Demonstration of Drawing by Robotic Arm 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: Artificial Intelligence & Robotics.
Type of the Paper: Simulation-based Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed In: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.5100536
Google Scholar Citation: IJAEML

How to Cite this Paper:

Chakraborty, Sudip, & Aithal, P. S., (2021). Demonstration of Drawing by Robotic Arm using RoboDK and C#. *International Journal of Applied Engineering and Management Letters (IJAEML)*, *5*(1), 153-158. DOI: https://doi.org/10.5281/zenodo.5100536

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.2581.7000.0099

Demonstration of Drawing by Robotic Arm 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: *Robots are transforming the world, and soon they will take part or assist in our all-daily life activities. There are several fields where robots are already doing well, like Surgery, painting, industrial automation, autonomous navigation, engraving, 3D printing, and many more. The robot is nothing but consists of some mechanical movement driven by functions and algorithms. To get the perfect result, the algorithm working behind the scene also needs to be perfect. In all the above fields, need to reach the end effector at the exact position. That is why we need serious research on it. Otherwise, it will remain a plaything, cannot be engaged in an important role. Before implementing it into an actual application, we need to test in a simulated environment, especially when writing new methods. The simulation environment is the safest place where anything that goes wrong cannot damage a physical entity. So, the robot researcher prefers the simulator. The RoboDK is one of the famous robot simulators. Its interface is excellent and easy to test any commercial robot inside the IDE. In our research will see how to keep marking the end-effector position using the graphical method. When our end-effector moves, it holds a footprint to inspect later. This one is a minimalistic approach. Through the simplified drawing method, we can observe arm movement. This demonstration will use an optimized version of C# API, which RoboDK provides.*

Design/Methodology/Approach: *In RoboDK IDE, we can create a new station or import existing built-in available stations. Our application drives the simulator robot. It is developed in C# language using Microsoft visual studio 2019 community edition. Both applications communicate through socket communication. Pressing the mouse's left click, when we move the mouse, it calculates the mouse pointer position and maps the value based on the robot drawing area. Then the values are sent to the simulated robot. Fetching the value, RoboDK moves into the desired position.*

Findings/result: *Using our research works, the researcher can get some references to enhance their research work. We keep our code as simple as possible to be understood quickly and easily integrate into their research work.*

Originality/Value: *We search the research work on RoboDK and C# language. Most of the documents are available in the python language, even in little bit documents present in the RoboDK example collection. So, we decided to migrate from python to our native language, C#. That is why this research work. After lots of hard work, we achieved it. If anyone wants to research RoboDK using C# API, it can reference their research work.*

Paper Type: Simulation-based Research.

Keywords: Robotic drawing, Robot simulator, RoboDK, Inverse Kinematics.

1. INTRODUCTION :

Nowadays, the robot is doing many activities. Some jobs need to take care of their perfections, and some need mechanical strength. Few of them need extreme care about all steps where from program to automatic movement demands a robust architecture. In medical robotics, where remote operation is happening, all physical activity of the robotic arm needs highly accurate. In the CNC, an engraving kind

of application also needs precise movement; otherwise, the quality of the output goes wrong. All the above processes need to exercise before the practical implementation of the robot. Otherwise, it can become a severe issue for property and human life. Moreover, machining activities can damage physical entities on improper data flow. That is why we need to work out in simulator first. For this purpose, we need a good robot simulator to test the application as naturally as possible. After much research, we found the RoboDK robot simulator. It has provided C# API to control from outside of the simulator environment. Studying the API, we optimized for our application to continue the research work. We designed a user interface for interacting with robots. When the end effector of the robotic arm moves, it draws a point. We can inspect the point for our algorithm's effectiveness. When the C# application starts, it communicates with the robot. It fetches the robot's name, robot type, etc., from RoboDK. The information is stored in the application variable for further communication. Every RoboDK object has a unique dynamic id. All commands are sent from the application to the RoboDK must be passed object id as an argument.

2. RELATED WORKS :

Patrick Tresset et al. present a robot, Paul, which is a human face drawing robot as a human does. They implement a visual feedback mechanism to permit the robot to augment and improve a drawing iteratively. The visual feedback is *computational* as it involves a purely internal (memory-based) representation of regions to render via shading by the robot [1]. S. Calinon et al., in their paper, presents a robot capable of drawing artistic portraits. Their research is based on face detection and image reconstruction and classical tools for trajectory planning of a 4 DOFs robot arm using speech recognition and speech synthesis to conduct the scenario [2]. G. Jean-Pierre et al. present an artist robot that draws portraits like a human artist. They use image processing at the back end [3]. R. Y. Putra *et al*., in their research, a 3 DOF arm drawing robot was built. An inverse kinematic model of the robot arm is made using the artificial neural network method. An artificial neural network model was implemented in a GUI application. The ANN model can work in real-time to control arm robot movement to reach specific coordinates. [4]. Chyi-Yeu Lin et al., in their research, developed a robot, which draws the face portrait [5]. S. Jain et al., in their paper, demonstrates a robot equipped with force sensing capability that can draw portraits on a non-calibrated, arbitrarily shaped surface. The robot can draw on a non-calibrated surface by orienting its drawing pen generally to the drawing surface. Several portraits were drawn successfully on a flat surface without calibration [6]. D. Song et al. present a semi-autonomous robotic pen-drawing system capable of creating pen art on an arbitrary surface with varying thickness of pen strokes without reconstructing the surface explicitly. Their robotic system relies on an industrial, seven-degree-of-freedom (7DoF) manipulator that can be both position- and impedance-controlled. They use a vector-graphics engine to take an artist's pen drawing as input and generate Bézier spline curves with varying offsets [7]. K. W. Lo et al. their paper describes the real-time capturing and data analysis of the brush footprint using the platform's new hardware and software capabilities. They include a transparent drawing plate and an underneath camera system, together with projective rectification and video segmentation algorithms [8]. Avinash Kumar Singh et al. research with NAO humanoid robot. They address the fundamental issue of defining a relationship between the points of the image plane and NAO end-effector position [9]. K. Mochizuki et al. paper mainly deals with robot developmental learning on drawing and discusses the influences of physical embodiment on the task. They developed incremental imitation learning to imitate and set the robot's drawing skill using basic shapes: circle, triangle, and rectangle [10].

3. OBJECTIVES :

This research aims to provide some reference information to the robot researcher for their study on robot simulators. They write methods and algorithms for some specific tasks on their robot. The newly implemented methods should test into the simulator Before implementing them into the practical environment. They always find some excellent simulators for the POC (proof of concept). Here we created a simulator environment to test the code. This research used the RoboDK simulator, which is one of the best simulators now. We control the arm movement from the external environment. We make our application using Microsoft visual studio that connects with the RoboDK through the TCP/IP socket. The RoboDK provides C# API. In our research, we created an optimized API which we discussed in the research and methodology section.

4. APPROACH AND METHODOLOGY :

Figure 1 depicts the architecture of our research methodology. Here the C# application is the coordinator between various parts of the project. When the application starts, create two main objects, "Drawing_Board_Csharp" and "Drawing_Board_RoboDK," with all other objects. The

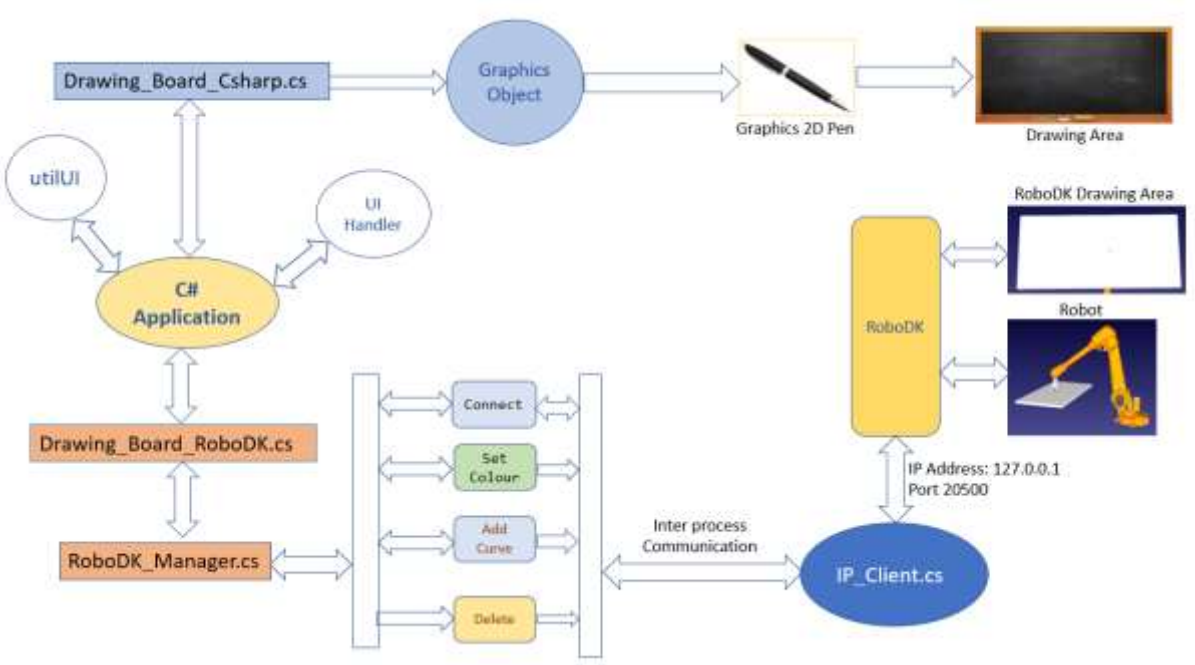

Fig. 1: Architecture of proposed methodology

"Drawing_Board_Csharp" object is responsible for local drawing. When the object is created, it also creates a graphics object from the instance of the panel. This graphics object helps draw the line on the user interface "Panel," which is available in the Visual Studio Toolbox. We add code inside the mouse move event on panel control. When the mouse moves on the panel, it fires the Mouse move event and captures the current mouse X and Y position to push into a point buffer. In every mouse move, the graphics object draws a line using the last two pushed point buffers.

Another important object is "Drawing_Board_RoboDK.," which creates another main object, the RoboDK manager class. It has several methods. "Connect" methods are responsible for connecting with the robot. The "Set color" method is used to change the drawing line color. The "Add Curve" methods are the main methods used to draw lines on the board attached to the robot. To delete the existing drawing line, we use the "Delete" method. All commands are sent to the robot through the TCP/IP communication channel. This channel is created when the robot manager class starts. When RoboDK runs, the TCP/IP server runs and waits for client requests. When any request comes into port 20500, it receives the request and response OK if the command is in the correct format. Several errors may be present inside the received packet. The most common issue we found object id does not match. When we send a command with an improper object id, it throws this type of exception.

When the mouse moves, local drawing is initiated by the C# drawing object. Parallelly, it draws in the remote places, i.e., inside the RoboDK simulator. The local mouse coordinate is converted into a remote coordinate and sends to the "Drawing_Board_RoboDK" object. The point is pushed into a points buffer. Then from the point buffer, fetching the last two-point, create a point object. This point object sends using the Add_Curve method. When drawing the curve, it returns a curve object id, which we store into the point buffer. It is required when we want to clear the drawing. Currently, we use this curve object to set the color object.

5. EXPERIMENT :

Now, let us experiment with our research work. We need two software: one RoboDK and the other one is C# application. For RoboDK, we need to download it from their website. For the C# application, we have already developed. The source code link can be found in recommends section. The two application is communicated through TCP/IP Socket communication. For the same PC, the IP address is 127.0.0.1, which is the localhost address. And the Port number is 20500. Open RoboDK application. Inside the File menu, open the "Drawing with a robot" file from "Example-07.d -Drawing with a robot". Alternatively, we can open it from the downloaded folder. After opening the file, the RoboDK environment looks like figure 2. No interaction is required to move the robot inside the RodoDK IDE. Everything will be controlled from our C# Application.

Fig. 2: RoboDK Environment

Now we need to open the C# application. It looks like figure 3. When the application tries to connect with available Robots in RoboDK, it returns the instance id if the connection is successful. It stores into a variable. In the C# application, we use the panel control to draw. When we move the mouse inside the panel, it shows the X and Y position of the mouse into the respective textbox. Still, we are seeing that nothing is drawing on the local or the remote board. Now Pressing the Mouse, we need to move anywhere inside the panel; it will draw as the mouse move. Figure 4 is the example of the user moving mouse inside the panel, and figure 5 depicts the remote robot has drawn the same thing as we move locally. Behind the scene, two methods are working. One is local drawing, and another one is remote drawing. Calculating the mouse coordinate, it sends to the robot to move. The robot receives the points and draws the line between two points. It is a simulated environment. That is why we need to send another packet to set the drawing color. The pen is connected to a real robot, and the line will draw automatically as the arm moves.

Fig. 3: Application Window

Fig. 4: Drawing Example in apps

Now let us see if something happens which is not expected. If not drawing inside the remote board. This error may happen if communication is not established between two applications. In our C# application, we provide one connect button. When the application starts, it checks the connection with RoboDK automatically. If the connection succeeds, The Back color of the button turns green. If not, turn red. Most of the time, we forget to start RoboDK first. That is why the error comes frequently. If we see a communication error after running the application starts, we can again press the button to communicate with RoboDK.

If multiple RoboDK applications are running, that may create a communication error. If not solved, We have to debug one by one, observing the flow of the code. If everything is OK, whatever drawing on the local board will also replicate on the remote panel. Here Z-axis movement set zero. For our custom application, we can send XYZ value so that it will create three positions. We can observe that our robot is moving to our expected location.

Fig. 5: Drawing in RoboDK

6. RECOMMENDATIONS :

Here is some suggestion for the researcher working on the robotic arm.

- This experiment can be used as a reference. Some customizations are required for the specific project.
- The complete project code for C# and RoboDK station is available. It can download and use it.

- The project source code download link: https://github.com/sudipchakraborty/Drawing-By-Robotic-Arm-using-RoboDK-and-C-Sharp
- The RoboDK software download link https://robodk.com/download
- For better results, the precision value can be adjusted on a need basis. Unnecessary precision creates computation overhead. So before decimal point adjustment, we need to justify the requirement of the high precision result. The computational overhead leads to slow updates where PC or laptop resources are not enough to process faster.

7. CONCLUSION :

For robotic arm research, we need to provide accurate Results to navigate the particular position of the end effector. Before implementing a new algorithm or change some mechanical specification, it should test in a safe environment. Using a robot simulator is the best way to test the experiment. Here, we experimented with a robotic arm end effector to navigate the target. We verified that our algorithm or complete software framework chain is working correctly through a simplistic drawing approach.

REFERENCES :

[1] Patrick Tresset, Frederic Fol Leymarie, (2013). Portrait drawing by Paul the robot. *Computers & Graphics*, *37*(5), 348-363.

[2] Calinon, S., Epiney, J. and Billard, A. (2005). A humanoid robot drawing human portraits. *5th IEEE-RAS International Conference on Humanoid Robots*, pp. 161-166, DOI: 10.1109/ICHR.2005.1573562.

[3] Jean-Pierre, G. and Saïd, Z. (2012). The artist robot: A robot drawing like a human artist. *IEEE International Conference on Industrial Technology*, pp. 486-491, DOI: 10.1109/ICIT.2012.6209985.

[4] Putra R. Y. *et al*. (2016). Neural network implementation for inverse kinematic model of arms drawing robot. *International Symposium on Electronics and Smart Devices (ISESD)*, pp. 153-157, DOI: 10.1109/ISESD.2016.7886710.

[5] Chyi-Yeu Lin, Li-Wen Chuang and Thi Thoa Mac (2009). Human portrait generation system for robot arm drawing. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 1757-1762, DOI: 10.1109/AIM.2009.5229810.

[6] Jain, S., Gupta, P., Kumar, V., and Sharma, K. (2015). A force-controlled portrait drawing robot. *IEEE International Conference on Industrial Technology (ICIT)*, pp. 3160-3165, DOI: 10.1109/ICIT.2015.7125564.

[7] Song, D., Lee, T. and Kim, Y. J. (2018). Artistic Pen Drawing on an Arbitrary Surface Using an Impedance-Controlled Robot. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4085-4090, DOI: 10.1109/ICRA.2018.8461084.

[8] Lo, K. W., Kwok, K. W., Wong, S. M. and Yam, Y. (2006). Brush Footprint Acquisition and Preliminary Analysis for Chinese Calligraphy using a Robot Drawing Platform. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5183-5188, DOI: 10.1109/IROS.2006.281655.

[9] Avinash Kumar Singh, Nandi, G. C. (2016). NAO humanoid robot: Analysis of calibration techniques for robot sketch drawing. *Robotics and Autonomous Systems*, *79*(1), 108-121.

[10] Mochizuki, K., Nishide, S., Okuno, H. G., and Ogata, T. (2013). Developmental Human-Robot Imitation Learning of Drawing with a Neuro Dynamical System. *IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 2336-2341, DOI: 10.1109/SMC.2013.399.

****