*Journal of Machine Engineering, 2019, Vol. 19, No. 1, 128–145*
*ISSN 1895-7595 (Print)* *ISSN 2391-8071 (Online)*

Received: *19 December 2018* / Accepted: *15 February 2019* / Published online: *11 March 2019*

*digital twins, industrial robots,*
*virtual factory, virtual reality*

Vladimir KUTS[1*]
Tauno OTTO[1]
Toivo TÄHEMAA[1]
Yevhen BONDARENKO[1]

# DIGITAL TWIN BASED SYNCHRONISED CONTROL AND SIMULATION OF THE INDUSTRIAL ROBOTIC CELL USING VIRTUAL REALITY

During the years common understanding of the possibilities and perspectives of Virtual Reality (VR) usage has been changed. It is thought that VR is mainly used in entertainment purposes, but it is being used already for many years in different industries, and now with easier access to the hardware it became a helpful and accessible tool that could be used and developed in any field of human activities. In manufacturing, immersive technologies are mainly used nowadays for the visualisation of processes and products combining those visuals into the factory Digital Twin (DT) which is possible to view from the inside look. This feature is already being used in several manufacturing simulation tools, which enable to view onto industrial line / robotic cells via Virtual Reality glasses. However, the potential of using simulations with VR in manufacturing is not fully uncovered. The main aim of this, industrial robotics targeted research is to enable besides simulation also universal control algorithms through Virtual Reality experience, produced by game engine Unity3D, which can be easily modified for a wide range of industrial equipment. The primary outcome of this work is the development of the synchronisation model of real and virtual industrial robots and experimental testing the developed model in Virtual Reality and shop floor labs

## 1. INTRODUCTION

Years ago, most people thought that Virtual Reality (VR) could be used only for gaming and another type of entertainment purposes, but actually, it is being used in research already few decades. Now VR could be used in any sphere of human activities. For example, it can be used in architecture or design, where people can firstly try their projects in a digital environment before going to real projects. This way can prevent dozens of mistakes and errors without any loss. Also, VR can be a good helper for education in schools and universities. Students will be able to see how theoretical knowledge from lessons could be implemented in real life. One more approach is professional work power

[1] Tallinn University of Technology, School of Engineering, Department of Mechanical and Industrial Engineering, Estonia, Tallinn
[*] E-mail: vladimir.kuts@taltech.ee
https://doi.org/ 10.5604/01.3001.0013.0464

training. VR technologies will be the proper apparatus to simulate working environments for medical, military or aviation purposes. To be more precise – there are different research areas, where control and improvement of how to better use and visualise different information, about the VR and Augmented Reality (AR). VR simulation could also be used in manufacturing, robotics and control systems [1–4] by simulating different algorithms and control methods. Manufacturing, robotics, and control Architecture [5] of varying scale rooms and buildings, for finding the most efficient way of design. Audio [6] – to use VR as a sound visualisation tool for the artist or blind people. Moreover, smart racks and simulations are done with force feedback [7, 8] for a better haptic feeling of tested in VR joysticks, devices or manipulators. So of course entertainment and education [9, 10] area for precise training simulations for different level workers and students.

From the other side, some issues may arise. First of them is the lack of people familiar to the VR field. As this technology is new and not so widely known, there is still a lack of developers, who have enough experience to create a ready-to-go project, because the production process of VR environment is hard work. In comparison to mobile applications, where one programmer or artist can do all the stuff, as pixel graphics and writing the code, VR requires high-quality 3D models, as well as, an understanding of human feelings - motion sickness. Also, it is essential to have a team, because developing in VR means testing application alongside with programming and doing it alone may be very time-consuming. The last, but not the least, it is hardware. Technologies such as VR requires very powerful newest personal computers (PC) with headsets and stations, such as HTC Vive or Oculus Rift, which can be very costly for small and medium enterprises (SME-s) and stand-alone developers. This entire means that integration of VR technologies to the SME-s is still complicated because of the lack of workforce or knowledge about the possibilities.

The practical aim of the research is to create Industrial Digital Twin (DT) – a digital copy of the real manufacturing system, which can be controlled and programmed in real-time directly from the computer application model of the industrial robot. It includes the creation of the precise model of the robot and developing a software package to control and program it directly from VR. The work also analyses how creating the DT can improve workspace awareness of the real robot without using any additional physical equipment, but an only accurate computer simulation. Moreover, our tool, developed during this research, with the usage of immersive technologies, is able to visualize in real scale manufacturing lines and robotics cells not only in a purpose of simulation and demo, but also for control of the actual work process – interactive online tools, which gives ability to re-program line in real-time. With it, downtime is reduced to the minimum, as all optimisations and new product production are being done in DT and then via network transferred to middle layer controller in seconds, allowing using the new program from a new loop of the process in reality. Thus, reduces downtime and money waste on re-programming and it increases the overall efficiency of the manufacturing process. Moreover, the aim of this project is a creation of the very flexible, user-friendly and modular environment, which can be easily modified, connected to the real manufacturing assets, and accessible by the broad public.

Toward this research, an experimental approach to the development of the methodology is being introduced. Both method, how-to and a use-case are combined into the main section. The outcomes of this work were tested and validated as a part of a more significant project in Tallinn University of Technology – a recreation of the Industrial Virtual and Augmented Reality Laboratory (TalTech IVAR laboratory) in VR.

## 2. DEVELOPMENT OF THE EXPERIMENTAL ENVIRONMENT

This part of the research describes the main development steps that were taken to implement the DT of industrial robot – Motoman GP8 in VR. Each of the steps, which include model preparation, programming and optimising for VR, their challenges and outcomes are analysed in detail in the corresponding sections. The primary software tool selected for the experimental research realisation is the Unity3D game engine. This approach can be developed in various similar engines, for example, Unreal Engine (UE), but because of the previous author's experience was chosen Unity, not to spend time on re-learning. Unity provides a simple but powerful development environment with a modular approach to programming and also offers integration with all commercially available VR systems, which is perfect for the defined task. 3DS Max and Maya from Autodesk were used as 3D modelling software for this project. All development cycles mentioned above are thoroughly explained in the corresponding sections of this paper. Also, the project described above was initially designed to work with the specific hardware: HTC Vive headset for VR capabilities and Yaskawa Motoman GP8 industrial robot for testing the DT concept. However, the software was developed with a modular programming approach in mind and, as a result, it can be easily extended to support other current VR platforms and to control DT of many different models of industrial robots and other equipment. Mainly, immersive technologies and tools of how to make those are being used for visualization and just a simulation of production processes [11], but more and more research is done on integration of different control inputs like Robot Operation System (ROS) to the different layers of simulations and modelling a test environment [12–15]. In the methodology described below, we intend to combine all mentioned above together and propose an alternative to the ROS environment.

### 2.1. PREPARATION OF THE ROBOT MODEL

The simulation required both models of an industrial robot and its training station. Digital models of the robots can be taken from the manufacturer website or related software libraries. Though the models imported, were built in proportions exactly corresponding to their real-world counterparts, there were several important issues to address before importing them to Unity3D game engine:

- Robot model had to be rigged. Rigging in related context means defining the location of pivot points in the models so that the program can get the axes around which robot's links are rotated. This operation had to be done precisely to keep the quality of the simulation.

- The robot stand model had to be simplified for VR because the original geometry was too "heavy" to be rendered in real time.
- Both models had the correct proportions when exported from manufacturer website or software, but their scale had to be checked and adjusted manually after importing to Unity3D.

Rigging of the robot model can be done using Blender, Autodesk 3DS Max or other similar software which can convert CAD models into .fbx file format – for this research 3DS Max were used. The process of rigging consisted of defining the correct coordinates of the model's pivot points (robot's axes) and aligning robot meshes (3D geometry) to them. Besides, it was essential to set up all links and joints of the robot into the correct hierarchy. Hierarchy allows the model to be controlled by game engine in the desired manner: when the parent geometry is moved (for example, the first joint of the robot is rotating), all child geometry follows along (part of the arm located above the actuated joint is rotating too). The hierarchy was set up in such a way that the robot's rig (relative locations of the pivot points) is not affected when adding and scaling link meshes to it. That is useful because the exported geometry itself consists of polygons, and never can correctly copy the dimensions of the real object. However, it is possible to give exact coordinates to the model's pivot points and maintain an ideal accuracy between the real robot and model in joints and end-effector positions.

Figure 1 demonstrates how pivot points of the model in 3D modelling software copy the robot's geometry from the mechanical drawing.

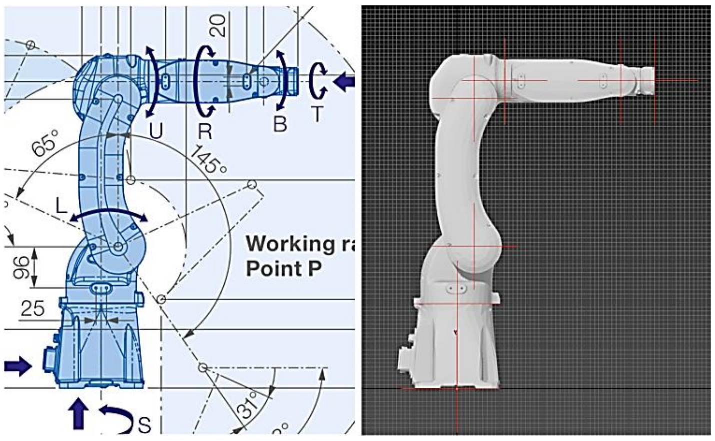


Fig. 1. Comparison of the robot's mechanical drawing and its pivot point representation in 3DS Max

Optimising of the industrial robot stand model was done in Autodesk Maya. It was chosen with reason as this 3D modelling software simplifies work with high-polygonal models and provides tools to simplify their geometry. The first part of optimisation was reducing the polygon count of the model, which, as a result, has decreased from more than 1.000.000 faces to around 80.000 without losing dimensional precision of the model. The second part of the optimisation process was minimising the count of materials used in the model after importing it to Unity3D. Materials of the model define the way it looks in a 3D application (i.e., colour, surface texture). The problem was in the number of materials generated for the model when it was exported from manufacturer software – several thousands of materials were created and added to the model because the program produced new material for each model's mesh. As a result, the model was unsuitable for real-time rendering, especially in VR. The solution was .to remove the generated materials and create several new ones to replace them. Achieved result is the model, which is ready to use in the simulation and without the significant loss of visual quality

Scaling the models was the last part of geometry preparation before the actual programming of DT. To represent the real robot station in the correct size in VR, both the stand and the robot models had to be downscaled to the right dimensions and positioned correctly relative to each other. The operation was non-complex due to the dimension system of Unity, where 1 unit of distance in the game scene equals 1 meter in the real world. This dimensioning is being preserved when inside a VR simulation, so the user can experience the robot’s model in 1:1 scale stereo environment, which gives to person very precise presence feeling.

## 2.2. ROBOT CONTROL SCRIPTS

To power the Digital Twin system created in this project, some scripts in C Sharp (C#) language was designed and tested in the Unity3D game engine. Unity3D uses a modular approach to application development, which is implemented as Game Objects (models, geometry, effects, etc.) and Components (C# scripts which control Game Object behaviour) attached to them. To make further future developments simpler was decided to maintain this modular approach when developing the project’s programming base as well.

The scripts used in the experiment can be separated into three major parts according to their functions:

- Control scripts family, which provides methods to control a generic industrial arm robot model. DT controller belongs here as well.
- Programming scripts, which allow creating simple systematic programs, which can be later, run on the robot models managed by Control scripts.
- Collision detection scripts, which monitor the position of the robots led by Control scripts to check for potential collisions with environment objects and stop when a possible collision is detected. These scripts are intended to be used with virtual robot models inside Unity3D.

Robot control scripts are built in a hierarchy structure, which can be seen in Fig. 2. The functions of each script are described in the following sections.

*Base control script*

RobotController is an abstract base class, which means that it contains no actual code to be run, but the definitions of methods and logic, which has to be implemented by any class inheriting from it.

RobotController class is the core of all application structure because it is an element, which enables the universality and consistency among the interfaces of all other Controller classes. Due to it any other script, for example, RobotProgrammer can send commands to the controlled robots (whether real or virtual) without the risk of producing an error because of a non-existing method called.

Here is the list of public (which can be accessed by any other scripts) methods and properties defined in base RobotController class:

- public bool isMoving – returns true if the controlled robot is currently moving,
- public void SetSpeed() – sets the speed of the robot (0 to 100%),
- public float GetJointAngle(int jointNumber) – returns the angle of the given robot joint,
- public List<float> GetJointsAngles() – returns the list with all current joint angle values of the robot,
- public void MoveJointToAngle(int jointNumber, float angle) – moves given robot's joint to a given angle with the currently set speed,
- public void MoveJointsToAngles(List<float> targetAngles) – accepts a list of angle values, and then moves all joints of the robot to these corresponding angles with the currently set speed,
- public abstract void MoveToEndpoint(Transform endPoint) – accepts Transform (Unity's representation of position in 3D space), then moves robot's end-effector to this point using Inverse Kinematics (IK) with the currently set speed,
- public abstract void MoveRobotToZero() – a shortcut command to move all robot's joints to their zero positions,
- public abstract void Stop(bool emergency = false) – stops the current robot movement as a default; if the given emergency parameter is true, stops the robot urgently (i.e., disables servos when controlling the real robot).

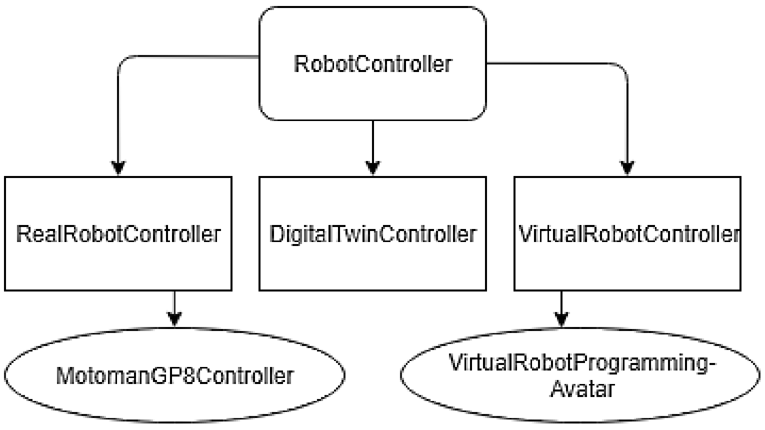


Fig. 2. Robot control scripts hierarchy

Also, RobotController defines some protected methods to be used only inside the deriving classes:

- protected abstract void InitializeRobot() – this method is run internally when the control script loads; intended for setting references and initialising all required parameters of the robot,
- protected abstract void MoveRobotJoints(List<float> targetJointAngles) – moves all joints of the robot to the corresponding angles set in targetAngles list; this function is

executed internally each time MoveJointToAngle(…), MoveJointsToAngles(…), MoveToEndpoint(…) or MoveRobotToZero () methods are called on the robot,
- protected IEnumerator RobotMovement_Coroutine(List<float> targetJointAngles) – an internal Coroutine (the method that is run in a loop over multiple game frames) which implements joints' movement process to the given angles with set speed,
- protected IEnumerator RobotMovement_Coroutine(Transform targetPoint) – an internal Coroutine, an analogue of the previous one, but is used for movement using IK.

Again, these methods are implemented in each class inheriting from RobotController. The code that is run inside these methods for each class can be different depending on the type of robot being controlled; however, the input and output parameters always follow the same pattern. Given this, if somebody creates a new controller script for a new robot model, it is going to work with this application – thus simplifying the development of new DT controllers.

*Virtual robot control*

VirtualRobotController is created to control a generic virtual industrial robot model inside Unity3D. It uses a supplementary VirtualRobotJoint script to manipulate its joints and can also connect to Inverse Kinematics solver to support MoveToEndpoint(...) command.

To set up a new virtual industrial robot for control in Unity using this script, a developer needs to execute several steps:
- Import a correctly rigged model of the desired robot to Unity.
- Add VirtualRobotJoint script to each joint of the robot model and define the joint’s rotation limits in this script’s Component interface.
- Add VirtualRobotController script to the root of the robot model hierarchy.
- (Optional) Add Inverse Kinematics solver script if MoveToEndpoint(...) command needs to be implemented.

The virtual robot has a default maximum speed, which can be set by the developer inside Unity; actual robot speed is set inside the application as a per cent of this maximum value. If at some point the speed of the VirtualRobotController is set to zero, it does not stop moving but instead moves immediately to the given joint angles or endpoint. This feature is implemented for the cases when a virtual robot needs to immediately synchronize its position according to some values (in case of DT application, sync with the real robot).

*Real robot control*

Regarding this work the control, Application Programming Interface (API) created by the industrial robot manufacturer was used to control the robot over the local network. Because the API itself is distributed under the Non-Disclosure Agreement (NDA), the code samples using it and the explanations of its inside functionality cannot be published in this work. However, it is sufficient to explain the underlying logic of how the real robot control is implemented in this DT project to understand the principle and apply it in other experiments.

The RealRobotController script is intended to be used for real industrial robot control over the network. It inherits directly from RobotController but is also declared as an abstract class, because it contains a couple of extra properties and methods which are dictated by the necessity to connect and continuously monitor the state of the linked real robot. These consist of the following:

- public bool isConnected – returns true if the connection with the robot is successfully established, returns false otherwise,
- Protected IEnumerator RobotStateMonitor_Coroutine() – an internal Coroutine which manages the link to the robot and periodically updates the status information about it (i.e., joint positions, system status, error messages).

These small additions create a base for writing scripts, which can be used to control and monitor real industrial robots from Unity3D. While the interface stays the same (RobotController), class methods can now utilise an internal protocol for communication with the real robot controller over the network, and the script can act following the state data received from the actual controller, creating a closed feedback loop with the robot. The protocol implementation depends solely on the company, which has produced the robot and can be integrated into the solution as a new script inheriting from RealRobotController. The control of researched robot was implemented in the form of MotomanGP8Controller class, which is inherited from RealRobotController and implements its methods.

It is also important to note that in the controller script it was managed to achieve the joint angle setting accuracy which is identical to the actual precision of the robot – which is as high as 0.001°. That means that any time the robot's joint angles are set from the simulation, the real robot moves to the target angles with the same precision.

*Digital Twin system*

With the programming basis of the project implemented inside RobotController family of scripts, namely VirtualRobotController and RealRobotController, creating a Digital TwinController script, which would enable Digital Twin functionality in the developed application, was near to a "plug-and-play" process. DT solution for industrial robots developed concerning this research work according to the following logic:

- DigitalTwinController script acts as a coordinator between two scripts in Unity: VirtualRobotController and MotomanGP8Controller.
- MotomanGP8Controller connects to Motoman GP8 robot over the local network and continuously monitors its state.
- DigitalTwinController reads position data received by MotomanGP8Controller and redirects it to VirtualRobotController script.
- VirtualRobotController synchronises the virtual robot's joints' positions according to received values, and, as a result, copies all movements of the real robot.

When a movement command is sent to DigitalTwinController (for example, MoveRobotToZero()), the script first sends this command to the real robot, and in the next update loop virtual robot gets synchronised with the real one again, creating a smooth DT experience. Moreover, if the teach pendant, which is the control panel attached to the real robot, overrides the real robot program control, the twin is still going to mimic the movements of its real counterpart. The solution also remains safe, because even when somebody is manipulating the robot from VR environment, the proximity sensor built into the experimental stand continues monitoring surroundings, and stops the robot if someone comes dangerously close in the real world. A schematic representation of the Digital Twin system can be seen in Fig. 3.

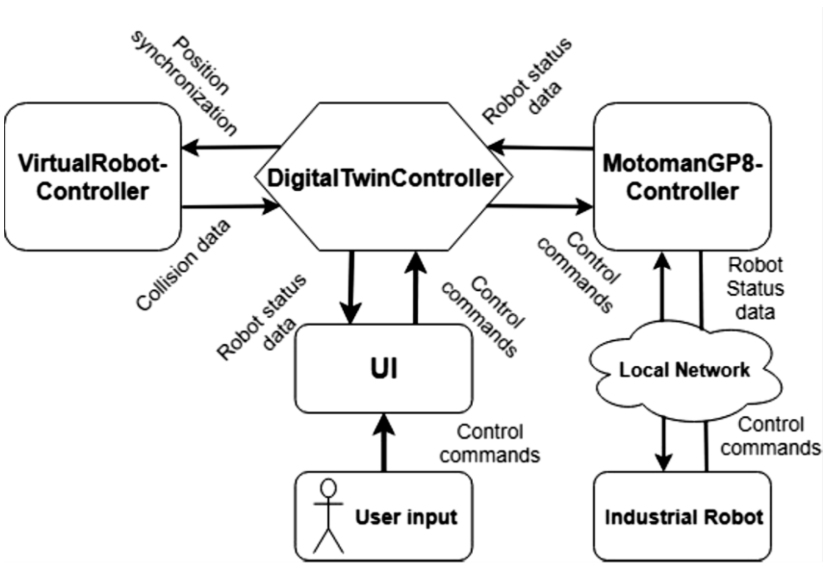


Fig. 3. Digital Twin system diagram

### 2.3. ROBOT CONTROL SCRIPTS

Robot programming scripts family contains three classes, which are responsible for providing robot programming and testing functionality as well as methods to simplify these processes. These scripts are:

- RobotProgrammer.
- RobotTester.
- VirtualRobotProgrammingClone.

*Main programming script*

The programming functionality for Control scripts is added using the Robot Programmer Component. It is a simple programming manager, which was created to demonstrate how it is possible to program robot directly from Unity without using the native commands, which can be different depending on the robot's model and implementation.

The script contains methods to create new programs for robot controllers and store them for later use. RobotProgrammer must be connected to a specific RobotController inside Unity to make it possible to run the created application. Currently, three types of commands can be added to the program using the following methods:

- public void AddPointToProgram(List<float> jointAngles) – adds a point the robot should move to; the point is given as the list of similar joints' angles,
- public void AddWaitToProgram(float waitTime) – adds pause which lasts for the number of seconds specified in the waitTime parameter,
- public void AddGripperAction (bool action) – adds gripper action to the program (if the given action parameter is true, the robot will close the gripper, if false open it).

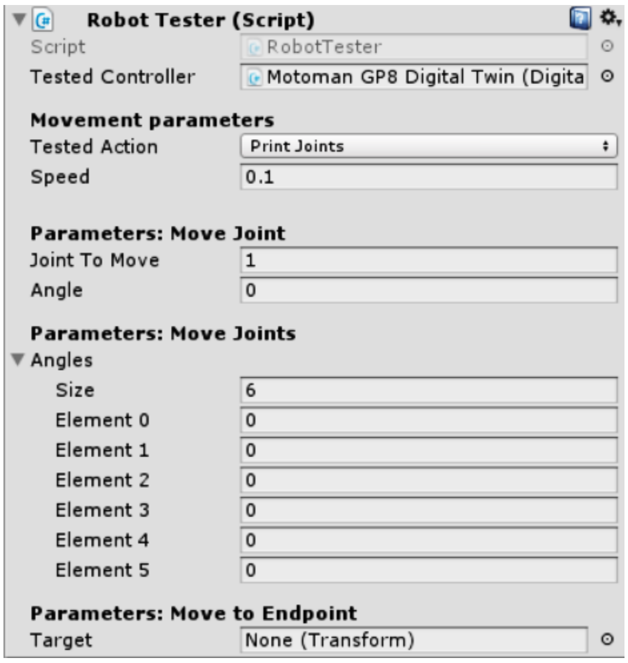
These three simple command types allow creating demonstrative which can be run on all robot controllers, from virtual to DT.

Internally, RobotProgrammer stores the programs as RobotProgram objects (a utility class declared inside RobotProgrammer script), which, in turn, contain the lists of ProgramAction objects (another utility class inside RobotProgrammer, which is responsible for storing and interpreting the steps added to the algorithm utilising methods described above.

To run the currently selected program in RobotProgrammer, one has to call a RunProgram(...) method from it. RunProgram(...) will automatically parse the currently selected program and send the corresponding commands to the connected robot using the same RobotController API, with a 32,875 ms of average delay between each command to ensure stability.

*Robot testing script*

A script called RobotTester was created to speed up the development cycle by providing a simple tool, which can be used to test the newly written RobotController scripts. It contains methods, which call the usual position and movement functions from RobotController, but provides a Unity3D interface, which helps to quickly switch between these methods, as, can be seen in Fig. 4.



**Robot Tester (Script)**

| | |
|---|---|
| Script | RobotTester |
| Tested Controller | Motoman GP8 Digital Twin (Digita |
| **Movement parameters** | |
| Tested Action | Print Joints |
| Speed | 0.1 |
| **Parameters: Move Joint** | |
| Joint To Move | 1 |
| Angle | 0 |
| **Parameters: Move Joints** | |
| Angles | |
| Size | 6 |
| Element 0 | 0 |
| Element 1 | 0 |
| Element 2 | 0 |
| Element 3 | 0 |
| Element 4 | 0 |
| Element 5 | 0 |
| **Parameters: Move to Endpoint** | |
| Target | None (Transform) |

Fig. 4. RobotTester script Component interface

As can be seen from Fig. 4, this component provides a simple interface where the developer can define the input parameters and select the tested method. Action to be tested is selected using a drop-down menu, which provides the next options: “Print Joints”, “Move Joint”, “Move Joints” and “Move To Endpoint”. Input parameters for each of these commands can be given further in the interface (joint number and angle for “Move Joint” command, joints' angles list for "Move Joints" and target Transform for "Move To Endpoint"). After the necessary action type and parameters are selected, the test can be performed by merely calling Test() method of this RobotTester script. This script has dramatically simplified the development and testing of new RobotController scripts during this research.

*Robot programming virtual representation (digital clone)*

The final goal in the development of robot programming system was to implement the way to make the programming process itself more understandable and straightforward for the user. It resulted in the creation of the VirtualRobotProgrammingClone script. This script is inherited from the VirtualRobotController, and its purpose is to create a copy of the programmed virtual robot with the same parameters placed in the same position as the original. This digital clone can be used to visually set the target points when programming the robot, with the need to move the original – this feature is especially useful when programming a real robot controller (see Fig. 5).

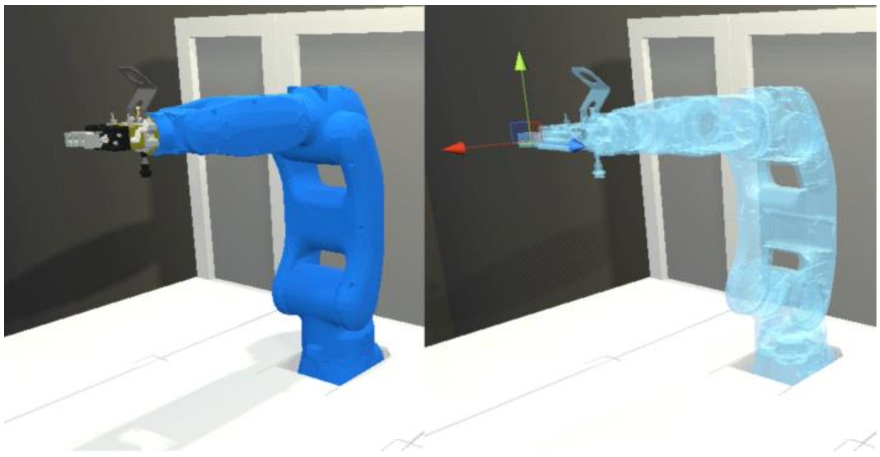


Fig. 5. Digital Twin robot (left) and its digital programming clone with IK endpoint target (right)

VitrualRobotProgrammingClone allows setting the positions of the joints as well as endpoints using steps, which makes it simpler to set the points from an interface. While joints are actuated the same way as it is done in other RobotController scripts, digital programming clone in this project also uses IK solver, which allows setting the robot position by moving its endpoint target. There was no need to develop the IK solver from scratch, as the Unity engine already provides different ready-to-go IK options.

The first available option is to use the built-in Unity IK system called Mecanim, but another tool was selected for this project – Final IK Unity plugin. Final IK is a full-packed IK system developed in Estonia specifically for Unity game engine. Though this plugin is not free, the license has to be purchased only one time, and it is more flexible and user-friendly than the Unity's built-in Mecanim system. Final IK also allows creating custom IK chains and can be used in the industrial robot model. So, Final IK solver was used to calculate the joints' positions of the digital programming clone when controlling it in IK mode. A schematic of the robot programming system can be seen in Fig. 6.

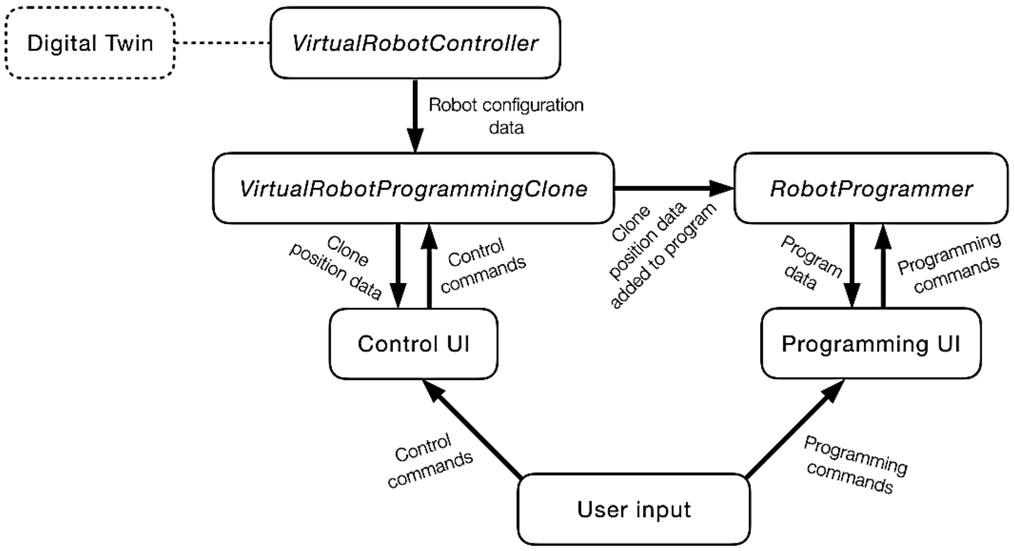
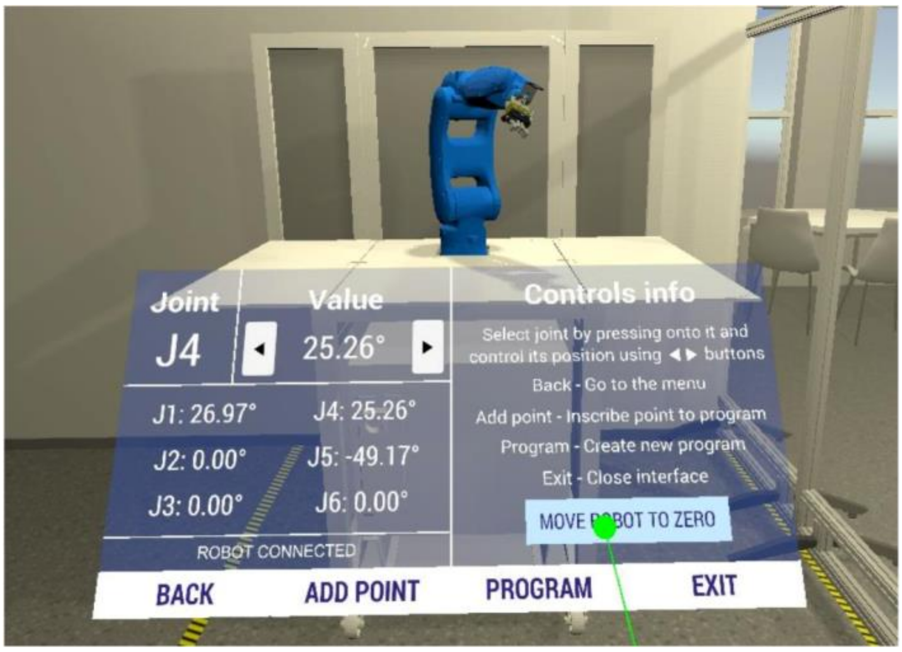


Fig. 6. Robot programming system diagram

## *Programming User Interface (UI)*

As a final goal, the DT system developed in this project was integrated into the full-scale simulation of the University's IVAR Laboratory. To the projected were implemented the User Interface (UI) for robot control and programming using APIs of the scripts created concerning this use-case development. It created a possibility to test the developed scripts right from VR, using HTC Vive headset, resulting in positive outcomes. The visual look of the UI is presented here for the reference of what can be done using the scripts developed concerning this project (see Figs 7 and 8).



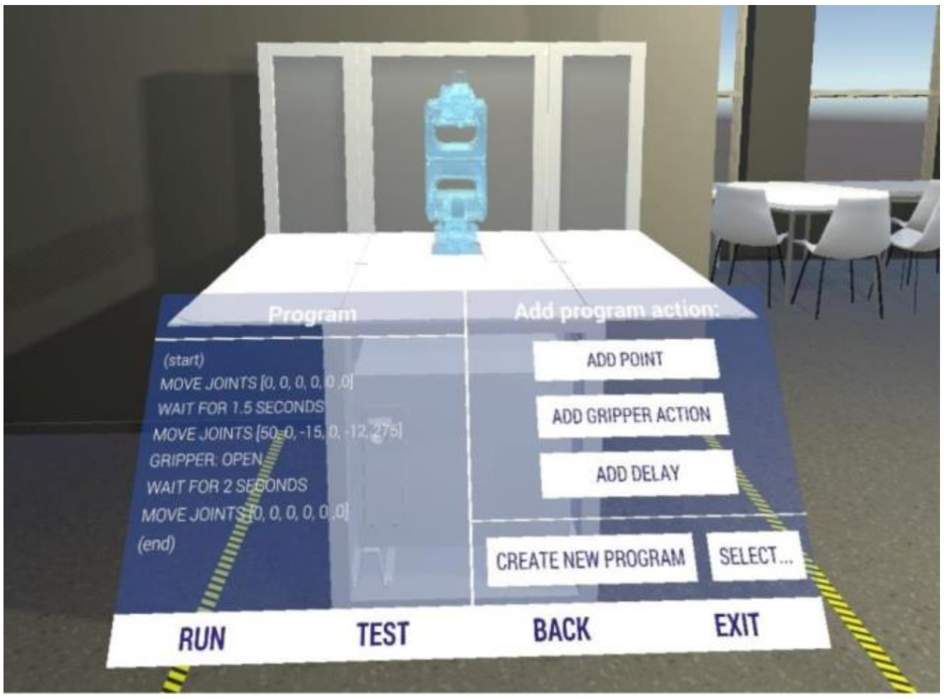
J4 ◀ 25.26° ▶

J1: 26.97° J4: 25.26°
J2: 0.00° J5: -49.17°
J3: 0.00° J6: 0.00°

ROBOT CONNECTED

Select joint by pressing onto it and control its position using ◀▶ buttons
Back - Go to the menu
Add point - Inscribe point to program
Program - Create new program
Exit - Close interface

Fig. 7. Digital Twin control menu



```
(start)
MOVE JOINTS [0, 0, 0, 0, 0 ,0]
WAIT FOR 1.5 SECONDS
MOVE JOINTS [50, 0, -15, 0, -12, 275]
GRIPPER: OPEN
WAIT FOR 2 SECONDS
MOVE JOINTS [0, 0, 0, 0, 0 ,0]
(end)
```

Fig. 8. Digital Twin programming menu (the virtual robot is replaced with its digital programming clone)

## 2.4. COLLISION DETECTION SCRIPTS

One additional goal of this research was to develop a method, which can improve the workspace awareness of the real robot by using its DT. As a result, two scripts were developed: RobotCollisionRollback and RobotCollisionAware. Both scripts have the same basic idea implemented in them: a DT is an exact digital copy of the real robot; given this, it should be possible to use the DT's geometry to monitor the position of the robot in its workspace and prevent it from accidental collisions with the environment using Unity physics system. If the models of both the robot and its working cell are made precisely (which is the case for this project), they can be used with a quite high certainty for collision prevention. Of course, the joint limits can always be set on the real-world robot itself, but this process takes time and requires thorough planning, and never guarantees that all possible collision scenarios are eliminated. Another option is to supplement the real-world robot with proximity sensors on each of its links and monitor collisions using these sensors – but this method is very costly and even more time-consuming. The DT collision detection approach, proposed in this work, provides almost the same level of reliability but does not require any extra physical equipment or significant setup time (given that the DT model is already made and has correct dimensions set). All that is needed for this approach to work is to generate colliders for the robot's model in Unity3D and attach one of the collision detection scripts described below.

The RobotCollisionAware script uses Unity physics system to detect collisions of the robot with the environment. Trigger colliders attached to the robot's model can detect collisions with other objects before the robot's geometry is going actually to encounter them. That can be used for example for prohibiting certain positions when using a digital programming clone and trying to add points, which can cause collisions to the program.

RobotCollisionRollback is used for the DT model itself. It is an enhanced version of RobotCollisionAware script, which is intended to prevent the real robot from colliding with its stand during direct control. In a case when the robot is about to bump into the stand, this script will call an emergency stop command, and then automatically move it back to the previous safe position. This method makes controlling the robot from VR safe for both the robot and its surroundings.

### 2.5. PROJECT OPTIMISATION FOR VIRTUAL REALITY

To enable the VR capabilities in the developed Unity project, Virtual Reality Toolkit (VRTK) Unity script library was used. VRTK is an exceptional example of open-source software. This library contains scripts, which simplify the process of building VR applications for Unity to a great extent, and furthermore, the applications developed using VRTK are elementary to modify and add support of different platforms. The project of this research was initially built to be used with the HTC Vive headset, but thanks to its VRTK basis it can be ported to all common VR platforms, including Oculus Rift and Microsoft Mixed Reality (MR).

During the DT control scripts tests in VR, it appeared that the code was causing a significant framerate drops when sending the commands to the connected robot. Framerate is the speed at which the application is rendered, measured in frames per second (FPS). The stable framerate of 90 FPS is crucial for good VR experience because framerate drops to the numbers lower than this are immediately causing discomfort or even nausea to the user of the application.

After the problem investigation, it was revealed that the problem was caused by industrial robot API used in MotomanGP8Controller to send commands to the robot – while the commands had to be sent asynchronously to avoid affecting framerate; API sent synchronous calls to the robot. Because of this, an application had to wait for the command to be successfully transmitted over the network and confirmed, causing it to freeze in the same frame until the command is executed. This effect caused the framerate to drop the framerate below 20 FPS – making the application practically unusable for VR. This problem had to be overcome to finish the experiment successfully.

The solution was to add multithreading functionality to the application. Unity API itself is single-threaded, so the multithreading solution had to be implemented independently of Unity scripts. After some experimental research, the solution was found – to create a ThreadedJob class, which can be extended into separate job classes, which run their code in parallel threads. With the usage of these classes as a base, the calls to the robot were moved into separate job classes called MotomanMonitorJob, MotomanMoveJointsJob, and MotomanStopJob. Because of such architecture change, all framerate problems cause by the robot API were eliminated – what made application to be able to run smoothly, and its performance does not depend on the commands sent to the robot.

## 3. EXPERIMENT/USE-CASE

As it was mentioned earlier in this paper, the VR environment for the robot's Digital Twin was created concerning the project of digitalisation of TalTech IVAR Laboratory. Thus said, before the system could be used for the demonstrations and educational purposes in the Laboratory, it had to undergo testing for safety and working stability.

The most critical part of the DT simulation was collision detection, as the safety of the real robot and its surroundings directly depended on this feature. After subsequent control tests, it was determined that the DT collision prevention system, described in section 2.4 of this article, was capable of timely stopping both virtual and real robots if the upcoming collision with the surrounding geometry was detected. It is important to note here, that the effectiveness of this system depends on the precision of the robot work cell/environment reproduced in VR simulation. If it is represented incorrectly, there is a chance of robot hitting objects in the real world. On the contrary, this hazard can be quickly eliminated by careful planning and dimensioning when reproducing the robot system in 3D.

Another essential feature of the system confirmed during testing was the possibility to override control over the system from inside the real world – whether by the teach pendant or by the proximity sensors installed in the robot's stand. In the first case, the robot could be set to manual control mode and manipulated directly from it's taught pendant, while the DT inside the running VR simulation would continue reproducing all movements of its real counterpart. In the second case, if some person approached too close, it would trigger the proximity sensor, automatically stopping the real robot from moving and thus guaranteeing safety to the people around the machine; again, the DT would stop as well, staying synchronised with the real robot. From the point of practical usability, the system has proven itself an excellent tool for DT technology demonstration and education, confirmed by the university's robotics specialist. DT makes interaction with the robot much safer, simpler and more engaging by utilising all the unique features the VR has to offer, i.e., real-size stereo picture, highly flexible 3D interfaces, etc. The system was also operated by numerous students and was said to be easier to understand and control for people unacquainted with robotics. Currently, it is being used in the TalTech IVAR Laboratory demo centre for research projects and demonstration purposes daily.

With certain improvements in stability and added features, such as support for custom industrial robot tools control, the developed concept can with the most certainty be used for industrial control and production purposes.

## 4. RESULTS AND FUTURE WORK

The simulation system developed in this work, which consists from the full synchronisation between real and virtual industrial robot, implements a universal software base, which can be extended to control and program DT of different industrial robots by the usage of the Unity3D game engine. It also presents an idea of how DT can improve the workspace awareness of its real-world counterpart using the collision prediction system inside the simulation – a viable alternative to cost- and time-consuming physical sensor solutions.

Overall developed DT system provides the following advantages for the manufacturing lines and robotic cells:

- Enlargement of equipment functionality with smaller cost – virtual sensors can control real machines, as with collision detection case, described in above sections.
- Flexible – modularity and universal and user-friendly UI adjustable for various type of equipment.
- Production monitoring – data visualisation from sensors, fast optimisation of production, preliminary maintenance – based on data received from DT.
- Historical data in real time – saved into log kinematics data allow to "travel back" in VR headset and check what went wrong or how processes could be optimised better for a previously produced product.
- Safety systems tests – human-machine interaction scenarios tested in the digital environment.
- Personnel training – training on the machine without interfering with real one, not causing loose of production time of it.
- Machines are not stopped while re-programming, which lead to a decrease of downtime, which reduces cost on re-programming.

This project, however, can be extended and improved to fit a much higher usability scale. The Unity3D script base of the project is built in a modular manner, which can allow industrial robot manufacturers and researchers to extend the current program and add support of their machines without exposing the internal control protocols, and still leaving the solution compatible with the interfaces used in this application. The project code development is being continued to improve stability and add new useful features.

Next step in the development is also the analysis of the connection and exploiting manufacturing equipment and virtual model telemetry [16, 17] data to find a most effective and precise way of the dual-way communication and create the framework for most effective connection between real and virtual environments.

## 5. CONCLUSION

Virtual and augmented reality technologies are indeed becoming the practical tools of Industry 4.0 and rapidly expanding their markets. While AR enhances the way industry workers can interact with the real-world equipment, VR presents another breakthrough concept – DT, a technology that allows overseeing and controlling existing manufacturing systems from a safe yet highly intuitive and interactive real-size stereo simulation. The goal of this research was to implement such a system on the example of a real industrial robot, and it was fulfilled successfully.

The DT robot created is the central framework part of the TalTech IVAR laboratory digitalisation project, where it works in tandem with the real industrial robot located in the actual laboratory. The solution was already assessed and tested by several students and robotics specialists, who confirmed its applicability for industrial demo and educational purposes.

As a conclusion, it can be stated that Digital Twin concept is a practically viable industrial solution, which can start driving control and management systems of enterprises in the nearest future. As for future development, the done environment will be redefined for the modular approach and will be continued work towards optimisation of synchronisation framework between two worlds, what is a part of ongoing research on existing DT model optimisation.

## ACKNOWLEDGEMENTS

*This research was supported by project AR16077 Smart Industry Centre (SmartIC) No. 2014-2020.4.01.16-0183, supported by the EU Regional Development Fund. The authors are grateful to the Integrated Engineering and System Control students of the Tallinn University of Technology – Ninel Dudnyk and Tengiz Pataraia, for their help in the experiments with Virtual Reality implementations and laboratory digitalisation.*

## REFERENCES

[1] OYEKAN J.O, HUTABARAT W., TIWARI A., et al., 2019, *The effectiveness of virtual environments in developing collaborative strategies between industrial robots and humans*, Robotics and Computer-Integrated Manufacturing, 55, 41–54.

[2] KOT T., NOVÁK P., BAJAK J., 2018, *Using HoloLens to Create a Virtual Operator Station for Mobile Robots*, 19th International Carpathian Control Conference (ICCC), Szilvasvarad, 422–427.

[3] CHEN C., SU B., GUO M., ZHONG Y., YANG Y., KUO H.L, 2018, *Applying virtual reality to control of logical control mechanism system*, IEEE International Conference on Applied System Invention (ICASI), Chiba, 520–523.

[4] COVACIU F., PISLA.A., CARBONE G., PUSKAS F., VAIDA. C., PISLA D., 2018, *VR interface for cooperative robots applied in dynamic environments*, IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, 1–6.

[5] KOSE A., PETLENKOV E., TEPLJAKOV A., VASSILJEVA K., 2017, *Virtual Reality Meets Intelligence in Large Scale Architecture*, De Paolis L., Bourdot P., Mongelli A. (eds) Augmented Reality, Virtual Reality, and Computer Graphics, AVR 2017. Lecture Notes in Computer Science, 10325. Springer, Cham, Ugento (Lecce), Italy.

[6] KOSE A., TEPLJAKOV A., ASTAPOV S., DRAHEIM D., PETLENKOV E.K., VASSILJEVA K., 2018, *Towards a Synesthesia Laboratory: Real-time Localization and Visualization of a Sound Source for Virtual Reality Applications*, Journal of Communications Software and Systems, 14/1, 112–120.

[7] BURDEA G.C., 1999, *Invited review: the synergy between virtual reality and robotics*, IEEE Transactions on Robotics and Automation, 15/3, 400–410.

[8] MCNEELY W.A., 1993, *Robotic graphics: a new approach to force feedback for virtual reality*, Proceedings of IEEE Virtual Reality Annual International Symposium, Seattle, WA, USA, 336–341.

[9] JI W., YIN S., WANG L., 2018, *A Virtual Training Based Programming-Free Automatic Assembly Approach for Future Industry*, IEEE Access, 6, 43865–43873.

[10] MINER N.E., STANSFIELD S.A., 1994, *An interactive virtual reality simulation system for robot control and operator training*, Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, USA, 2, 1428–1435.

[11] GUCWA K.J., CHENG H.H, 2018, *RoboSim: a simulation environment for programming virtual robots*, Engineering with Computers, 34/3, 475–485.

[12] KUTS V., SARKANS M., OTTO T., TÄHEMAA T., 2017, *Collaborative Work Between Human And Industrial Robot in Manufacturing by Advanced Safety Monitoring System*, Proceedings of the 28th DAAAM International Symposium, Vienna.

[13] VUNDER V., VALNER R., MCMAHON C., KRUUSAMÄE K., PRYOR M., 2018, *Improved Situational Awareness in ROS Using Panospheric Vision and Virtual Reality*, 1th International Conference on Human System Interaction (HSI), Gdansk, 471–477.

[14] SURESH A., GABA D., BHAMBRI S., LAHA D., 2019, *Intelligent Multi-fingered Dexterous Hand Using Virtual Reality (VR) and Robot Operating System (ROS)*, Kim JH. et al. (eds), Robot Intelligence Technology and Applications 5, RiTA 2017, Advances in Intelligent Systems and Computing, 751, 459-474.

[15] ROLDÁN J.J., PEÑA-TAPIA E., GARZÓN-RAMOS D., JORGE DE LEÓN, GARZÓN M., JAIME DEL CERRO, 2019, *Multi-robot Systems, Virtual Reality and ROS: Developing a New Generation of Operator Interfaces, Koubaa A. (eds)*, Robot Operating System (ROS), Studies in Computational Intelligence, 778, 29–64.

[16] MODONI G.E., SACCO M., TERKAJ W., 2016, *A telemetry-driven approach to simulate data-intensive manufacturing processes*, 49th CIRP Conference on Manufacturing Systems, Stuttgart.

[17] KUTS V., MODONI G.E., TERKAJ W., TÄHEMAA T., SACCO M., OTTO T., 2017, *Exploiting Factory Telemetry to Support Virtual Reality Simulation in Robotics Cell*, Augmented Reality, Virtual Reality, and Computer Graphics, 1: 4th International Conference, *AVR 2017*, Ugento, Italy.