Journal of Machine Engineering, 2020, Vol. 20, No. 2, 53–64
ISSN 1895-7595 (Print) ISSN 2391-8071 (Online)

Received: *31 January 2020* / Accepted: *24 March 2020* / Published online: *24 June 2020*

virtual reality, digital twin,
industrial robots,
inverse kinematics

Vladimir KUTS[1*]
Natalia CHEREZOVA[1]
Martins SARKANS[1]
Tauno OTTO[1]

DIGITAL TWIN: INDUSTRIAL ROBOT KINEMATIC MODEL INTEGRATION TO THE VIRTUAL REALITY ENVIRONMENT

Digital Twin (DT) concept nowadays is shown via the simulations of the manufacturing systems and included those production processes and parametric 3D models of the product. It is the primary method for planning, analysing and optimising the factory layout and processes. Moreover, work on management via the simulation in real-time is already done using Virtual Reality (VR) tools from a safe and remote environment. However, there is a list of limitation of such kind of digital systems, as connectivity speed and precision of the digital environment. The primary goal of this study is to access second listed limitation and on the example of the fully synchronised physical with its digital replica industrial robot, increase the level of precision of the developed DT environment. The proposed approach introduces transfer of the mathematical model to the virtual environment, thus creating a precise and scaled visual model of the Industrial Robot.

1. INTRODUCTION

Manufacturing world robotisation and digitalisation are essential pillars of Industry 4.0 paradigm [1], where the simulation tools are used to design layouts and changes of the equipment placement as well as for simulation of manufacturing processes and product assembly. This concept is called Digital Twins (DT) and was used by NASA already from 1967. However, in the scope of Industry 4.0 paradigm, it was introduced by Grieves [2]. DT is a virtual replica of manufacturing system, processes and product in the means of simulations.

Effective DT is a virtual factory with the ability of not only simulation but also re-configuration and management of the real factory environment, which needs a specific preparation. Various researchers are already implementing this concept in order to obtain precise scalable and controllable simulation environments for direct control over the factory remotely [3–6].

[1] Tallinn University of Technology, School of Engineering, Department of Mechanical and Industrial Engineering, Tallinn, Estonia
[*] E-mail: vladimir.kuts@taltech.ee
https://doi.org/10.36897/jme/120182

Main points of adequate DT preparation are:

1. Precision and Level of Details (LoD);
2. Data acquisition and validation;
3. Data Model;
4. Synchronisation.

Related work refers to the precision of the DT on the example of Industrial Robots. The aim is to control robots from the Virtual Reality (VR) environment in a precise manner in order to enable a local and remote collaborative and safe programming environment [7–11].

The VR solutions are widely used for gaming, training, simulation, programming and other areas, also taking into account industrial needs. One of the directions in the development of VR solutions is the use of the concept of DT that enables to connect the physical device with its digital twin in VR environment. The need for DT is, for example of the programming of the physical device through a VR environment. Industrial robots are widely used for this purpose. To make it possible to "import" the physical industrial robot into VR, its digital twin has to be created. Depending on the needed functionality, different parts of the physical system has to be described. For programming purposes, the kinematics of the industrial robot has to be defined. Definition of inverse kinematics can be done in several ways, like neural network based, soft computing algorithm based and Denavit-Hartenberg (D-H) methodology, naming a few of them. Each methodology has its advantages and shortcomings. In this research, the D-H methodology is used for the analysis of 6-DOF industrial robot, as this methodology requires less computational power and enables to describe the kinematics of the robot by knowing its dimensions and joint configurations. In addition, it makes it possible to describe different robot models in fast and efficient way.

For the Digital Twin solution of the ABB IRB1600-10/1.2, the simulation model of the robot movement is implemented in the VR environment of the Industrial Virtual and Augmented Reality laboratory (IVAR lab) [12]. This paper presents a study on kinematics and programmable motion control of the robotic arm and its implementation using the Unity game engine.

The paper is organised as follows. Kinematic model of the robot is described in Section 2. Inverse kinematics solution is presented in Section 3. Implementation of the derived solution in the VR environment is delivered in Section 4. Evaluation of the results is discussed in Section 5. Conclusions are derived in the last section.

2. ROBOT SPECIFICATIONS AND KINEMATIC MODEL

ABB IRB1600-10/1.2 is an industrial robot with 6 axes, handling the capacity of 10 kg, and reach of 1.2 m. Main application areas of the robot are assembly, arc welding, material handling, cleaning, spraying, dispensing, and packaging. The movement range of the robot axes and axes maximum speeds are presented in Table 1.

The kinematic modelling of the robot was done according to the Denavit-Hartenberg D-H convention; a systematic approach that makes the analysis of the kinematics problem considerably simple and well defined [14–16]. D-H parameters of the ABB IRB1600-10/1.2 robot are presented in Table 2.

Table 1. The movement range of the robot axes and maximum speed [13]

Axis	Type of motion	Range of movement	Axis maximum speed
1	Rotation motion	+180° to –180°	180°/s
2	Arm motion	+110° to –63°	180°/s
3	Arm motion	+55° to –235°	185°/s
4	Rotation motion	+200° to –200°	385°/s
5	Bend motion	+115° to –115°	400°/s
6	Turn motion	+400° to –400°	460°/s

Table 2. Parameters of ABB IRB1600-10/1.2 according to the Denavit-Hartenberg method

Joint number	Link twist angle α_i	Link length a_i	Joint distance d_i	Joint angle θ_i
1	– 90°	a_1	d_1	θ_1
2	0	a_2	0	θ_2
3	– 90°	0	0	θ_3
4	90°	0	d_4	θ_4
5	– 90°	0	0	θ_5
6	0	0	d_6	θ_6

From the robot documentation, parameters values are a_1 = 150 mm, a_2 = 475 mm, d_1 = 486.5 mm, d_4 = 600 mm, d_6 = 65 mm.

The kinematic model of the robot is presented in Fig. 1.

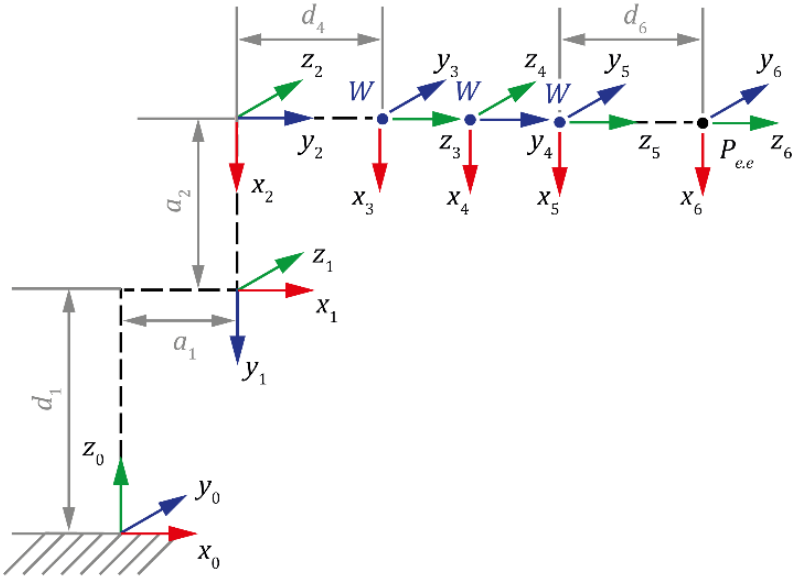

Fig. 1. Kinematic model of the robot: W – wrist point; $P_{e.e}$ – end-effector position

According to D-H convention, each link of the robot is assigned with the coordinate frame. Therefore, the position and rotation of the ith coordinate frame concerning the $i-1$ coordinate frame can be expressed with the use of the homogeneous transformation matrix.

$$\boldsymbol{T}_{\boldsymbol{i-1}}^{\boldsymbol{i}} = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

This way, the successive multiplication of transformation matrices will give the transformation matrix of the end-effector concerning the base coordinate system:

$$\boldsymbol{T}_{\boldsymbol{0}}^{\boldsymbol{n}} = \prod_{i=1}^{n} \boldsymbol{T}_{\boldsymbol{i-1}}^{\boldsymbol{i}} = \begin{bmatrix} R_{e.e} & P_{e.e} \\ 0 & 1 \end{bmatrix}, \tag{2}$$

where $R_{e.e}$ = [n s o] is a 3×3 rotation matrix of the end-effector; $P_{e.e}$ is a 3×1 position vector of the end-effector.

3. INVERSE KINEMATICS SOLUTION

3.1. JOINT VALUES CALCULATION

Kinematics considers the length of each link and joint limits to specify all the possible positions that the robot can achieve. It is split into two parts: forward kinematics and inverse kinematics. Inverse kinematics (IK), on the other hand, determines the joint variables, given the position and orientation of the tool or end-effector and is required to command a robot to a particular point within its workspace. In most cases, there would be several possible solutions for the inverse kinematics problem. However, inverse kinematics is necessary to control the robot through a programme code [14, 17].

The general solution of the inverse kinematics problem, using a homogeneous transformation matrix (2), gives a system of twelve non-linear equations with six unknowns, which will take a significant amount of computation time. For the real-time application, this solution is inappropriate due to its complexity.

However, it is possible to significantly simplify the solution of the inverse kinematics problem, since the robotic arm has a spherical wrist – three consecutive rotational joints with the axes intersecting in one point [18, 19]. This way, the problem can be divided into two parts. First part is the linear solution for the first three joints, based on the position of the so-called wrist point. Wrist point is the point of axes intersection of the joints forming the spherical wrist. The second part is the orientation solution for the spherical wrist, which calculates the angles of the last three joints.

The position of the wrist point can be found as follows:

$$W = P_{e.e} - d_6 o. \tag{3}$$

The geometrical approach was used for both parts.

Geometrical solution for the first three joints. Geometrical model of the problem is presented in Fig. 2.

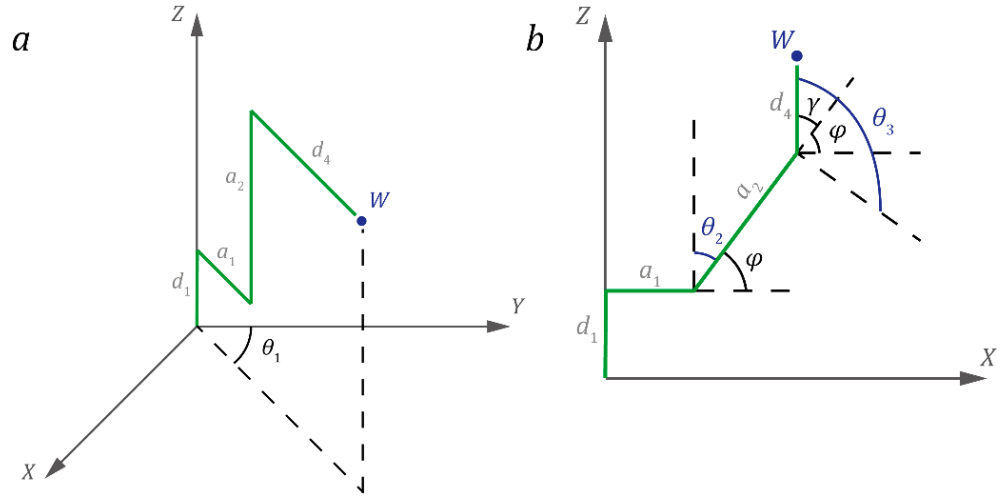

Fig. 2. A geometrical model for the inverse kinematics solution for the first three joints: a) 3D model for the θ_1, b) 2D model for the θ_2 and θ_3

From the figure, it is evident that the solution for the first joint angle is $\theta_1 = \text{atan2}(y_w, x_w)$. The second and third joints angles can be found by solving a system of equations with two unknowns: additional angles φ and γ.

$$\begin{cases} x_w = a_1 \cos\theta_1 + \cos\theta_1(a_2 \cos\varphi + d_2 \cos(\varphi + \gamma)) \\ y_w = a_1 \sin\theta_1 + \sin\theta_1(a_2 \cos\varphi + d_2 \cos(\varphi + \gamma)) \\ z_w = d_1 + a_2 \sin\varphi + d_4 \sin(\varphi + \gamma) \end{cases}. \tag{4}$$

Rearranging the terms gives

$$(x_w - a_1 \cos\theta_1)^2 + (y_w - a_1 \sin\theta_1)^2 + (z_w - d_1)^2 = a_2^2 + d_4^2 + 2a_2 d_4 \cos\gamma. \tag{5}$$

This way,

$$\cos\gamma = \frac{(x_w - a_1 \cos\theta_1)^2 + (y_w - a_1 \sin\theta_1)^2 + (z_w - d_1)^2}{2a_2 d_4}, \tag{6}$$

$$\pm \sin\gamma = \pm\sqrt{1 - \cos^2\gamma}, \tag{7}$$

$$\gamma = \text{atan2}(\sin\gamma, \cos\gamma) \quad \text{or} \quad \gamma = \text{atan2}(-\sin\gamma, \cos\gamma). \tag{8}$$

To find φ, equations from the system (4) were rearranged to correspond to the matrix form of the system of linear equations $A\mathbf{x} = \mathbf{b}$, where A is a matrix of coefficients, $\mathbf{x}$ is a column vector of unknowns, and $\mathbf{b}$ is a column vector of constant terms.

$$\begin{pmatrix} x_w \cos\theta_1 + y_w \sin\theta_1 - a_1 \\ z_w - d_1 \end{pmatrix} = \begin{pmatrix} a_2 + d_4 \cos\gamma & d_4 \sin\gamma \\ d_4 \sin\gamma & a_2 + d_4 \cos\gamma \end{pmatrix} \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix}, \tag{9}$$

where

$$A = \begin{pmatrix} a_2 + d_4 \cos\gamma & d_4 \sin\gamma \\ d_4 \sin\gamma & a_2 + d_4 \cos\gamma \end{pmatrix},$$

$$\mathbf{x} = \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix},$$

$$\mathbf{b} = \begin{pmatrix} x_w \cos\theta_1 + y_w \sin\theta_1 - a_1 \\ z_w - d_1 \end{pmatrix}.$$

Since A is a square 2×2 matrix and has full rank, the solution to the system is $\mathbf{x} = A^{-1}\mathbf{b}$. Therefore,

$$\begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} = \frac{1}{(a_2 + d_4\cos\gamma)^2 + d_4^2 \sin^2\gamma} \times \\ \times \begin{pmatrix} a_2 + d_4\cos\gamma & d_4\sin\gamma \\ -d_4\sin\gamma & a_2 + d_4\cos\gamma \end{pmatrix} \begin{pmatrix} x_w\cos\theta_1 + y_w\sin\theta_1 - a_1 \\ z_w - d_1 \end{pmatrix}, \tag{10}$$

$$\cos\varphi = \frac{(a_2 + d_4\cos\gamma)(x_w\cos\theta_1 + y_w\sin\theta_1 - a_1) + d_4\sin\gamma\,(z_w - d_1)}{a_2^2 + 2a_2 d_4\cos\gamma + d_4^2}, \tag{11}$$

$$\sin\varphi = \frac{-d_4\sin\gamma\,(x_w\cos\theta_1 + y_w\sin\theta_1 - a_1) + (a_2 + d_4\cos\gamma)(z_w - d_1)}{a_2^2 + 2a_2 d_4\cos\gamma + d_4^2}. \tag{12}$$

This way,

$$\varphi = \text{atan2}(\sin\varphi\,, \cos\varphi) \quad \text{or} \quad \varphi = \text{atan2}(-\sin\varphi\,, \cos\varphi). \tag{13}$$

Geometrical solution for the last three joints. To simplify the solution and to reduce the computational time, it was decided not to calculate the sixth joint angle. Since fourth and sixth joints are rotated the same way, that is to say, their axes of rotation are parallel; the specified TCP position can be achieved rotating only one of them. The rotation of the sixth joint is essential in the situations when the tool should additionally rotate while the robot arm is moved along some path. However, for the demonstration purpose, this complexity is unnecessary.

To find the fourth and the fifth joints angles, it was decided to use polar coordinate system equations (Fig. 3). For that, coordinates of the end-effector concerning the wrist point are used. The polar coordinate system is a two-dimensional coordinate system based on a distance from a reference point and an angle from a reference direction on the plane. For the θ_4, the plane of interest would be *ZX*-plane; for the θ_5, it would be the moving $Y\rho$-plane.

Therefore,

$$\theta_4 = \text{atan2}(x_e, z_e), \tag{14}$$

$$\theta_5 = \text{atan2}(\rho_e, y_e), \tag{15}$$

where

$$\rho_e = \sqrt{(x_e)^2 + (z_e)^2}. \tag{16}$$

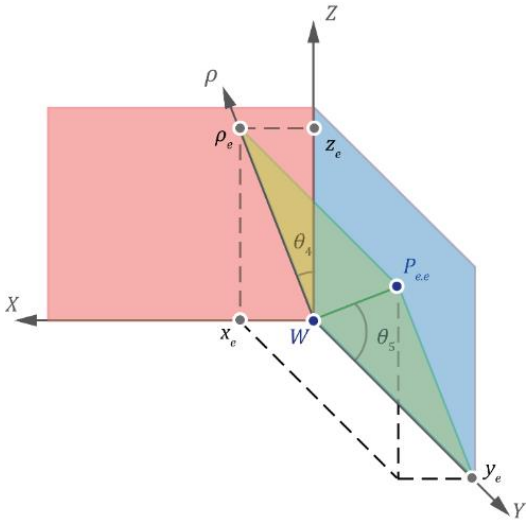

Fig. 3. A geometrical model for the fourth and fifth joints

3.2. VELOCITY CALCULATIONS

Another problem connected to the inverse kinematics is the joints velocity calculations. Since the programmed commands specify the linear velocity of the end effector, before starting the movement, it is necessary to calculate the joints rotational velocities. Usually, the Jacobian matrix composed of the partial derivatives of the functions $x_{e.e}(\theta_i), y_{e.e}(\theta_i)$ and $z_{e.e}(\theta_i)$ is used. Jacobian represents the relationship between the linear motion of the end-effector and the rotational motion of individual joints [14, 17].

However, due to the nature of the Unity engine, which runs the application in frames, it is possible to avoid the calculation of Jacobian and to use the time between the frames. This way, for the linear movement, where the path of the end-effector is known, velocity is calculated simply dividing the angle, by which each joint should rotate to get to the specified point, by the time it should take the end-effector to get to that point with the specified linear velocity. This method gives an approximation of the joints' velocities changes throughout the linear movement.

For the joint movement, where the path of the end-effector is not necessarily linear and not known, the joints' velocities were calculated in the following manner. Since all the joints should start and end the movement at the same time, first, the change of the joints' values is normalised by the common factor, and relative joints' velocities concerning each other are found. As a common factor, the most significant change among the joints is taken. Obtained values will be referred to as a step. Then, using forward kinematics, the position of the end-effector, if the joints will rotate one step from the current position, is found. The distance d between the two positions is calculated. The time it will take the end-effector to move through this distance t_{step} is calculated as:

$$t_{\text{step}} = \frac{d}{v_{e.e}}, \tag{17}$$

where: $v_{e.e}$ is the linear velocity of the end-effector.

The rotational velocity r_i of each joint is then

$$r_i = \frac{\theta_{\text{step}}}{t_{\text{step}}}, \tag{18}$$

where θ_{step} is the step value the joint needs to rotate to.

4. IMPLEMENTATION OF THE VR SIMULATION

The simulation was implemented using Unity engine, widely used for game production, however, offering great opportunities and toolsets for manufacturing visualisation and simulation. One-to-one scale robot model with precisely placed pivot points acting as original points of joints' frames are placed in the virtual environment of the IVAR laboratory (see Fig. 4). Implemented virtual robot control gives the possibility to simulate the writing of a simple RAPID program using movement functions (MoveL for linear movement and MoveJ for joint movement) to be executed inside the virtual environment. RAPID is a proprietary programming language used to control ABB robots. Moreover, this solution enables to simulate robotics systems without connection with physical equipment; this way, it can be used in a safe manner for training of new operators of the robots.

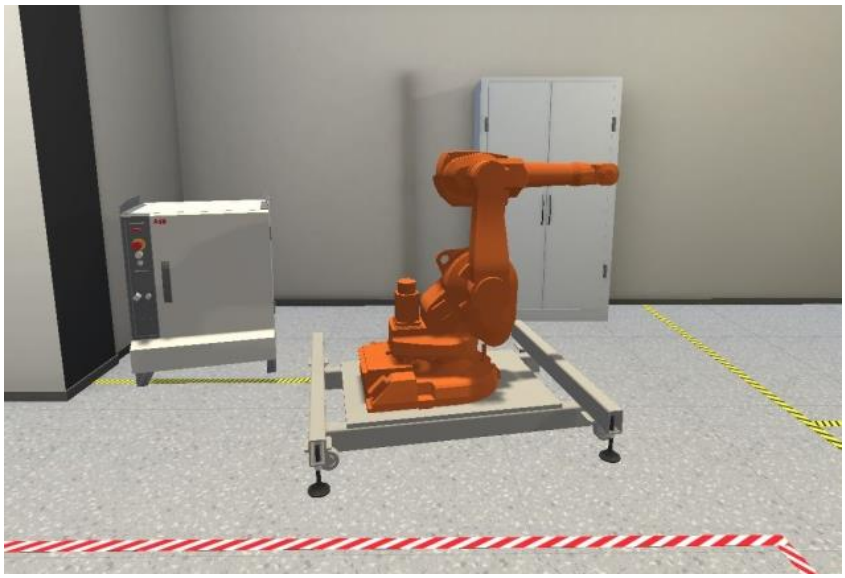

Fig. 4. ABB IRB1600-10/1.2 model in the VR environment

The syntax for the MoveL and MoveJ instructions is very similar:

MoveL Point, Speed, Zone, Tool,
MoveJ Point, Speed, Zone, Tool,

where: Point defines the target point of the movement, it is given as the coordinates of the central tool point (TCP) or end-effector usually in the base coordinate system; Speed defines

the linear speed of the TCP in mm/s; Zone defines a corner zone in mm for the point approaching motion; and Tool defines the tool the robot is using [20]. Zone data is used to smooth the long continuous motion of the robot arm through several defined points, to avoid high mechanical loads on the manipulator due to the sudden change in TCP movement. However, it can also be used to make sure that the TCP will reach the exact point position if it is of importance.

Inverse kinematics solution is handled by the script, containing two main methods:

- public void FirstThreeJoints (calculates rotational angles for the first three joints; takes the coordinate vector of the target wrist point and references to the θ_1, θ_2, and θ_3 as a parameter);
- public void LastThreeJoints (calculates rotational angles for the last three joints; takes the coordinate vector of the TCP point concerning the wrist point coordinate system and references to the θ_4, θ_5, and θ_6 as a parameter).

Writing of the RAPID programme is handled by another script that implements two customised types:

- structure Parameters (the structure that contains the parameters of the movement functions);
- class UIprocessing (class used to generate the RAPID programme).

The programme is stored in the public static list of type KeyValuePair <string, Parameters> objects called functionList. This way, each object of the list is a pair of the movement function name (as a string object) and its parameters (as a Parameters object). Elements are added to list consequently as the user adds commands to the programme.

Since the inverse kinematics solution is done based on two points coordinates: wrist point position for the first three joints and TCP position concerning wrist point for the last three joints. Equation (3) shows how to obtain the coordinates of the wrist point from the TCP frame coordinates; however, with the Unity nature, it is not necessary. The position of each point can be obtained using its Transform parameters; additionally, the position of the point relative to the specific coordinate frame can be found using InverseTransformPoint method.

Programmable control of the virtual robot model is fulfilled by the robot control script implementing. The script is attached to the parent object of the robot. The attributes of the class keep information about each joint object, joints' angles and current speeds, the target position of the TCP and target angles of the joints.

The automatic movement of the joints is performed by the following functions:

- IEnumerator ProgramHandler (executes the RAPID programme by starting the necessary coroutine and passing the arguments to it);
- IEnumerator MoveL (executes the linear movement function of the robot arm);
- IEnumerator MoveJ (executes the joint movement function of the robot arm);
- private void MoveJoints (moves joints automatically to the target angle; utilised by the joints movement coroutine);
- private void FirstThreeJointsLinearMovement (performs the rotation of the first three joints for the linear movement);
- private void LastThreeJointsLinearMovement (performs the rotation of the last three joints for the linear movement).

Coroutine MoveJ successively calls the inverse kinematics solver functions to find the target angles for all the joints, calculates appropriate velocities for each joint and then executes the MoveJoints function.

Coroutine MoveL executes the linear motion in the following way. First, the direction vector of the path line is found, the length of the line, and the time it will take the wrist point to get from the starting point to the target point. Then the movement of the arm and, therefore, the solution is divided into two parts: the linear motion of the wrist point driven by the first three joints and rotational motion of the end-effector driven by the last two joints. This approach significantly reduces the computational time of the inverse kinematics calculations during the motion, since the angles for the fourth and fifth joints should be calculated only once when the linear movement starts.

For the linear movement of the wrist point, several points are calculated along the path line with the step of 50 mm, thus avoiding overload of the application and not forcing to do the joints' angles calculations every frame, but only once in a while. Step range can be reduced based on application needs, but as for experiment and proof of concept gathered results were enough for evaluation. For each new points, the joints variables are calculated using the approach described in the previous section. Then the time needed for the joints to reach the target angles is calculated, and the speed for each joint is derived. After that, each joint starts to move towards the calculated point. Therefore, essentially linear movement is composed of several short joint movements, forming, in the end, the approximation of the straight line.

5. ANALYSIS

A set of ten points was tested to inspect the accuracy of the implemented algorithm for the virtual simulation. Mean absolute error (MAE) was calculated. It is an average of the absolute errors calculated by the formula

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|, \tag{19}$$

where e_i is the absolute error of ith observation; $\hat{y}_i$ is the ith observed value; y_i is the ith original value; n is the number of observations [21].

The results are presented in Table 3 in the following form: the coordinates of the TCP that the robot was sent to are compared to the coordinates of the TCP that the robot model has reached using the implemented IK solver.

It can be seen that MAE is around 4 mm; therefore, the implemented inverse kinematics algorithm is sufficient in experiment for robot being used for 3D inspection purposes. However, to reduce it deeper optimization of the kinematic model accuracy shall be performed, which is one of the future steps for related research. The precision can be increased by optimising mathematical model by Generic Evolutionary Inverse Kinematics for Unity3D when using Bio IK algorithm.

Table. 3. MAE of the implemented algorithm test

Point number	Original point, m			Obtained point, m		
	X	*Y*	*Z*	*X*	*Y*	*Z*
1	1.07125	0.81961	–0.12217	1.06272	0.80799	–0.12375
2	1.15335	0.93264	0.12659	1.14030	0.90149	0.11096
3	0.30981	1.23879	0.81669	0.31499	1.22345	0.81719
4	0.07158	1.46461	0.70210	0.07142	1.44980	0.70888
5	–0.74896	0.93126	–0.03171	–0.74968	0.93367	–0.03177
6	–0.70893	0.97986	0.49418	–0.70946	0.98237	0.49444
7	0.96523	1.33437	0.56625	0.96524	1.33454	0.56625
8	1.25531	1.01683	–0.17813	1.25546	1.01753	–0.17816
9	0.63438	1.38530	0.73518	0.63439	1.38519	0.73519
10	1.00014	0.67190	0.58504	1.00034	0.67265	0.58517
MAE	0.00435					

6. CONCLUSION

Implemented simulation is a part of the Digital Twin solution for the ABB IRB1600-10/1.2 robot. Connection was set between the real robot and its digital copy making it possible to control the real robot using virtual model kinematics, and obtain the data from the real robot, while it is controlled from the Flex Pendant, for example, to copy its movement to the virtual one.

Moreover, it was possible to simulate the movements without connecting to the real robot; this way, it can be used in future for the training and show-case purposes to demonstrate how the programmable control of the robot works.

ACKNOWLEDGEMENTS

This research was supported by project AR16077 Smart Industry Centre (SmartIC) No. 2014-2020.4.01.16-0183, supported by the EU Regional Development Fund. The authors are grateful to the student of the Tallinn University of Technology – Yevhen Bondarenko, for his help in the experiments with Virtual Reality implementations and laboratory digitalisation.

REFERENCES

[1] LU Y., 2017, *Industry 4.0: A survey on technologies, applications and open research issues*, Journal of Industrial Information Integration, 6, 1–10.

[2] GRIEVES M.W., 2015, *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*, White Paper.

[3] TERKAJ W., TOLIO T., URGO M., 2015, *A virtual factory approach for in situ simulation to sup-port production and maintenance planning*, CIRP Annals – Manufacturing Technology, 451–454.

[4] TAO F., ZHANG M., 2017, *About the importance of Autonomy and Digital Twins for the future of manufacturing*, IEEE Access – Special Section on Key Technologies for Smart Factory of Industry 4.0, 20418–20427.

[5] KUTS V., OTTO T., TÄHEMAA T., BONDARENKO Y., 2019, *Digital Twin based synchronised control and simulation of the industrial robotic cell using Virtual Reality*, Journal of Machine Engineering, 19/1, 128–145.
[6] MAHMOOD K., SHEVTSHENKO E., 2015, *Analysis of machine production processes by risk assessment approach*, Journal of Machine Engineering, 15/1, 112−124.
[7] KUTS V., OTTO T., TÄHEMAA T., BUKHARI K., PATARAIA T., 2018, *Adaptive industrial robots using machine vision*, ASME International Mechanical Engineering Congress and Exposition, Pittsburgh, Pennsylvania, USA.
[8] SELL R., OTTO T., 2008, *Remotely controlled multi robot environment*, 19th EAEEIE Annual Conference, Tallinn.
[9] SELL R., 2013, *Remote Laboratory Portal for Robotic and Embedded System Experiments*, International Journal of Online Engineering, 9, 23−26.
[10] KUTS V., SARKANS M., OTTO T., TÄHEMAA T., 2017, *Collaborative work between human and Industrial robot in manufacturing by advanced safety Monitoring System*, Proceedings of the 28th DAAAM International Symposium, Vienna.
[11] KUTS V., TÄHEMAA T., OTTO T., SARKANS M., LEND H., 2016, *Robot manipulator usage for measurement in production areas*, Journal of Machine Engineering, 16/1, 57−67.
[12] Industrial Virtual and Augmented Reality Laboratory Homepage, http://ivar.ttu.ee/, last accessed 2020/01/27.
[13] ABB Robotics, 2018, *Product specification IRB 1600/1660*, Sweden.
[14] ASADA H.H., 2005, *Introduction to Robotics*, Lecture Notes. Massachusetts Institute of Technology.
[15] SHAH S.V., SAHA S.K., DUTT J.K., 2012, *Denavit-Hartenberg Parameterization of Euler Angles*, Journal of Computational and Nonlinear Dynamics, 7/2, 146–152.
[16] TZAFESTAS S.G., 2013, *Introduction to Mobile Robot Control*, 1st ed. Elsevier, Amsterdam.
[17] CRAIG J., 2005, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Pearson Education International, New Jersey.
[18] MEGAHED S.M, 1992, *Inverse kinematics of spherical wrist robot arms: Analysis and simulation*, Journal of Intelligent and Robotic Systems, 5/3, 211–227.
[19] KUCUK S., BINGUL Z., 2004, *The inverse kinematics solutions of industrial robot manipulators*, Proceedings of the IEEE International Conference on Mechatronics, Istanbul, 274–279.
[20] ABB Robotics, 2017, *Technical reference manual: RAPID Instructions, Functions and Data Types*, Sweden.
[21] WILLMOTT C., MATSUURA K., 2005, *Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in assessing average model performance*, Climate Research, 30, 79–82.