Forward and Inverse Kinematics Demonstration using RoboDK and C#

Sudip Chakraborty[1] & P. S. Aithal[2]

[1]Post-Doctoral Researcher, College of Computer science and Information science, Srinivas University, Mangalore-575 001, India

OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2]ViceChancellor, Srinivas University, Mangalore, India

OrcidID: 0000-0002-4691-8736; E-mail: psaithal@gmail.com

Subject Area: Engineering.
Type of the Paper: Simulation-based Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed In: OpenAIRE.
DOI: http://doi.org/10.5281/zenodo.4939986
Google Scholar Citation: IJAEML

How to Cite this Paper:

Chakraborty, Sudip, & Aithal, P. S., (2021). Forward and Inverse Kinematics Demonstration using RoboDK and C#. *International Journal of Applied Engineering and Management Letters (IJAEML)*, *5*(1), 97-105. DOI: http://doi.org/10.5281/zenodo.4939986.

International Journal of Applied Engineering and Management Letters (IJAEML)
A Refereed International Journal of Srinivas University, India.

Crossref DOI: https://doi.org/10.47992/IJAEML.2581.7000.0095

Forward and Inverse Kinematics Demonstration using RoboDK and C#

Sudip Chakraborty[1] & P. S. Aithal[2]

[1]Post-Doctoral Researcher, College of Computer science and Information science, Srinivas University, Mangalore-575 001, India

OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2]ViceChancellor, Srinivas University, Mangalore, India

OrcidID: 0000-0002-4691-8736; E-mail: psaithal@gmail.com

ABSTRACT

Purpose: *Robot researchers need a simulator to understand better the algorithm on path planning, arm movement, and many more. They need a good simulator. RoboDK is an excellent simulator to fulfill the research work. It has calibration facilities, so it is industrial-grade software. Its forward and inverse kinematics accuracy is better than any competing software. The main advantage is all robots under one IDE. When we use an industrial robot, and we must use their software environment to operate the robot. But the RoboDK covers most of the robots and runs under one roof. And we need to learn only one IDE. The RoboDK online library is full of the standard robot. And all robot's operation procedure is the same. So, the learning curve of new robots is easy. It is easy to simulate, and it can connect with a practical robot to execute the task. Using this software, we can quickly create digital twins for the industry. Now we think about control the robot from our application. When we use to control the robot from an external environment or remote software, we need the use the API to control the robot. Here we will see how easily we can operate the robot from our custom application. We adopted RoboDK C# API and integrated it into Visual studio using a User interface to control the robot movement. Keeping this research as a reference, the robotic arm researcher can add value to their research. Our primary purpose is to shorten the learning curve to integrate the RoboDK with their custom application.*

Design/Methodology/Approach: *Taking the RoboDK C# API they provided, we customized it according to our purpose with minimal components. After developing a graphical user interface, we interact through API. Then, opening both RoboDK IDE and C# application, we can send the End effector position using the sliding movement.*

Findings/Result: *After our research, we found that RoboDK is a good IDE for our research on the robotics arm. We can easily integrate the C# API they provided with our custom application for research purposes.*

Originality/Value: *If we want to test robotic arm movement in the simulator, we need an excellent simulator like RoboDK. Integrating the RoboDK C# API is a little bit time-consuming. Using our approach, the researcher can continue their research in a minimal period. And find adequate information here to integrate easily into their project.*

Paper Type: *Simulation-based Research.*

Keywords: Forward kinematics, Inverse kinematics, 6 degrees of freedom robot, RoboDK,

1. INTRODUCTION :

Nowadays, most industries adopted robotics process automation. Most enterprises use robotics arms for their process like pick and place, packaging. For robot operation, we need to position the end effector at an exact point. The robot researchers need to calculation and operate the arm very efficient manner. Before implementing their algorithm, they need to test in a safe environment; otherwise, it can create unpredictable situations and damage property and human life. The simulator is the safest environment to test the result of the newly implemented algorithm. A good simulator can give us the output as practical as possible. The most trusted software RoboDK is the best simulator now. Its 3d model is as

natural as possible. Forward (FK) and inverse kinematics (IK) are very efficient. It can also integrate with a real-life robot. It has lots of standard robot 3d model which we just download and experiment on it. In practice, the real robot is so costly, and cannot afford different vendors robot. However, in the simulator, we can easily simulate real robots with lots of variants. RoboDK is free to download. It has a student/researcher version which is a little bit costly. It also has a one-month trial version. We can experiment within one month without any charges. If we want to continue research, we can purchase the education version using our university mail-id.

Now we see the demonstration of FK and IK using RoboDK. Here we will use the Niryo One, six degrees of freedom (DoF) robot. We operate the robot from outside of the simulator environment. We use RoboDK C# API to communicate with the simulator. We create an IP client in our software and communicate with c# using the address "127.0.0.1," and the port is 20500. After connecting, we verify that we genuinely connected with robots. After confirming that we connected with the simulator, we get the information about which robots we are connected to. First, we get API name, API version, build version. And then, we get the robot name, Item id, etc. now, our application is ready for further operation. We can get or set joints value.

Moreover, get or set the robot end-effector position. Inverse kinematics generally consume a lot of calculation overhead. However, we can drive the real-world robot with a minor CPU footprint driver by getting the joints value. The standard industrial robot provides API and a control card with a built-in processor to calculate the robot's position. But in the case of the custom-made robot, lots of facts need to account. So, using this method, we can minimize the calculation overhead utilizing the getting parameter from RoboDK.

2. RELATED WORKS :

A. Garbev et al. presented a comparative analysis of known systems for creating robot control programs of any type. The study aims to analyze the programs and their way of creation [1]. Martin Pollák et al., in their paper, describe the design of equipment for the implementation of 5-axis milling with the help of a robot arm ABB IRB 140. Their experiment was carried out using RoboDK to create a robotic arm control program [2]. Ribeiro et al., in their paper, demonstrate the aims to describe an innovative solution for implementing robotic simulation for AM experiments using a robot cell, which is controlled through a system control application (SCA) [3]. S. Pieskä et al., in their paper, described their experiences in simulation and programming of collaborative robots (cobots). They also demonstrated the use of these devices and software with novice or inexperienced users. This study asked novice users to perform simple pick-and-place tasks under varying perception and planning capabilities. The goal was to study how small- and medium-sized enterprises (SMEs) can effectively utilize cobots in production [4]. M. Pollák et al. in their article describes the design of a 3D print head working with the ABB IRB 140 [5]. C. Lin and M. Li proposed a method where a particle swarm optimization with the charge search system (CSS) to find the optimal path planning with obstacle avoidance is implemented by the real-time experiments of the UR3 [6]. Tarek Al-Geddawy in their paper describes a method to make simulated changeable learning to create a digital twin [7]. Margaria T. research on Industry 4.0. they provide an example of how the new model-powered and integrated thinking can disrupt the status quo, empower a better understanding, and deliver more automatic management of the many cross-dimensional issues that future connected software [8]. In their paper, Li L et al. establish a kinematics model for a six-degree-of-freedom (DOF) manipulator and verify its correctness through simulation and experiment [9]. R. Beloiu explains how trajectory generation and the robot OLP are computer-generated in many cases. Robot programs for unregulated shapes bodies can be obtained with either of the programs without the need to master the theoretical complex mathematical moving algorithms [10]. Esperto, V. et al. research a software tool that can automate fiber fabrics and tissues [11]. Adrián Peidró, Oscar Reinoso, in their paper, presents a virtual laboratory to simulate the dynamic control of parallel robots [12].

3. OBJECTIVES :

This research aims to familiarize to control the RoboDK from the external environment. First, creating the application using visual studio connects with the RoboDK. Then we discuss some basic APIs for communication and basic functionality like getting joints, set joints, get the position, update the end-effector position, etc.

4. APPROACH AND METHODOLOGY :

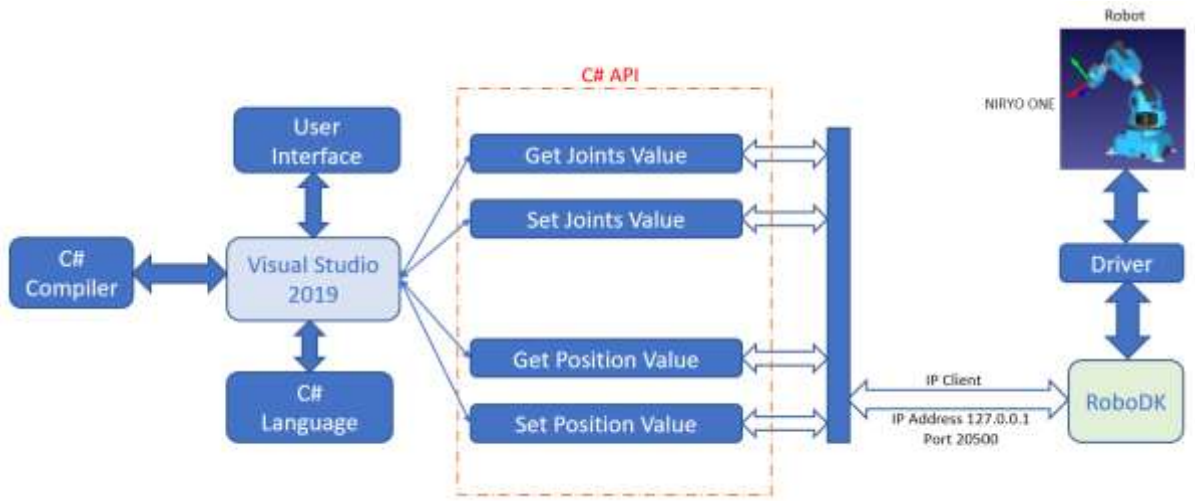

Fig. 1: The methodology of our research work

Figure 1 picture depicts the complete methodology of our research work. There are two sides to our research work. The left side is visual studio IDE, and the right side is RoboDK IDE. RoboDK provides C# API to communicate from the visual studio. The left side constitutes the User interface, C# language, and C# compiler. We created a customized API from provided API. We accumulate minimal API to communicate and created RoboDK_Manager.cs. To change the robot parameter, we created a GUI with some slider, button, and textbox. The basics functions of the API as below:

Connected_With_RoboDK() – We place this function inside the form load function. When the form is load, the state machine starts. When RoboDK runs, start an IP server in its background. The server waits for client requests. When our application starts, first, create an IP client socket with port number 20500. For the same machine, we can use the loopback address "127.0.0.1". If the RoboDK runs on a different device, we need to put the IP address in which the RoboDK is running. If the connection is successful, then we send the further command. Here basically a few types of control commands available to ship RoboDK. Send line command, send bytes until "\n" is found in transmit buffer. Send int. It sends 4-byte data. Double send function, send 8 bytes data. All send function names are the same. Due to function overloading, it will detect the exact function and send it for the specific process. In the receive function, we have a couple of functions as transmit. Read line function read bytes data until '\n character finds. Read int function reads 4-byte data. Read double, reads 8 bytes of data. In reading status, we read 4-byte data which can detect the status of the robot message. Another is long bytes read. It is needed when we read pos data. The end effector data consists of 128 bytes of data. So after connection success, we will verify that the connection is truly connected. So we send "RDK_API" followed by 4 bytes of data consists of zero. Then we should receive API =" RDK_API," API version :1, and build 20593. The values may vary from their version build. After that, we read the status. If the return value is zero, no error happened. If other than zero means something happens. Most of the time, we see that the robot cannot move for a particular pose. Then we can get "Target is unreachable." we assume that we get zero means no error found. Then we will go to receive The robot details, which are connected with the robot driver. To get the robot details, we should send the series of the command like "PickItem\n," "Select a robot\n," send 2(4 bytes). After sending the sequence of commands, we now receive the 8 bytes. i.e., ItemId, which is the essential parameter for further command. Then we can get 4 bytes of data which is the item type. Here we get value 2 for robot object. Then read the command status. Zero reception means no error, other than zero means something goes wrong. We can see the last read message to the error message. To read the robot name. We send "G_Names\n" and 8 bytes Item id. Receive bytes till "\n.". Here we get the Name "Niryo One" because we selected Niryo One robot in RoboDK IDE. After any data exchange, we read the status to confirm that nothing goes wrong (Figure 2).

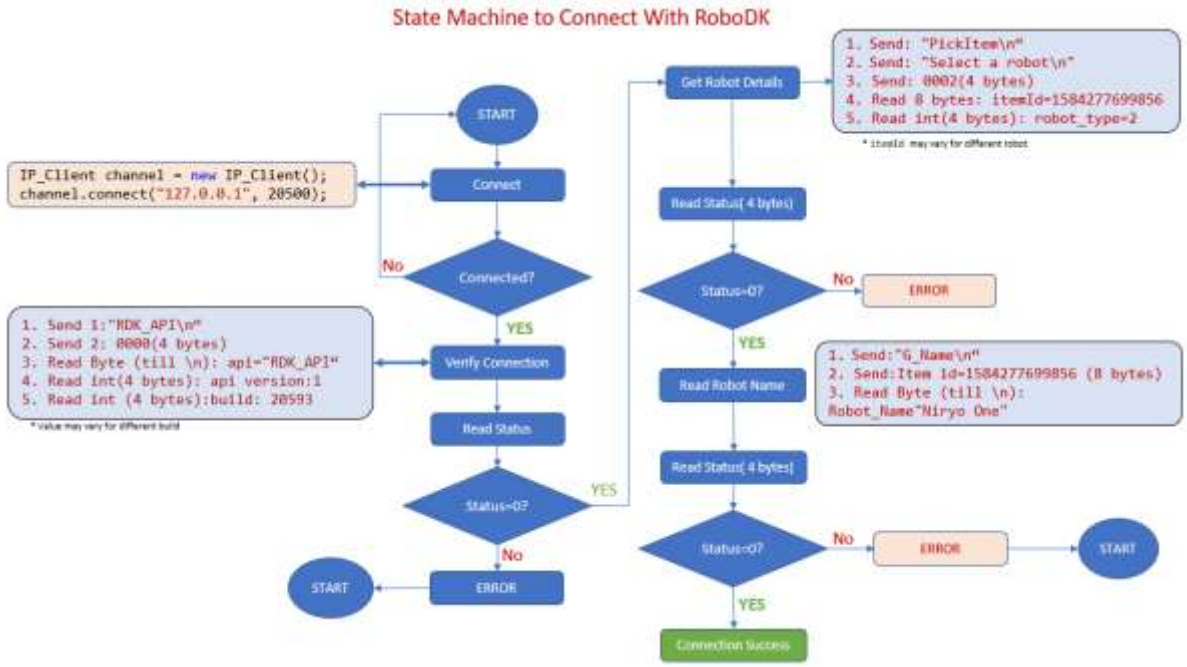

Fig. 2: State machine to connect with RoboDK.

Public string read_joints(): Figure 3 shows the series of commands to read the joints values. To read the current joints, we should call the function. It takes five steps. At first, send "G_Thetas\n." then send the item id we received at the connection steps. Then read the number of joints having in the robot. In our selected robot, we get a value of six because our robot is six degrees of freedom. Next, we read the joints value. The number of bytes is the number of joints multiplied by 8, one double takes an eight-byte length. Here we receive 48 bytes. After receiving all bytes, every eight bytes packed into one double. The C# has a built-in function to convert from bytes to double using the Bitconverter class. At the final stage, to display the data, we need to convert it into the string to display the joints value into the textbox or any other control. These joints value also can be used for digital twins. If the robot is not directly observable, can be used for remote monitoring purpose.

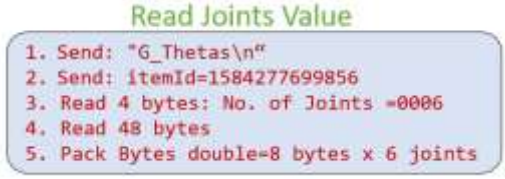

Fig. 3: Command to read joints.

Public void Move_Joints(string strvalues): Figure 4. depicts the sequence of updated joints value to the RoboDK. We need to send some commands to update the robot's joint value. At first, we send "MoveX\n." then move type 1. The One is sent as 4 bytes of data. Next, send joints to count. After that, send all joints value. Our robot is 6 degrees of freedom, so we need to send six joints values multiplied by 8 bytes. So we send 48 bytes of data. Then send itemID followed by 8 bytes of value zero. That is it. Now the time to read to ensure that no error happened. Read 4-byte status. If the return value is zero means everything on the transaction is ok. We can observe that our robot is moved to a new position.

Fig. 4: Command to send the joints.

Public string read_Pose(): Now, we will introduce another essential command is read position. On specific joints, we can get the position values. We get the end effector translation value of Tx, Ty, Tz, rotation value Rx, Ry, and Rz. In figure 5, we listed the steps. The item ID may vary when we do our experiment. Whenever we close and open RoboDK, this value generally changed, and if more than one object presents in the workspace, the application assigned object id dynamically. So it is not fixed.

Read Position Value

```
1. Send: "G_Hlocal\n"
2. Send: itemId=1584277699856 (8bytes, it may vary)
3. Read 128 bytes
4. Read int(4 bytes)  // Read Status
5. 128 bytes are packed into 4x4 matrix (each element is double i.e. 8bytes)
6. Convert matrix to IxIyIzRxRyRz value.
7. Convert to string to display into UI element.
```

Fig. 5: Read position value sequence.

Now we will see, after getting the 128 bytes, what next to process. We take 8 bytes of data, packed as double, and push into a 4x4 matrix as figure 6 first read color value enclosed with the third bracket is row and column number. The green color value surrounded by the first bracket is received Packet Buffer position. Every data be 8 bytes long that are finally converted into a double value.

	C0	C1	C2	C3
R0	[0,0] (0-7) -3.751072591459578E-15	[0,1] (32-39) -6.1232339957366E-17	[0,2] (64-71) 1	[0,3] (96-103) 248.00000000000045
R1	[1,0] (8-15) 5.54222631166383E-16	[1,1] (40-47) 1	[1,2] (72-79) 6.1232339957368141E-17	[1,3] (104-111) 4.89858719658986356-16
R2	[2,0] (16-23) -1	[2,1] (48-55) 5.54222631166383196-16	[2,2] (80-87) -3.7510725914595756E-15	[2,3] (112-119) 422.99999999999937
R3	[3,0] (24-31) 0	[3,1] (56-63) 0	[3,2] (88-95) 0	[3,3] (120-127) 1

Fig. 6 : Convert Receive bytes to matrix

After creating the matrix, now the time to extract the End effector position. The X, Y and Z position is easy to extract from the matrix table described in figure 7. X value will be the fourth column-first row value. Y position value will be the fourth column-second value. Finally, the Z value will be the fourth column-third row value.

x = [0, 3]	248
y = [1, 3]	4.90E-16
z = [2, 3]	423

Fig. 7: Get Translation Data from Matrix

To extract the rotation value, we need to process it further. Figure 8 is showing how to process the rotation value. Here C [0,2] is the comparison factor. Suppose this value 1, or C = -1, and other than this value, calculate a different way. Finally, we got the end effector value.

Now we convert from radian to degree value for X, Y, and Z rotation value. Figure 9 shows how we transform into a degree. No need to process further the Translation value. Just it is straightforward from the calculation. Moreover, we can convert to string value from double using the toString command.

```
C[0, 2]==1
ry1 = 0.5 * Math.PI;
rx1 = 0;
rz1 = Math.Atan2(_mat[1, 0], _mat[1,

C==-1
ry1 = -Math.PI / 2;
rx1 = 0;
rz1 = Math.Atan2(-mat[1, 0], -mat[1,
1]);

Else

var sy = c;
var cy1 = +Math.Sqrt(1 - sy * sy);
var sx1 = -d / cy1;
var cx1 = e / cy1;
var sz1 = -b / cy1;
var cz1 = a / cy1;
rx1 = Math.Atan2(sx1, cx1);
ry1 = Math.Atan2(sy, cy1);
rz1 = Math.Atan2(sz1, cz1);
```

Fig. 8: process position Value

```
xyzwpr[0] = x;
xyzwpr[1] = y;
xyzwpr[2] = z;
xyzwpr[3] = rx1 * 180.0 / Math.PI;
xyzwpr[4] = ry1 * 180.0 / Math.PI;
xyzwpr[5] = rz1 * 180.0 / Math.PI;
return xyzwpr;
```

Fig. 9: convert radian to degree.

Public void Move_pos(string strvalues): for inverse kinematics operation, this is a handy command. We send the end-effector position and orientation value. So, the robot can move according to our desired place. We have to follow some series of the command sequence. At first, the string value must convert from string to double [6] array because string value can not participate in any mathematical operation. From double [6] we convert a 4x4 matrix whose element is double. Then we convert from matrix to double array. Now almost ready to send out EF (End Effector) value to the robot. Start sending the data to the robot driver. The sequence starts with sending "MoveX\n." then send move type, i.e., 1. Send two in 4 bytes int format. Then send how much double wants to send. Send 128 bytes of data to the robot. Now the time to know if anything happens wrong. Send 8 bytes zero. Then send itemId and read 4 bytes. Pack into an int, and if we find a value other than zero, we assume that something goes wrong. Then try again. After healthy transmit, we can observe that the robot is moving according to our desired position. Lots of theories we learned. Now we need to do some practical experiments.

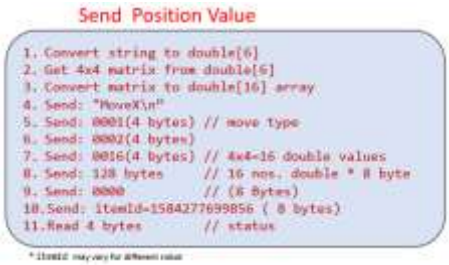

Fig.10: Sequence to send position

5. EXPERIMENT :

We need two software for our experiment. One is RoboDK, and another one is Visual studio. In the recommendation section, we can get the download link of both software. After installing the software, we can download the project's source code from the GitHub repository. Open RoboDK. From File\Open, we need to select the RoboDK file, which is inside the source code folder. After opening the RoboDK file, we can see it like in figure 11. we can choose or download different robots online also. One side is ready. Next, we must ready another side of our research work.

Fig. 11: Niryo One robot

For the Control application, we can do it in two ways. One is, we can run the ..\IK_Test\bin\x64\Debug\IK_Test.exe file. Alternatively, we can open a visual studio. Then open the project and select the project from the download folder. The project will open. Now we build the project by pressing the build button in the top menu bar. If the build is successful, then it can run. The application will appear on the screen that looks like figure 12.

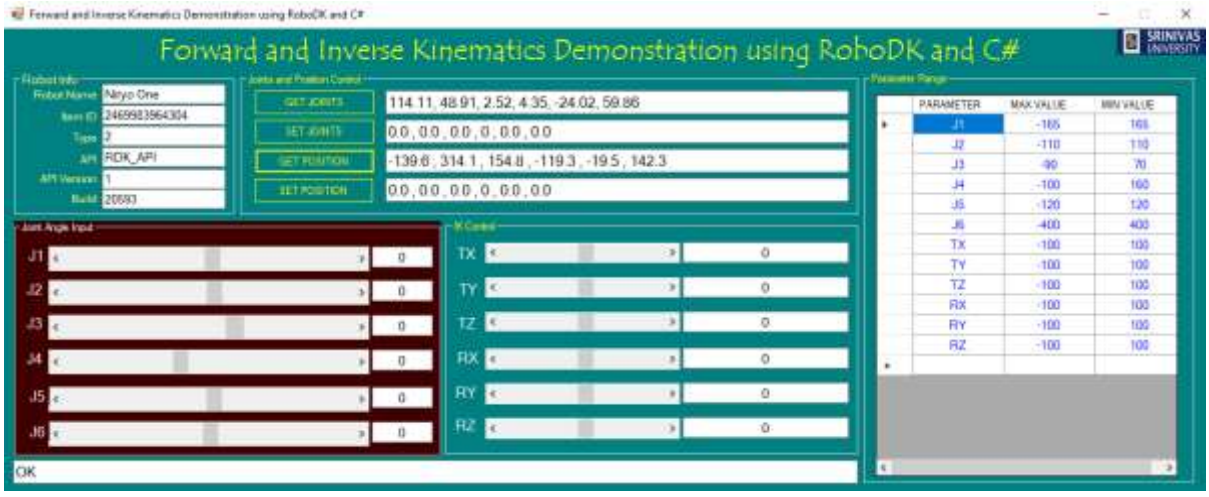

Fig. 12: Robot Control Application

If the windows appear successfully, then we can send the command to the robot through Inter-process communication. When the form loads without any command send, we can see the robot details in the

textbox. It means communication is successful. We can get the joints to value by pressing the "GET JOINTS" button. We can set our robot position using Set joints value. We need to maintain a comma with every value; otherwise, the application cannot segregate the value and cause the error. We are not completed catching all errors. Then, we can test the control application with the different robots. Of course, we need to select any six degrees of freedom robot. Because in all variables, the array we assumed that it is a six DoF robot. For further experiments, we can customize according to our requirements.

6. RECOMMENDATIONS :

After completed our research, we realize some suggestions to get better and perfect results. –

- For visual studio community edition, we can download from https://visualstudio.microsoft.com/downloads/
- Download RoboDK software from https://robodk.com/download
- The RoboDK C# API can be downloaded from https://github.com/RoboDK/RoboDK-API
- The Complete source code is available from https://github.com/sudipchakraborty/FK-and-IK-Using-RoboDK-and-C-
- This experiment is our fundamental approach how to communicate with the RoboDK simulator. We recommend further research to develop more knowledge on robot research work.
- When we input the joint's angle or position, maintain the comma, which may raise an exception.
- Sometimes we get a very long result. We need to process results further to compatible with our input. In C#, lots of functions can be used like substring, padded left, trim, etc.

7. CONCLUSION:

At present, the industry uses a massive amount of robotics arm for repetitive work. The robot can work around the clock with perfection and full potential entire its life cycle. Just need to calibrate after one or two-year interval. Hence, the massive demand for the robotic arm is gradually increasing the demand for robotics arm researchers. The latest technology like Artificial Intelligent (AI) and 3D image processing are value addition, so the robot becomes more acceptable to the uncovered industry day by day. For robotics arm research, we need an excellent simulator to understand our work better. Only a good simulator can provide a satisfactory result. RoboDK is one of the Best for a comprehensive industrial accepted robot. Here we experimented with RoboDK. Creating our custom application, we communicated and controlled the robot end-effector position using the user interface. This experiment is the basic approach. We can research further for better results.

REFERENCES

[1] Garbev, A. and Atanassov, A. (2020). Comparative Analysis of RoboDK and Robot Operating System for Solving Diagnostics Tasks in Offline Programming. International Conference Automatics and Informatics (ICAI), 2020, pp. 1-5, doi: 10.1109/ICAI50593.2020.9311332.

[2] Martin Pollák, Monika Telišková, Marek Kočiško and Petr Baron (2019). Application of industrial robot in 5-axis milling process. MATEC Web Conf., 299 (2019) 04004, DOI: https://doi.org/10.1051/matecconf/201929904004.

[3] Ribeiro, F. M., Pires, J. N. and Azar, A. S. (2019). Implementation of a robot control architecture for additive manufacturing applications. *Industrial Robot*, *46*(1), 73-82.

[4] Pieskä, S., Kaarela, J. and Mäkelä, J. (2018). Simulation and programming experiences of collaborative robots for small-scale manufacturing. 2nd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS), pp. 1-4, DOI: 10.1109/SIMS.2018.8355303.

[5] Pollák, M., Török, J., Zajac, J., Kočiško, M. and Telišková, M. (2018). The structural design of 3D print head and execution of printing via the robotic arm ABB IRB 140. 5th International Conference on Industrial Engineering and Applications (ICIEA), 2018, pp. 194-198, DOI: 10.1109/IEA.2018.8387095.

[6] Lin, C. and Li, M. (2018). Motion planning with obstacle avoidance of a UR3 robot using charge system search. 18th International Conference on Control, Automation and Systems (ICCAS),

2018, pp. 746-750.

[7] Tarek Al-Geddawy (2020). A Digital Twin Creation Method for an Opensource Low-cost Changeable Learning Factory. *Procedia Manufacturing*, *51*(1), 1799-1805.

[8] Margaria T., Schieweck A. (2019). The Digital Thread in Industry 4.0. In: Ahrendt W., Tapia Tarifa S. (eds) Integrated Formal Methods. IFM 2019. Lecture Notes in Computer Science, Vol 11918. Springer, Cham. https://doi.org/10.1007/978-3-030-34968-4_1.

[9] Li, L., Huang, Y., Guo, X. X. (2019). Kinematics modelling and experimental analysis of a six-joint manipulator. *Journal Européen des Systèmes Automatisés*, *52*(5), 527-533.

[10] Beloiu, R. (2021). Virtualization of robotic operations. 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE), pp. 1-4, DOI: 10.1109/ATEE52255.2021.9425336.

[11] Esperto, V., Gambardella, A., Pasquino, G., Tucci, F., Durante, M., & Carlone, P. (2021). *Modeling and Simulation of the Robotic Layup of Fibrous Preforms for Liquid Composite Molding*. Paper presented at ESAFORM 2021. 24th International Conference on Material Forming, Liège, Belgique. DOI: 10.25518/esaform21.475.

[12] Adrián Peidró, Oscar Reinoso, Arturo Gil, José M. Marín, Luis Payá (2015). A Virtual Laboratory to Simulate the Control of Parallel Robots, *IFAC-Papers OnLine*, *48*(29), 19-24.

****