proposed that four parameters assign orthogonal frames to each link of a manipulator and these DH parameters have become the most common standard method for describing the robot kinematics.

DH parameters θ_i, a_i, d_i, α_i are the parameters of link *i* and joint *i*. The DH parameters give the angles and displacements between frames.

- a_i (link length), is the offset distance between z_{i-1} to z_i axes, measured along x_i;
- α_i (link twist), is the angle from z_{i-1} axis and z_i axis measured about x_i axis;
- d_i (link offset), is the distance from the origin of frame $i-1$ to the x_i axis along z_{i-1} axis;
- θ_i (joint angle), is the angle between x_{i-1} and x_i, measured about z_{i-1} axis.

Note. The z-axis should lie on the axis of rotation, for a revolute joint, the *x*-axis should lie along the "common normal", which is the shortest orthogonal line between the previous *z*-axis and the current *z*-axis.

In representing the frame, the axes are drawn using the following color *z*-axis (blue), *x*-axis (red) and *y*-axis (green).

The *generalized* joint coordinate denoted by q_i, corresponds to the angular displacement around z_i if the *i*-th joint is revolute.

The Direct Geometric Model is the set of relations that defines the location of the end-effector of the robot as a function of its joint configurations.

For a serial structure, it may be represented by the transformation matrix ${}^0T_n = {}^0T_1\,{}^1T_2\ \ {}^{n-1}T_n = F(q)$ where each transformation T_i is represented as a product of four basic transformations $T_i = Rot_{z,\theta i}\, Trans_{z,di}\, Trans_{x,ai}\, Rot_{x,\alpha i}$;

$$ {}^{i-1}T_i = \begin{vmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{1} $$

Finally, transformation is given by the following matrix multiplication and is generally represented as follow:

$$^{0}T_6 = {}^{0}T_1\, {}^{1}T_2 \quad {}^{5}T_6 = \begin{vmatrix} l_x & m_x & n_x & p_x \\ l_y & m_y & n_y & p_y \\ l_z & m_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{2}$$

where *l*, *m*, *n* are orthogonal vectors and they represent the orientation (rotation element), *p* is a vector which represents the position. The fourth column of this matrix defines the coordinates of the origin of the end-effector frame referred to the reference frame. The first column of this matrix defines the projections of the normal axis of the tool frame onto the *x*, *y* and *z* axes of the reference frame axes. The second column defines the projections of the orientation vector *y* onto the *x*, *y* and *z* axes of the reference frame axes. The third column defines the projections of the vector *z* on to the *x*, *y* and *z* axes of the reference frame axes.

4. MODELLING A 6_DOF MANIPULATOR USING MATLAB SOFTWARE

In this paper we will refer to the industrial robots IRB4600-60 and IRB7600-500/2.55 installed in the INCAS SpaceLab Laboratory. The control of these robots is made by Industrial RobotController S4Cplus [11], [29-30]. It admits the programming of robots in RAPID language, TCP/IP communication or other protocols. The mechanical system of both serial robots consists of a configuration of 6+1 rigid bodies, linked to each other successively through the 6 rotating joints. These 6 joints define the 6 degrees of freedom (DOF).
For the modelling of the IRB robots installed in the laboratory were used the following two Matlab tools:

- Robotics_Vision_Control-Rvctools. The development of this toolbox respects the terminology of Denavit and Hartenberg [20], [22-23].
- Matlab Robotics system Toolbox [21], [24] .

In addition, the simulation of the kinematic behaviour of the robot using Matlab instruments was compared with that made with RobotStudio. The table below shows data extracted from the documentation attached to the ABB7600-500/2.55 product.

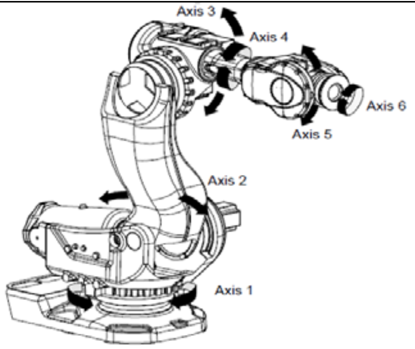


Fig. 2a Robot ABB7600-500/2.55

Alpha	a(m)	theta	d(m)
-pi/2	0.41	0	0.78
0	1.075	-pi/2	0
-pi/2	0.165	0	0
pi/2	0	0	1.056
-pi/2	0	0	0
0	0	pi	0.25

Fig. 2b Denavit and Hartenberg Parameters (DH) for IRB7600-500/2.55 robot

Table 1. ABB7600-500/2.55 Joint position

Joint position			Local ref. frame(xyzl)			joint axis		
0	0	0	0	0	0	0	0	1
a(1)	0	d(1)	a(1)	0	d(1)	0	1	0

a(1)	0	d(1)+a(2)	0	0	a(2)	0	1	0
a(1)	0	d(1)+a(2)+a(3)	0	0	a(3)	1	0	0
a(1)+d(4)	0	d(1)+a(2)+a(3)	d(4)	0	0	0	1	0
a(1)+d(4)+d(6)	0	d(1)+a(2)+a(3)	d(6)	0	0	0	0	1

Table 2. ABB7600-500/2.55 motion range (joint motion range)

J1		J2		J3		J4		J5		J6	
-180	180	-60	85	-180	60	-300	300	-100	100	-360	360

Based on this data, the rigid body tree model was created using Matlab classes Link, SerialLink or robotics.RigidBodyTree. Every rigid body tree has a base. The base defines the world coordinate frame and is the first attachment point for a rigid body. The base cannot be modified. The base origin is [0 0 0]. Each rigid body has a joint that defines how that body moves relative to its parent in the tree. Specify the transformation from one body to the next by setting the fixed transformation on each joint [4]. Robotics System Toolbox assumes that the positions and orientations are defined in a right handed Cartesian Coordinate System [19]. Using the codes below, two models were created for the same ABB7600 robot represented in figures 3a and 3b.

Create robot body tree model using SerialLink [20].

```
dhparams=[theta;d;a;alpha]';
L(i) = Link ([dhparams(i,:)   0      0] , 'standard');    i=1:6
IRB7600SL = SerialLink(L, 'name', 'IRB7600SL INCAS');
```

Create robot body tree model using RigidBodyTree[21].

```
type='revolute';
for ilink=1:nr_links
     body(ilink)=robotics.RigidBody(strcat('body',num2str(ilink)));
      joint (ilink) = robotics.Joint(strcat('joint' ,num2str(ilink)), type);
      setFixedTransform(joint(ilink),trvec2tform(xyzl(ilink,:) ));
      joint(ilink).JointAxis = JAxis(ilink,:);
      joint(ilink).PositionLimits= [limitsrad(ilink,1) limitsrad(ilink,2)];
      body(ilink).Joint = joint(ilink);
      if ilink==1
          addBody(IRB7600RB, body(ilink), 'base');
      else
          addBody(IRB7600RB,body(ilink), strcat('body',num2str(ilink-1)))
      end
```

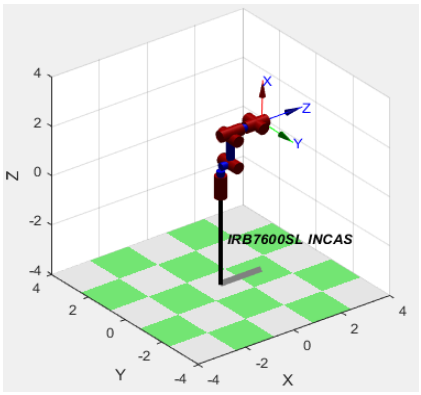


Fig. 3a Robot model using SerialLink

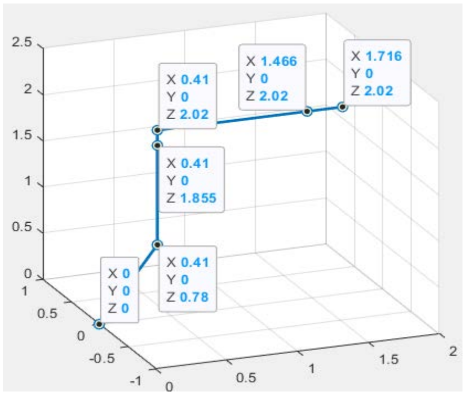


Fig. 3b Robot model using RigidBodyTree

Table 3. The models for ABB4600-60/2.05 are analogous where DH parameters are in the table

alpha	-pi/2	0	-pi/2	pi/2	-pi/2	0
a	0.175	0.9	0.175	0	0	0
theta	0	-pi/2	0	0	0	pi
d (m)	0.495	0	0	0.96	0	0.135

Table 4. ABB4600-60/2.05 motion range (joint motion range)

J1		J2		J3		J4		J5		J6	
-180	180	-90	150	75	-180	-400	400	-125	120	-400	400

Regarding the ABB7600-500/255 robot, we start with the description of the direct kinematics as serial link using (1) and (2) that have been implemented in Matlab programs. The kinematic chain F(q) that describes the direct kinematics of the serial link robot IRB7600 is the following set of matrices:

${}^{0}T_1$	$\begin{array}{cccc} 1.00 & -0.00 & -0.00 & 0.41 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & -1.00 & 0.00 & 0.78 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{array}$	${}^{1}T_2$	$\begin{array}{cccc} 0.00 & 1.00 & -0.00 & 0.00 \\ -1.00 & 0.00 & -0.00 & -1.07 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{array}$	${}^{2}T_3$	$\begin{array}{cccc} 1.00 & -0.00 & -0.00 & 0.17 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & -1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{array}$
${}^{3}T_4$	$\begin{array}{cccc} 1.00 & -0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & -1.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 1.06 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{array}$	${}^{4}T_5$	$\begin{array}{cccc} 1.00 & -0.00 & -0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & -1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{array}$	${}^{5}T_6$	$\begin{array}{cccc} -1.00 & -0.00 & 0.00 & -0.00 \\ 0.00 & -1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.25 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{array}$

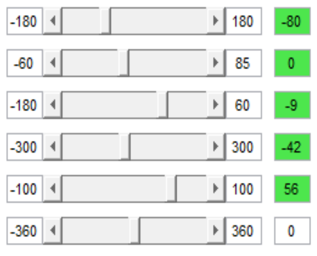
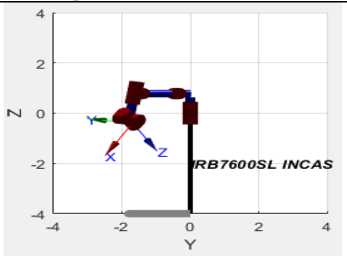
$$\begin{array}{l|cccc} {}^{0}T_6 = {}^{0}T_1\,{}^{1}T_2 \ldots {}^{5}T_6 = & -0.00 & -0.00 & 1.00 & 1.72 \\ & -0.00 & 1.00 & 0.00 & 0.00 \\ ({}^{0}T_6 = {}^{robot\text{-}base}T_{end\text{-}effector}) & -1.00 & -0.00 & -0.00 & 2.02 \\ & 0.00 & 0.00 & 0.00 & 1.00 \end{array}$$

For the initial joint configuration q=[0 0 0 0 0 0]; TCP=[1716 0 2020], and for q=[0 0 0 0 30 0]; TCP=[1682.51 0 1895]. These results were confirmed with RobotStudion solution. The tool center/centre point (TCP) is the point in relation to which the entire positioning of the robot is defined.

Using Matlab guide we created an interface that allowed the user to determine the robot end-effector position and orientation for different joint configurations [31].

The functions used are robot.fkine for serial link model and setFixedTransform and replaceJoint for rigid.body model.

Table 5. User interface for setting joint configuration and observing the result

Joint's configuration	Robot position (model serial link)	end-effector pose including TCP
 -180 180 -80; -60 85 0; -180 60 -9; -300 300 -42; -100 100 56; -360 360 0		0.4623 -0.6171 -0.6368 0.1557 -0.4670 -0.7799 0.4168 -1.6814 -0.7538 0.1047 -0.6487 -0.3994 0 0 0 1.0000 q=[-80 0 -9 -42 56 0];

5. MOTION CONTROL OF ROBOT

The main application in which we will use the two robots installed in the INCAS SpaceLab Laboratory is the simulation of an autonomous spatial rendezvous and docking. This process consists of a series of orbital manoeuvres and controlled trajectories, which successively bring the active vehicle (chaser) into the vicinity of, and eventually into contact with, the passive vehicle (target) [5].

Equations of motion in the orbital plane can be used for trajectory analysis until the chaser vehicle is in the close vicinity of the target.

For relative navigation it becomes more convenient to keep one of the spacecraft as a fixed point (target). We distinguish two stages for the simulation of an autonomous spatial rendezvous using robots.

First, we create applications for the offline control of robots, then these applications will be implemented on real robots.

In this paper we are concerned with the control of the robot end-effector to describe a certain trajectory in its own motion envelope [12-14].

The work is focused on establishing the procedures and software programs to control the movement of one of the robots (IRB7600) in order to move it through a trajectory. To achieve this we used procedures specific to the inverse kinematics [2,] [3], [6], [7], [10].

Inverse Kinematics involves determining a set of appropriate joint configurations for which the end effectors move to desired positions as smoothly, rapidly, and as accurately as possible [8-9].

Trajectory generation consists in parameterizing in time during the path planning but our reference trajectory is defined in terms of coordinates in robot workspace.

Once a trajectory is generated, the Inverse Kinematics block is used to translate his Cartesian space to a joint-space trajectory, which can then be used to simulate the dynamics of the robot manipulator.

Using inverse kinematic procedure $q=F^{-1}$(end-effector pose) the joint configuration for several trajectories was obtained.

Figure 4a shows a trajectory through several points through which the end effector of the robot will pass.

In those points its orientation is required as in Figure 4a. Fig. 4b shows the Joint angles along the trajectory that are obtained with inverse kinematics procedure.

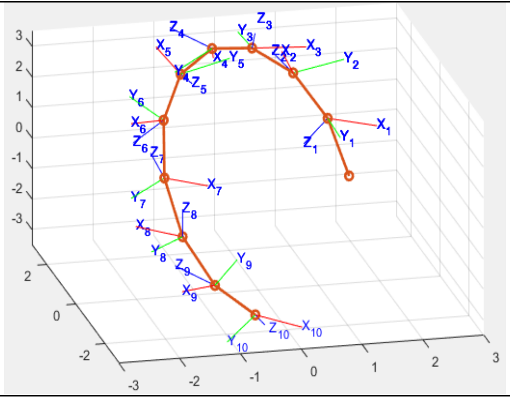


Fig. 4a Cartesian trajectory between a sequence poses

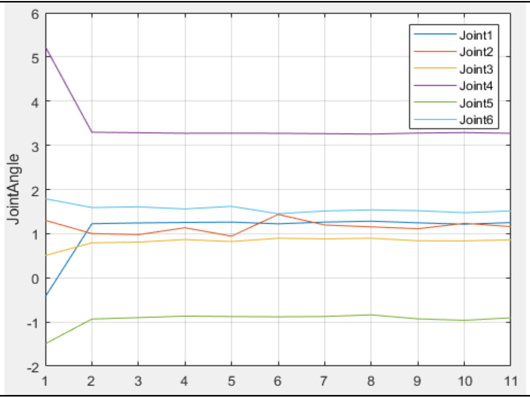


Fig. 4b Joint configuration along the trajectory

For the IK analysis with MATLAB tools we used serial link and rigid body models and for the ABB model (from ABB database) RobotStudio Simulator [11]. Matlab's Inverse Kinematics tools are robotics InverseKinematics and GeneralizedIK classes. These classes are builted based on iterative algorithms Broyden–Fletcher–Goldfarb–Shanno ((BFGS)) or Levemberg –

Marquardt [21]. The IK simulation with RobotStudio used the RAPID. Figures 5 capture a moment of movement of the three models in a trajectory.

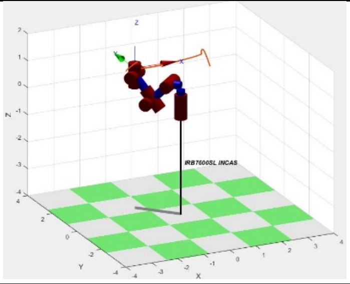
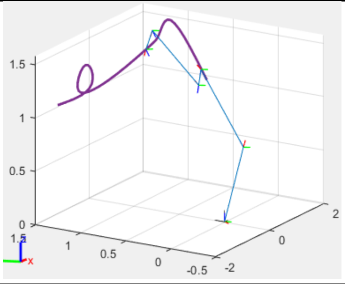
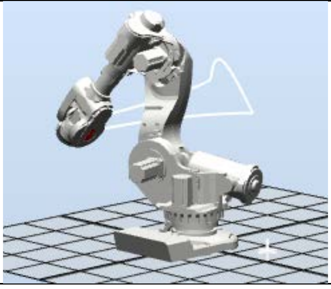
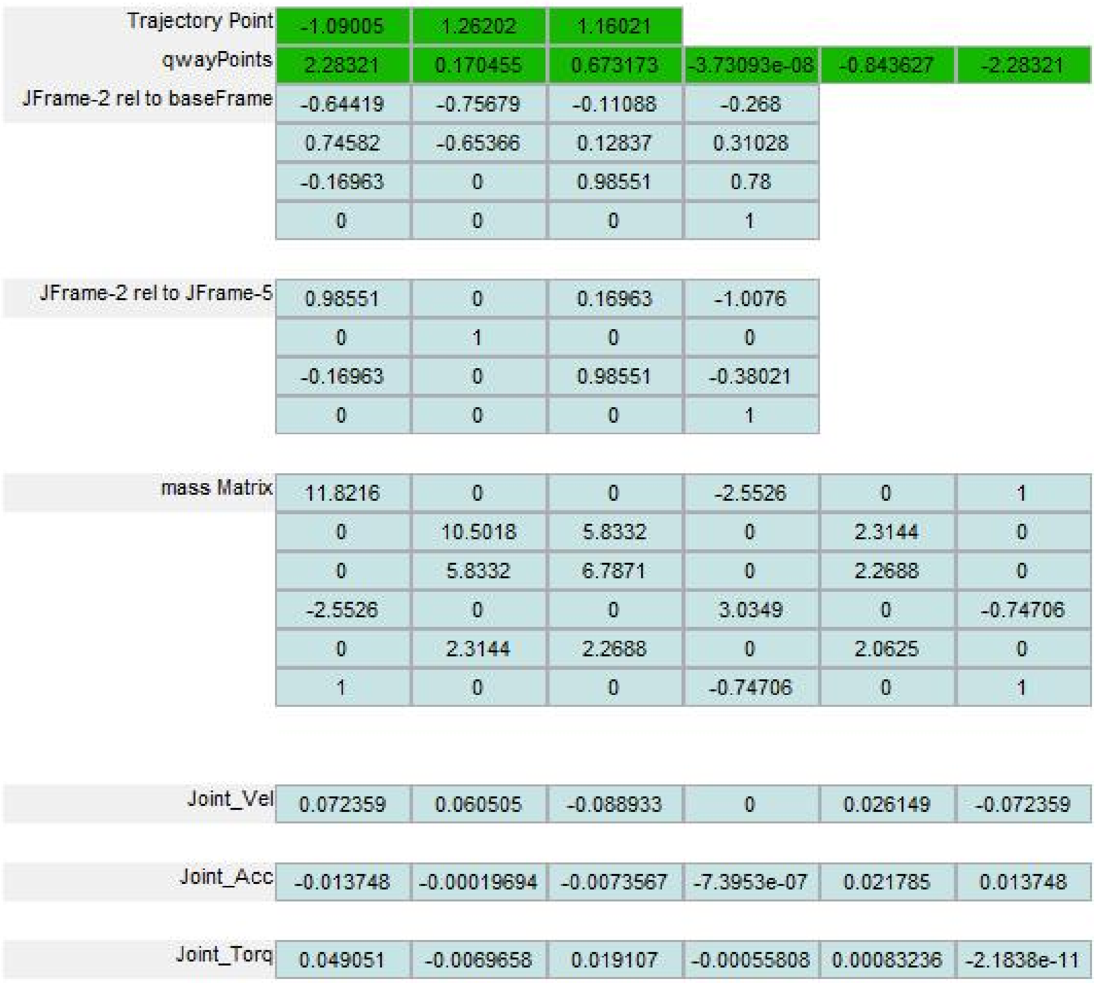
		

Fig. 5a Robot serial link browse a date path	Fig. 5b Robot Rigid body browse a date path	Fig. 5c Robot RobotStudio browse a date path

A MATLAB interface gives the user the opportunity to choose a point on the trajectory and one or two of the joints and will get the information below:



Trajectory Point	-1.09005	1.26202	1.16021			
qwayPoints	2.28321	0.170455	0.673173	-3.73093e-08	-0.843627	-2.28321
JFrame-2 rel to baseFrame	-0.64419	-0.75679	-0.11088	-0.268		
	0.74582	-0.65366	0.12837	0.31028		
	-0.16963	0	0.98551	0.78		
	0	0	0	1		

JFrame-2 rel to JFrame-5	0.98551	0	0.16963	-1.0076
	0	1	0	0
	-0.16963	0	0.98551	-0.38021
	0	0	0	1

mass Matrix	11.8216	0	0	-2.5526	0	1
	0	10.5018	5.8332	0	2.3144	0
	0	5.8332	6.7871	0	2.2688	0
	-2.5526	0	0	3.0349	0	-0.74706
	0	2.3144	2.2688	0	2.0625	0
	1	0	0	-0.74706	0	1

Joint_Vel	0.072359	0.060505	-0.088933	0	0.026149	-0.072359
Joint_Acc	-0.013748	-0.00019694	-0.0073567	-7.3953e-07	0.021785	0.013748
Joint_Torq	0.049051	-0.0069658	0.019107	-0.00055808	0.00083236	-2.1838e-11

Fig. 6 Control values on robot Trajectory

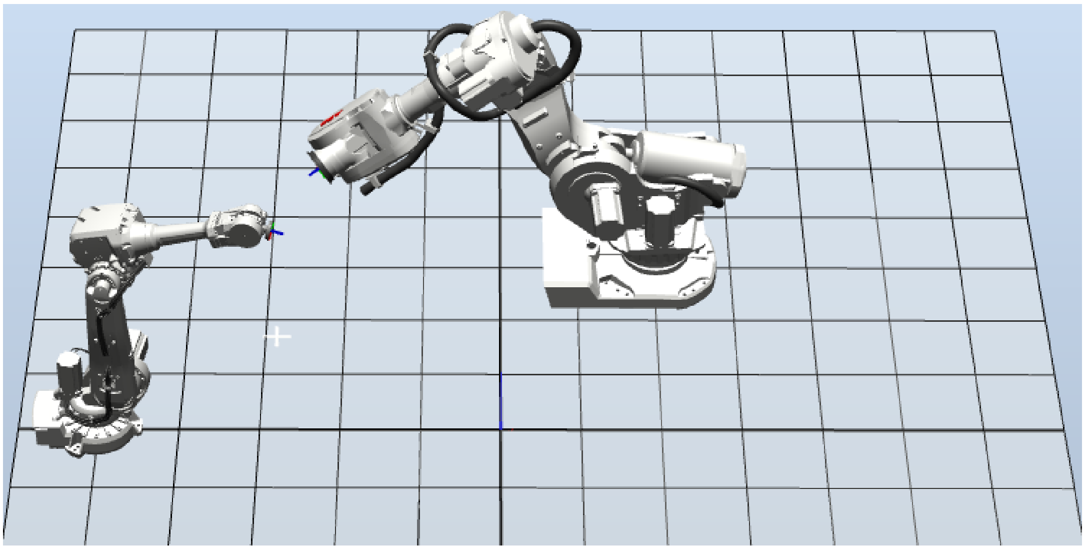


Fig. 7 Chaser seek for Target (in RobotStudio)

The relation between the Cartesian space of the end-effector and joint space of the joint angles can be also achieved using jacobian matrix. The jacobian of any robot manipulator structure is a matrix that relates the *endeffector* linear and angular Cartesian velocities with the individual joint velocities

$$J(q).\dot{q} = \begin{bmatrix} v \\ w \end{bmatrix}, \qquad \dot{q} = J^{-1}(q).\dot{X} \ , \qquad \dot{X} = \begin{bmatrix} v \\ w \end{bmatrix}. \tag{3}$$

where J(q) is the jacobian matrix of the robot manipulator, $\dot{q} = [\dot{q}_1 \, , \ \dot{q}_2, \ \dot{q}_3, \ \dot{q}_4 \, , \ \dot{q}_5, \ \dot{q}_6]$ the joint velocity vector, $v = [v_1, \ v_2, \ v_3]^T$ is the *end-effector* linear velocity vector, and $w = [w_1, \ w_2, \ w_3]^T$ is the *end-effector* angular velocity vector [8].

The $J^{-1}(q)$ calculation process is a two-phase iterative process. The first phase consists in calculating the transformation matrices (1) to obtain the end-effector position. Then the end-effector position is changed in the desired one. The second phase calculates Jacobian matrix inversion and changes joint angles using (3). The mentioned phases repeat until the difference between the current and the desired end-effectror position comes below a defined value or until the given number of iteration steps is reached.

6. CONCLUSIONS

The simulation of an autonomous spatial rendezvous and docking processes using ABB robots can be accomplished successfully. This process consists in controlling the movements of both robots. In general, the control of a robot involves solving stability problems, positioning control issues, trajectory tracking and even optimisation of its movement. The motion control problem consists in making the end-effector go to a specified point regardless of the trajectory (point to point method). Analyzing the results of the developed simulation programs we conclude that they can be used to control the Kinematic movement of any robot and they can be implemented on real robots. These gives correct joint angles so that the robot arm with its end- effector can easily moved to any reachable positions and orientations for performing a pick and place task. Also these programs can be used to solve kinematics for other robotics arm.

REFERENCES

[1] J. Denavit & R. S. Hartenberg, A kinematic notation for lower-pair mechanisms based on matrices, *Journal of Applied Mechanics*, Vol. **1**, pp. 215-221, June 1955.

[2] J. J. Craig, *Introduction to Robotics: Mechanics and Control* (3rd Edition), ISBN 978-0201543612.

[3] S. Kucuk and Z. Bingul, Robot Kinematics: Forward and Inverse Kinematics, *Industrial-Robotics-Theory-Modelling-Control*, ISBN 3-86611-285-8, pp. 964, ARS/plV, Germany, December 2006.

[4] L. A. Radavelli, E. R. De Pieri, D. Martins, R. Simoni, *A screw dual quaternion operator for serial robot kinematics*, Proceedings of 14th Pan-American Congress of Applied Mechanics - PACAM XIV, Santiago, Chile, March 24-28, 2014.

[5] W. Fehse, *Automated Rendezvous and Docking of Spacecraft*, Cambridge University Press, ISBN-13 978-0-511-07086-0 eBook (EBL), 2003.

[6] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*, Springer-Verlag London 2000.

[7] B. Sicilano, L. Sciavicco, L. Villani, G. Oriolo, *Robotics: Modelling, Planning and Control*, Springer, 2009.

[8] R. Kelly, V. Santibáñez and A. Loría, *Control of Robot Manipulators in Joint Space*, Springer-Verlag London Limited 2005, ISBN-10: 1852339942.

[9] A. H. Shabeeb, L. A. Mohammed, *Forward Analysis of 5 DOF Robot Manipulator and Position, Eng. & Tech. Journal*, Vol. **32**, Part (A), No. 3, pp. 617-628, 2014.

[10] S. Cubero, *Industrial Robotics-Theory, Modelling and Control*, ISBN 3-86611-285-8, 2006.

[11] P. Neto, *A Guide for ABB Robot Studio*, January 2014.

[12] D. B. Marghitu, M. Dupac, *Analytical and Numerical Calculations with MATLAB*, ISBN 978-1-4614-3475-7 (eBook).

[13] D. B. Marghitu, *Mechanisms and Robots Analysis with MATLAB*, Springer Dordrecht Heidelberg, ISBN 978-1-84800-390-3.

[14] D. Marghitu, *Kinematic chains and machine components design*, Elsevier Academic Press, ISBN 0-12-471352-1, 2005.

[15] D. K. Chaturvedi, *Modeling and Simulation of Systems Using Matlab and Simulink,* International Standard Book Number: 978-1-4398-0672-2 (Hardback).

[16] J. Angeles, *Fundamentals of Robotic Mechanical Systems. Theory, Methods, and Algorithms*, Second Edition, ISBN 0-387-95368-X, 2003.

[17] S. Chauhan, S. Chaudhari, R. Sethiya, S. Ahire, D. Malche, Modular Robotic Arm, *International Research Journal of Engineering and Technology (IRJET)*, Volume **02**/ May 2015.

[18] K. H. Wurst, The conception and construction of a modular robot system, *International Journal of Scientific and Research Publications*, 1986.

[19] J. Funda, R. H. Taylor & R. P. Paul, On homogeneous transforms, quaternions, and computational efficiency. *IEEE Trans.Robot. Automat.*, Vol. **6**, pp. 382–388, June 1990.

[20] P. Corke, *Robotics Toolbox for Matlab*, Release 9, 2011.

[21] * * * *Robotics System Toolbox*, User's guide, R2018a, Matworks.

[22] P. I. Corke, *A computer tool for simulation and analysis the Robotics Toolbox for MATLAB.*

[23] P. Corke, *Resources for robotics education*, February 2017: http://petercorke.com/wordpress/

[24] * * * *Reference Robotics System Toolbox*™Reference- R2018a.

[25] * * * *Computer Vision System Toolbox*™Reference.

[26] Ch. Yang, H. Ma, M. Fu, *Advanced Technologies in Modern Robotic Applications*, Springer Science Press Beijing.

[27] D. K. Chaturvedi, *Modeling and Simulartion of Systems Using Matlab and Simulink*, CRC Press Taylor & Francis Group, 2010.

[28] K. Perutka, *MATLAB for Engineers –Applications in Control, Electrical Engineering, IT and Robotics*, ISBN 978-953-307-914-1

[29] * * * *Robot Kinematics*, February 2017, http://www.wikipedia.com.

[30] * * * The ABB group. http://www.abb.com/, Oct 2003.

[31] * * * *KUKA System Software (KSS). Expert Programming*