The autonomous control of landing on mobile platforms

Nicolae APOSTOLESCU*,[1], Ion TOMESCU[2], Dragos Daniel Ion GUTA[1], Radu BOGATEANU[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, bogateanu.radu@incas.ro
[2]Systems - Business Plus S.R.L.,
ion.tomescu@businessplus.ro

DOI: 10.13111/2066-8201.2021.13.3.1

Received: 24 June 2021/ Accepted: 10 August 2021/ Published: September 2021

Abstract: *In this paper, we propose a simulation application that allows an aerial vehicle to land autonomously on a moving platform in the presence of uncertainties and disturbances. We have tested our method with various speeds and positions for the landing platform. In the context of this article, the autonomous control of landing on mobile platforms consists in synchronizing the movement of an aerial vehicle with the movement of the mobile platform. As a first step, the Spacelab INCAS laboratory group has developed an offline simulation application that allows an ABB robot to receive information on the movement of a Stewart-type mobile platform in order to conduct a landing process. The application can initiate a landing process on the mobile platform and guide the vehicle for perfect docking on the platform. Offline simulation allows the study of several scenarios of a robot working cell - the mobile platform before setting up the production cell. The offline application has a distributed client-server structure. The client communicates with the server through specific communication protocols. The client and server can reside on the same computer. The client application is developed in the Matlab environment and has as object the simulation and programming of the PS-6TL-1500 platform; the server one simulates and programs an ABB 7600-500/2.55 robot that moves on the track, in the RAPID language under RobotStudio ABB simulator.*

Key Words: *robot modelling, simulation, client - server communication*

1. INTRODUCTION

In this work, we propose a simulation application that allows an aerial vehicle to land autonomously on a moving platform in the presence of uncertainties and disturbances. We have tested our method with various speeds and positions for the landing platform. In the context of this article, the autonomous control of landing on mobile platforms consists in synchronizing the movement of an aerial vehicle with the movement of a single mobile platform. As a first step, the Spacelab INCAS laboratory group has developed an offline simulation application that allows an ABB robot to receive information on the movement of a Stewart type mobile platform in order to conduct a landing process. The application can initiate a landing process on the mobile platform and guide the vehicle attached to the robot end effector for a perfect docking on the platform (see Fig. 4). Offline simulation allows to study several scenarios of a robot working cell - the mobile platform before setting up the real cell.

It is also often quicker and easier to change some parameters in a model built in a computer simulation than to change parameters in a real process.

Here, the offline application has a distributed client-server structure. The client communicates with the server through specific communication protocols. The client and server can reside on the same computer.

The client application is developed in the Matlab environment and has as object the simulation and programming of the PS-6TL-1500 platform. This platform is assimilated to a fully parallel robot that is composed of a mobile platform connected to a fixed base by a set of six identical kinematic chains which are called legs. The desired mobile platform position can be obtained by changing the leg lengths. The PS-6TL-1500 product is based on a Stewart platform. The 6 DoF architecture of motion platform is mainly intended for professional applications in defence sector, flight and driving simulation and entertainment industry.

The general specification: system performance, payload specification, main dimensions, and power requirements can be found in [18], [19].

PS-6TL-1500 PERFORMANCE	PS-6TL-1500
SURGE -0.63, 0.53 m SWAY -0.53, 0.53 m HEAVE -0.41, 0.37 m ROLL -26.0°, 26.0° PITCH -24.0°, 24.0° YAW -34.5°, 34.6°	

Fig. 1 PS-6TL-1500

Using PS_6TL-1500 data it was possible to model the platform in Matlab environment. Its movement according to a given model, allows the determination of the normal vector to the upper plane of the platform originated at a specified point. This vector will give the robot endeffector position and orientation.

The server application simulates the behavior of an ABB 7600-500/2.55 robot for the proposed task using a) Matlab tools or b) RobotStudio simulator.

In the developed server application using Matlab tools, we need to calculate the required joint angles so that the end effector reaches a specific position and orientation of the mobile platform. Knowing the geometry of the robot and all its joint positions, it is possible to do mathematics and figure out the position and orientation of any point on the robot. The serial manipulator geometries are described using the well-known Denavit-Hartenberg (D-H) parameters [1], [2]. Therefore, the joint positions can be controlled to place the end effector (tool) of the robot in 3D space. First, based on the robot specifications data, the rigid body tree model was created, using Matlab classes Link, SerialLink or robotics.RigidBodyTree [8]. Every rigid body tree has a base. The base defines the world coordinate frame and is the first attachment point for a rigid body. Each rigid body has a joint that defines how that body moves in relation to its parent in the tree. The transformation from one body to another is specified by setting the fixed transformation on each joint. Robotics System Toolbox assumes that the positions and orientations are defined in a right handed Cartesian Coordinate System [6], [15]. The process of calculating the parameters of joints that determine a specified position of the end effector is known as inverse kinematics (IK) [13]. Matlab's Inverse Kinematics tools are

robotics InverseKinematics and GeneralizedIK classes [2], [10], [11]. The solution of inverse kinematic is more complex than direct kinematics and there is no global method of analytical solution. The robot joint positions must be within the position limits of the robot model and must not violate any constraints of the robot. The robot workspace is the total volume swept out by the end effector when it executes all possible motions [3], [5]. The position control problem consists in driving the robot end effector to the desired position, regardless of the initial posture.

To ensure that the robot TCP is inside its working space, a routine Matlab does the check before any test. For example, the bottom figure contains the envelope in the xz plane (green) and a point outside the envelope in the yz plane (red); therefore, this point cannot be reached by the robot. The specifications, performance, work space (envelope) of the ABB 7600-500/2.55 robot are presented in documents [14], [15], [17].

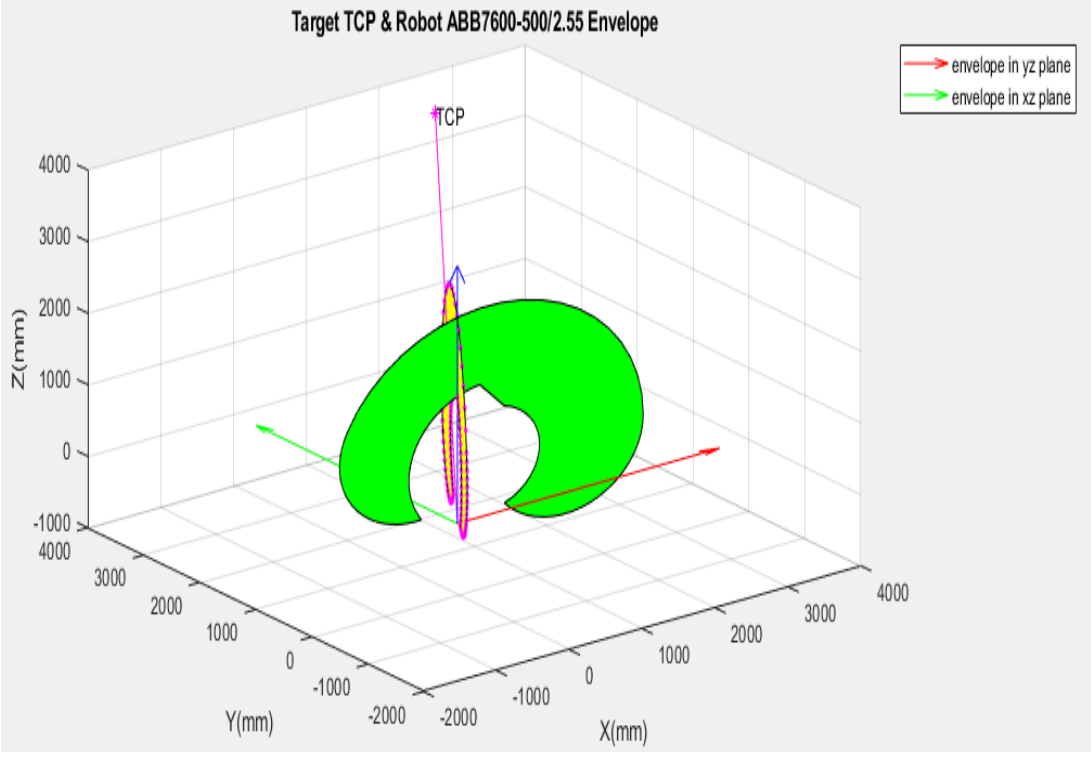


Fig. 2 The robot's workspace

The communication between the two Matlab applications uses the TCP/IP protocol. The server application first calculates the Euclidean distance between the initial position of the robot and the platform [4]. Then, the server waits and receives the information from the client application.

The information received consists of the position and orientation of the vector normal to the upper plane of the platform.

A program calculates the robot's joint angles using the inverse kinematics Matlab functions and move the end effector of the robot to a specific location and orientation. Finally, we assess the difference between the position of the normal vector of the platform and the position of the robot end effector.

Running the above steps 100 times with the initial position and orientation of platform randomly generated, the accuracy of these solutions is reasonably fair.

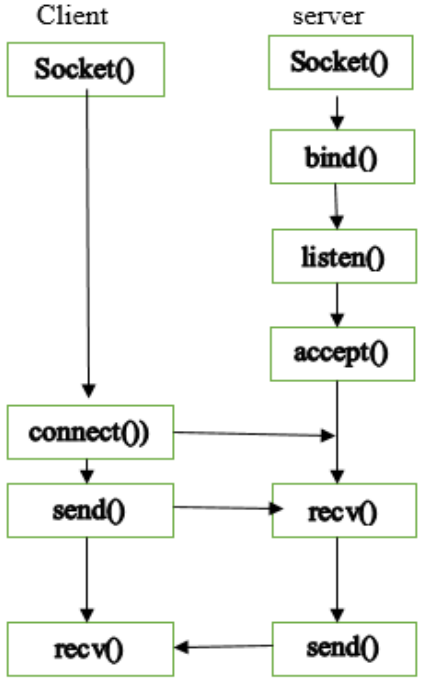
The RobotStudio simulator can also be used to simulate and program the ABB robot as a server application. This ABB simulator is a useful tool for developing the robot behaviour and it provides a fast and efficient means for testing real or virtual robots. Software such as Robotics Developer Studio built 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 [9], [17].
RobotStudio provides all models of ABB robots. The RAPID program calculates the required joint angles so that the end effector reaches a specific position and orientation of the mobile platform.

2. MATLAB – ROBOTSTUDIO COMMUNICATION

The developed application has a client-server structure. The basic mechanisms of client-server applications is the follow: a client application sends a request to a server application, the server application returns a reply.

The basic data communications between the client and the server are data or files. The tcpclient as client and tcpserver objects are build. The tcpclient object is always the client and cannot be used as a server.

The remote host can be a server or hardware that supports TCP/IP communication, and must already exist.



SocketCreate is used to create a new socket for connection based communication or connectionless communication.
SocketConnect is used to connect the socket to a remote computer in a client application.
SocketAccept is used to accept incoming connection requests. SocketAccept can only be used for server applications.
SocketBind is used to bind a socket to the specified server IP-address and port number. SocketBind can only be used for server applications.
SocketListen is used to start listening for incoming connections, i.e. start acting as a server. SocketListen can only be used for server applications.
SocketReceive is used for receiving data from a remote computer.
SocketReceive can be used for both client and server applications.
SocketSend is used to send data to a remote computer. SocketSend can be used for both client and server applications.

Fig. 3 Client Server communication scheme

3. THE APPLICATION DESCRIPTION

Matlab made it possible to build an advanced simulation model using a user-friendly graphical interface. The application has two working modes: manual and autonomous. In manual mode, the position and orientation of the upper plane of the PS-6TL-1500 platform are set as input using sliders, and sent to the RobotStudio simulator which will use them in a RAPID program to drive the ABB robot to the required state.

Fig. 7 suggests the current position (green) of the platform compared to the initial position (black). The interface makes us the robot answer, the TCP positions, its orientation, the angles of the six joints and torques.

In the autonomous mode, the status of the platform (position and orientation) is given by a Matlab wave simulation program. This datum is transmitted sequentially to the server which will move the ABB robot according to this requirement, ie the TCP status is identical to the platform status.

The robot movement is driven by a virtual robot controller provided by RobotStudio. The development of this client-server application aimed at making the RAPID program explicit and with as few command instructions as possible. For this, the Matalb client program provided the server with requests in the most appropriate form, for example it does not transmit angles but quaternions, checks if the TCP position is in the robot work envelope, commands linear movements to near the platform [7].

The data transmitted to the server by the client is grouped into Matlab strings. To be "understood" correctly they are grouped into strings of fixed length. Each string contains seven data (position and orientation) of 10 characters. The server receives the string as rawbytes (special RAPID type of data).

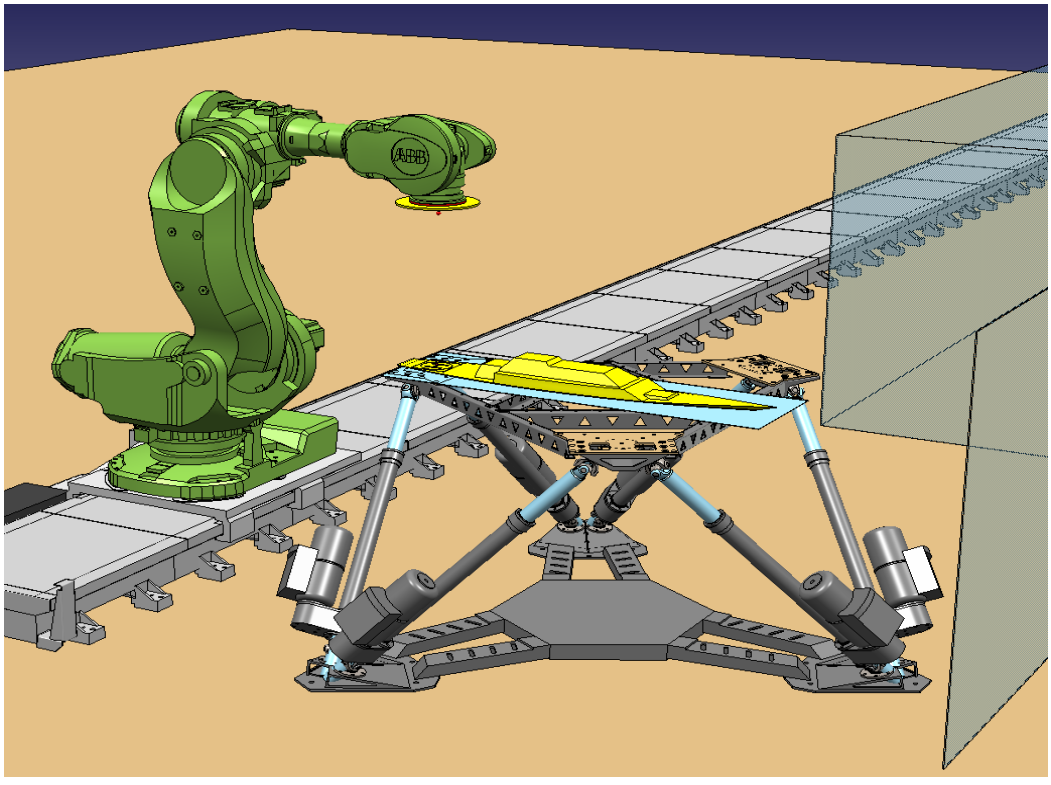


Fig. 4 Landing process on the mobile platform

A fast procedure is responsible for avoiding the collision with the platform which, for this reason, is inscribed in a virtual cylinder.

When TCP touches the surface of the cylinder, the simulation is stopped and accompanied by a message.

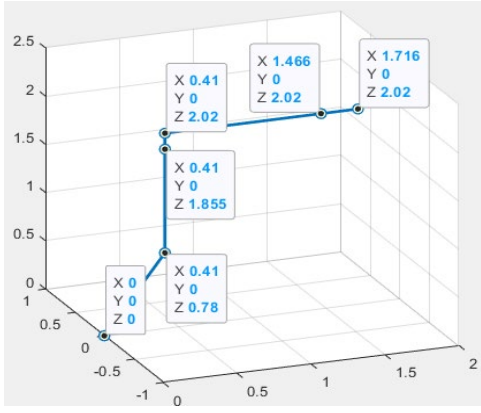


Fig. 5a Robot model using RigidBodyTree

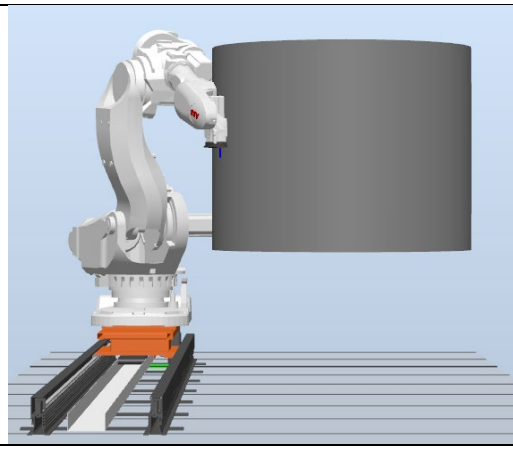


Fig. 5b The robot collision detect

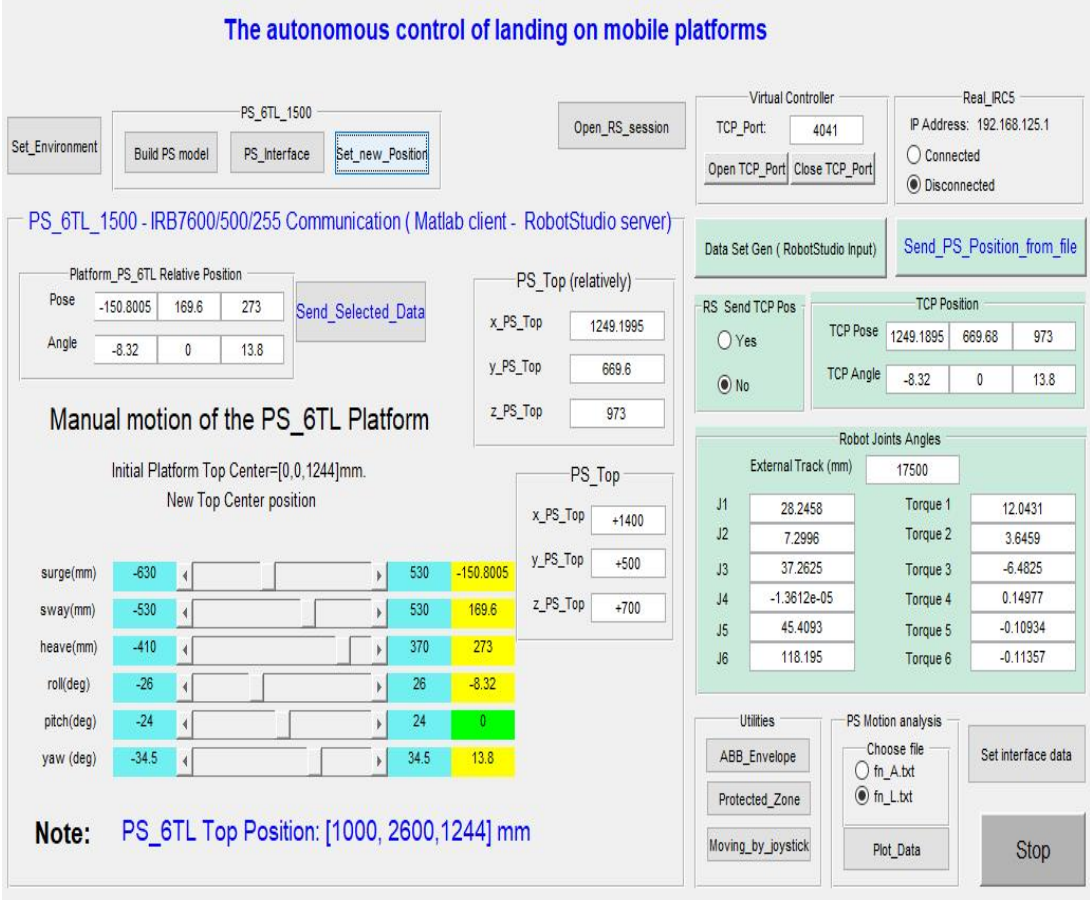
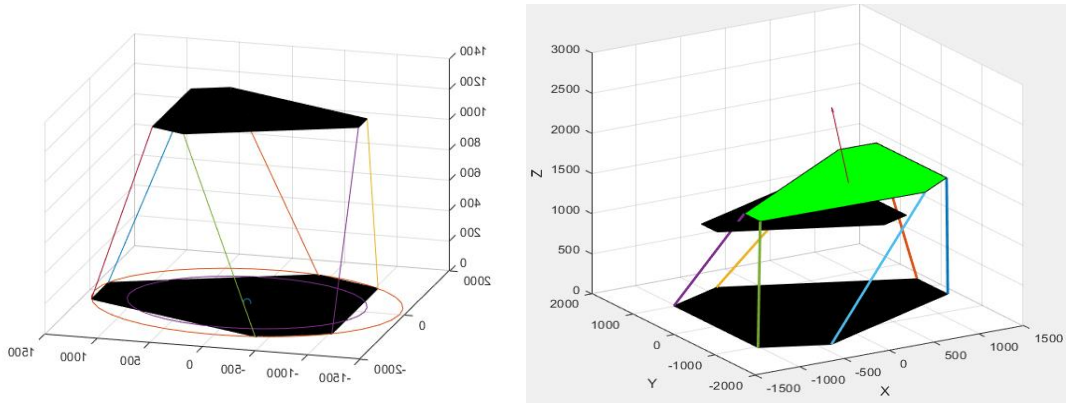


Fig. 6 The application GUI



a) b)

Fig. 7 Platform PS-6TL-1500 initial position a) by sliders position b)

The server application is a module in the RAPID language and controlled by the virtual simulator RobotStudio [12], [16], [17]. The module may contain several procedures.

The collision avoidance procedure was treated independently. The results of the manual simulation can be viewed directly in the interface and by virtual flexpendant.

In case of automatic simulation the results are entered in the file/s text for further analysis.

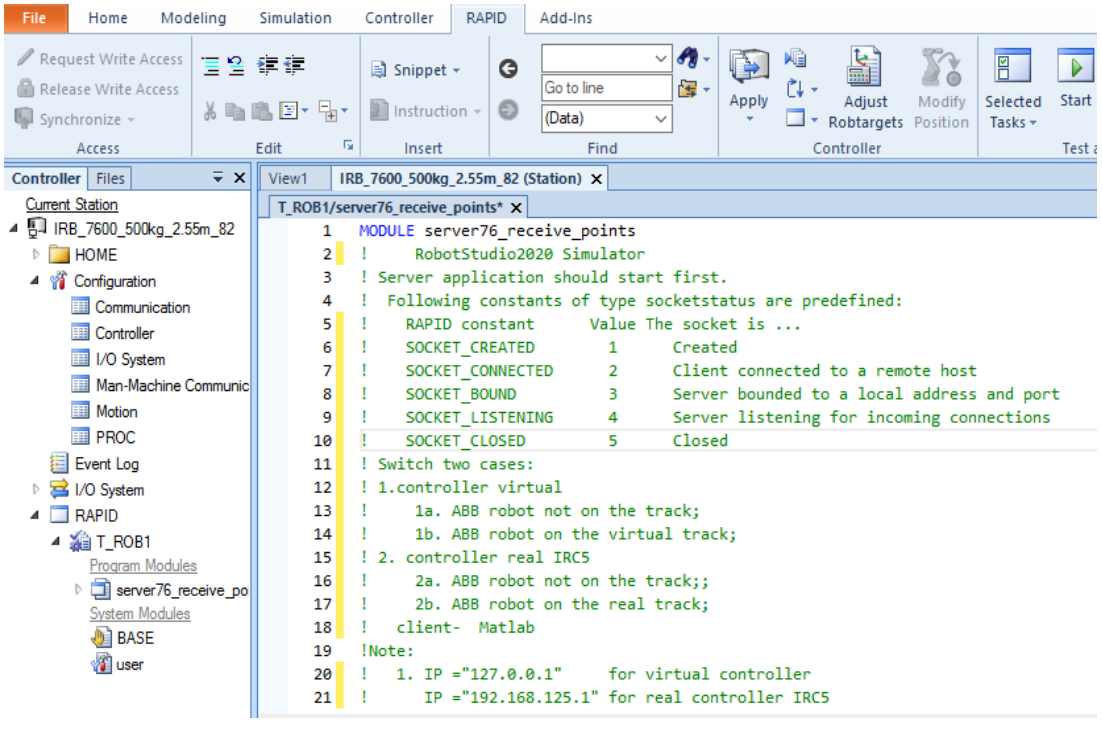


Fig. 8 RobotStudio Interface

```
WHILE TRUE DO
      SocketClose   serverSocket;

         server_state:=SocketGetStatus(serverSocket);
         Write logfile," server_state="+ValToStr(server_state);
      SocketCreate  serverSocket;
      SocketBind    serverSocket,"127.0.0.1", TCP_Port;
      SocketListen  serverSocket;
         server_state:=SocketGetStatus(serverSocket);
         TPWrite " server_state="+ValToStr(server_state);

      SocketAccept serverSocket, clientSocket \Time:=360;
         client_state:=SocketGetStatus(clientSocket);
      SocketReceive clientSocket,\RawData:=raw_data;
         bytes_available:=RawBytesLen(raw_data);
         TPWrite " bytes_available="+ValToStr(bytes_available);

         UnpackRawBytes raw_data,   1,   chx \ASCII:=10; ok:=StrToVal(chx,px);
         UnpackRawBytes raw_data,  11,   chy \ASCII:=10; ok:=StrToVal(chy,py);
         UnpackRawBytes raw_data,  21,   chz \ASCII:=10; ok:=StrToVal(chz,pz);
         UnpackRawBytes raw_data,  31,   chx \ASCII:=10; ok:=StrToVal(chx,q1);
         UnpackRawBytes raw_data,  41,   chy \ASCII:=10; ok:=StrToVal(chy,q2);
         UnpackRawBytes raw_data,  51,   chz \ASCII:=10; ok:=StrToVal(chz,q3);
         UnpackRawBytes raw_data,  61,   chz \ASCII:=10; ok:=StrToVal(chz,q4);

    SocketSend clientSocket \Str:="ok!";
         points.trans:=[px,py,pz];
         points.rot  :=[q1,q2,q3,q4];
    MoveJ points,v1000,fine,tool0\WObj:=wobj0;
         robotdata   := CRobT(\TaskRef:=T_ROB1ID);
         XYZ    :=   ValToStr(robotdata.trans);
         TPWrite "TCP Position= "+XYZ;
    SocketClose   clientSocket;
    SocketClose   serverSocket;
ENDWHILE
```

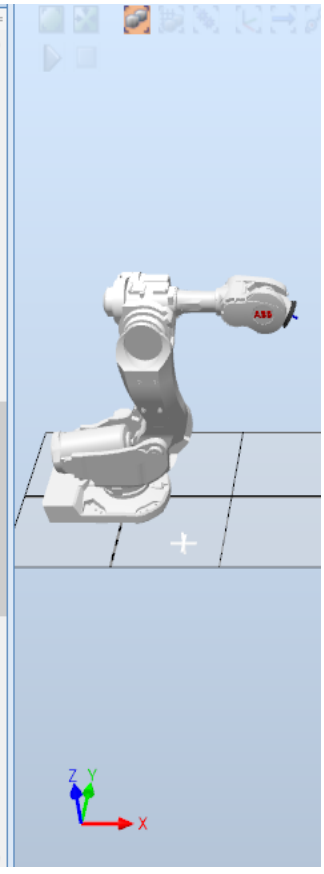


Fig. 9 RAPID statements from moving procedure

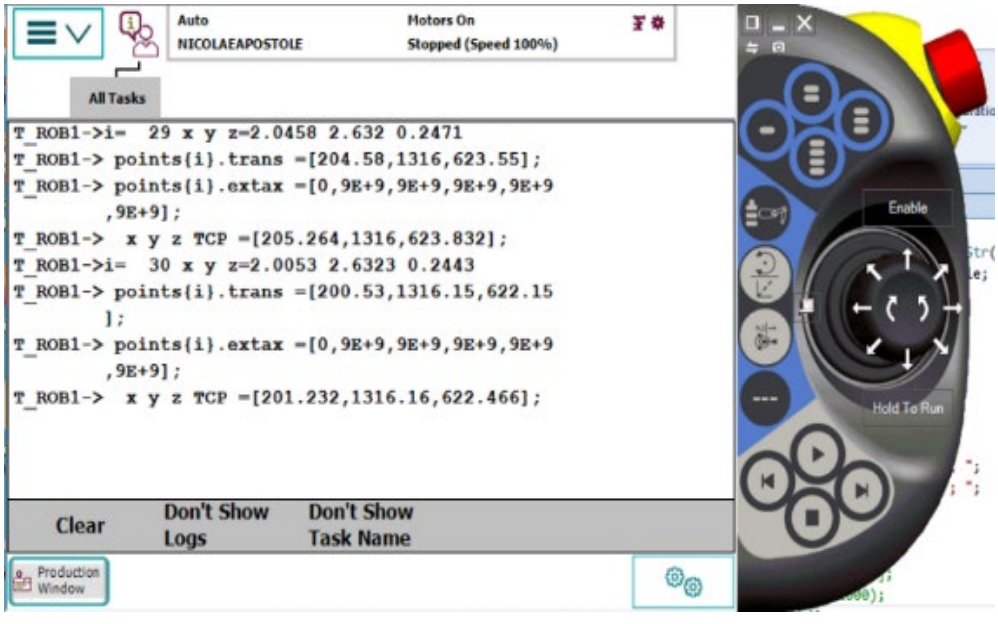


Fig. 10 Virtual Controller FlexPendant

4. EXPERIMENTAL RESULTS

Running the above steps 100 times with the initial position and orientation of platform randomly generated, the accuracy of these solutions is reasonably fair.

Table 1. Experimental results

PS-6TL-1500 position (normal vector)	ABB Endeffector position
P(1) = 1420.00 620.00 1060.00 P(11) = 1306.86 1075.95 1516.18 P(21) = 707.17 1258.32 1207.93	Pw(1, :) = [1420.1, 620.225, 1060.34] Pw(11, :) =[1306.9, 1075.82, 1516.56] Pw(21, :) =[707.78, 1258.29, 1207.91]
PS-6TL-1500 Angles (deg)	
Angle- 1: 46.886 67.626 51.472 Angle- 11 32.323 77.781 60.592 Angle- 21 29.826 84.753 60.733	
Quaternions from client Matlab	
q 1 = [0.7828, 0.1316, 0.6034, 0.0761] q11 = [0.7336, 0.2262, 0.6300, -0.1171] q21 = [0.7036, 0.2112, 0.6581, -0.1652]	
ABB TCP _rotation (deg)	
Angle- 1: 46.888, 67.619, 51.474 Angle-11: 32.350, 77.785, 60.612 Angle-21: 29.861, 84.755, 60.764	
ABB Joints Angles (deg)	
Jw(1, :) =[27.3691, 13.2151, 46.7237, 24.6944, -77.7101, 4.21074] Jw(11, :) =[41.4432, 2.38402, 27.4518, 20.3773, -42.279, -11.2679] Jw(21, :) =[66.4138, -0.61104, 48.0937, 41.7428, -60.714, -24.2151]	
ABB Torques(Nm)	
Tw(1, :) = [2.51576, 2.61331, -6.56382, 0.528206, 0.701321, -0.384403] Tw(11, :) =[-3.04503, 6.98147, -4.73071, 0.423618, 0.558045, 0.348339] Tw(21, :) =[-10.1109, 1.3892, -7.17613, -0.172197, 0.695508, 0.247572]	

5. CONCLUSIONS

Analysing 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 give correct joint angles so that the robot arm with its end effector can easily move to any reachable positions and orientations to allows an aerial vehicle to land autonomously on a moving platform in the presence of uncertainties and disturbances. The application can initiate a landing process on the mobile platform and guide the vehicle for a perfect docking on the platform.

ACKNOWLEDGEMENT

This work is a part of the "Unmanned platforms with dedicated capabilities and support infrastructure for national security mission applications" project contract no. 216/26.01.2017 under funds UEFISCDI, PNCDI III, programme 2, topic 2.1., 2020-2022.

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] N. Apostolescu, T. Savu, D. D. I. Guta, A. Ionita, Industrial robotics for spacecraft rendezvous and docking simulation, *INCAS BULLETIN*, (print) ISSN 2066–8201, (online) ISSN 2247–4528, ISSN–L 2066–8201, vol. **11**, issue 4, https://doi.org/10.13111/2066-8201.2019.11.4.3, 2019.
[3] J. J. Craig, *Introduction to Robotics: Mechanics and Control* (3rd Edition), ISBN 978-0201543612, August 6, 2004.
[4] W. Fehse, *Automated Rendezvous and Docking of Spacecraft*, Cambridge University Press, ISBN-13 978-0-511-07086-0 eBook (EBL), 2003.
[5] R. Kelly, V. Santibáñez and A. Loría, *Control of Robot Manipulators in Joint Space*, Springer-Verlag London Limited 2005, ISBN-10: 1852339942.
[6] D. B. Marghitu, *Mechanisms and Robots Analysis with MATLAB*, Springer Dordrecht Heidelberg, ISBN 978-1-84800-390-3, 2009.
[7] 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.
[8] P. Corke, *Robotics Toolbox for Matlab*, Release 9, 2011.
[9] P. Neto, *A Guide for ABB Robot Studio*, January 2014.
[10] * * * *Robotics System Toolbox,* User's guide, R2019b, Matworks.
[11] * * * *Reference Robotics System Toolbox*™Reference- R2019b.
[12] * * * *Computer Vision System Toolbox*™Reference.
[13] * * * *Robot Kinematics*, February 2017, http://www.wikipedia.com.
[14] * * * The ABB group. http://www.abb.com/, Oct 2003.
[15] * * * *Product specification IRB 7600* Document ID: 3HAC023934-001.
[16] * * * *Operating manual Troubleshooting IRC5 RobotWare 6.08* Doc ID: HAC020738-001.
[17] * * * R*ef manual RAPID Instructions, Functions and Data Types*; Doc ID: HAC050917-001.
[18] * * * Motion Platform PS-6TL-1500 (6DoF, 1500kg) - Motion Systems.
[19] * * * Platform Manager - ForceSeatPM - Motion Systems.