Digital Twins of Robotic Systems: Increasing Capability for Industrial Applications

Tran Tuan Anh, Nguyen Thanh Tan, Dinh Than Le, Le Chi Hieu, Jamaluddin Mahmud, M. J. A. Latif, and Nguyen Ho Quang

Abstract Digital twin is one of the emerging areas of research and technology development and the enabling technologies of Smart Manufacturing and Industry 4.0. This study aims to develop and demonstrate a proof-of-concept prototype with a case study of the digital twin of a robotic system. The system has two main elements: the virtual element and the physical or the real element. The virtual element of system has been built based on the Unity platform, which is a cross-platform game engine developed by Unity Software Inc., and the physical element was built with the use of two servomotors and the NVIDIA® Jetson Nano™ Developer Kit. The virtual and the physical elements are connected and communicated via using the TCP socket protocol suite. A digital twin model of the ABB IRB 120 robot was successfully

T. T. Anh · N. T. Tan · D. T. Le · N. H. Quang (✉)
Institute of Engineering and Technology, Thu Dau Mot University, Thu Dau Mot City, Binh Duong Province, Vietnam
e-mail: quangnh@tdmu.edu.vn

D. T. Le
e-mail: than.ld@ieee.org

D. T. Le
Artificial Intelligence Laboratory, Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam

L. C. Hieu
Faculty of Engineering and Science, University of Greenwich, Kent ME4 4TB, UK
e-mail: c.h.le@gre.ac.uk

J. Mahmud
School of Mechanical Engineering, College of Engineering, Universiti Teknologi MARA, Shah Alam, Malaysia
e-mail: jm@uitm.edu.my

M. J. A. Latif
Fakulti Kejuruteraan Mekanikal, Universiti Teknikal Malaysia Melaka (UTeM), Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia
e-mail: juzaila@utem.edu.my

Advanced Manufacturing Centre (AMC), Universiti Teknikal Malaysia Melaka (UTeM), Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

T. D. L. Nguyen and J. Lu (eds.), *Machine Learning and Mechanics Based Soft Computing Applications*, Studies in Computational Intelligence 1068,
https://doi.org/10.1007/978-981-19-6450-3_23

developed and demonstrated. The collected data include the joint angle position values of the physical and virtual models, and they are stored both locally and in the cloud for the future system development, which can be used as for minimizing the errors between the physical and virtual models of digital twins of robotic systems. The successfully developed digital twin model can be considered as the cost-effective solutions for demonstrating and evaluating potential applications of digital twins in industrial practices as well as in higher educations and research.

Keywords IoT · Coffee disease · Coffee farm · Environmental factor

1 Introduction

A digital twin [1, 2] is a digital duplicate of a physical entity whose base is infrastructure and data and at its core are algorithms and models and applications software and services. This technology along with robotics has been growing rapidly in recent years and is considered by research and industry to be the main driving factors for Industry 4.0. Under the development of artificial intelligence (AI) [3–7], robotics [8–11] has made breakthrough developments in the capabilities of robots such as performing complex jobs with high danger, taking care of health, cooperation with people, and more. The combination of digital twin and robot applying new technologies and algorithms will be the foundation for building and developing smart manufacturing. The trends of using AI to analyze data and make prediction are being special care.

The Robot Operating System (ROS) platform is a collection of tool software libraries that help build robotic applications with source code [12]. The control algorithms tend to apply ML to increase performance and optimization process. In this study, ROS is used as middleware to connect the virtual and real parts and makes it possible to apply platforms like motion planning [13–16], autonomous mobile robotics [17, 18].

A digital twin provides various features such as visualization, simulation with real data, and performance monitoring. To do that, the amount of data collected must be stored. Firebase is a cloud-based database service, accompanied by an extremely powerful server system of Google. Firebase provides capabilities such as analytics, databases, activity reports, and error reporting for easy development. It has versatile services and trusted security. Therefore, the data in this project, namely the joint values and the execution time, are all stored using Firebase.

The main contribution of this paper includes data storage cloud which are the limitations of the recent research papers [19–22]. In addition, we build virtual environment, calculated the kinematics, assure real time two-way data transmission between virtual and analyze and evaluate the obtained data.

In this paper, we divide into five main sections. We first build a virtual model and a virtual environment for the robot. Second, we build the real part system for the robot. Third, we connect and transfer data between the real and the virtual models.

Fourth, we calculate the kinematics for the robot. Finally, the data will be transmitted in both directions to control, evaluate errors, and store the data on both local storage and the cloud.

2 Related Works

2.1 Proposed Framework

We have built a digital duplicate for a robot architecture. It is divided into two components: virtual and real parts. The virtual element is a software built on the Unity platform. For the real part, we use two servomotors with built-in encoder to represent the two joints of the robot. These two motors are controlled via a PID controller. In addition, we also use Jetson Nano with integrated ROS as a central processor. The virtual element and the real part are connected to each other through the TCP socket protocol suite.

Jetson Nano is a small but very powerful computer that allows to run multiple neural networks in parallel for applications such as image classification, object detection, segmentation, and speech processing. Being in one platform, it is easy to use and consumes less than 5 W. The Jetson Nano also delivers 472 GFLOPS to run modern AI algorithms quickly, with a 64-bit ARM quad-core CPU, an onboard 128-core NVIDIA GPU, as well as 4 GB of LPDDR4 memory. It is possible to run multiple neural networks in parallel and handle several high-resolution sensors simultaneously. Here, users can control from the virtual model to real model and vice versa. First, we will present the control process from virtual to real model. When the user changes the robot's joint position via the Unity controller, the matching angle data will be sent through the central processor. The central processor will calculate the forward kinematics to give the position of the end of the manipulator. Next, the matching angles will be controlled by the Virtual robot model, and the data is also given to the user interface to display the joint angle values and the coordinates of the robot's manipulator end point. After that, the data will be sent to the Physical robot to control the real model through the TCP socket protocol suite.

In this study, we take two servomotors representing two joints of the robot. The data will be sent to the central processor and then sent to the controller to control the servomotor. Then the controller will take the position value of the servomotor and send it to the central processor. The central processor will process the data through the aforementioned formula and calculate the error between the matching angles of the real and imaginary model. The data will then be stored on Cloud and sent back to Unity which calculates the delay and then saves the data including the error, the position of the real and virtual model's joints, and the delay to local storage. For the control process from real to virtual object, it will reverse the above process and start from the controller at ROS. In this process, the Unity side controller will be ignored

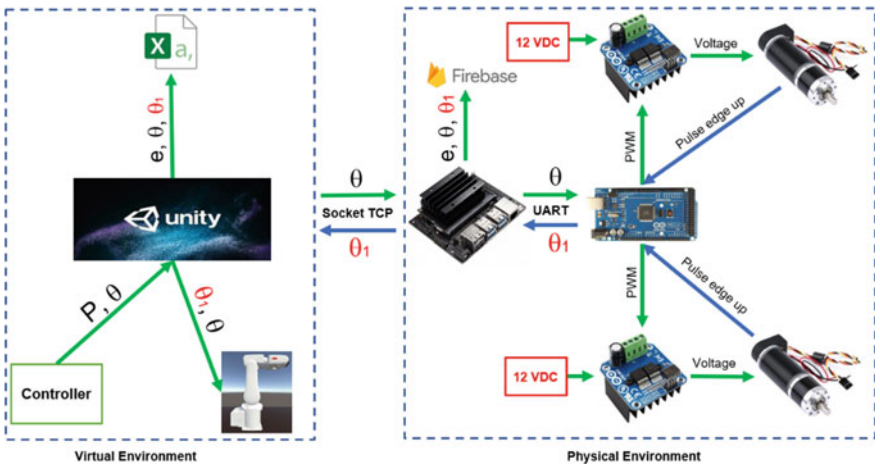


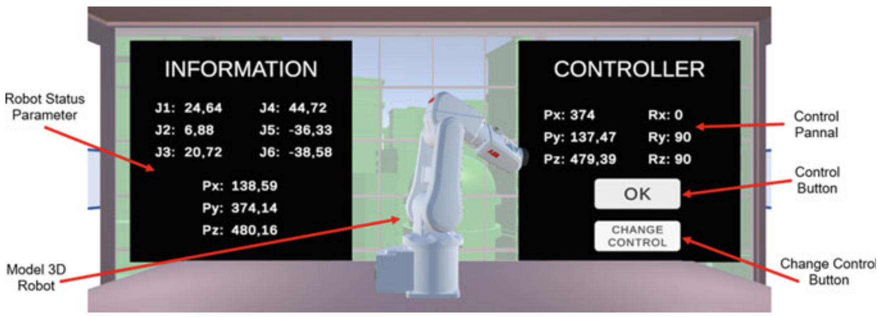
Fig. 1 System overview. The system consists of two parts: real and virtual models. On the left is a virtual element system built on the Unity platform. On the right is the real part system

and the robot's matching angle values will be fed directly into the Virtual robot model through Unity's central processor. Figure 1 is the architecture of the system.

2.2 *Virtual Environment*

We use the Unity platform to build and develop virtual environments. Unity is software that specializes in games, so building a virtual environment is almost easy. In addition, Unity also supports virtual reality and augmented reality for future development of the system. To build a virtual environment on Unity, we used available tools and some other graphics software like Blender. After building the objects of the virtual environment, we started to set up features such as data communication, kinematics, and rotation controller for the robot. In order to build a robot model and put it into a virtual environment, we first downloaded the details of the ABB IRB 120 robot model from the ABB homepage. Second, we edit and stitch the parts together into a robot ABB IRB 120. Third, we set the coordinate axes for each joint of the robot model. Fourth, we set the parameters of the model. Fifth, we exported the model as a URDF file and stored it in the directory where the project is located. Finally, we bring that model into the Unity virtual environment.

In the virtual environment with the Virtual robot model, the central processor will process the data, store the data in local memory, and transmit the data to the real model, meanwhile the controller helps the user to change the angles. This virtual environment allows the data visualization, monitoring, and control. Figure 2 shows the virtual environment built.



INFORMATION

J1: 24,64 J4: 44,72
J2: 6,88 J5: -36,33
J3: 20,72 J6: -38,58

Px: 138,59
Py: 374,14
Pz: 480,16

CONTROLLER

Px: 374 Rx: 0
Py: 137,47 Ry: 90
Pz: 479,39 Rz: 90

Fig. 2 Virtual environment in Unity includes robot model, robot status panel, and control panel

3 Approach: Digital Twin of Joint Angle of Robot Arm

3.1 Compare Inverse Kinematics Between the Construction Algorithm and Robot Studio

After building the inverse kinematics algorithm, we compare the inverse kinematics with Unity, MATLAB, and Robot Studio. We found that the coordinates of Unity, the algorithm we built, and Robot Studio were interchanged. The joints values have very low errors. These errors are due to the rounding process in the algorithm's calculation. The results showed that our inverse kinematic calculation is valid within very acceptable limits of error. Table 1 is the test value obtained after testing.

3.2 Build a PID Controller

A PID controller is built for the purpose of controlling the motor rotation angle. Planetary GP36 Planetary Reducer DC Servomotor with 1:27 deceleration ratio with 145 rpm integrated Optical Encoder 500 CPR (count per round) and two channels A–B has been used. From here, we get the formula to convert between pulse count to angle with angle being the current angle and p being the count of pulses, c is the number of pulses of the rising or falling edge when the motor is turn 1 revolution, and r is rate of motor reducer:

$$\alpha(x) = \frac{p}{\mathrm{c} \times \mathrm{r}} \times 360 \tag{1}$$

$$= \frac{p}{500 \times 27} \times 360 = \frac{p}{37.5} \tag{2}$$

The PID controller for the motor includes three parameters: K_p; K_i; and K_d. The above parameters are determined by the method of manual adjustment through

Table 1 Evaluation of inverse kinematics

No	Manipulator end point coordinates	Rotation angle (°)	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Software
1	X: 374	R_X: 0	24.48	6.8	20.96	44.34	−36.35	−38.21	Robot Studio
	Y: 137.47	R_Y: 90	24.47	6.8	20.96	44.34	−36.35	−38.21	Unity
	Z: 479.39	R_Z: 90	24.48	6.8	20.96	44.34	−36.35	−38.21	MATLAB
2	X: 374	R_X: 0	−38.4	17.87	7.78	−61.37	−45.05	52.3	Robot Studio
	Y: −239.4	R_Y: 90	−38.4	17.87	7.78	−61.37	−45.05	52.3	Unity
	Z: 479.39	R_Z: 90	−38.4	17.87	7.78	−61.37	−45.05	52.3	MATLAB
3	X: 420.12	R_X: 0	−22.93	19.01	−26.64	72.59	24.1	−71.04	Robot Studio
	Y: −147.3	R_Y: 90	−22.93	19.01	−26.64	72.59	24.1	−71.04	Unity
	Z: 654.72	R_Z: 90	−22.93	19.01	−26.64	72.59	24.1	−71.04	MATLAB
4	X: 420.12	R_X: 0	25.41	20.95	−29.2	−73.21	26.63	71.34	Robot Studio
	Y: 165.37	R_Y: 90	25.41	20.95	−29.2	−73.21	26.63	71.34	Unity
	Z: 654.72	R_Z: 90	25.41	20.95	−29.2	−73.21	26.63	71.34	MATLAB
5	X: 535.17	R_X: 0	−14	38.4	−28.05	−54.23	−17.35	52.96	Robot Studio
	Y: −115.5	R_Y: 90	−14	38.4	−28.05	−54.23	−17.35	52.96	Unity
	Z: 516.22	R_Z: 90	−14	38.4	−28.05	−54.23	−17.35	52.96	MATLAB
6	X: 389.93	R_X: 0	26.49	22.06	−44.05	−53.08	33.91	47.84	Robot Studio
	Y: 158.43	R_Y: 90	26.49	22.06	−44.05	−53.08	33.91	47.84	Unity
	Z: 718.23	R_Z: 90	26.49	22.06	−44.05	−53.08	33.91	47.84	MATLAB
7	X: 454.39	R_X: 0	−24.88	25.73	−23.7	−85.64	−24.95	85.19	Robot Studio
	Y: −177.3	R_Y: 90	−24.88	25.73	−23.7	−85.64	−24.95	85.19	Unity
	Z: 592.51	R_Z: 90	−24.88	25.73	−23.7	−85.64	−24.95	85.19	MATLAB

experiment, respectively, $K_p = 0.01$; $K_i = 0.002$; and $K_d = 0.001$. The built-in PID controller is used to control the rotation position of the motor with the feedback signal being the number of pulses of the encoder. Through the PID algorithm, it will determine the voltage to be supplied to the motor (the desired voltage value). U(V) is the value of voltage to be supplied to the motor is shown by the formula:

$$U = K_p \times e + K_i \times \left(e_{\text{pre}} + e \times \left(t - t_{\text{pre}}\right)\right) + \frac{K_d \times \left(e - e_{\text{pre}}\right)}{t - t_{\text{pre}}} \tag{3}$$

In which, K_p, K_i, K_d, respectively, are the PID coefficients declared above. t is the current sampling time, and t_{pre} is the immediate previous sampling time. t and t_{pre} have units of seconds (s). e is the error between the desired motor angle (p) and the current motor angle (p_{cur}). e_{pre} is the number of times before.

$$e = p - p_{\text{cur}} \tag{4}$$

Since the power supply for the motor driver (U_{Pow}) is 12 V source and the microcontroller is 8-bit, the formula for converting voltage to control pulse (Pwm) is

$$\text{Pwm} = \frac{U \times 255}{U_{\text{Pow}}} = \frac{U \times 255}{12} = U \times 21.25 \tag{5}$$

3.3 Two-Way Data Transfer Between Real and Virtual Models

Here, we install ROS on a Jetson Nano and set up nodes and topics to transmit and receive data between Arduino and ROS and between ROS and Unity. ROS is an intermediary that supports sending and receiving data between Arduino and Unity. We use TCP socket protocol for two-way data transmission between ROS and Unity and UART for two-way data transmission between ROS and Arduino.

A TCP end point running as a ROS node facilitates message passing to and from Unity and ROS. The message being passed between Unity and ROS is expected to be serialized as ROS would internally serialize them.

4 Experiences and Results

4.1 Compare the Forward Kinematics Between the Construction Algorithm and Robot Studio

After building the forward kinematics algorithm, we compare the forward kinematics with Unity and Robot Studio. We found that the coordinates of Unity, the algorithm we built, and Robot Studio were interchanged. Here, Unity's X-axis is the Y-axis of our algorithm and Robot Studio's X-axis. Unity's Y-axis is the Z-axis of the algorithm we built and Robot Studio's Z-axis. Unity's Z-axis is the X-axis of the algorithm we built and the Y-axis of Robot Studio. The obtained values have very low errors. These errors are due to the rounding process in the algorithm's calculation. That proved our algorithm can accept. Table 2 is the test value obtained after testing.

4.2 Error and Latency

The error between the virtual model and the real model is calculated by the central processing unit of Physical robot. We take the value of the virtual model matching angles minus the value of the real model matching angles. Here, E_m is the value of the joint virtual model and E_n is the value of the joint physical model with m and n are from 1 to the last retrieved data stored at the system. From there, we calculate the $\Delta E(t)$ of the system using the formula:

$$\Delta\mathbb{E}(t) = \mathbb{E}_m - \mathbb{E}_n \tag{6}$$

To calculate the delay, we took the time of data sent from Virtual robot to Physical robot set as t_1 and time of data sent from Physical robot to Virtual robot set as t_2. From there, we calculate the $\Delta t(s)$ (latency) of the system using the formula:

$$\Delta t(s) = t_2 - t_1 \tag{7}$$

4.3 Data Storage

The data is stored locally, in the Firebase Cloud and Google sheet. Cloud Firestore's data transmission and reception rate is 10 Hz, and a limit of 50,000 data updates and edits in one day for the free version. We also tried storing data on Google Sheets, and it also receives and sends data at 10 Hz and is limited to 100 fetches and updates per minute. The data is divided into three parts including the collection, the directory and the file. The collection we named data contains folders inside. The name of each

Table 2 Evaluation of forward kinematics

No	Joint 1–3	Joint 4–6	P_X	P_Y	P_Z	Software
1	Joint1: 0	Joint4: 0	374	374	374	Robot Studio
	Joint2: 0	Joint5: 0	0	630	0	Unity
	Joint3: 0	Joint6: 0	630	0	630	MATLAB
2	Joint1: 10	Joint4: 100	413.94	135.34	414	Robot Studio
	Joint2: 30	Joint5: 60	135.37	413.94	135	Unity
	Joint3: 15	Joint6: 0	341.91	341.98	341	MATLAB
3	Joint1: 10.46	Joint4: 5.59	257.61	50.61	258	Robot Studio
	Joint2: −18.76	Joint5: 25.7	135.37	257.64	51	Unity
	Joint3: 6.02	Joint6: 2.86	341.91	665.05	665	MATLAB
4	Joint1: 20	Joint4: 35	315.36	143.03	315	Robot Studio
	Joint2: 25	Joint5: 40	143.01	315.35	143	Unity
	Joint3: 30	Joint6: 45	260.52	260.54	260	MATLAB
5	Joint1: 21	Joint4: 15	373.27	146.74	373	Robot Studio
	Joint2: 30	Joint5: 10	146.74	373.25	147	Unity
	Joint3: 24	Joint6: 18	256.2	256.19	255	MATLAB
6	Joint1: 26	Joint4: 18	321.37	177.03	321	Robot Studio
	Joint2: 24	Joint5: 55	176.99	321.39	177	Unity
	Joint3: 20	Joint6: 60	308.18	307.44	307	MATLAB
7	Joint1: 71	Joint4: 30	5.24	124.13	5	Robot Studio
	Joint2: 58	Joint5: 100	124.12	525	124	Unity
	Joint3: 56	Joint6: 90	165.12	165.11	165	MATLAB
8	Joint1: −131.53	Joint4: −63.18	−270.96	−255.33	−271	Robot Studio
	Joint2: 5.04	Joint5: 31.64	−255.13	−271.17	−255	Unity
	Joint3: 22.36	Joint6: −70.25	438.77	438.036	438	MATLAB
9	Joint1: −29.44	Joint4: −90.96	−130.88	21.15	−131	Robot Studio
	Joint2: −47.9	Joint5: 39.75	21.01	−131.12	21	Unity
	Joint3: −18.86	Joint6: 76.07	827.31	827.25	827	MATLAB
10	Joint1: 92.25	Joint4: −37.11	−13.8	−136.52	−14	Robot Studio
	Joint2: −3.69	Joint5: −26.07	−134.91	−13.74	−136	Unity
	Joint3: −89.73	Joint6: 70.26	919.8	919.44	919	MATLAB

folder is the time that the data inside that folder is stored in the Cloud. And the file is inside the directory. The data including the time, the matching position of the virtual model, the rotational position of the servomotor and the error between the matching position of the virtual model and the position of the virtual model, and rotation position of the servomotor at that time was collected.

We also used one more local data storage. With this local data storage, the frequency is around 165 Hz and is limited to memory on local storage only. This is not a concern because local storage can expand memory and it also takes a long time to fill this local storage. Where the STT column is the ordinal number of the data sorted in ascending order based on the time that the data was stored, the Time column is the column that stores the data about the time that the data was stored. The Error column is the column used to represent the error between the matching position of the virtual model and the real one. The Joints Unity column is the column representing the joint position of the virtual model, and the last column is the Joints Servo column, which represents the rotation position of the servomotor.

4.4 Results

The analyzed data showed that the system takes about 2 s for the system to stabilize after booting. Here, the sampling frequency is about 165 Hz. They are shown in Fig. 3. Here, the vertical axis is the value of the number of samples taken in one second and the horizontal axis is the time the sample is taken. The system latency is about 30 ms with an error of ± 2 degrees. They are shown in Fig. 4. Figure 4 has the vertical axis being the error between the matching angle of the virtual and real model (degrees) and the horizontal axis being the time (ms).

Here, we have simulated servomotor power failure. Figure 5 shows the difference of the data when the fault occurs and when the failure occurs. When there are 3000 data stored, we have disconnected the servomotor and powered up again when 3150 data is stored, and we have also disconnected the servomotor when there is 4665 data until it stops completely system. Error is error of joint between real and virtual when the system is operating normally.

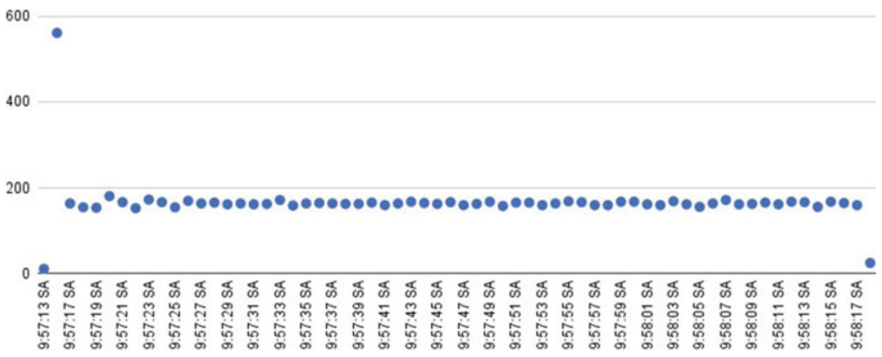


Fig. 3 Number of samples received in 1 s

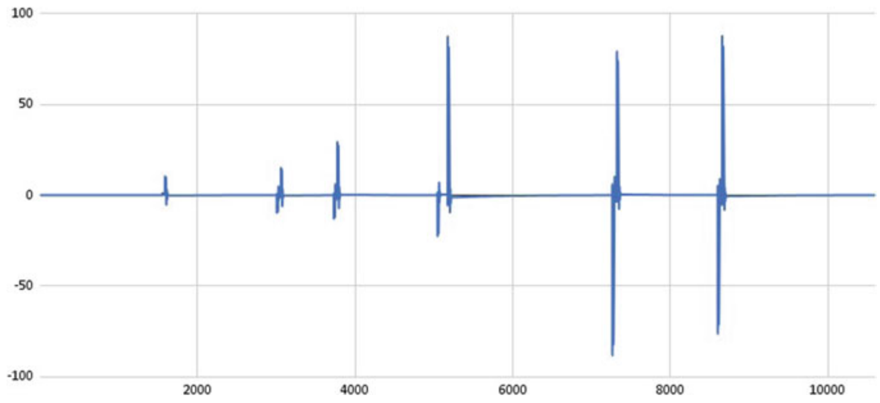


Fig. 4 Error between real and virtual model

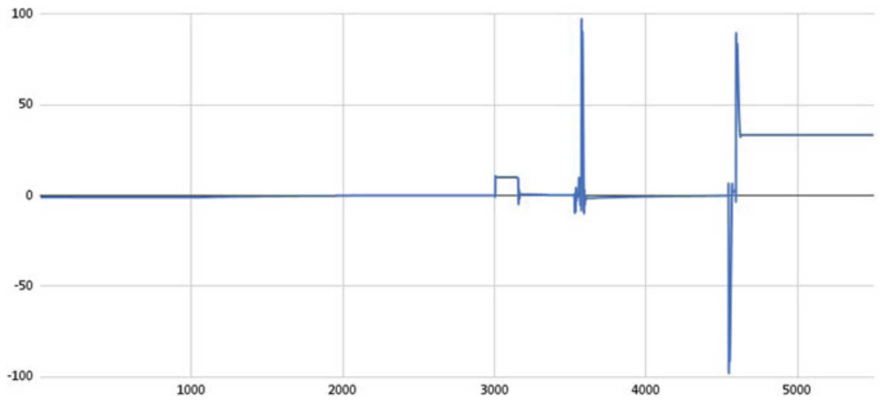


Fig. 5 Error between real and virtual model when there is a problem

5 Conclusion

In this paper, we have successfully built a digital twin model of ABB IRB 120 robot. The digital twin model has developed communication between real and virtual through the TCP socket protocol suite. We also calculated the forward kinematics of the robot and compared it with the forward kinematics of Robot Studio. In addition, we also build a PID controller for the engine and store the data on Cloud and local storage for later machine learning development.

The system operates stably with a maximum frequency of 165 Hz when not storing data and 10 Hz when putting data in the cloud. Calculation of kinematics gives results with very small error due to rounding of values in the calculation. The PID controller is stable and has been able to control the position of the motor as desired. Data is transmitted bidirectionally with very small errors due to rounding during conversion in calculations.

In the future, we will increase the frequency of sending data to the cloud and use machine learning to analyze the collected data to help monitor and evaluate the system. We will then make digital copies of typical mobile robots like manipulator platform and rehabilitation robots that we plan to do in the near future.

Appendix

Kinematics DH

We use ABB IRB 120 robot model as a model to calculate forward and experimental kinematics. We have set representation of the coordinate positions and orientation based on the joints of the robot model as shown in Fig. 6. Figure 6 is a coordinate axis location diagram with the red axis being the X-axis, the orange axis being the Y-axis, and the blue axis is the Z-axis.

The DH parameters of the ABB IRB 120 for the specified joint frames in Fig. 6 are presented in Table 3. Here, θ_i is the relative rotation of the stitches, a_i is the length, α_i is the twist angle of the stitch, and d_i is the distance between the stitches.

According to the manufacturer's specifications, the limits of joints are shown in Table 4.

Each link can bethink as a coordinate transformation from the previous coordinate system to the next coordinate system. That transformation is described as a product of translations along and rotations about X- and Z-axes. They are expressed in formula 33:

$$ {}^{n-1}T_n = \begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & a_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & a_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8} $$

Table 3 DH parameters of robot ABB IRB 120

Joints	θ_i (°)	a_i (mm)	α_i (°)	d_i (mm)
1	$\theta_1(+90)$	0	−90	290
2	$\theta_2(-90)$	270	0	0
3	θ_3	70	−90	302
4	θ_4	0	90	0
5	θ_5	0	−90	0
6	θ_6	0	0	72

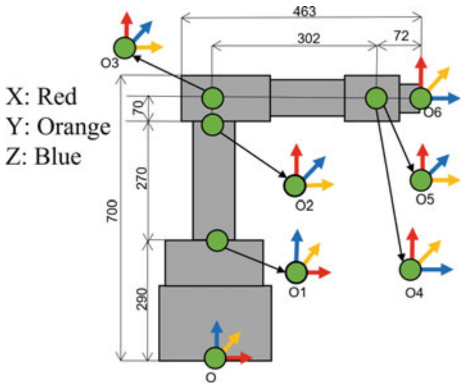


Fig. 6 Schematic and frames assignment of ABB IRB 120. $^0P = [^0R_1.^1R_2.^2R_3.^3R_4.^4R_5.^5R_6].^6P = {}^0R_6.^6P$ is the direction and coordinate location

Table 4 Table ranges of motion in every axes

Joints	Lower limit [deg]	Upper limit [deg]
1	−165	165
2	−110	110
3	−110	70
4	−160	160
5	−120	120
6	−400	400

Forward Kinematics

The purpose of this part is to find the absolute position and orientation of each frame (which is attached to each joint/servo) in reference to the global coordinate system, which is the very first frame attached to the shoulder. Apply Table 1 attributes to Eq. 33, we get transformation for each link.

Now that we have each transformation, we can calculate the position and orientation of each frame in reference to the "global coordinate system."

$$^0T_6 =^0 T_1^1 T_2^2 T_3^3 T_4^4 T_5^5 T_6 \tag{9}$$

$$= \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

In matrix 35, the parameters r form the rotation matrix of the robot and the parameter P is the coordinate position of the end point of the manipulator. From that, we can derive the formula for determining the position of the manipulator's end point coordinates as follows:

$$p_x = d_5c_1s_{234} + d_4s_1 - d_6c_1c_{234} + a_2c_1c_2 + d_6c_5s_1 + a_3c_1c_2c_3 - a_3c_1s_2s_3$$

$$p_y = d_5s_1s_{234} - d_4c_1 - d_6s_1c_{234} + a_2c_2s_1 - d_6c_5c_1 + a_3c_2c_3s_1 - a_3s_1s_2s_3$$

$$p_z = d_1 - d_6s_{234}s_5 + a_3s_{23} + a_2s_2 - d_5c_{234}$$

Inverse Kinematics

A very large number of joints make it difficult to solve the trigonometric equation, we can divide 6 joints into two groups of joints: Group one is from joint 1 to joint 3, and group two is from joint 4 to joint 6. For joint 1 to joint 3: P_c point coordinates are shown in Fig. 7. Parameters d_1, a_2, a_3, and d_4 are taken in Table 3. Change to new end point $P_c = \left[P_{cx}; P_{cy}; P_{cz}\right]$ with the matrix a being the cosine of the three Euler angles:

$$P_c = P[x; y; z] - d_6a\left[a_x; a_y; a_z\right] \tag{11}$$

Perform inverse matrix multiplication:

$$\left({}^0T_1\right)^{-1} \times \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left({}^1T_2\right)\left({}^2T_3\right)\left({}^3T_4\right) \tag{12}$$

From 2, we have the formula

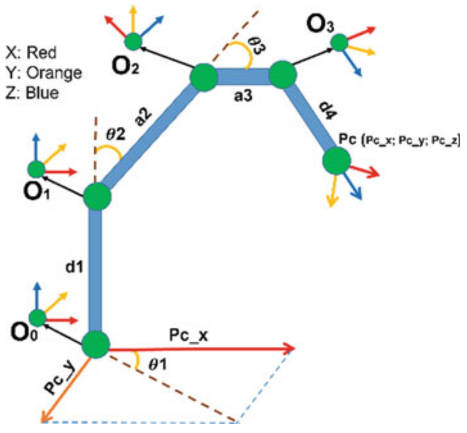


Fig. 7 Point coordinates P_c $(Pc_x; Pc_y; Pc_z)$

$$P_{cy}\cos(\theta_1) - P_{cx}\sin(\theta_1) = 0 \tag{13}$$

$$=> \theta_1 = a\tan 2\big(P_{cy}, P_{cx}\big) \tag{14}$$

$$2a_2a_3\cos(\theta_3) - 2a_2d_4\sin(\theta_3) = n_x^2 + n_y^2 - \big(a_2^2 + a_3^2 + d_4^2\big) \tag{15}$$

After simplifying, we get

$$\theta_3 = a\tan 2\left(\pm\sqrt{1 - (\frac{D}{\rho})^2}, \frac{D}{\rho}\right) - a\tan 2(d_4, a_3) \tag{16}$$

with

$$D = n_x^2 + n_y^2 - \big(a_2^2 + a_3^2 + d_4^2\big) \tag{17}$$

$$\rho = 2\sqrt{(a_2a_3)^2(a_2d_4)^2} \tag{18}$$

$$n_x = \frac{P_{cx}}{cos(\theta_1)} \tag{19}$$

$$n_y = P_{cz} - d_1 \tag{20}$$

Continuing to multiply the inverse matrix, we have

$$({}^1T_2)^{-1}({}^0T_1)^{-1} \times \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = ({}^2T_3)({}^3T_4) \tag{21}$$

We have the equation

$$d_4\cos(\theta_3) + a_3\sin(\theta_3) = \cos(\theta_2)(d_1 - P_{cz}) + P_{cy}\sin(\theta_1) + \sin(\theta_2)P_{cx}\cos(\theta_1)$$

After simplifying, we get

$$\theta_2 = \varphi - a\tan 2\left(\pm\sqrt{1 - (\frac{M}{\sigma})^2}, \frac{M}{\sigma}\right) \tag{22}$$

with

$$M = d_4\cos(\theta_3) + a_3\sin(\theta_3) \tag{23}$$

$$\sigma = \sqrt{p_x^2 + p_y^2} \tag{24}$$

$$\varphi = a\tan 2\left(p_y, p_x\right) \tag{25}$$

$$p_x = P_{cx}\cos(\theta_1) + P_{cy}\sin(\theta_1) \tag{26}$$

$$p_y = d_1 - P_{cz} \tag{27}$$

For joint 4 to joint 6: We can calculate the transformation matrix from joint 4 to joint 6

$$^3T_6 =^3 T_4^4 T_5^5 T_6 = \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5c_6 - s_4c_6 & c_4s_5 & c_4s_5d_6 \\ s_4s_5c_6 + c_4s_6 & -s_4s_5s_6 + c_4c_6 & s_4s_5 & s_4s_5d_6 \\ -s_5c_6 & s_5s_6 & c_5 & c_5d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We get rotated matrix

$$^3R_6 = \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5c_6 - s_4c_6 & c_4s_5 \\ s_4s_5c_6 + c_4s_6 & -s_4s_5s_6 + c_4c_6 & s_4s_5 \\ -s_5c_6 & s_5s_6 & c_5 \end{bmatrix} \tag{28}$$

but

$$^0R_6 =^0 R_3^3 R_6 \tag{29}$$

or

$$^3R_6 = (^0R_3)^T(^0R_6) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{30}$$

From matrixes 18 and 20, we have:

$$\cos(\theta_5) = r_{33} \tag{31}$$

$$\sin(\theta_5) = \pm\sqrt{1 - r_{33}^2} \tag{32}$$

$$=> \theta_5 = a\tan 2\left(\pm\sqrt{1 - r_{33}^2}, r_{33}\right) \tag{33}$$

$$\frac{\sin(\theta_5)\sin(\theta_6)}{-\sin(\theta_5)\cos(\theta_6)} = \frac{r_{32}}{r_{31}} => \theta_6 = a\tan\left(\frac{-r_{32}}{r_{31}}\right) \tag{34}$$

$$\frac{\sin(\theta_5)\sin(\theta_4)}{\sin(\theta_5)\cos(\theta_4)} = \frac{r_{23}}{r_{13}} => \theta_4 = a\tan\left(\frac{r_{23}}{r_{13}}\right) \tag{35}$$

From Eqs. (14), (16), (22), (33), (34), (35), we have obtained the formula to calculate the joint angle values to complete the inverse kinematics problem for the ABB IRB 120 robot. From here, we will build a robot controller based on this algorithm.

References

1. Wu, Y., Zhang, K., & Zhang, Y. (2021). Digital twin networks: A survey. *IEEE Internet of Things Journal, 8*(18), 13789–13804.
2. Lim, K. Y. H., Zheng, P., & Chen, C.-H. (2020). A state-of-the-art survey of digital twin: Techniques, engineering product lifecycle management and business innovation perspectives. *Journal of Intelligent Manufacturing, 31*, 1313–1337.
3. Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd edn.). Prentice Hall.
4. Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag.
5. Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org
6. Le, T. D., Huynh, D. T., & Pham, H. V. (2018). Efficient human-robot interaction using deep learning with mask R-CNN: Detection, recognition, tracking and segmentation. In *2018 15th International conference on control, automation, robotics and vision (ICARCV)* (pp. 162–167).
7. Le, T. D., & Pham, H. V. 2020. *Intelligent data analysis* (Chap. 5, pp. 85–114). Wiley. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119544487.ch5
8. Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). *Robotics: Modelling, planning and control*. Springer Publishing Company.
9. Siciliano, B., & Khatib, O. (2007). *Springer handbook of robotics*. Springer-Verlag.
10. Nguyen, H. V., Le, T. D., Huynh, D. D., Nauth, P.: Forward kinematics of a human-arm system and inverse kinematics using vector calculus. In *2016 14th International conference on control, automation, robotics and vision (ICARCV)* (pp. 1–6).
11. Than, L., & An, L. (2020). Manipulation-based skills for anthropomorphic human-arm system based on integrated anfis and vector calculus. *bioRxiv*. [Online]. Available: https://www.biorxiv.org/content/early/2020/02/10/2020.02.10.941344
12. Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). Ros: An open-source robot operating system. In *ICRA workshop on open source software*.
13. Palmieri, L., & Arras, K. O. (2015). Distance metric learning for rrt-based motion planning with constant-time inference. In *2015 IEEE International conference on robotics and automation (ICRA)* (pp. 637–643).
14. Palmieri, L., & Arras, K. O. (2014). A novel rrt extend function for efficient and smooth mobile robot motion planning. In *2014 IEEE/RSJ International conference on intelligent robots and systems* (pp. 205–211).
15. Le, T. D., Bui, D. T., & Pham, V. H. (2018). Encoded communication based on sonar and ultrasonic sensor in motion planning. *IEEE Sensors, 2018*, 1–4.

16. Le, T., & Le, T. D. (2018). Search-based planning and replanning in robotics and autonomous systems. in R. Róka (Ed.), *Advanced path planning for mobile entities*. IntechOpen, 2018, Chap. 4. [Online]. Available: https://doi.org/10.5772/intechopen.71663
17. Le, T., Hung, B. T., Van Huy, P. (2021) *Search-Based Planning and Reinforcement Learning for Autonomous Systems and Robotics* (pp. 481–501). Springer International Publishing. [Online]. Available: https://doi.org/10.1007/978-3-030-77939-9_14
18. Le, T. D., Le, A. T., Nguyen, D. T. (2017). Model-based q-learning for humanoid robots. In *2017 18th International conference on advanced robotics (ICAR)* (pp. 608–613).
19. Garg, G., Kuts, V., & Anbarjafari, G. (2021). Digital twin for fanuc robots: Industrial robot programming and simulation using virtual reality. *Sustainability, 13*(18). [Online]. Available: https://www.mdpi.com/2071-1050/13/18/10336
20. Mengacci, R., Zambella, G., Grioli, G., Caporale, D., Catalano, M. G., & Bicchi, A. (2021). An open-source Ros-gazebo toolbox for simulating robots with compliant actuators. *Frontiers in Robotics and AI, 8*. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2021.713083
21. Dröder, K., Bobka, P., Germann, T., Gabriel, F., & Dietrich, F. (2018). A machine learning-enhanced digital twin approach for human-robotcollaboration. *Procedia CIRP, 76*, 187–192.
22. Wang, X., Liang, C. -J., Menassa, C., & Kamat, V. (2020) Real-time process-level digital twin for collaborative human-robot construction work. In F. H. T. K. Osumi Hisashi (Ed.), *Proceedings of the 37th International symposium on automation and robotics in construction (ISARC)*, (pp. 1528–1535). International Association for Automation and Robotics in Construction (IAARC).