Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 176 (2020) 1221–1230

Procedia Computer Science

www.elsevier.com/locate/procedia

24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Augmented reality in robot programming

Fengxin Zhang[a], Chow Yin Lai[b]*, Milan Simic[a], Songlin Ding[a]

[a]School of Engineering, RMIT University, Melbourne, VIC 3000, Australia
[b]University College London, Gower Street, Bloomsbury, London WC1E 6BT, United Kingdom

Abstract

Industrial robots have been traditionally programmed using teaching pendants, whereas offline programming methods are getting increasingly popular in recent years. Although the above two methods are widely-used in the industry, they both have certain disadvantages. For instance, the teaching pendant method requires a shutdown of the production line during the programming process, while offline programming method requires 3D CAD models of both the robot and the workpiece. In this paper, an augmented reality (AR) application which alleviates the aforementioned problems was proposed for robot programming purposes. The application is created using commercially available AR software, with the addition of our JavaScript code. The use of commercially available software allows an easier sharing and widespread adoption of the application.

Keywords: industrial robot; augmented reality; programming, commercially-available software.

1. Introduction

The principle advantage of industrial robots over conventional "hard" automation is the programmability. One single industrial robot can be programmed to do different tasks, such as pick-and-place, painting, assembly, machining and welding. Unfortunately, programming of robots is not a straightforward task and can take hours or days,

* Corresponding author. Tel: +44-2076792911
E-mail address: c.lai@ucl.ac.uk

10.1016/j.procs.2020.09.119

particularly when the geometries of the workpiece are complicated, e.g., robotic machining of turbine blades [1-2], cutting tools and blade disks [3-5]. The recent development of socially assistive robots to help manage chronic health conditions such as chronic obstructive pulmonary disease [6-7] makes the issue of robot programming more important. Conventionally, robots have been programmed using teaching pendants [8], and at the present moment, over 90% of robots are still programmed this way. This method however has the drawback that the production line has to be shut down to facilitate the programming process, resulting in a loss of valuable production time.

Recognising this problem, offline programming methods have been developed to avoid stopping the production line [9]. This approach means programming robots outside of the robot's working environment using specialized software. However, offline programming requires 3D CAD models of both the robot and the workpiece. If the robot is used for maintenance, repair and overhaul (MRO), each part would have suffered different damages. It is therefore challenging to obtain a correct CAD model of the workpiece. While this can be solved by performing a 3D scan of the object, it is very time-consuming to join the point cloud data into surfaces for the purpose of offline programming.

Augmented reality (AR) offers a great opportunity to alleviate the issue in requiring CAD models of the workpiece. The actual object and the virtual robot model can both co-exist in the AR environment, and the workers can now jog the virtual robot to reach target points on the actual workpiece, while making sure that there is no collision or robot singularities. After teaching of the robot in AR software, an executable robot code can then be generated and uploaded to the real robot. The human-computer interaction of AR technology can help inexperienced users to deal with complex robot operations and programming tasks [10]. Moreover, programming using AR also means that the shop floor is not affected during the programming process.

At present, AR is not widely-used in robot programming, despite it offering many advantages as mentioned above. Some of the limited examples are as follows: In [8], a teaching pendant has been created on smart phone, through which users can see the virtual robot movement. A green box or a red box will appear on the virtual robot, depending on whether the robot movement will be safe or unsafe. If the move is safe, users then send a command for the actual robot to follow the same motion as the virtual robot. It is however unclear from the paper whether a full robot code is generated or only a single movement. In the work reported by Rastogi and Milgram [11], AR technology allows the operator to preview the robot's intended motion, and the task is transferred to the real robot if the operation is the same as expected. An example of a pick-and-place task has been demonstrated.

The authors in [12] proposed using AR to train students to operate industrial robots, as well as for sales, for planning the installation, and for servicing and maintaining the robot. In [13], AR is used to program robot and plan trajectory, while taking dynamic constraints of the robot including overshoot into account. In [14], the researchers presented an AR robotic system for trajectory interaction, and compared the programming method with kinesthetic teaching. It was found that it takes less teaching time using the proposed AR robotic interface than performing kinesthetic teaching. Specifying more complex paths e.g. sine curves is much easier using the AR interface.

AR technology observes virtual models in the real world through optical devices. However, because there are still unresolved problems in measuring distance, it is difficult for AR technology to project paths on complex surfaces. Therefore, a new approach is to use interactive laser projection to achieve precise spatial interaction between the programmer and the robot [15]. This method allows the programmer to edit the target coordinates of the robot directly on the surface of the workpiece. By combining optical equipment, handheld devices and AR display technology, the user can draw the target coordinates onto the surface of the work piece just like using a real pen. At the same time, the user can also use buttons on the handheld device to provide additional commands, such as selection or confirmation. This method greatly reduces the difficulty of use. In one experiment, nine engineers with the same experience used a traditional way to complete the path of 12 target points before using a new way of combining with AR. The average time required for the traditional method is 480 seconds, while the AR interface only takes 90 seconds.

Most of the methods described in the above literature require additional equipment, such as tracking equipment and projection equipment, which greatly increases the cost of implementation. Also, most researchers developed their own special software to create the AR environment, and thus are usually not as robust (bug-free) and easily-distributed as commercial software. The aim of our project presented here therefore is to use state-of-the-art commercial CAD software (CREO), AR development software (Vuforia Studio), AR experience software (Vuforia View) and programming software (e.g. Matlab, Python) to create a robot programming app, which can then be run on smartphones or Microsoft Hololens. With a strong user base of these software and hardware, the sharing of the apps will be much easier.

The paper is organized as follows. In Sections 2 and 3, the development of the AR robot programming application will be described. The postprocessing of the target points into robot code will then be detailed in Section 4. The paper

ends with conclusions in Section 5.

2. Creating the Virtual Environment

The first step in creating a robot in the AR environment is to prepare 1:1 CAD models of each individual robot links using the CREO software. These links should not be assembled prior to being imported into Vuforia Studio, as each link needs to be able to move independently. Some robot CAD models can also be downloaded from the internet, but the robot still needs to be "dismantled" to obtain the individual links.

In the next step, CAD models of the links are imported into Vuforia Studio, as shown in Fig. 1(a). Initial positions and orientations of the links are then given so that the robot "looks" complete, as shown in Fig. 1(b). However, there is no constrained kinematic relationship between the links, i.e. no joints are defined. Each link is merely given a position and orientation to look as if it forms the complete robot.

A virtual identification pattern called "ThingMark" is attached to or placed next to the virtual robot, see Fig. 1. Later (after programming is completed so as not to disturb the production line), a physical printout of the ThingMark will also be attached to or placed next to the real robot at exactly the same position as that on the virtual robot. The ThingMarks serve as references for the position of objects and the robot in AR and physical environment.

(a) (b)

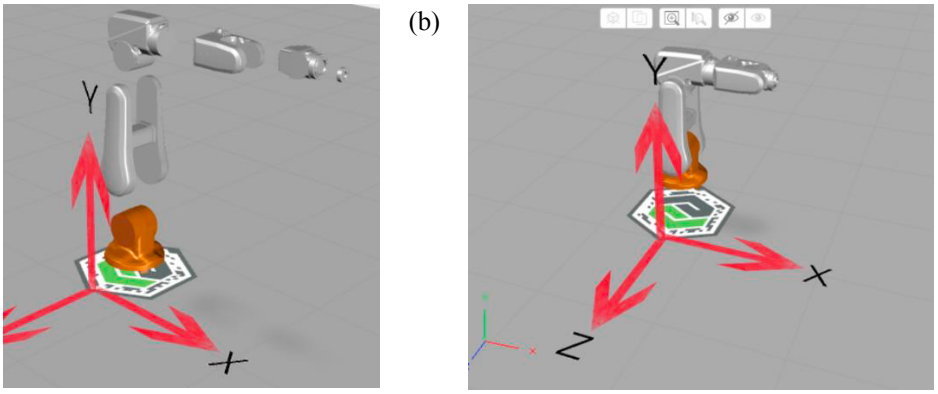

Fig. 1: (a) CAD of robot links imported into Vuforia Studio, (b) Links "assembled" into a complete robot; Virtual ThingMark attached to the robot.

3. Manipulating the Virtual Robot in Vuforia Studio using Javascripts

After the virtual robot is imported into Vuforia Studio, Javascripts will need to be written within Vuforia Studio to manipulate the robot and visualize its movement. To clarify what tasks the Javascripts are required to perform, it would be useful to first understand the process flow of the AR robot programming application, as shown in Fig. 2.

Firstly, the user opens the "Vuforia View" app on smartphones or Microsoft Hololens. By directing the camera view towards the physical printout of the ThingMark, the user will see the virtual robot in its initial configuration as shown in Fig. 1(b) and Fig. 3(a). At this initial configuration, the joint angles are known since these are the default values set by the user. The values of the joint angles (T1 to T6, where "T" stands for Theta) are also shown on the AR app.

User then places a real object beside the virtual robot, as shown in Fig. 3(a). By using arrow buttons on the AR robot programming application, the user can either directly command the change of joint angles, or command the virtual end-effector to move in the x, y, z direction and rotate about the x, y or z axis. For the latter, an inverse kinematics will be invoked to calculate the joint angle needed to move the end-effector as required by the user.

Using the commanded or calculated joint angles, the position and orientation of each link and end-effector will be updated based on forward kinematics and reflected visually as a movement of the virtual robot. The user continues pressing the buttons until the end-effector reaches the target position, see Fig. 3(b), at which point the user will record the Cartesian position and orientation of the end-effector, along with the robot joint angles. The process is repeated

until all the target points are taught. At the time of writing, the recording of the end-effector positions and joint angles is done manually, i.e. reading the values on the AR screen and typing into a .txt or .csv file. Automating this will be focus of our future work.

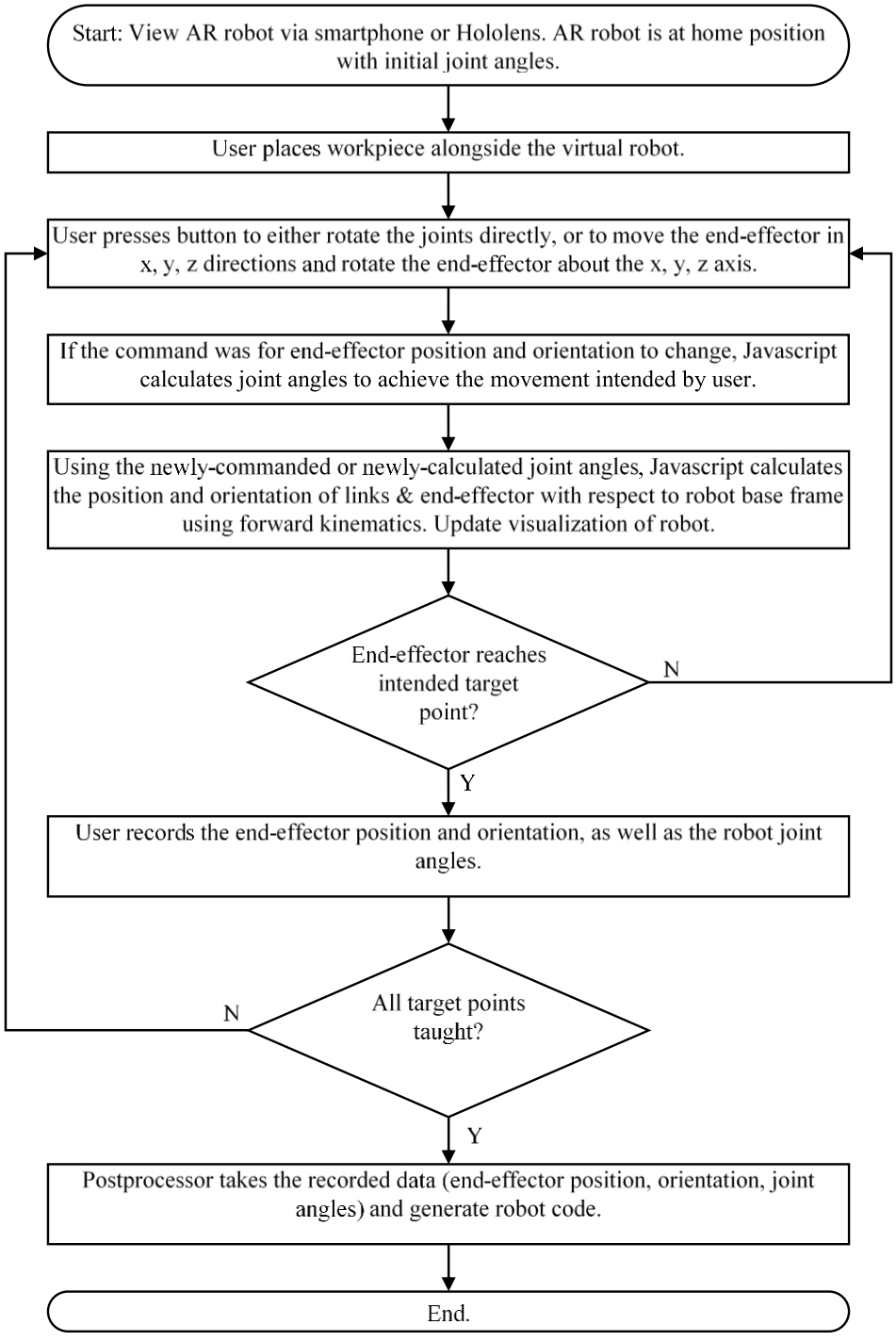

Fig. 2: AR Programming Flowchart

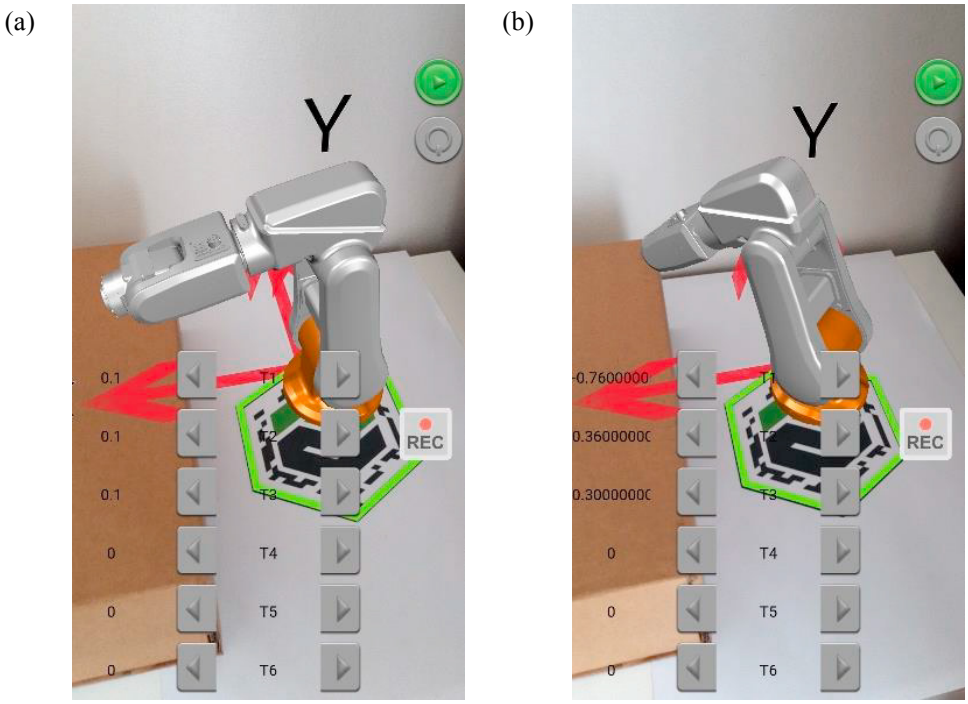

Fig. 3: (a) Virtual robot with real object; (b) Robot pointing to one corner of object

As can be seen, Javascripts will be required for computation of inverse kinematics and forward kinematics. As an example, the forward kinematics for an ABB IRB120 robot, as shown in Fig. 4, is derived here to show how the Javascripts are coded.

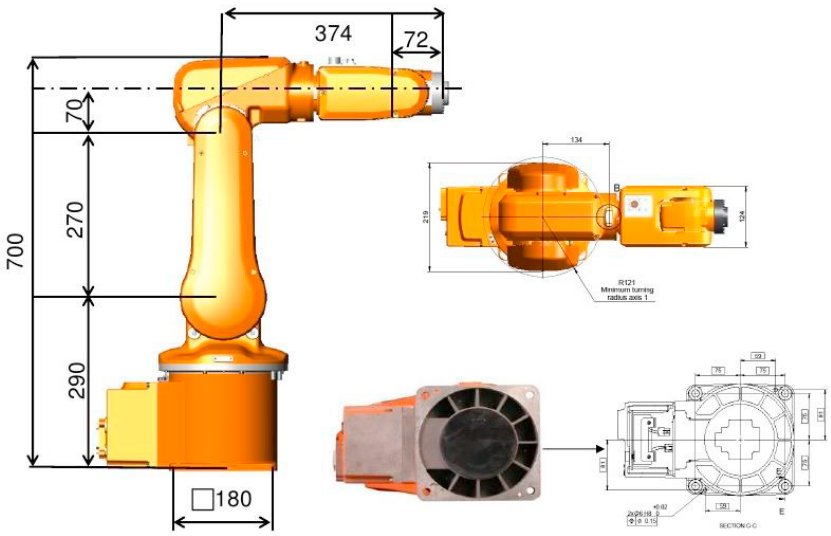

Fig. 4: Dimensions of an ABB IRB120 robot [16].

While the kinematic models of robots are generally well-understood and can be derived using the well-known DH parameters, unfortunately the axis definition in Vuforia Studio is not exactly the same as in robotics textbooks / literature. For instance, rotation axes are always defined as z-axis in robotics texts, but this is not necessarily the case in Vuforia Studio – It depends on how the CAD models were drawn in the first place. Therefore, it is necessary to re-derive the kinematic model from scratch.

To obtain the kinematic model, the relationship between frame *i*-1 and frame *i* need to be calculated, and it is done as follows:

From frame 0 to 1: It was observed that the joint 1 rotates by $-\theta_1$(variable) about the y-axis in Vuforia studio. The transformation matrix from frame {0} to {1} is therefore:

$$ {}^{0}_{1}T = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ 0 & 1 & 0 & 0 \\ -S_1 & 0 & C_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

where c_i and s_i are abbreviations for $\cos\theta_i$ and $\sin\theta_i$ respectively, θ_i being the angle of joint i.

From frame 1 to 2: Rotation by 90º about the x-axis, followed by a rotation by θ_2(variable) about the y-axis and a translation by the link lengths of L_1 in the y-direction:

$$ {}^{1}_{2}T = \begin{bmatrix} C_2 & 0 & S_2 & 0 \\ -S_2 & 0 & C_2 & L_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

Combining the above, we have the compound transformation from frame 0 to frame 2 as:

$$ {}^{0}_{2}T = {}^{0}_{1}T \cdot {}^{1}_{2}T = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ 0 & 1 & 0 & 0 \\ -S_1 & 0 & C_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_2 & 0 & S_2 & 0 \\ -S_2 & 0 & C_2 & L_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_1C_2 & -S_1 & C_1S_2 & 0 \\ -S_2 & 0 & C_2 & L_1 \\ -S_1C_2 & -C_1 & -S_1S_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3} $$

An excerpt of the Javascript showing transformation matrices ${}^{0}_{1}T$ and ${}^{0}_{2}T$ is shown in Fig. 5.

```
$scope.app.params.R0111 = Math.cos($scope.app.params.jointAngle1);
$scope.app.params.R0112 = 0;
$scope.app.params.R0113 = Math.sin($scope.app.params.jointAngle1);
$scope.app.params.R0121 = 0;
$scope.app.params.R0122 = 1;
$scope.app.params.R0123 = 0;
$scope.app.params.R0131 = -Math.sin($scope.app.params.jointAngle1);
$scope.app.params.R0132 = 0;
$scope.app.params.R0133 = Math.cos($scope.app.params.jointAngle1);

$scope.app.params.R0211 = Math.cos($scope.app.params.jointAngle1)
      *Math.cos($scope.app.params.jointAngle2);
$scope.app.params.R0212 = -Math.sin($scope.app.params.jointAngle1);
$scope.app.params.R0213 = Math.cos($scope.app.params.jointAngle1)
      *Math.sin($scope.app.params.jointAngle2);
$scope.app.params.R0221 = -Math.sin($scope.app.params.jointAngle2);
$scope.app.params.R0222 = 0;
$scope.app.params.R0223 = Math.cos($scope.app.params.jointAngle2);
$scope.app.params.R0231 = -Math.sin($scope.app.params.jointAngle1)
      *Math.cos($scope.app.params.jointAngle2);
$scope.app.params.R0232 = -Math.cos($scope.app.params.jointAngle1);
$scope.app.params.R0233 = -Math.sin($scope.app.params.jointAngle1)
      *Math.sin($scope.app.params.jointAngle2);
```

Fig. 5: Excerpt of Javascript for forward kinematics.

The rest of the transformations is as follows:

$$ {}^{0}_{3}T = \begin{bmatrix} C_1C_{23} & -S_1 & C_1S_{23} & C_1S_2L_2 \\ -S_{23} & 0 & C_{23} & C_2L_2 + L_1 \\ -S_1C_{23} & -C_1 & -S_1S_{23} & -S_1S_2L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

$$ {}^{0}_{4}T = \begin{bmatrix} C_1C_{23} & -S_1C_4 + C_1S_4S_{23} & S_1S_4 + C_1S_4S_{23} & C_1C_{23}L_3 + C_1S_{23}L_4 + C_1S_2L_2 \\ -S_{23} & C_{23}S_4 & C_{23}C_4 & -S_{23}L_3 + C_{23}L_4 + C_2L_2 + L_1 \\ -S_1C_{23} & -C_1C_4 - S_1S_4S_{23} & C_1S_4 - S_1C_4S_{23} & -S_1C_{23}L_3 - S_1S_{23}L_4 - S_1S_2L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5} $$

$$ {}^{0}_{5}T = \begin{bmatrix} \begin{bmatrix} C_1C_{23}C_5 - S_1S_4S_5 \\ -C_1C_4S_5S_{23} \end{bmatrix} & [-S_1C_4 + C_1S_4S_{23}] & \begin{bmatrix} C_1C_{23}S_5 + S_1S_4C_5 \\ +C_1C_4C_5S_{23} \end{bmatrix} & \begin{bmatrix} C_1C_{23}L_5 + C_1C_{23}L_3 \\ +C_1S_{23}L_4 + C_1S_2L_2 \end{bmatrix} \\ [-S_{23}C_5 - C_4S_5C_{23}] & [C_{23}S_4] & [-S_{23}S_5 + C_{23}C_4C_5] & \begin{bmatrix} -S_{23}L_5 - S_{23}L_3 \\ +C_{23}L_4 + C_2L_2 + L_1 \end{bmatrix} \\ \begin{bmatrix} -S_1C_{23}C_5 - C_1S_4S_5 \\ +S_1C_4S_5S_{23} \end{bmatrix} & [-C_1C_4 - S_1S_4S_{23}] & \begin{bmatrix} -S_1S_5C_{23} + C_1S_4C_5 \\ -S_1C_4C_5S_{23} \end{bmatrix} & \begin{bmatrix} -S_1C_{23}L_5 - S_1C_{23}L_3 \\ -S_1S_{23}L_4 - S_1S_2L_2 \end{bmatrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

$$ {}^{0}_{6}T = \begin{bmatrix} \begin{bmatrix} C_1C_{23}C_5 - S_1S_4S_5 \\ -C_1C_4S_5S_{23} \end{bmatrix} & \begin{bmatrix} -S_1C_4C_6 + C_1S_4C_6S_{23} \\ +C_1S_5S_6C_{23} + S_1S_4C_5S_6 \\ +C_1C_4C_5S_6S_{23} \end{bmatrix} & \begin{bmatrix} S_1C_4S_6 - C_1S_4S_6S_{23} \\ +C_1C_6C_{23}S_5 + S_1S_4C_5C_6 \\ +C_1C_4C_5C_6S_{23} \end{bmatrix} & \begin{bmatrix} C_1C_{23}C_5L_6 - S_1S_4S_5L_6 \\ -C_1C_4S_5S_{23}L_6 + C_1C_{23}L_5 \\ +C_1C_{23}L_3 + C_1S_{23}L_4 \\ +C_1S_2L_2 \end{bmatrix} \\ [-S_{23}C_5 - C_4S_5C_{23}] & \begin{bmatrix} C_{23}S_4C_6 - S_5S_6S_{23} \\ +C_4C_5S_6C_{23} \end{bmatrix} & \begin{bmatrix} -C_{23}S_6S_4 - S_5C_6S_{23} \\ +C_{23}C_4C_5C_6 \end{bmatrix} & \begin{bmatrix} -S_{23}C_5L_6 - C_4S_5C_{23}L_6 \\ -S_{23}L_5 - S_{23}L_3 \\ +C_{23}L_4 + C_2L_2 + L_1 \end{bmatrix} \\ \begin{bmatrix} -S_1C_{23}C_5 - C_1S_4S_5 \\ +S_1C_4S_5S_{23} \end{bmatrix} & \begin{bmatrix} -C_1C_4C_6 - S_1S_4C_6S_{23} \\ -S_1S_5S_6C_{23} + C_1S_4C_5S_6 \\ -S_1C_4C_5S_6S_{23} \end{bmatrix} & \begin{bmatrix} C_1C_4S_6 + S_1S_4S_6S_{23} \\ -S_1C_6C_{23}S_5 + C_1S_4C_5C_6 \\ -S_1C_4C_5C_6S_{23} \end{bmatrix} & \begin{bmatrix} -S_1C_{23}C_5L_6 - C_1S_4S_5L_6 \\ +S_1C_4S_5S_{23}L_6 - S_1C_{23}L_5 \\ -S_1C_{23}L_3 - S_1S_{23}L_4 \\ -S_1S_2L_2 \end{bmatrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

where c_{ij} and s_{ij} mean $\cos(\theta_i + \theta_j)$ and $\sin(\theta_i + \theta_j)$ respectively, and L_k is the length of link k. The inverse kinematics of the robot can then be calculated using the method shown in [17], and is also coded in Javascript, though it is not shown here due to page limitations.

4. Postprocessing of Target Points into Robot Code

In Section 3, it was mentioned that the end-effector's position and orientation as well as the robot joint angles were recorded. These can be provided in any standard format such as .txt or .csv, an example of which is shown in Fig. 6. A postprocessor code, written in standard software such as Matlab or Python, will then transform the data points into actual robot code. An example of the Matlab code is given in Fig. 7 and Fig. 8, whereas the output file is shown in Fig. 9. This output file can eventually be uploaded to the actual robot.

T1	T2	T3	T4	T5	T6	x	y	z	rx	ry	rz
-10.15	-5.63	21.53	31.67	16.89	-35.67	550.226	-49.872	349.549	-180	21	-180
10.35	-5.59	21.5	-32.18	16.99	36.26	550.226	50.917	349.549	-180	21	-180
15.9	-27.32	37.89	-35.02	24.42	40.65	450.226	50.917	349.549	-180	21	180
-15.59	-27.38	37.93	34.49	24.27	-40	450.226	-49.869	349.549	-180	21	180

Fig. 6: An example list of joint angles (T1 to T6, where T stands for Theta), target positions (x, y, z) and orientation (rx, ry, rz) in a .txt file.

```
%% Read in Data
fid = fopen('Data.txt','rt');
indata = textscan(fid,'%f %f %f %f %f %f %f %f %f %f %f %f','HeaderLines',1,... % skip header
    'Delimiter',',','TreatAsEmpty',{'NA','na'},'CommentStyle','//'); % use delimiter
fclose(fid);

celldisp(indata);
completedata = [indata{1},indata{2},indata{3},indata{4},indata{5},indata{6},indata{7},indata{8},...
    indata{9},indata{10},indata{11},indata{12}];
```

Fig. 7: Matlab code to read in data from .txt file

```
%% Postprocess Row by Row and Write into Output File

[m,n] = size(completedata);
for i = 1:m
    x = completedata(i,7); % Positions in x y z
    y = completedata(i,8);
    z = completedata(i,9);
    rx = completedata(i,10); % Rotations in roll pitch yaw
    ry = completedata(i,11);
    rz = completedata(i,12);
    quat = angle2quat(rz, ry, rz); % Change rotations in quaternion
    quat1 = quat(1);               % as that's ABB format
    quat2 = quat(2);
    quat3 = quat(3);
    quat4 = quat(4);

    if completedata(i,1) >= 0 && completedata(i,1) < 90 % Configuration in ABB format
        conf1 = 0;
    elseif completedata(i,1) >= 90 && completedata(i,1) < 180
        conf1 = 1;
    elseif completedata(i,1) >= 180 && completedata(i,1) < 270
        conf1 = 2;
    else
        conf1 = -1;
    end
    if completedata(i,2) >= 0 && completedata(i,2) < 90 % Configuration in ABB format
        conf2 = 0;
    elseif completedata(i,2) >= 90 && completedata(i,2) < 180
        conf2 = 1;
    elseif completedata(i,2) >= 180 && completedata(i,2) < 270
        conf2 = 2;
    else
        conf2 = -1;
    end
    fileID = fopen('RobotCode.txt','a+'); % write and expand
    formatSpec = ['MoveL [[%f,%f,%f],[%f %f %f %f],[%f %f %f %f],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]]'...
        ',v10,z1,tool0\\Wob:=wobj0;\n'];
    fprintf(fileID,formatSpec,x,y,z,quat1,quat2,quat3,quat4,conf1,conf2,conf3,conf4);
    fclose(fileID);
end
```

Fig. 8: Matlab code to write ABB code from .txt file (calculation conf3 and conf4 not shown due to page limitation).

```
MODULE MOD_MainProgram

PROC Main()

MoveL [[550.226000,-49.872000,349.549000],[-0.798553 0.161896 -0.556680 0.161896],[-1.000000 -1.000000
    0.000000 0.000000],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]],v10,z1,tool0\Wob:=wobj0;
MoveL [[550.226000,50.917000,349.549000],[-0.798553 0.161896 -0.556680 0.161896],[0.000000 -1.000000 0.00000
    -1.000000],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]],v10,z1,tool0\Wob:=wobj0;
MoveL [[450.226000,50.917000,349.549000],[-0.798553 -0.161896 -0.556680 -0.161896],[0.000000 -1.000000
    0.000000 -1.000000],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]],v10,z1,tool0\Wob:=wobj0;
MoveL [[450.226000,-49.869000,349.549000],[-0.798553 -0.161896 -0.556680 -0.161896],[-1.000000 -1.000000
    0.000000 0.000000],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]],v10,z1,tool0\Wob:=wobj0;

ENDPROC

ENDMODULE
```

Fig. 9: Robot code generated by postprocessor.

5. Conclusion

In this paper, we have reported our research activities on the development of the robot programming applications using augmented reality (AR). This approach enables robot programming without stopping the production line, as well as without the need for 3D CAD model design of the workpiece. This is extremely important when using robots for maintenance, repair and overhaul (MRO) in industry environment where it is not easy to obtain the CAD model of damaged workpiece. The use of commercial AR software, combined with our JavaScript addition, has the advantage that it is more robust than creating special software from scratch, and it would also have more widespread use by users around the world because of a larger user base. Our new AR application does not require additional tracking and projection equipment and can therefore improve efficiency and reduce manufacturing costs.

Acknowledgements

The authors would like to thank Mr. Bryan Kirchner from LEAP Australia for providing technical support and advice on the use of Vuforia Studio to complete the project.

References

[1] Ding, S., Yang, D.C.H., Han, Z. (2005) “Boundary-conformed machining of turbine blades”, *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* 219(3), pp. 255-263.

[2] Lai, C.Y., Villacis Chavez, D.E., Ding, S. (2018) “Transformable parallel-serial manipulator for robotic machining”, *International Journal of Advanced Manufacturing Technology,* 97(5-8), pp. 2987-2996.

[3] Bani Melhem, M.K., Simic, M., Lai, C.Y., Feng, Y., Ding, S. (2020) “Fuzzy control of the dual-stage feeding system consisting of a piezoelectric actuator and a linear motor for electrical discharge machining”, *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* 234(5), pp. 945-955.

[4] Melhem, M.B., Simic, M., Feng, Y., Ding, S. (2020) “Dual stage control of a servo system consisting of a piezoelectric actuator and a linear motor for electrical discharge machining”, *IOP Conference Series: Materials Science and Engineering*, 715(1),012051.

[5] Ding, S. and Jiang, R. (2004) “Tool path generation for 4-axis contour EDM rough machining”, *International Journal of Machine Tools and Manufacture*, 44(14), pp. 1493-1502.

[6] Broadbent E, Garrett J, Jepsen N, Ogilvie VL, Ahn HS, Robinson H, Peri K, Kerse N, Rouse P, Pillai A, MacDonald B （2018）Using robots at home to support patients with chronic obstructive pulmonary disease: pilot randomized controlled trial”, *Journal of Medical Internet Research*，20(2):e45.

[7] Inthavong, K., Tu, J., Ye, Y., Ding, S., Subic, A., Thien, F. (2010) “Effects of airway obstruction induced by asthma attack on particle deposition”, *Journal of Aerosol Science,* 41(6), pp. 587-601.

[8] Abbas, S. M., Hassan, S. and Yun, J. (2012). “Augmented reality-based teaching pendant for industrial robot”, in *Proceedings of the12th International Conference on Control, Automation and Systems*, Jeju Island, South Korea, 2012, pp. 2210-2213.

[9] Tang, X. H., and Drews, P. (2005). “3D-visualized offline-programming and simulation system for industrial robots.” *Hanjie Xuebao/Transactions of the China Welding Institution* **26(2)**: 64-68.

[10] Andersson, N., Argyrou, A., Naegele, F., Ubis, F., Campos, U. E., de Zarate, M. O., and Wilterdink, R. (2016). “AR-Enhanced Human-Robot-Interaction - Methodologies, Algorithms, Tools.” *Procedia CIRP* **44**: 193-198.

[11] Rastogi, A., Milgram, P. (1995). "Augmented telerobotic control: a visual interface for unstructured environments", in *Proceedings of the KBS/robotics conference*, Montreal, Canada, 1995. pp. 16–18.

[12] Bischoff, R., and Kazi, A., "Perspectives on Augmented Reality Based Human-Robot Interaction with Industrial Robots", in *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 28 – Oct 2, 2004, pp. 3226-3231.

[13] Fang. H. C., Ong. S. K., and Nee. A. Y. C. (2012) "Interactive Robot Trajectory Planning and Simulation using Augmented Reality." *Robotics and Computer-Integrated Manufacturing*, **28**: 227-237.

[14] Quintero, C. P., Li, S., Pan, M. K. X. J., Chan, W. P., Van der Loos, H. F. M., and Croft, E. "Robot Programming through Augmented Trajectories in Augmented Reality", in *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 1-5, 2018, pp. 1838-1844.

[15] Zaeh, M.F., and Vogl, W. (2006). "Interactive laser-projection for programming industrial robots", in *Proceedings of the IEEE/ACM International Symposium on Mixed and Augmented reality 2006 (ISMAR2006),* Santa Barbara, USA, 2006, pp. 125–128.

[16] ABB (2019), "Product Specification: IRB 120".

[17] Craig, J. J. (1986), "Introduction to Robotics: Mechanics and Control, Second Edition." Addison Wesley, pp. 131-136.