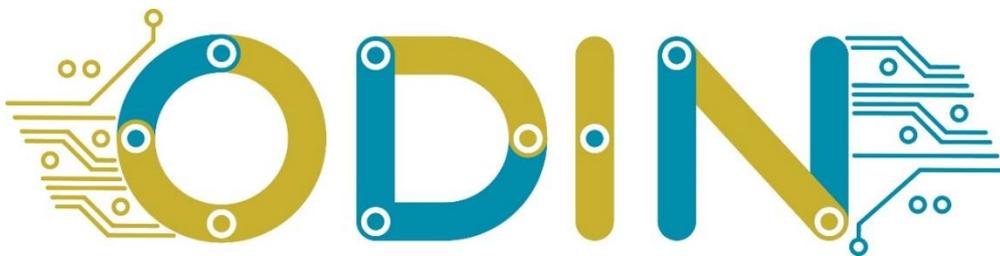


Open-Digital-Industrial and Networking pilot lines using modular components for scalable production

Grant Agreement No : 101017141
Project Acronym : ODIN
Project Start Date : 1st January, 2021
Consortium : UNIVERSITY OF PATRAS – LABORATORY FOR MANUFACTURING SYSTEMS AND AUTOMATION
FUNDACION TECNALIA RESEARCH & INNOVATION
KUNGLIGA TEKNISKA HOEGSKOLAN
TAMPEREEN KORKEAKOULUSAATIO SR
COMAU SPA
PILZ INDUSTRIELEKTRONIK S. L.
ROBOCEPTION GMBH
VISUAL COMPONENTS OY
INTRASOFT INTERNATIONAL SA
GRUPO S21SEC GESTIÓN, S.A.
FUNDACION AIC AUTOMOTIVE INTELLIGENCE CENTER FUNDAZIOA
DGH ROBOTICA, AUTOMATIZACION Y MANTENIMIENTO INDUSTRIAL SA
PSA AUTOMOBILES S.A.
AEROTECNIC COMPOSITES SL. U.
WHIRLPOOL EMEA SPA
WHIRLPOOL MANAGEMENT EMEA SRL



Title : ODIN Digital Component virtual commissioning framework – Final Version
Reference : D3.4
Availability : Public
Date : 22/01/2024
Author/s : VIS, TECNALIA, LMS

Summary:

The purpose of this document is to present the final version of the integrated digital tools for virtual commissioning and control.

Table of Contents

LIST OF FIGURES	3
LIST OF TABLES	4
1. EXECUTIVE SUMMARY	5
2. INTRODUCTION.....	6
3. VIRTUAL COMMISSIONING.....	7
3.1. Virtual Commissioning in ODIN	8
3.2. Open interface	9
3.2.1. ROS 2 Connectivity.....	9
3.3. Communication interface	10
3.3.1. UR plugin	11
3.3.2. KUKA plugin	12
4. ROBOT VIRTUAL COMMISSIONING	14
4.1. Workflow for virtual commissioning.....	15
4.1.1. Robot programing and simulation.....	15
4.1.2. Post-process to the robot language	16
4.1.3. Upload the robot program to the VRC	19
4.1.4. Configuration of the robot in the VRC.....	19
4.1.5. Position set up	20
4.1.6. Program validation in the VRC	20
4.1.7. Signals mapping	21
4.1.8. Virtual validation.....	21
5. CONTROL SYSTEMS COMMISSIONING	23
6. CONCLUSIONS.....	24
7. GLOSSARY.....	25
8. REFERENCES.....	26

LIST OF FIGURES

Figure 1: Engineering project with and without Virtual Commissioning [3]	7
Figure 2: System lifecycle used in ODIN with and without virtual commissioning	7
Figure 3: Overview of the communication interfaces used in ODIN for Virtual Commissioning	8
Figure 4: Overview of the open interfaces and current development within ODIN	9
Figure 5: Connectivity schema between Visual Components and ROS2	9
Figure 6: Workflow of ROS2 message to VIS sending joint goal to obtain interpolation time	10
Figure 7: Screenshots of the configuration and programming of a KUKA robot within VC 4.0	12
Figure 8: Screenshot of the configuration window within the simulation for RCS and KRL	13
Figure 9: Robot program editor available in VC 4.0 (r4.8)	14
Figure 10: Screenshot of the robot executor properties UI	15
Figure 11: Robot programming within VC 4.0	16
Figure 12: Screenshot of VC 4.0 UI showing access to Post Process	16
Figure 13: Screenshot of the KUKA SUNRISE - Java post-process	17
Figure 14: Screenshot of the post-process tab available at VC 4.0 for UR, PP Type URP (left) and PP Type Script (right)	17
Figure 15: Screenshot of the post-processing of robot program of the COMAU Aura robot	19
Figure 16: Upload process of the robot program into the virtual robot controller	19
Figure 17: Configuration of the virtual controller with the same operation parameter than the simulation	20
Figure 18: Position synchronization of the robots in VC 4.0	20
Figure 19: Validation of the robot controller	21
Figure 20: Process of connecting the UR VRC and the robot in the simulation through the communication interface	21
Figure 21: UR robot in VC 4.0 connected to the virtual controller through the communication interface displaying the program uploaded in the VRC	22
Figure 22: KUKA Robot in VC 4.0 connected to the virtual controller through the communication interface displaying the program uploaded in the VRC	22
Figure 23: Screenshot of the virtual environment using the OPC communication with control (PLC) validation vs simulation	23

LIST OF TABLES

Table 1: Commands provided by the communication feature of Visual Components 4.0	10
Table 2: Statements available to program a robot in the VC 4.0	14

1. EXECUTIVE SUMMARY

The target of this document is to present the work being developed in Task 3.5 of the ODIN project. This task targets the virtual control and commissioning of the pilots. The task extends the work developed within WP3 to the development of interfaces to validate pilots' requirements using virtual control and commissioning.

The document contains the work towards the development of communication interfaces for the OpenFlow and Robot, for virtual control and commissioning. Virtual commission is developed in one line to address the validation of the automation systems supported by the virtual control with the development of the interfaces and connectivity.

The deliverable provides details about the development of the communication interfaces implemented on top of the simulation platform used in ODIN, Visual Components 4.0 (VC 4.0) release 4.8, and the development of communication interfaces for robots used in the project pilots, COMAU, UR and KUKA.

The deliverable ends with the workflow introduced for virtual commissioning and the requirements regarding access to virtual robot controllers (VRC).

2. INTRODUCTION

This deliverable presents the work developed in task 3.5 towards the virtual control and commissioning of the pilots. In this document the work completed is presented by starting with a brief overview of virtual commissioning and the relevance under the ODIN project.

Aligned with the objectives of ODIN project, the further development of technologies and methodologies that support virtual commissioning and control and the introduction of them in the pilots will enhance productivity, accelerating the deployment of automation and robotics technologies.

Section 3 starts with virtual commissioning, which is the phase before the commissioning of a manufacturing system. As presented in section 3, it is an important phase of the automation system lifecycle, and the availability of technologies that enables will enhance productivity, avoiding costly mistakes, accelerating production ramp-up and achieving quality. The section continues with the work developed towards the integration with OpenFlow developing ROS 2 Connectivity (section 3.2).

The deliverable continues with section 3.3 and the work completed towards the development of interfaces for communication, for enabling the virtual control and commissioning. The development of the control has been done using the open interfaces provided by VC 4.0, and the communication interface for the UR and KUKA plugins for virtual commissioning.

The deliverable ends with the robot virtual commissioning in section 4, which includes the workflow for robot virtual commissioning.

3. VIRTUAL COMMISSIONING

To have a common understanding about virtual commissioning, it is required to define what commissioning means in discrete manufacturing automation. According to [1], commissioning is defined as “the task to put the mounted products on time in readiness for operation, to verify their readiness for operation and, if readiness for operation is not given, to establish it”. Commissioning is the final part of the system development and delivery process that results in a fully operational and tested system ready to use, which can be delivered to the customer [2].

In practice, commissioning of automation systems includes various procedures to check, inspect and test every operational component of the system, from physical fit of components and connections of electrical wiring to correct operation of work cells and the system as a whole. For controls’ commissioning the activities include correction of software errors, correction of addressing failures, teaching of sensor positions and adjustment of parameters such as speeds [2]. Out of the total commissioning phase, time spent on control software and electrics is by far the most time-consuming part with up to 90% share. As Figure 1 shows, the utilization of virtual commissioning reduces considerably the time improving production ramp-up, integration and avoidance of errors. [2], [3].

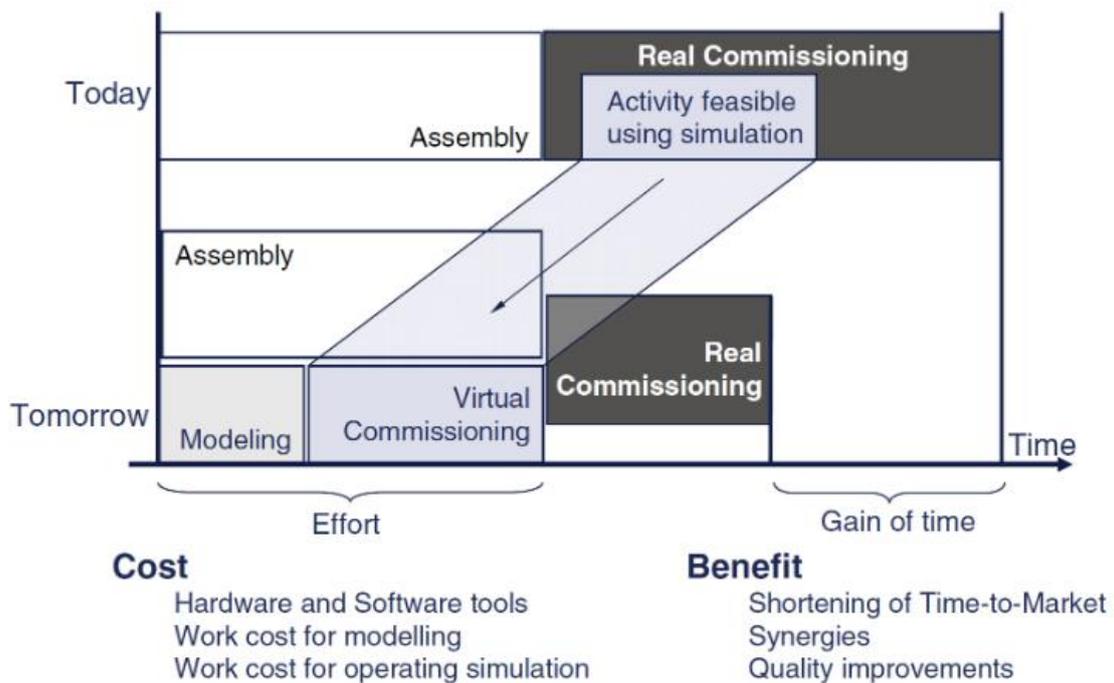


Figure 1: Engineering project with and without Virtual Commissioning [3]

The system lifecycle is presented in deliverable 3.2 and used along the project. Figure 2 introduces the virtual commissioning phase in the overall system lifecycle, and its integration within the simulation environment through connectivity technologies and its application in the digital twin.

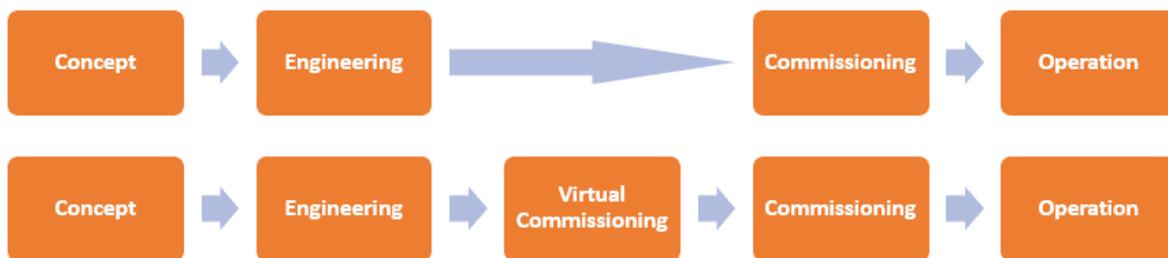


Figure 2: System lifecycle used in ODIN with and without virtual commissioning

3.1. Virtual Commissioning in ODIN

Task 3.5 in ODIN has extended the work developed in previous WP3 tasks to virtual control and commissioning of the pilots. To achieve this objective, new communication interfaces have been developed and integrated inside the digital simulation platform, provided by Visual Components 4.0 (VC 4.0), which allows communicating the DC with the OC to enable virtual control of the pilot in the simulation environment. The developments within the project have been integrated in the different release of the platform, being at the moment of completing this deliverable the release 4.8 (VC 4.0 (r.4.8) the one in use.

During the tasks development, the work has been focused on the identification of the interfaces, and data models used in other tasks, and extend it with the pilot requirements of work package 5 (WP5). As mentioned in the description of work, the interfaces developed will be validated in the three pilots to ensure their validity during the project. The development of the virtual commissioning in ODIN is considering the pilot requirements, in addition to considering the interoperability requirements with the rest of the modules developed in the project. The three pilots in ODIN contain different automation equipment which can be validated during the virtual commissioning phase:

- Robot control
- PLC
- Sensors
- Cameras
- etc.

Visual Components 4.0 Premium, particularly the release 4.8, used in the final stage of task 3.5, provide the communication feature, described in more detail in section 3.3, which allows the development of communication plugins (Figure 3). Currently OPC UA is available and allows the control, validation and virtual commissioning of systems that supports that communication protocol.

Two plugins over the communication interface have been developed and deployed for supporting the task in the ODIN project in the virtual commissioning of the robots, the UR plugin (3.3.1) and the KUKA plugin (3.3.2). In addition to the communication interface, Visual Components 4.0 provides two open interfaces (3.2), .Net and Python. The open interfaces are the based to develop the plugin for ROS2 connectivity (3.2.1) that in addition to connect with ROS 2 is the base for the communication through OpenFlow.

Figure 3 summarizes the interfaces developed and used in ODIN under the scope of task 3.5. Despite the task being completed, its utilization continues and the final integration within the pilots will be reported in D5.5.

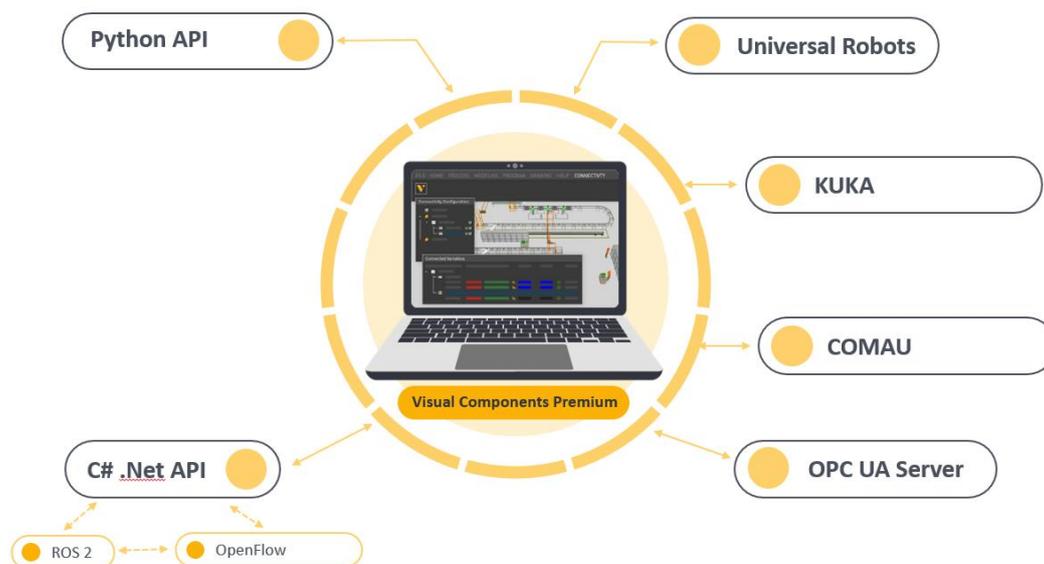


Figure 3: Overview of the communication interfaces used in ODIN for Virtual Commissioning

While the virtual control and validation has been focused on the development of ROS2 through the .Net interface, the virtual commissioning of the robots (UR and KUKA) have been achieved using dedicated plugins to support the full virtual commissioning in the white goods and aeronautics pilots and partially in the case of the aerospace pilot as it is explained in section 4.1.2.3.

3.2. Open interface

VC 4.0 provides two open interfaces, .Net and python (Figure 4), which allow the development of the required interfaces. Both interfaces provide an API for developers available through VC 4.0 user interface, which is extensively documented in the help menu of VC 4.0.

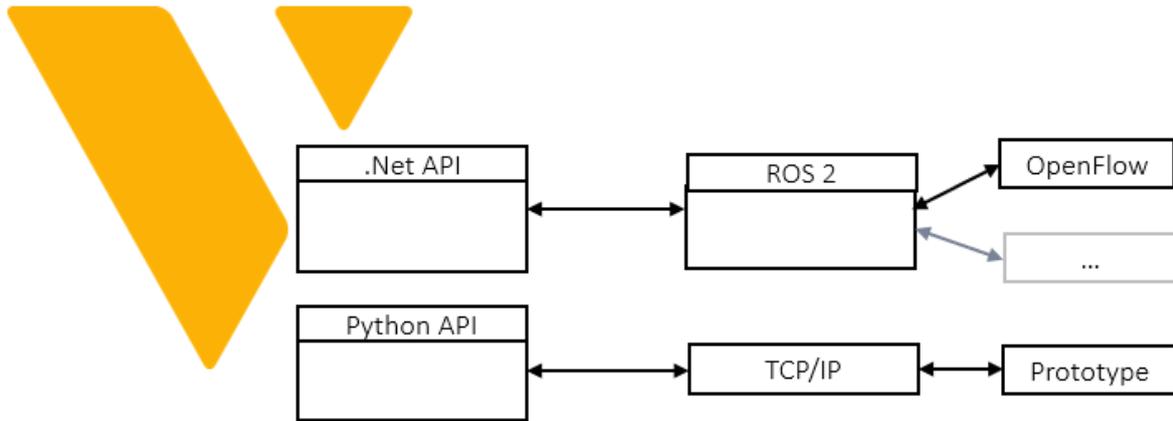


Figure 4: Overview of the open interfaces and current development within ODIN

While the python interface allows to create situation components and the development of add-ons, like the post-processors developed and integrated in the Programming tab, the .Net interface is targeting the development of plugins and extensions.

Within the ODIN project, the python interface is mainly used for the development of simulation models, but also for prototyping initial communication interfaces through TCP/IP or dedicated interfaces. The .Net interface has been used, mainly for the development of ROS 2 connectivity and the robotics communication interface.

3.2.1. ROS 2 Connectivity

Using the .Net interface, a plugin for ROS 2 connectivity has been developed. This plug-in has been under active development until the end of the task and has integrated the ODIN communication requirements through OpenFlow. It supports, publishes, and subscribes to ROS2 topics as showed in Figure 5.

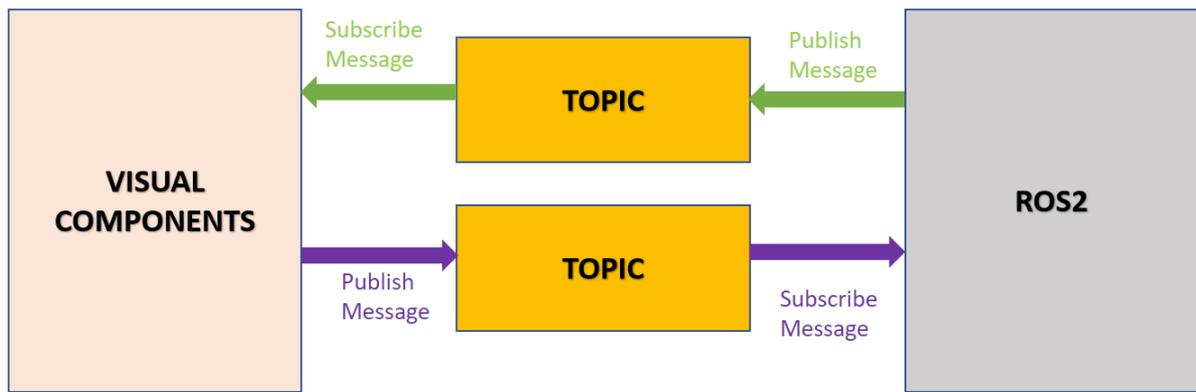


Figure 5: Connectivity schema between Visual Components and ROS2

The communication through ROS2 also allows the connectivity between VC 4.0 and OpenFlow, which also supports ROS2 interface. In that way it is integrated through the ODIN network component provided by OpenFlow with the rest of the ODIN connected components.

An example of the completed implementation is the use case showed in Figure 6. In this use case, joint goals are received from ROS2 to VC 4.0. The message is received through the topic and VC 4.0 creates trajectory points and statements. Once these are created, within VC 4.0 is checked reachability, joint configuration and singularity. After this, VC 4.0 sends the interpolation time through the topic, which is received by ROS2, request id and success message (Figure 6).

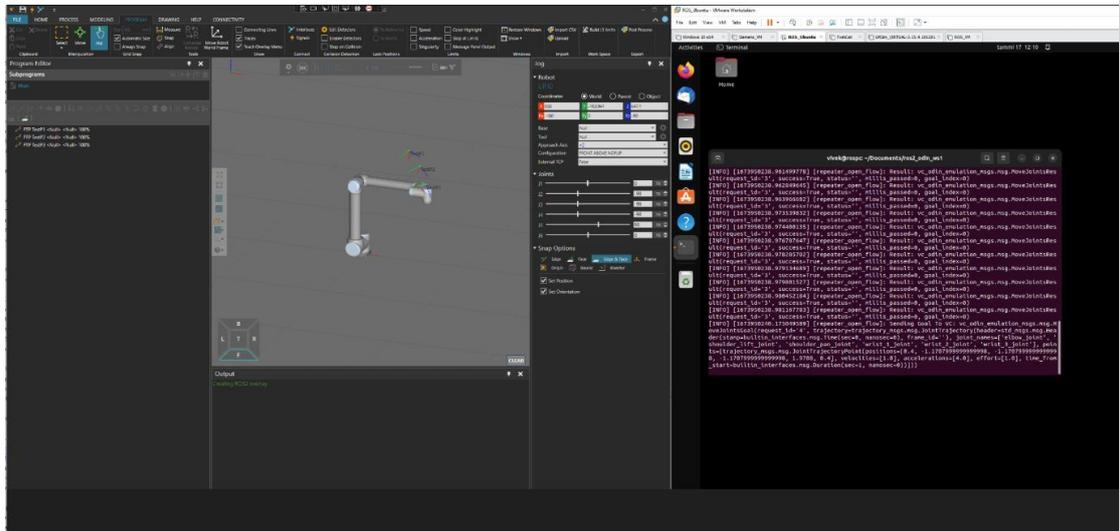


Figure 6: Workflow of ROS2 message to VIS sending joint goal to obtain interpolation time

Additionally, development and integration efforts have been devoted specifically for the functionalities of connecting the digital simulation in VC 4.0 with the Digital Twin module of KTH as well as the AI task planner of LMS reported in D3.3.

3.3. Communication interface

The communication interface is a feature provided by VC 4.0 that allows the development and deployment of connectivity plug-ins. This feature and its commands can be accessed from the Connectivity tab accessible through the user interface of VC 4.0.

The feature provides the commands described in Table 1, and allows communication between the virtual systems (sensors, actuators, conveyors, robots, machines, etc.) in the virtual environment and the controllers which can be real or virtual.

Table 1: Commands provided by the communication feature of Visual Components 4.0

Commands	Description
Add Group	Adds and lists a new variable group with a selected connection.
Add Server	Adds a new connection for a selected plugin.
Add Variables	Opens an editor for connecting simulation variables to server variables.
Clear	Removes all connections for each plugin.
Disconnect	Disconnects Visual Components Premium 4.8 from a selected connection.
Edit Connection	Displays options in a task pane for editing or troubleshooting a selected connection.
Export	Exports the configuration of all connections in an XML format.
Import	Imports an XML or CFG file that defines the configuration of one or more connections.

Commands	Description
Reconnect	Attempts to reconnect Visual Components Premium 4.8 to a selected connection.
(Server) Remove	Removes a selected connection.
(Variable) Remove	Removes a selected variable group.
Restore Windows	Restores the workspace of the current view to its default setting.
Show	Displays a list of panels that can be shown/hidden from the current view of the workspace.
Show Variables	Shows a panel for managing the connection between simulation and server variables.

The work developed towards the development of communication interfaces for ODIN are targeting the connectivity with UR (3.3.1) and KUKA (3.3.2) to match the robot requirements towards virtual commissioning in the white goods pilot and in the aeronautics pilot.

The base for the development has been the RRS (Realistic Robot Simulation) robot controller provided within VC 4.0 API, which visualizes within the simulation environment the realistic robot motions by using the native controller of a robot, particularly in ODIN the VRC controllers as presented in section 4.1.

3.3.1. UR plugin

Support for UR connectivity has been developed within ODIN, including a plugin which allows connecting the Universal Robot controller through RTDE interface ¹(Real Time Data Exchange). The plugin for the communication interface enables seamless communication between the virtual robot and the virtual robot controller (VRC).

The Real-time data exchange (RTDE) interface provides a cyclic stream of value updates from the controller and listens for inputs. The interface updates (sends and handles data packages) at a fixed frequency and is based on a binary application-level protocol transmitted over (insecure) TCP/IP socket communication. The robot controller uses TCP port 30004 for the interface. The connection plugin's RTDE client implementation developed uses an automatically assigned port (either by .NET or Windows) for the socket.

The basic operation principle involves two modes, configuration and run. First, the client configures with the server the data it wants to receive and data it wants to send. This is done in configuration mode. After configuration has been set, the client can request the controller to enter run mode where the controller sends the requested data at the fixed 125 Hz frequency and the client can send its data at a preferred rate. Run mode can also be paused by request of the client to return to configuration mode.

The data packages the client and controller send to one another are defined with input and output recipes in configuration mode:

- Input is data flow from client to controller.
- Output is data flow from controller to client.

The recipes contain one or more variables from a known fixed set, and the associated data packages contain values for all variables in the recipe.

The current version of the RTDE protocol supports only a single output recipe per client, but up to 255 input recipes can be defined per client. Furthermore, it is not possible to remove an input recipe without disconnecting and creating an entirely new one. This means that adding/removing a variable pair or

¹ More information about the RTDE interface can be found at <https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/>

activating/deactivating a variable group always causes a recipe update. Since the recipe update can only be done in configuration mode, the RTDE client implementation automatically requests pausing from the controller, and then either a) redefines the output recipe shared between all variable groups or b) registers a new input recipe for the activated variable group.

The RTDE protocol does not provide any way to poll the controller for value updates. This causes some limitations that differentiate the RTDE connection plugin from others. The RTDE connection plugin manages a local cache of the variable values for all configured recipes (active variable groups). This allows using cyclic update mode to read output recipe values at any desired frequency and sending whole input recipe data to the controller in event-based update mode. However, since the output recipe updates are received and input recipe data is sent asynchronously, the update delay timing functionality of Connectivity core does not really work with the RTDE plugin. The times measured are only processing times to get data in or out of the cache. That is, they do not include the network delay or even how old the received output recipe data is when the cache is read using cyclic update mode.

The plugin has been extended with the development of the post-processor which converts the robot program statements from VC 4.0 to the UR language (.urp). To validate the program Universal Robots provides, free of charge, the URSim robot controller simulator² which allows to load the robot program and validate to later visualize for virtual commissioning purposes.

3.3.2. KUKA plugin

The KUKA communication plugin has been developed targeting connectivity to KUKA RCS (KRCS-KUKA Robot Control System). This plugin can be activated when necessary and requires a license for the KUKA VRC.

The seamless integration within the VC 4.0 UI facilitates its use, as the user only needs to concentrate in the creation of the simulation, choosing the robot, adding the tools, configuring robot, tool and signals and creating the program within VC 4.0 (Figure 7).

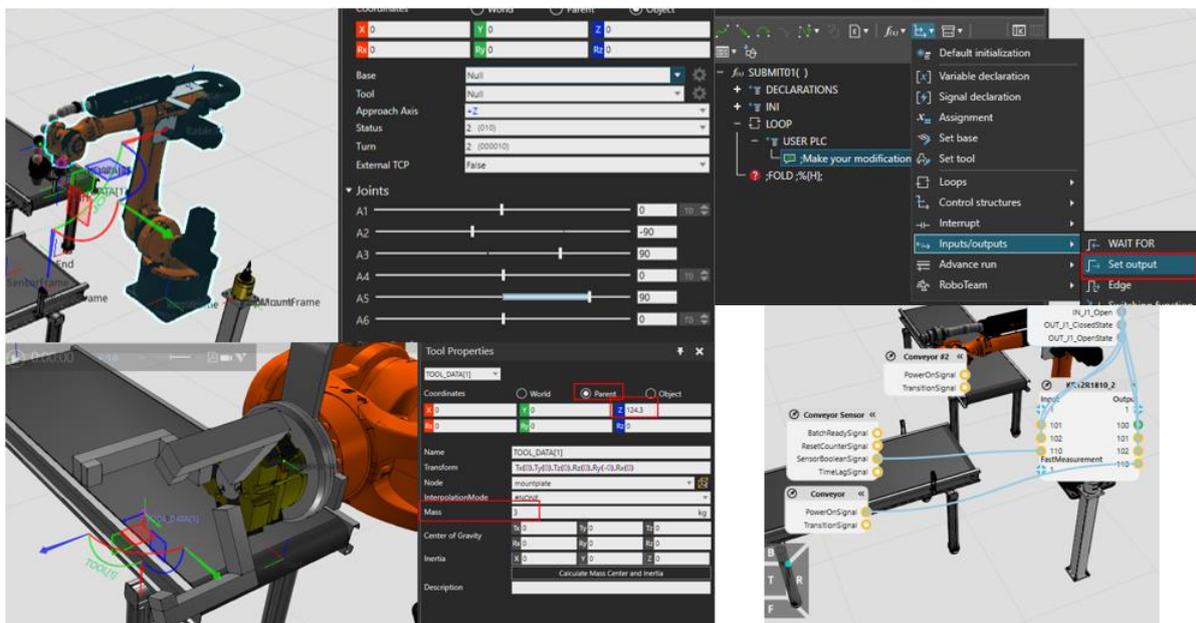


Figure 7: Screenshots of the configuration and programming of a KUKA robot within VC 4.0

² Available for downloading from <https://www.universal-robots.com/download/software-e-series/simulator-non-linux/offline-simulator-e-series-ur-sim-for-non-linux-594/>

The post-processing is done according to the configuration defined in the simulation components, that includes the KRL and the RCS (Figure 8).

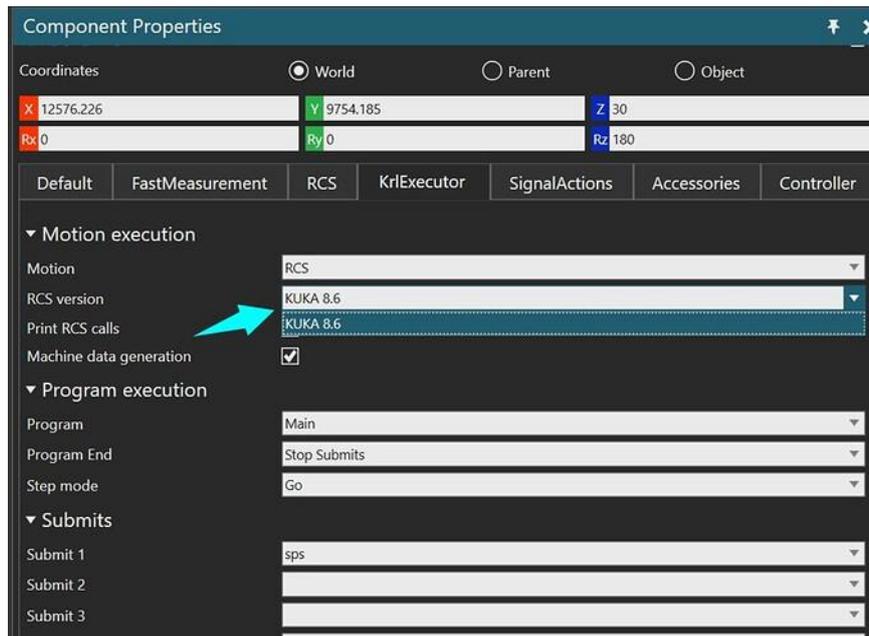


Figure 8: Screenshot of the configuration window within the simulation for RCS and KRL

4. ROBOT VIRTUAL COMMISSIONING

The virtual environment provided within VC 4.0 (r4.8) provides the functionalities to program a robot and simulate the robot. The robot program editor presented in D3.2 deploys the functionalities to create a robot program or modify existing ones or generated with path planning and other available tools within VC 4.0. Table 2 shows the statements available in VC 4.0 (r4.8) to program a robot through the robot program editor.

Table 2: Statements available to program a robot in the VC 4.0

Statement	Description
Break	Ends the execution of loop.
Call Sequence	Executes a specified subroutine in program.
Comment	Leaves a comment in program.
Continue	Sets loop to continue from next iteration.
Define Base	Sets the properties of a base frame in robot.
Define Tool	Sets the properties of a tool frame in robot.
Delay	Delays the execution of program.
Halt	Stops the execution of program.
If	Defines an if-then-else condition for executing one group of statements if the condition is True or another group of statements if the condition is False.
Joint Motion	Executes a point-to-point motion to a position by interpolating joint values.
Linear Motion	Executes a linear motion to a position based on current configuration.
Print	Sends feedback to be printed in the Output panel.
Program Synchronize	Synchronizes program execution with other programmable components using a matching sync message.
Return	Ends the execution of routine.
Set Binary Output	Sets the value of a digital signal connected to a robot output or signals an action in the robot.
Touch-up	Updates the properties of a motion statement for a selected robot position.
Wait for Binary Input	Waits for a digital signal connected to a robot input to reach a specific value.
While	Defines a condition for executing a group of statements in a loop.

While programming a robot in the virtual layout the user can just add the statements in the desired orders. Statements can be edited and rearranged in through the user interface just moving them in the desired order through the program editor UI (Figure 9).

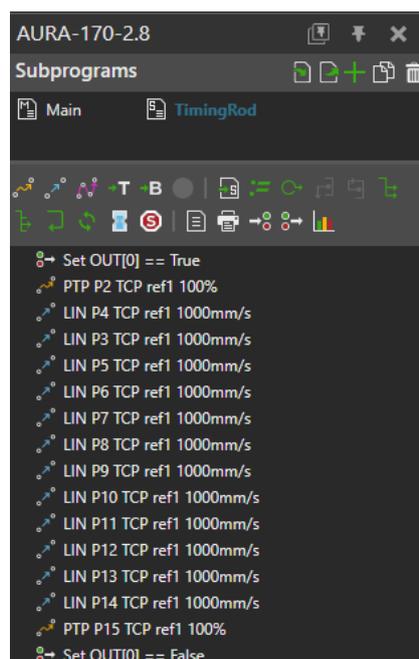


Figure 9: Robot program editor available in VC 4.0 (r4.8)

Once the robot program has been created and the initial simulations have been created, the statements can be simulated using the executor available in VC 4.0, which has been extended during the project. The executor in VC 4.0 reads, write and execute a robot program providing the properties showed in Figure 10, which can be modified by the user.

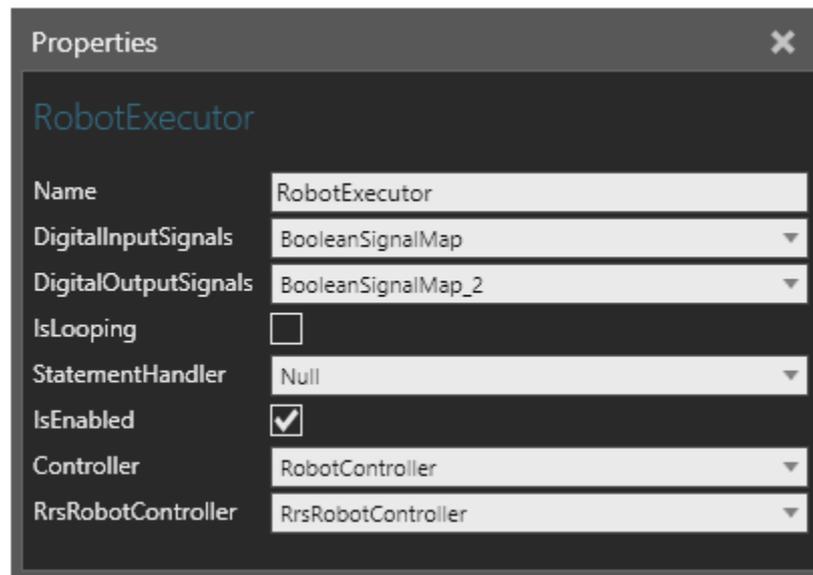


Figure 10: Screenshot of the robot executor properties UI

Once the robot is set up, configured in the layout, and programmed, the robot program routines must be transferred to a language that the robot can understand, action known as post-process. Every robot manufacturer has its own robot program language that runs in its proprietary robot controller. The post-process transfers the robot program created during the simulation stage into the language that the robot can run into its controller.

The work developed in ODIN project has been focused on the post-processors for UR, KUKA and COMAU aligned with the white goods pilot, the aeronautics pilot and the automotive pilot.

After post-processing the robot program can be uploaded in the robot controller and start testing the program, but as mentioned in section 3 and aligned with the target of the Task 3.5, the use of virtual commissioning will allow to validate the code and identify possible errors. To achieve the virtual commissioning, the workflow presented in the next section has been followed.

4.1. Workflow for virtual commissioning

Once the pilot layout has been created in the virtual environment and the initial simulations have been completed, it is possible to start the virtual commissioning or the robots. For achieving virtual commissioning, a workflow of eight steps presented below has been set up.

4.1.1. Robot programing and simulation

The robot is programmed in VC 4.0 using the robot programming statements, presented in Table 2 using the UI as showed in Figure 11 which provides access to the program editor. Modification to the robot programming to change parameters and create new programs can be also done with the robot program editor (detailed in Figure 9).

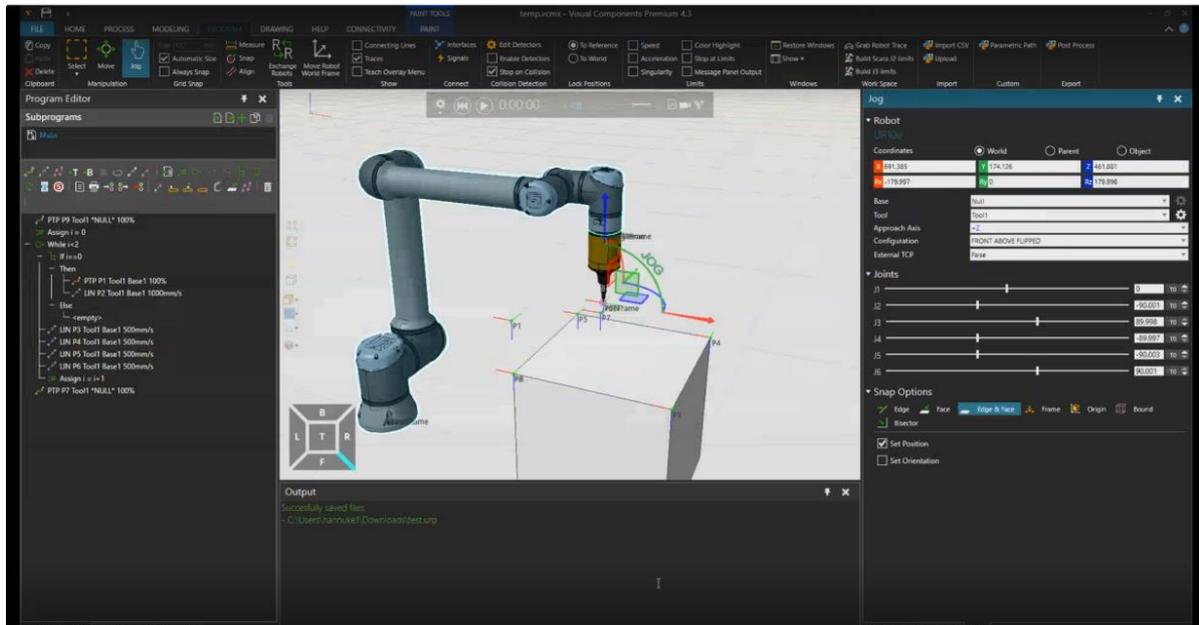


Figure 11: Robot programming within VC 4.0

In VC 4.0 all the robots are programmed with the same methodology, independent of the brand. This facilitates the learning of the software and the adaptation to different brands. During the concept and engineering phase this characteristic allows the user to analyze the performance of different robot brands and models allowing them to choose the one which adapts better to the requirements of the tasks, without adding additional robot programming effort as the program only requires to be created once and is reused when simulating the different robots.

4.1.2. Post-process to the robot language

When the simulation results match the operational requirements, the robot program in the simulation is postprocessed to the robot language. The access to the post processor is through the UI, PROGRAM tab/ Post Process button (Figure 12).

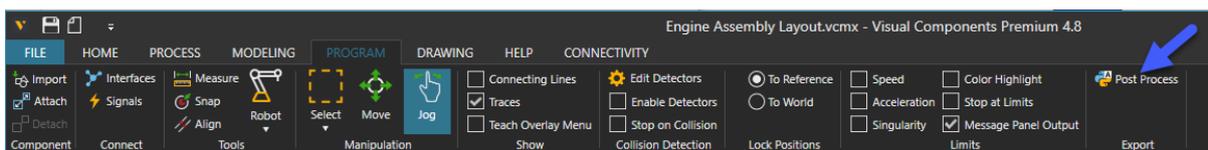


Figure 12: Screenshot of VC 4.0 UI showing access to Post Process

Once clicked, the Post Process button the generation of the specific code for the robot brand/model starts. The developed add-on selects the correct post processor, based on the properties defined of the robot within the virtual space.

As mentioned at the beginning of this section, each robot brand has its own programming language. Furthermore, depending on the model of controller the robot is using, has different characteristics. Within the ODIN project the development has been focused on the development and maintenance of the program post-processor as well as the communication plugins (section 3.3) required in the project.

4.1.2.1. KUKA

For the KUKA robot, used in the aeronautics pilot, the KRL is supported (KUKA Robot Language). The post-processor generates one *.src and one *.dat file. Base/tool frame definitions are written at the beginning of the main routine, which can be commented on the robot settings definition in VC 4.0 UI.

Furthermore, the post-processor also supports SUNRISE -Java for the required KUKA robot controllers defined within the simulation. One *.java and one RoboticsAPI.data.xml file is generated. The *.java file contains the program itself and the base/tool frame definitions as well as position frames are written into the RoboticsAPI.data.xml file (Figure 13). Simulation IOs should be mapped to real IOs using wrapper functions GetDO and GetDI in .java file so that for given simulation IO port those functions should return desired Output or Input object.

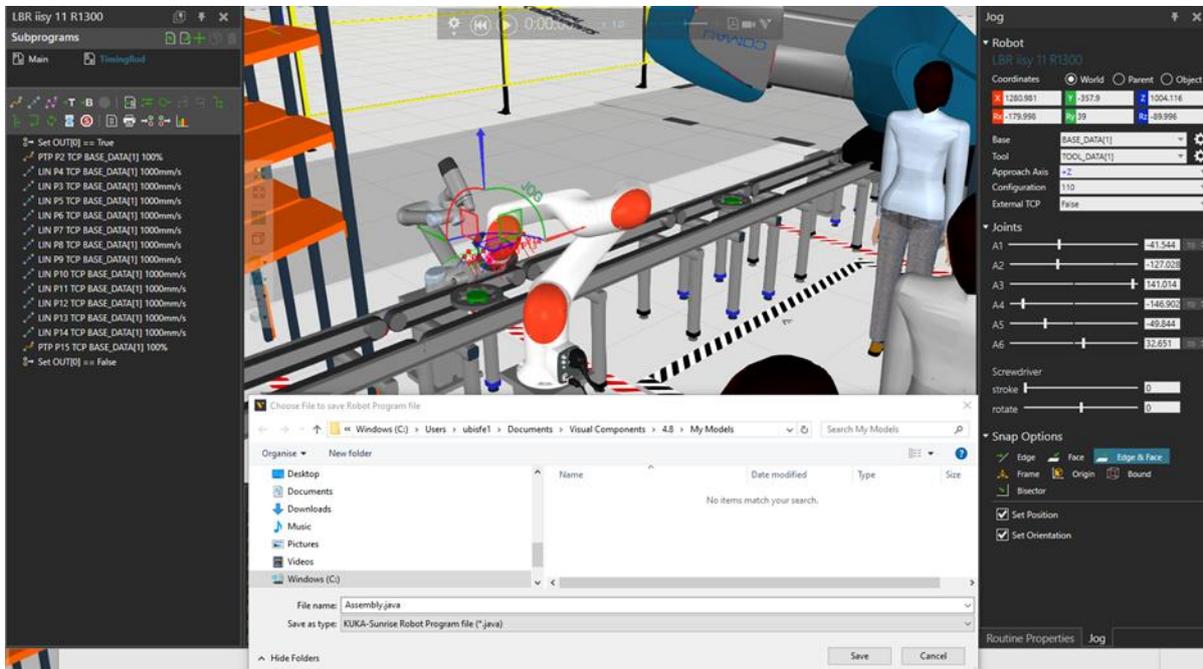


Figure 13: Screenshot of the KUKA SUNRISE - Java post-process

4.1.2.2. UR

For the UR robot used in the white goods pilot, the post-processor produces a *.urp file for each routine defined in VC 4.0 program editor. But before creatin the *.urp file, is required to set up the configuration in the window opened after clicking the Post Processor button (Figure 14).

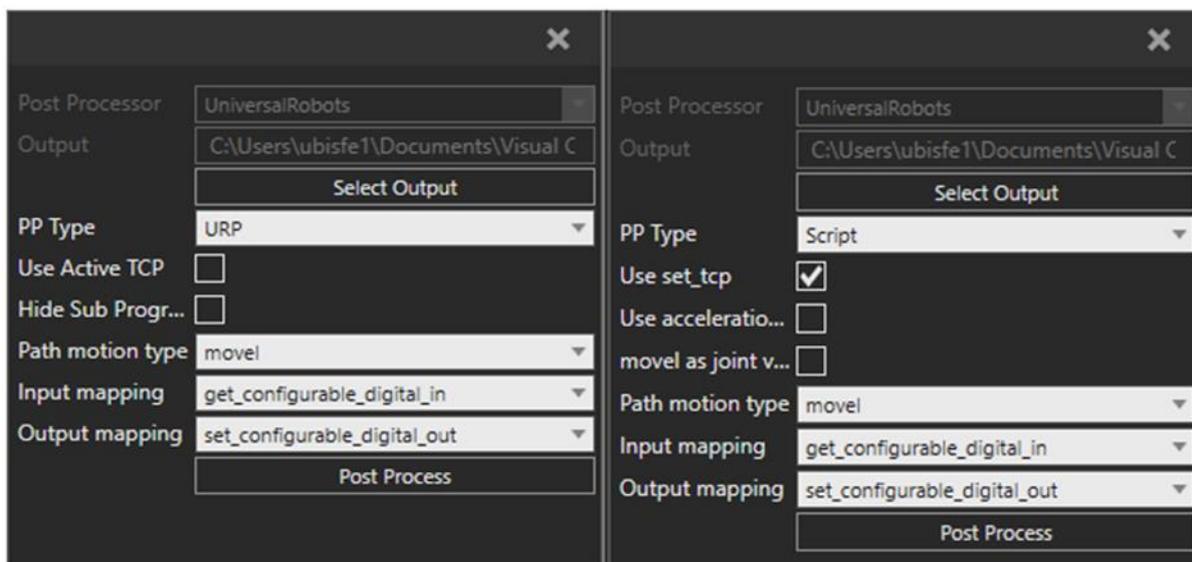


Figure 14: Screenshot of the post-process tab available at VC 4.0 for UR, PP Type URP (left) and PP Type Script (right)

The configuration includes:

- "PP Type"
 - If set to "Script" produces on .script file instead, which that can be imported into a script command.
 - "URP" is supported only for e-Series UR models. (Note that in .urp, subroutine calls are only possible from the main routine)
- Tool definition:
 - The TCP name used in VC 4.0 should be defined in Polyscope under INSTALLATION > TCP
 - VC Tool > X, Y, Z > TCP Position X, Y, Z
 - VC Tool > Rx, Ry, Rz > TCP Orientation Unit (RPY in degree)
 - *NULL* TCP in VC 4.0 will create 'Tool0' in MoveL and MoveJ . DEFINE THE TCP NAME as 'Tool0' IN POLYSCOPE UNDER INSTALLATION > TCP with position and orientation as '0'
- Sequence (routines):
 - During post-process, each subroutine in VC 4.0 is created as *.urp file
 - Loading the *.urp (main) file in Polyscope, Call statement and SubProg with the (sequence) routine name is created.
 - Users should manually navigate to the *.urp(sequence) file from their file system and select it. Also assign it to Call statement.
- Supported VC 4.0 statements:
 - PTP/LIN/Path, Wait Input, Set output, Halt, Comment, Call, Assign, If, While
- Settings:
 - Use Active TCP (URP): sets if default TCP is used or if TCP is specified in statement.
 - Hide Sub Program Tree (URP): Hide/show subroutine tree in URP.
 - Use set_tcp (Script): Use set_tcp function to set active tool pose. Use this if there are many tool frames in your program created in VC 4.0.
 - Use acceleration values (Script): Use optional parameter for acceleration on motions.
 - movel (Figure 14) as joint values (Script): Post movel as joint values instead of cartesian pose.
 - Path motion type (URP/Script, Figure 14): Post path statement as movel or movep.
 - Input mapping (URP/Script, Figure 14): Select IO type where wait input statements are mapped.
 - Output mapping (URP/Script, Figure 14): Select IO type where set output statements are mapped.

4.1.2.3. COMAU

For the COMAU robot used in the automotive pilot, the post-processor generates one *.pdl file including main and subroutines and one *.lsv file containing global variables such as positions. Base/Tool definition are written at the beginning of the program before the actual main routine is called. (Figure 15).

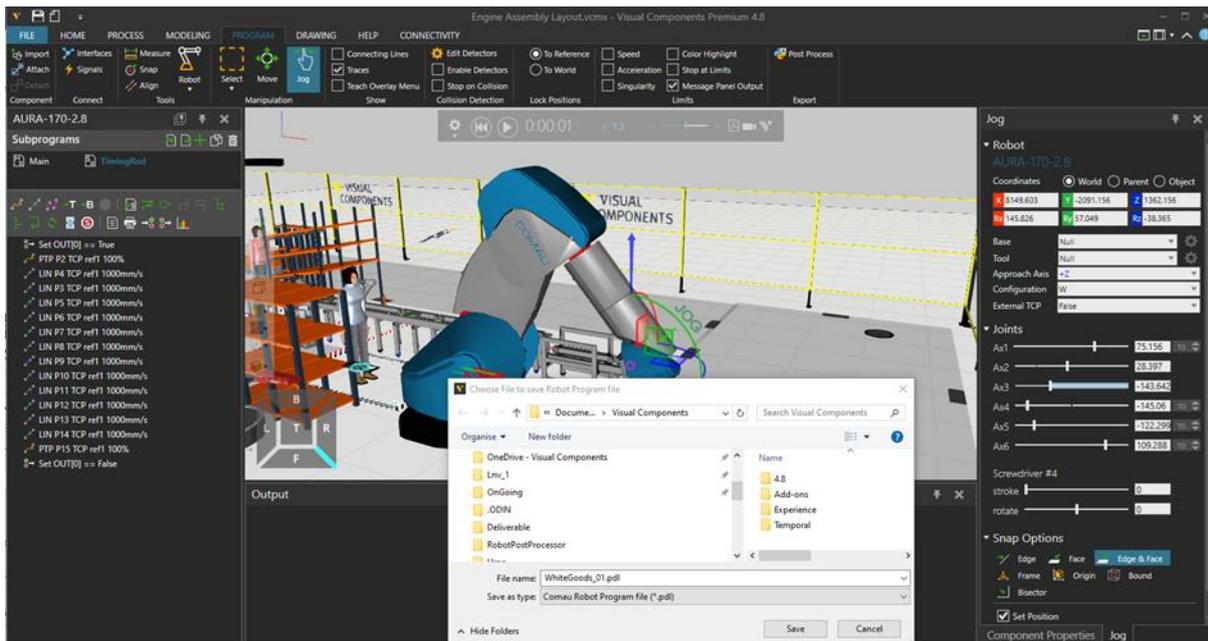


Figure 15: Screenshot of the post-processing of robot program of the COMAU Aura robot

4.1.3. Upload the robot program to the VRC

After the program has been post-processed into the language of the robot, the file is uploaded to the robot VRC. Similar processes are followed in the UR and in the KUKA.

In the case of the COMAU robot, the communication interface (section 3.3) has not been developed because the VRC is not available so this step and the following for virtual commissioning are followed and by uploading the post-processed file directly to the real robot as it has been presented in D5.4.

Figure 16 shows the process of uploading the *.urp file into the VRC for the UR robot.

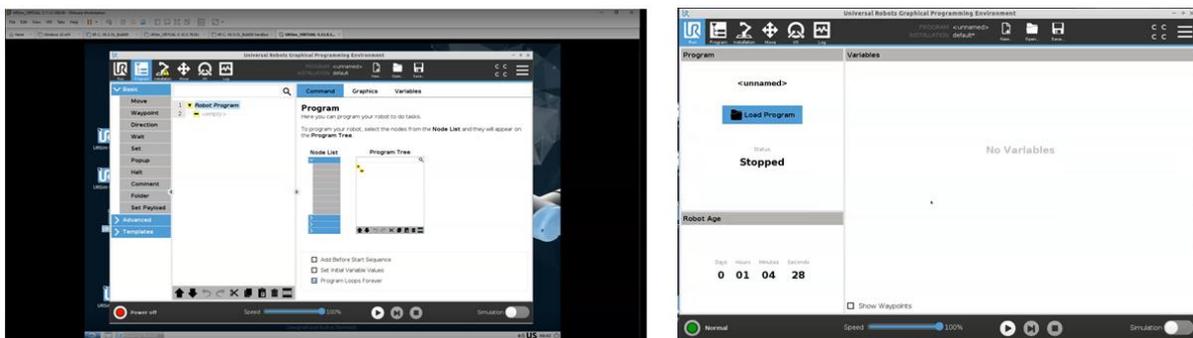


Figure 16: Upload process of the robot program into the virtual robot controller

4.1.4. Configuration of the robot in the VRC

The VRC, which mirrors the real controller virtually, should be configured with the same operational parameters as the robot in the simulation. Figure 17, shows how the operational are adjusted in the VRC, in this case in the UR.

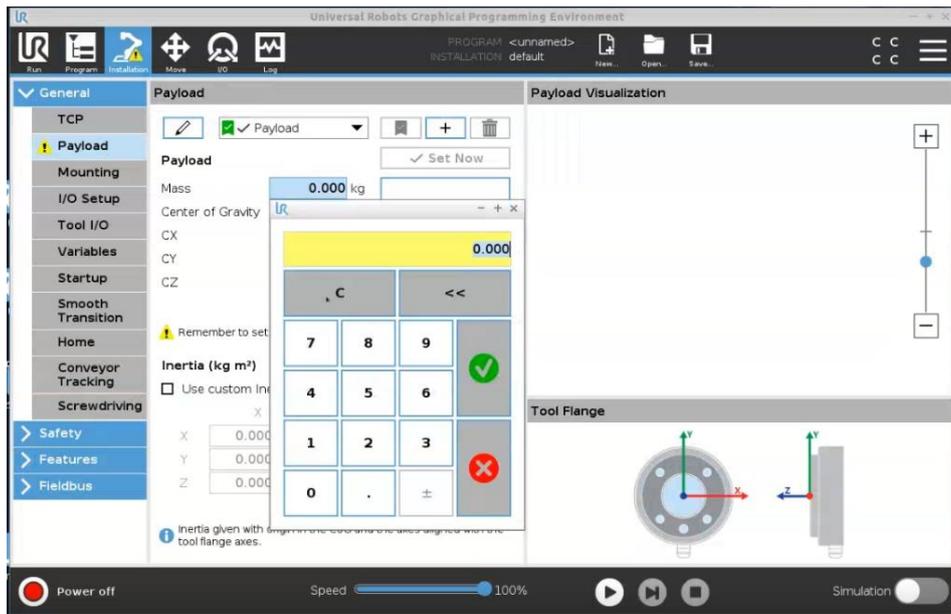


Figure 17: Configuration of the virtual controller with the same operation parameter than the simulation

4.1.5. Position set up

In addition to the robot configuration, both robots, the one in the simulation and the one in VRC should be in the same initial position before starting the virtual commissioning process (Figure 18).

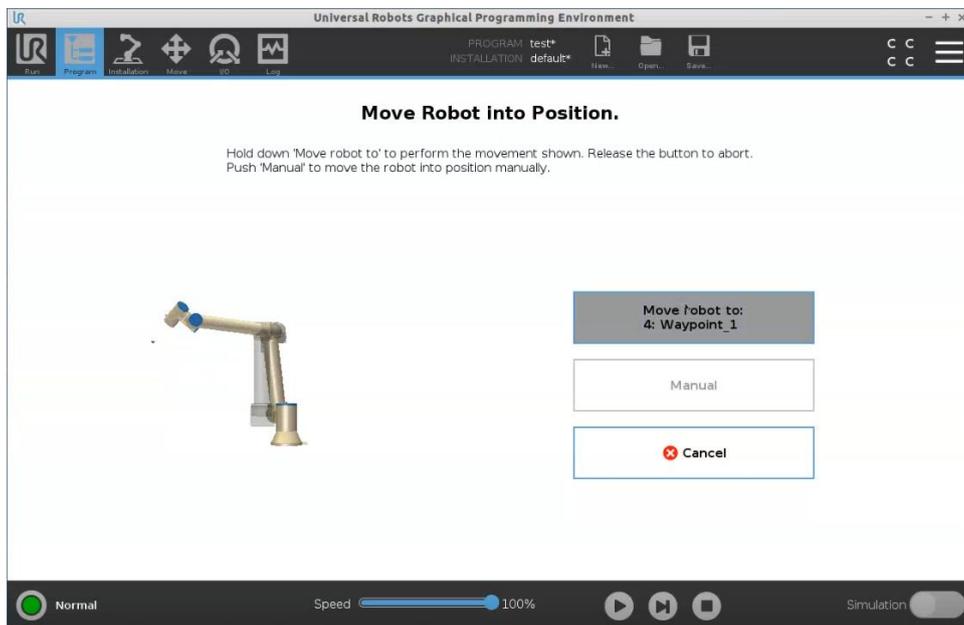


Figure 18: Position synchronization of the robots in VC 4.0

4.1.6. Program validation in the VRC

VRC allows to test and validate the code upload, this facilitates to verify the robot program post-processed into the virtual controller (Figure 19).

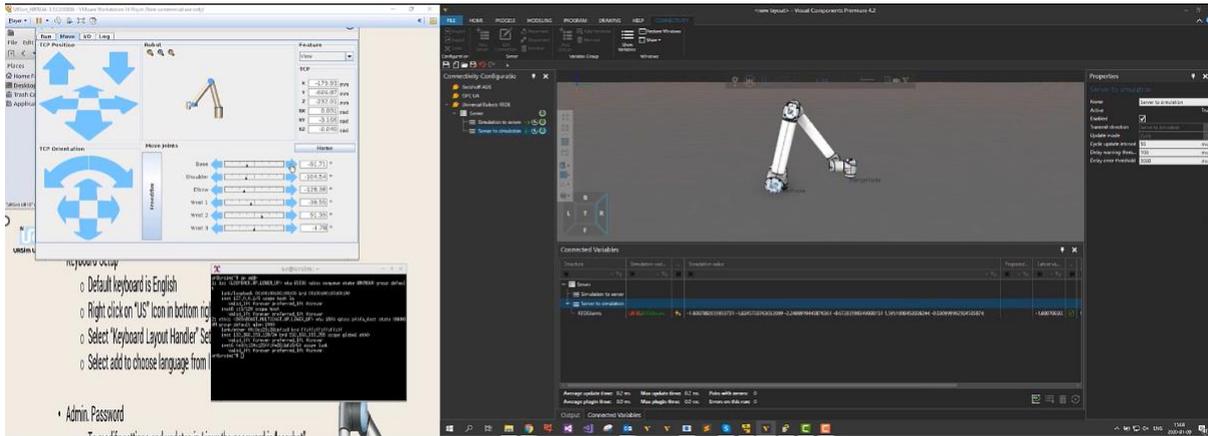


Figure 21: UR robot in VC 4.0 connected to the virtual controller through the communication interface displaying the program uploaded in the VRC

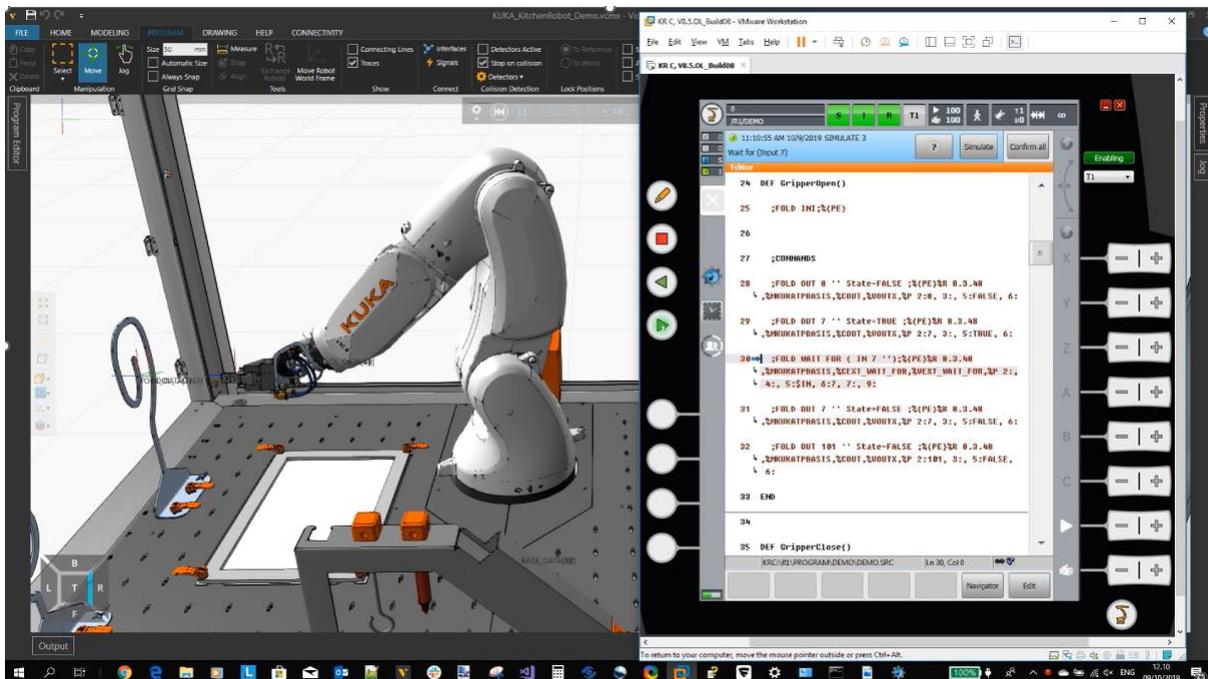


Figure 22: KUKA Robot in VC 4.0 connected to the virtual controller through the communication interface displaying the program uploaded in the VRC

5. CONTROL SYSTEMS COMMISSIONING

In addition to ROS 2 connectivity developed to connect through the OpenFlow, VC 4.0 allows the connectivity through OPC UA allowing the control of the systems connecting directly to the PLC (Figure 23). The utilization of this interface for control and validation has been discussed for the white goods pilot for controlling the PLC in the pilot.

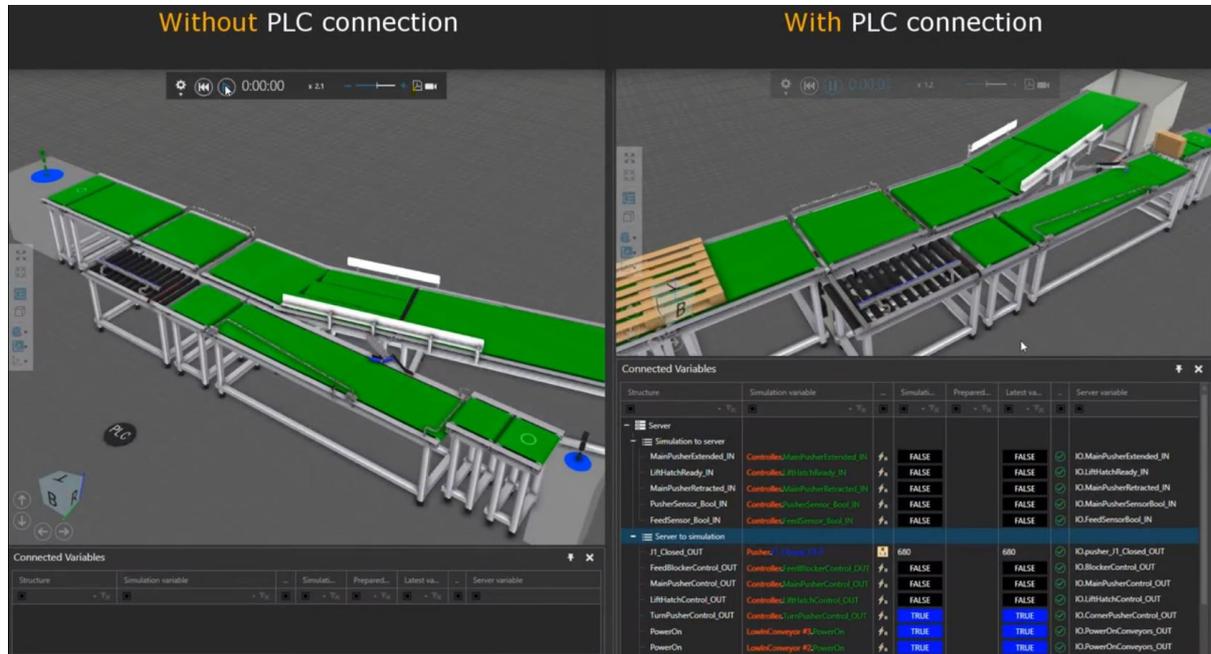


Figure 23: Screenshot of the virtual environment using the OPC communication with control (PLC) validation vs simulation

6. CONCLUSIONS

The work developed in Task 3.5 towards virtual control and commissioning has been focused on identifying the initial requirements for virtual commissioning of robots, targeting the robots used in the white good pilot and aeronautics pilot. The work has been completed and the results obtained are the base to continue the work in the final part of the ODIN project, particularly WP5.

The connectivity plug-in for ROS2 to integrate with OpenFlow has been deployed and is operational for robot movement and messages through OpenFlow has been completed. Connectivity with other modules such as the KTH's digital twin and LMS's AI task planner has been deployed as reported in D3.3.

Despite the task has been completed, the integration work will continue in WP5 for the three pilots giving the opportunity to extend the functionalities.

7. GLOSSARY

API	Application Program Interface
IP	Internet Protocol
TCP	Transmission Control Protocol
OC	Open Component
OPC UA	Open Platform Communication Unified Architecture
DC	Digital Component
RTDE	Real Time Data Exchange
UR	Universal Robots
.urp	universal robot program (extension)
RCS	Robot Control System
ROS	Robot Operating System
RRS	Realistic Robot Simulation
VC 4.0	Visual Components 4.0
VRC	Virtual Robot Controller
WP	Work Package

8. REFERENCES

- [1] S. Bangsow and U. Günther, 'Creating a Model for Virtual Commissioning of a Line Head Control Using Discrete Event Simulation', in *Use Cases of Discrete Event Simulation*, S. Bangsow, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 117–130. doi: 10.1007/978-3-642-28777-0_7.
- [2] S. Bangsow, Ed., *Use Cases of Discrete Event Simulation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-28777-0.
- [3] Z. Liu, C. Diedrich, and N. Suchold, *Virtual Commissioning of Automated Systems*. INTECH Open Access Publisher, 2012. Accessed: Jul. 02, 2015. [Online]. Available: http://cdn.intechopen.com/pdfs/37992/InTech-Virtual_commissioning_of_automated_systems.pdf