

EMULATION WITH VC PREMIUM/ROBOTICS AND POLYSCOPE(UR ROBOT GRAPHICAL PROGRAMMING ENVIRONMENT)

Table of Contents

Getting started with PolyScope (Universal robots graphical programming environment)	3
How to Post-Process a robot program from Visual Components Premium 4.0.....	8
Reading the imported robot program into PolyScope	11
Server-Client connection and emulation.....	16
Appendix.....	19
Universal Robots RTDE connection plugin	19
RTDE protocol operation principles	19
Connection settings.....	20
Controller address space	21
Supported data types	23
Performance	23
Q&A	24
Virtual Robot Controller (VRC)	24
Offline Robot Programming (OLP).....	25
Post Processor for robot.....	25

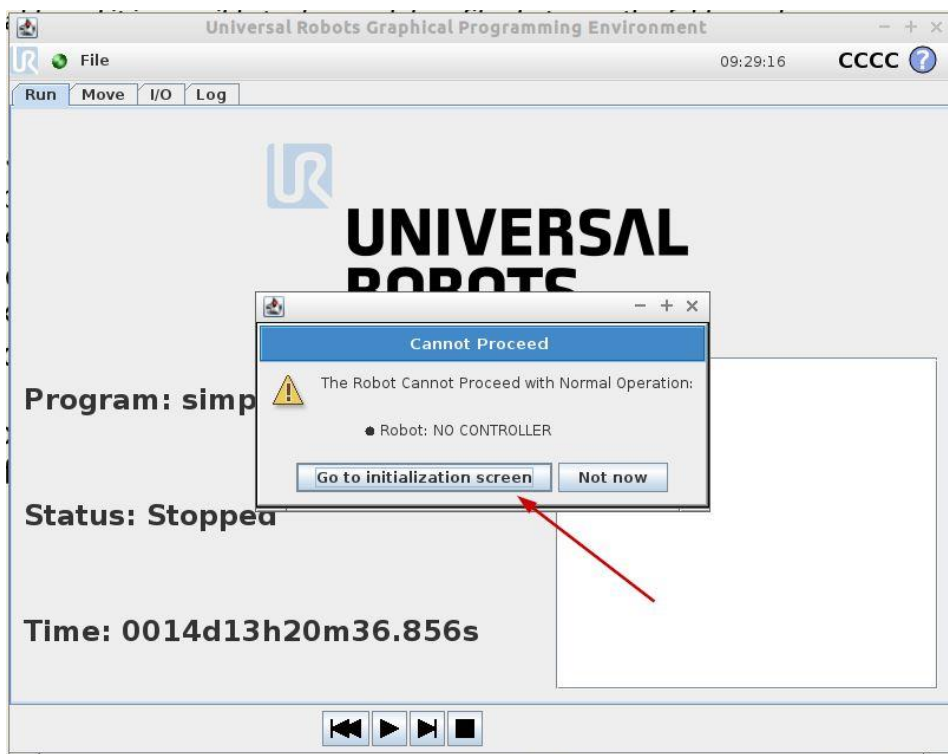
Getting started with PolyScope (Universal robots graphical programming environment)

First thing the user needs to do is to get the Virtual Machine where the URSim programs are installed /stored. There are 3 different programs for the 3 different versions of Universal Robot. URSim UR3, URSim UR5 and URSim UR10 are the different controllers for the different instances of robots. You can get the Virtual Machine from S Drive.

Open the virtual machine and double click on the icon [URSim UR5]



Select the [GO to initializing screen]

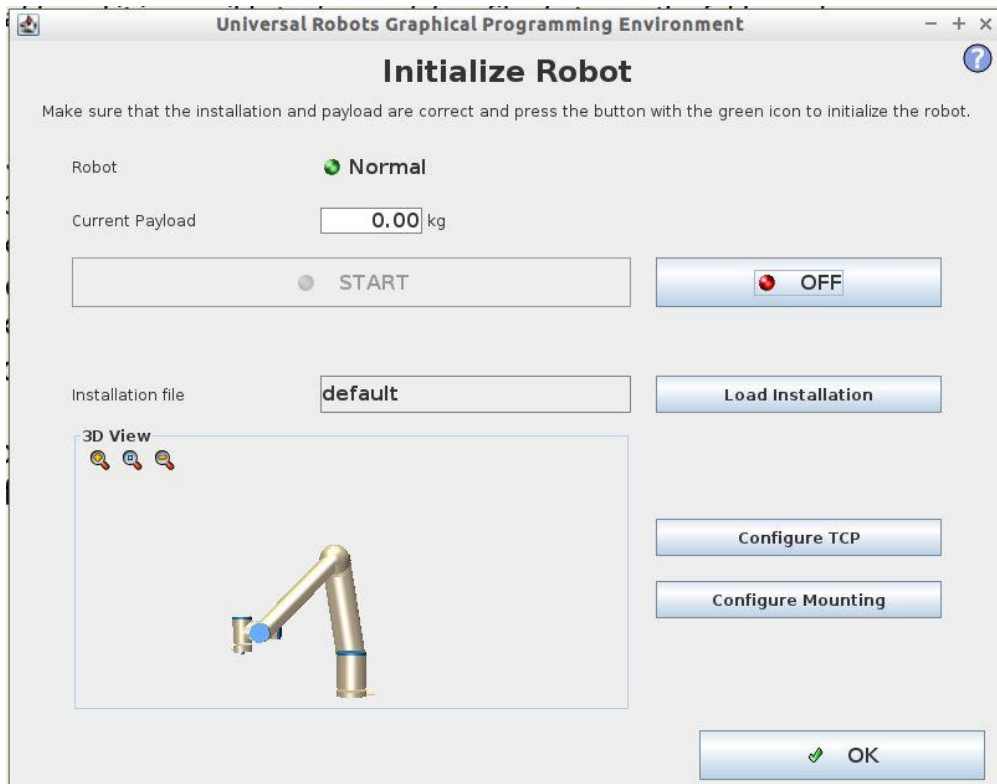


First press [OFF] button

Second press [ON] button

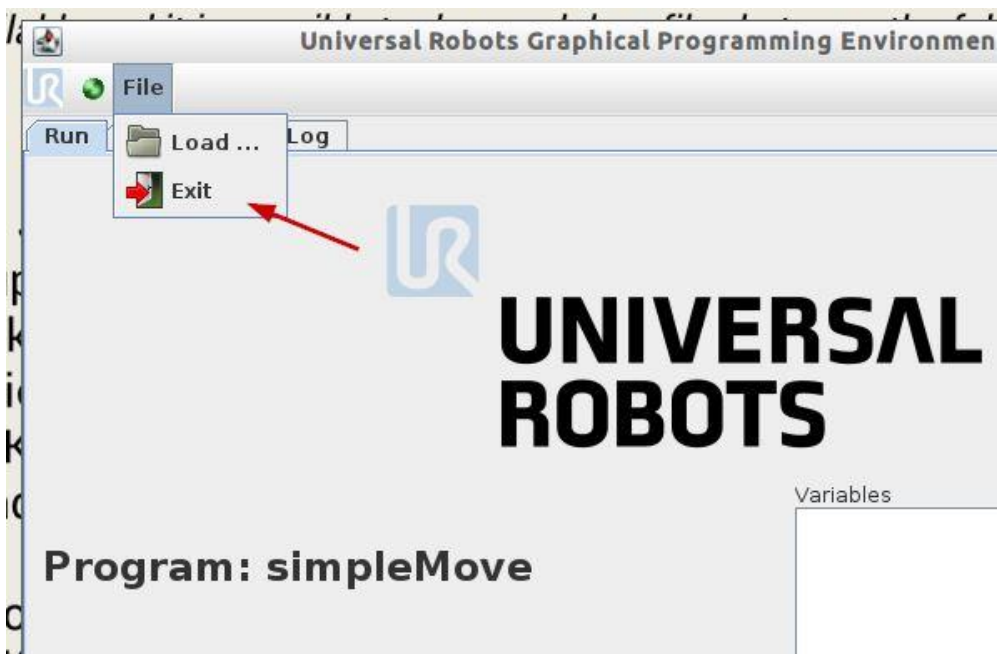
Third press [START] button

Final state looks like this –

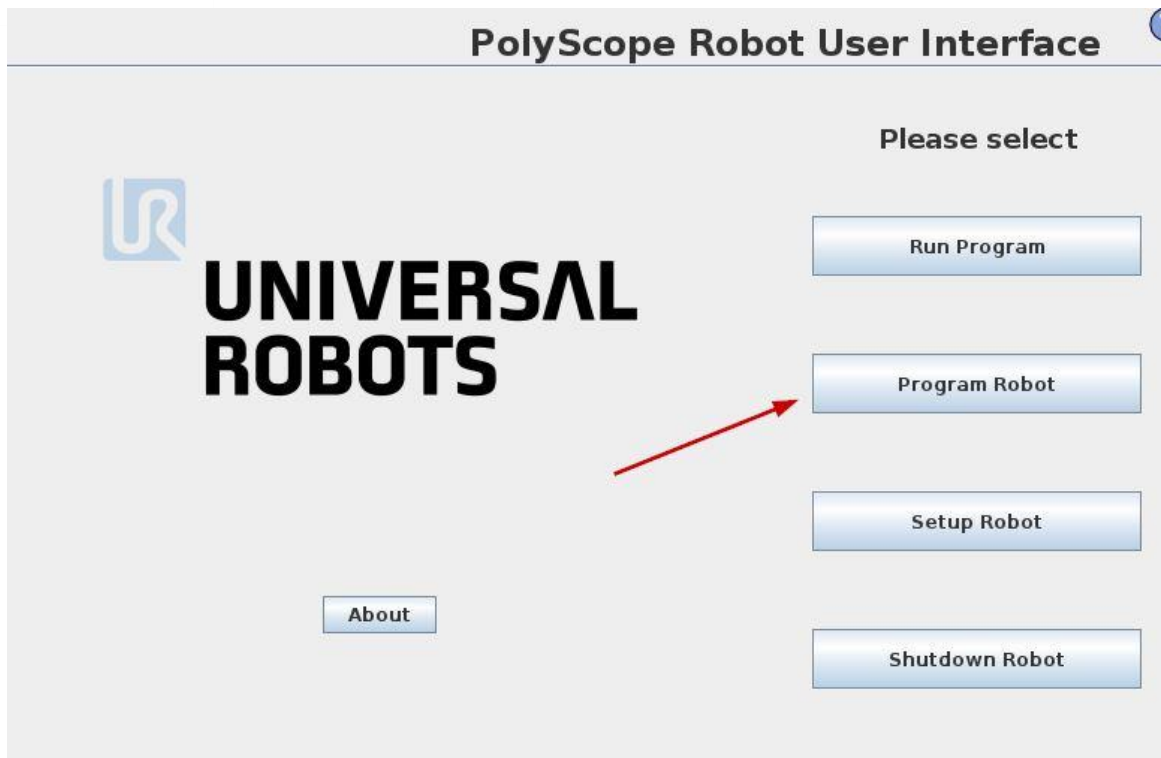


Finally press the [OK] button.

Go to File > Exit.



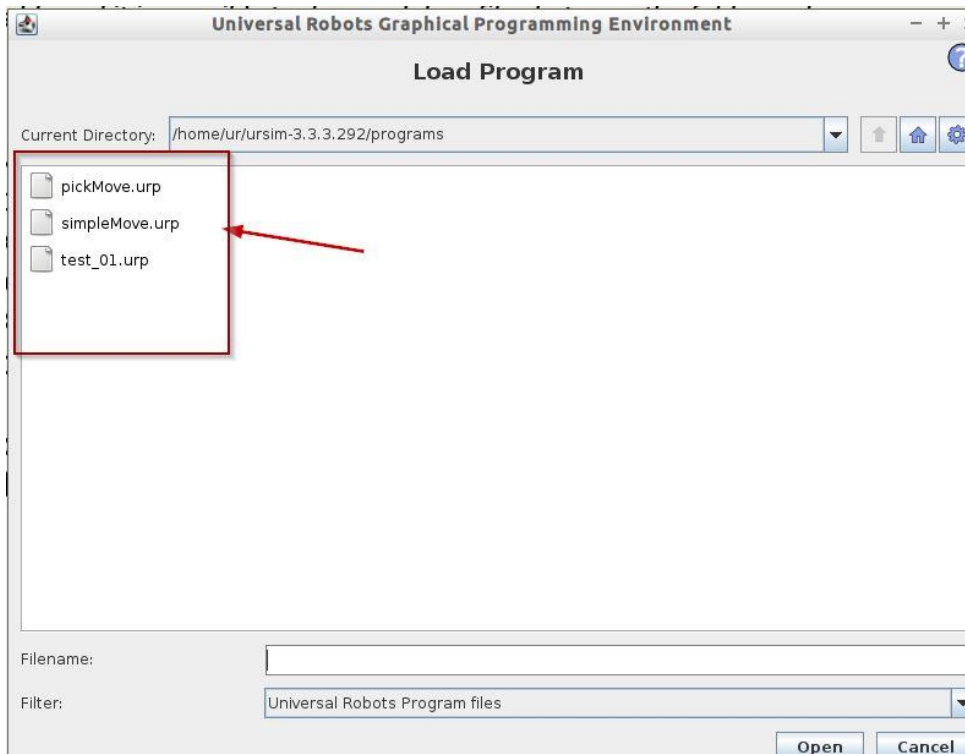
Press [Program Robot]



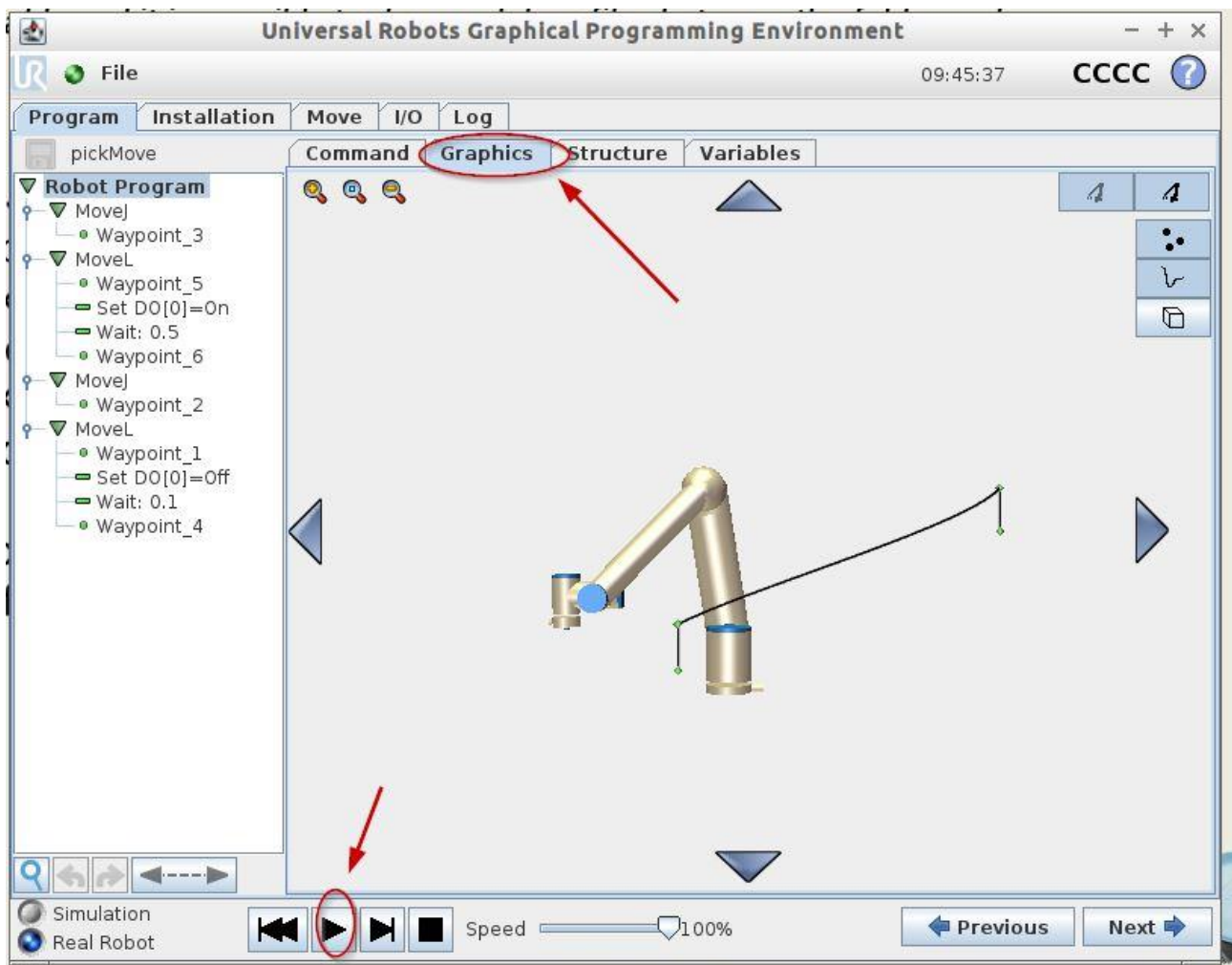
There are 3 different folders for storing the robot programs for UR3, UR5 and UR10. In the VM workstation already there some programs stored in those folders for the ease of user.

Now Press [Load Program] and select any of the programs stored in the Folder [Programs UR5] (*as we are programming UR5 in this example*) –

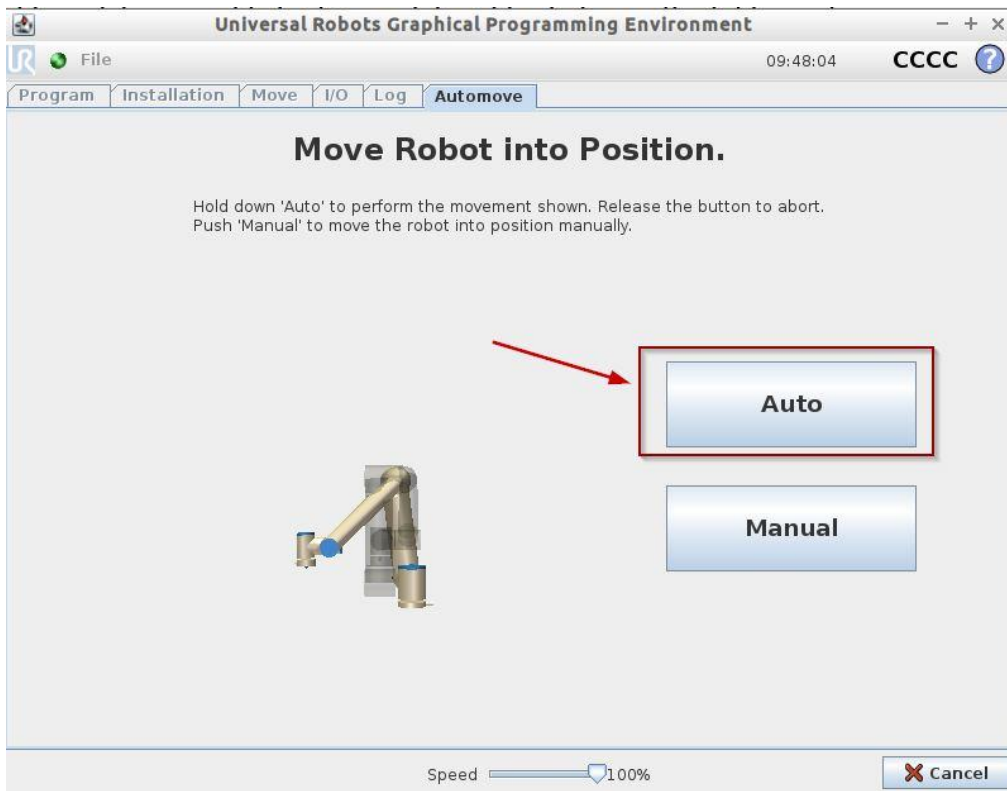




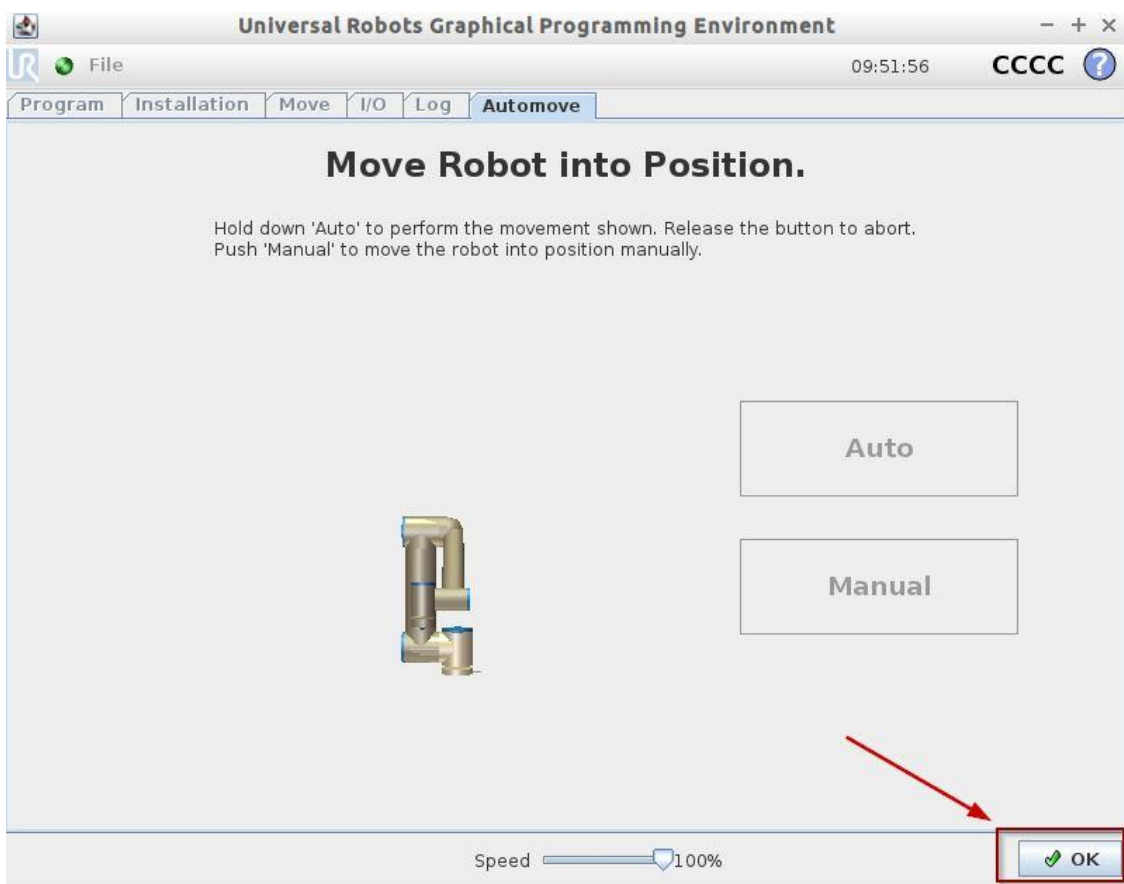
Go to the Tab > Graphics and then press Play



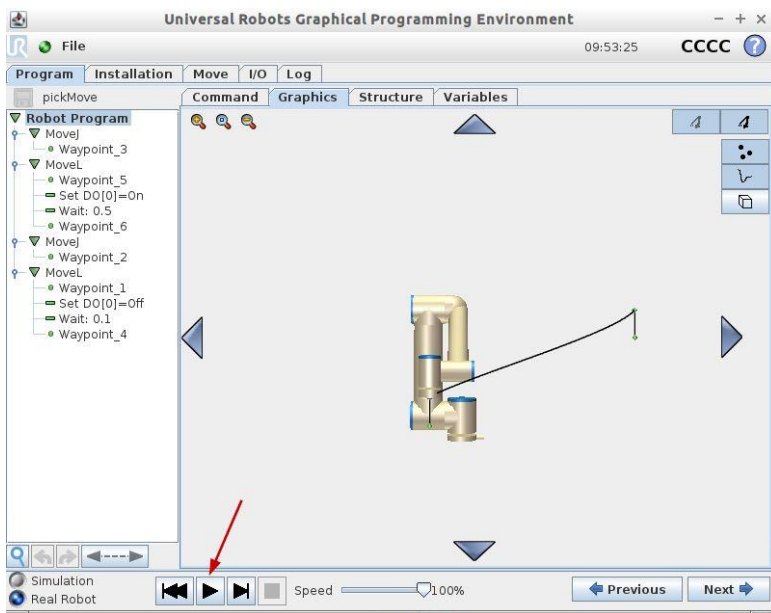
Pressing the [play] button will automatically take us to the Tab > [Automove]. Press and Hold the [Auto] button until the robot is ready for simulation.



Now the final state looks like this, the robot controller is ready for simulation. Press the OK Button.



Now we can run the simulation by pressing the Play button.

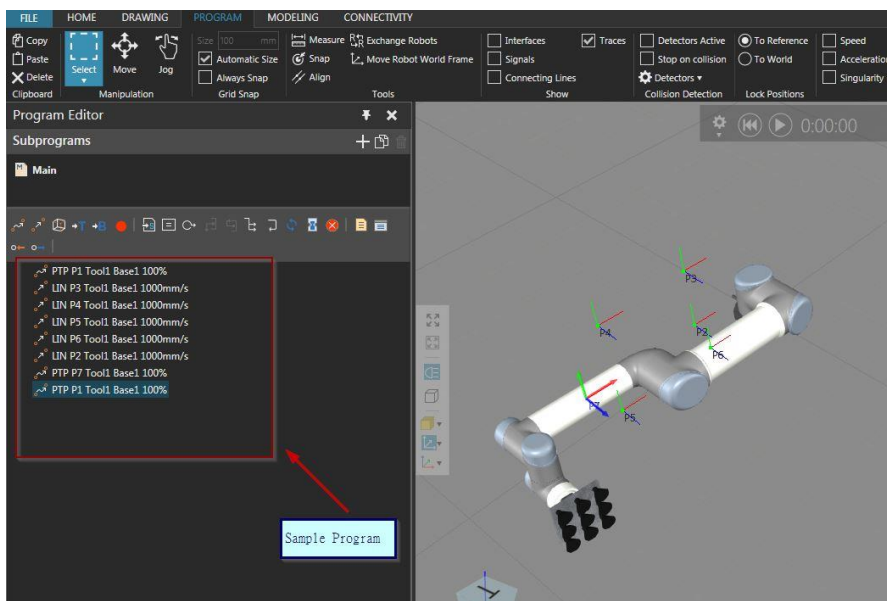


How to Post-Process a robot program from Visual Components Premium 4.0

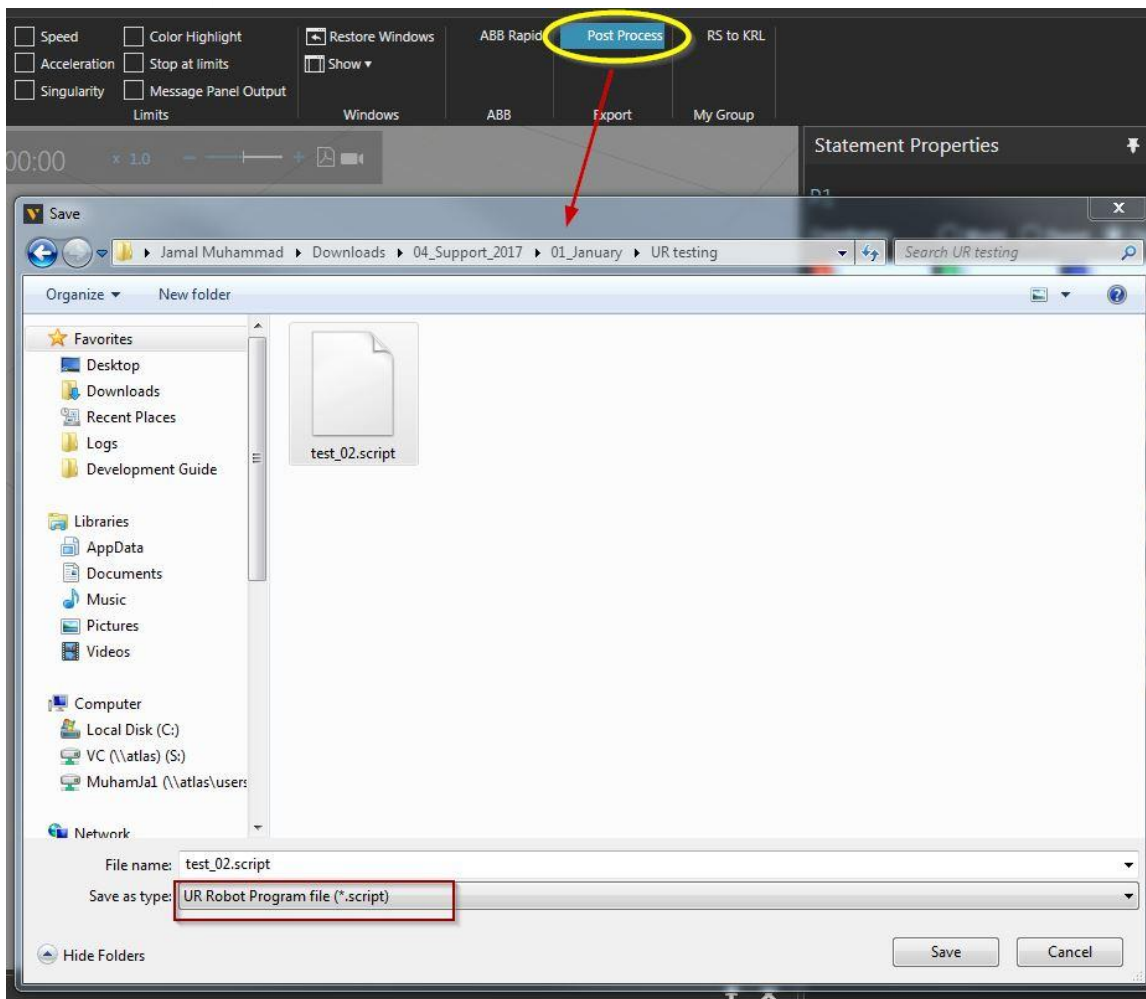
Open the Layout [UR5_testing.vcmx], there is a sample program with the UR5 robot in the robot. The target here are –

- Program the UR5 robot in VC Premium/Robotics.
- Post process the UR5 robot program using the Post-Processor. User can find the Post-Processor from Visual Components community.
- Establish connection in between VC Premium/Robotics(Main machine) and PolyScope(in Virtual Machine)
- Export the post-processed program to Virtual Machine and load the program in PolyScope.
- Run robot program in PolyScope and emulate the same movement in VC Premium/Robotics.

Run the program in VC Premium/Robotics and observe the robot movement.



Now using the post-processor export the robot program from VC to UR robot program file (*.script)



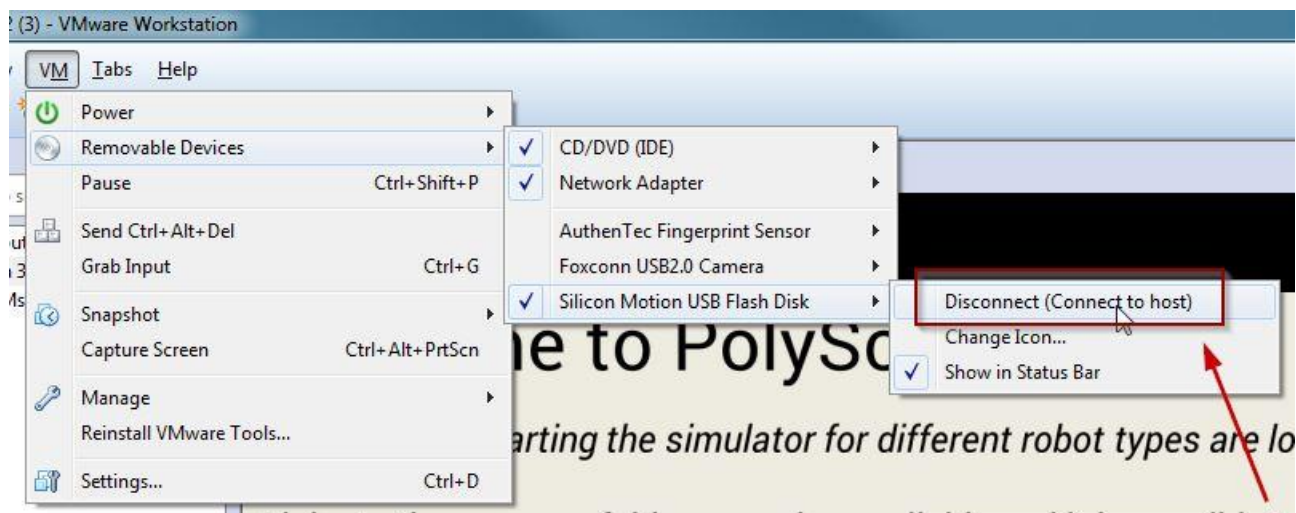
You can find the Post Processor in Program tab as shown in above screenshot. When we open the post processed file it shall be something like below.

```
File Edit Selection Find View Goto Tools Project Preferences Help
UR_script.py x test_02.script x RTDE_reader.py x
1 def test_02():
2     movej([-0.014417,-0.782716,0.776956,-0.011221,0.040680,0.006559],v=6.283185)
3     movel(p[-0.367653, -0.181292, 0.466071, 1.569944, -0.035081, -0.051454],v=1.000000)
4     movel(p[-0.622141, -0.181292, 0.466071, 1.569944, -0.035081, -0.051454],v=1.000000)
5     movel(p[-0.622141, -0.181292, 0.193585, 1.569944, -0.035081, -0.051454],v=1.000000)
6     movel(p[-0.331112, -0.181292, 0.193585, 1.569944, -0.035081, -0.051454],v=1.000000)
7     movel(p[-0.367653, -0.181292, 0.296552, 1.569944, -0.035081, -0.051454],v=1.000000)
8     movej([-0.014417,-0.782716,0.776956,-0.011221,0.040680,0.006559],v=6.283185)
9 end #test_02
10
11
12 test_02()
13
```

Copy the program to a USB Drive. The way we will be transferring the robot program from our main machine to virtual machine is by using USB drive.

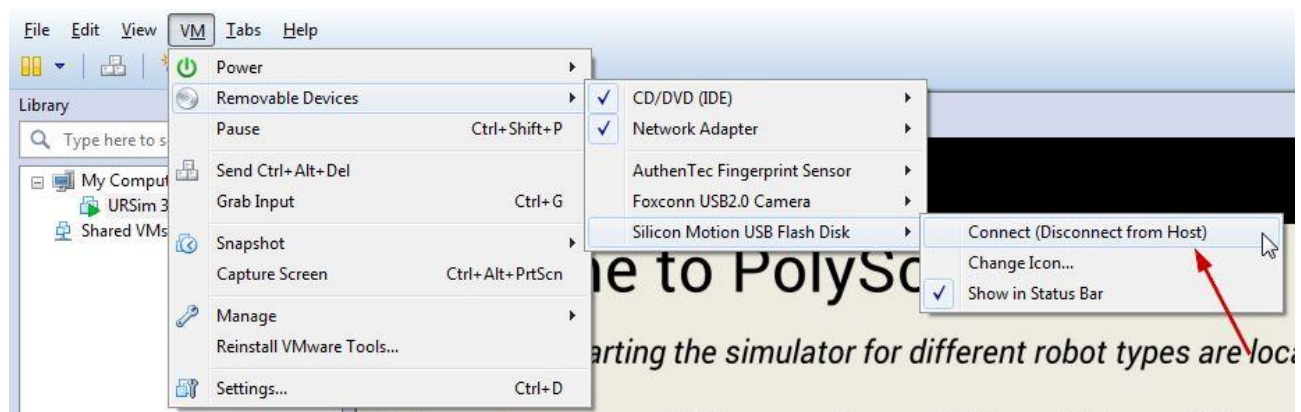
When we insert a USB drive it is usually found in the virtual machine, now the user needs to disconnect it from virtual machine and connect it to the main machine. It can be done following way –

- a. Go to VMware Workstation
- b. Go to VM > Removable Devices > Silicon Motion USB Flash Disk(it will be different in your case) > Disconnect(Connect to host)



Now user needs to copy-paste the program into the USB disk and then re-connect the USB disk with the VMware. Now user needs to use the tool –

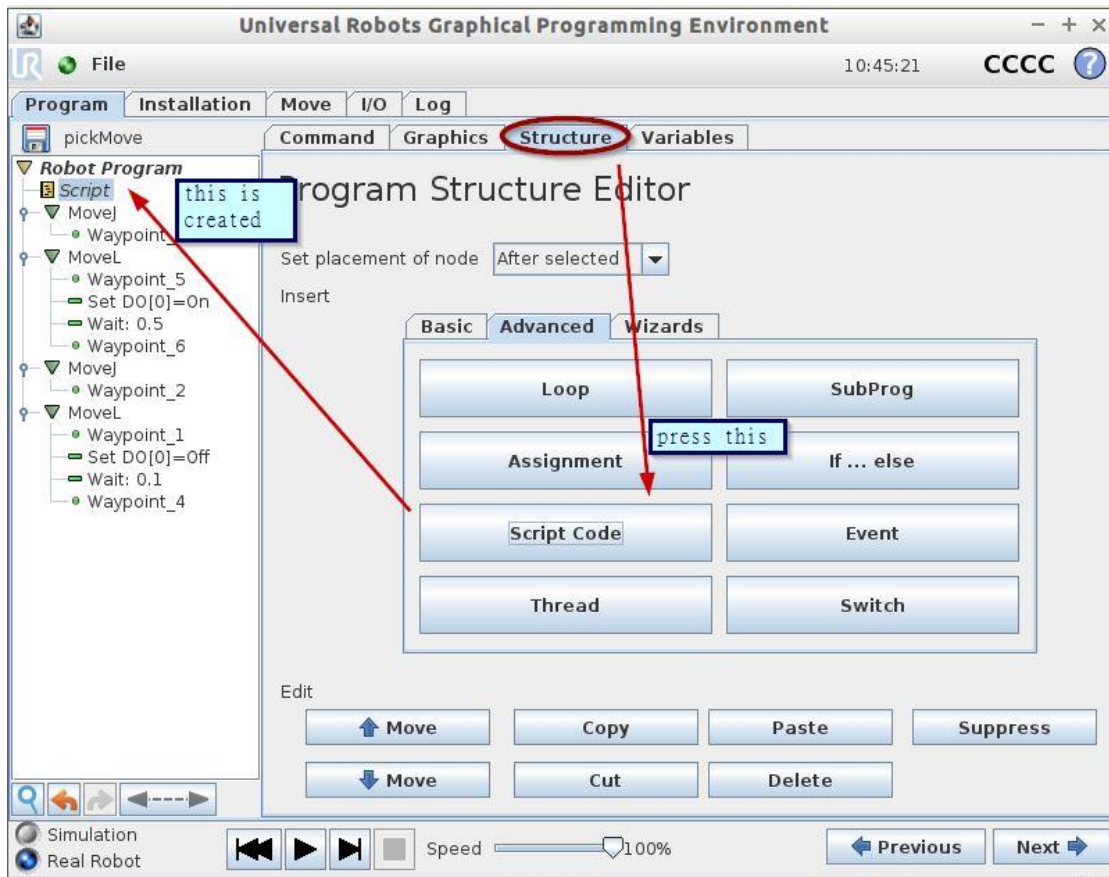
VM > Removable Devices > Silicon Motion USB Flash Disk (it will be different in your case) > Connect(Disconnect from host)



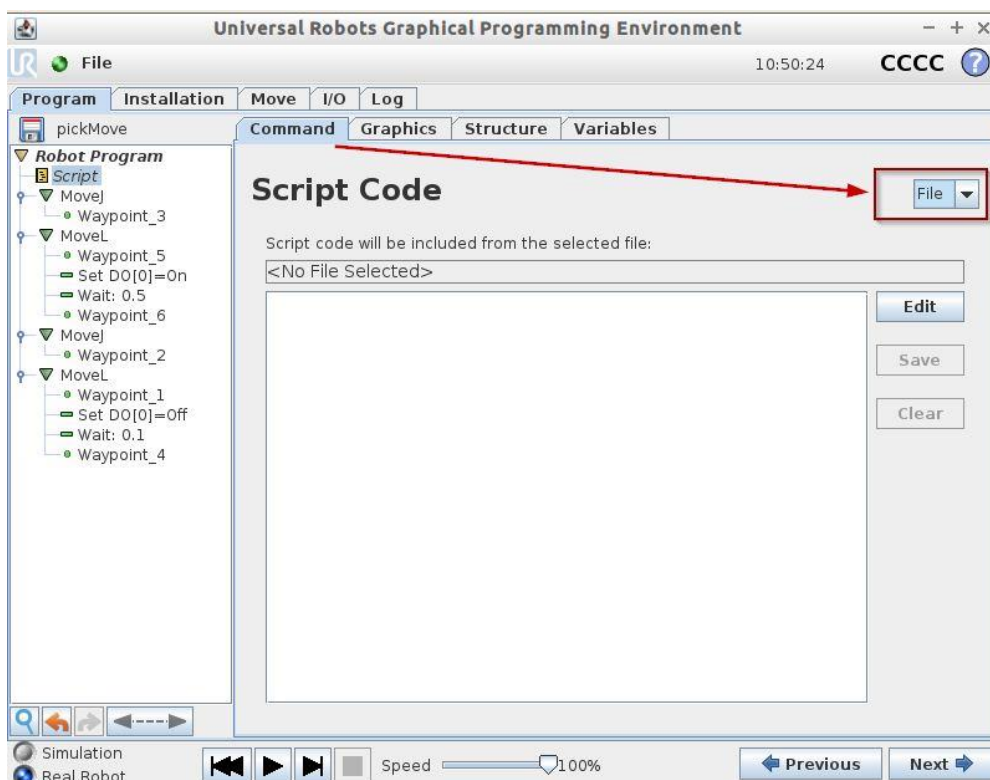
Later Visual Components will develop a FTP connection to send/receive files in between Main machine & Virtual Machine.

Reading the imported robot program into PolyScope

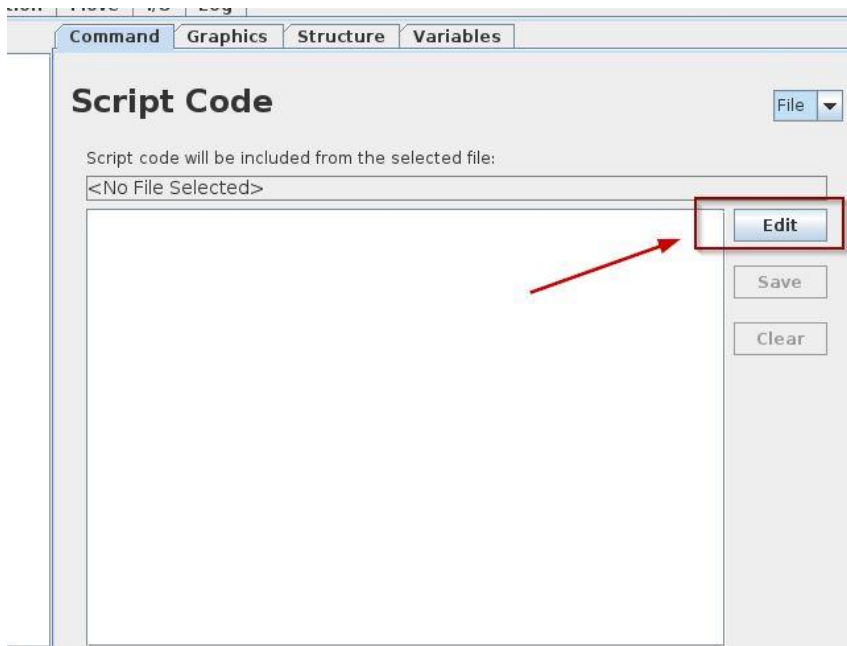
Go to [Structure] > Press [Script Code] and this will create a (Script) in the Robot Program tree.



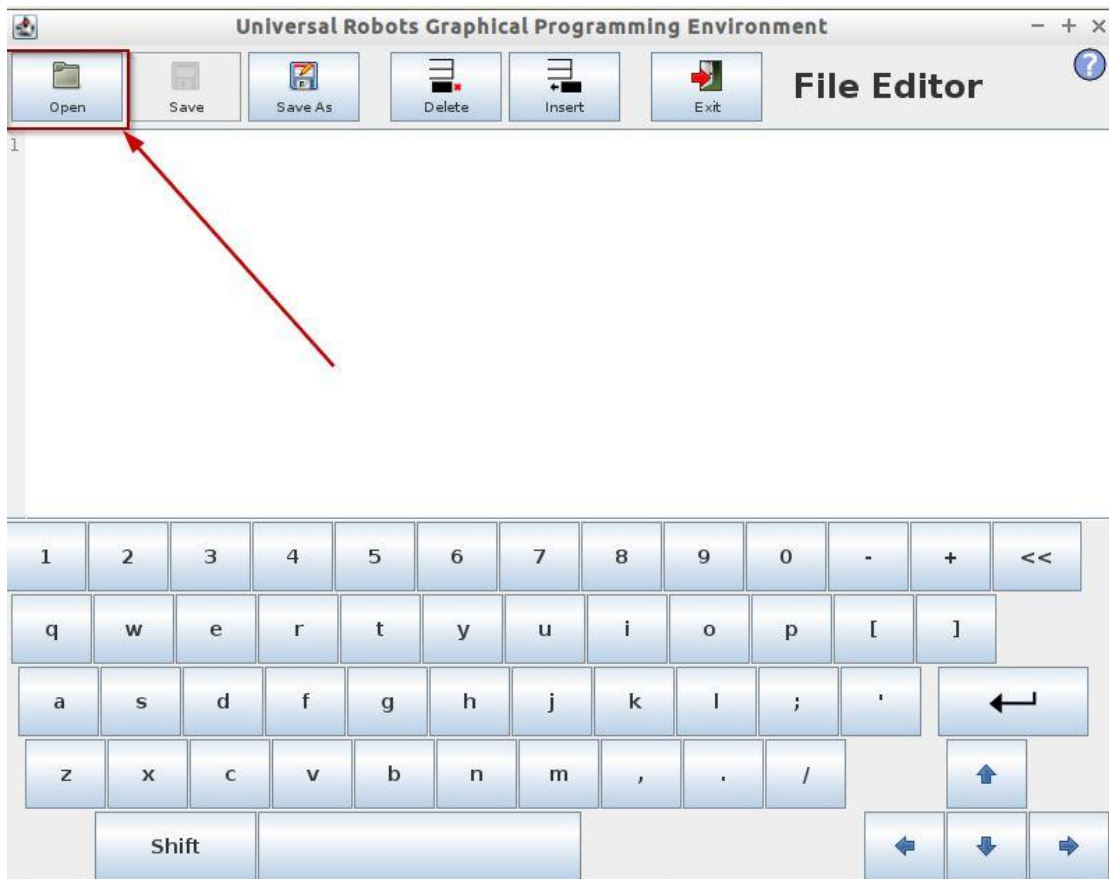
Go to Command tab and select File from the dropdown list



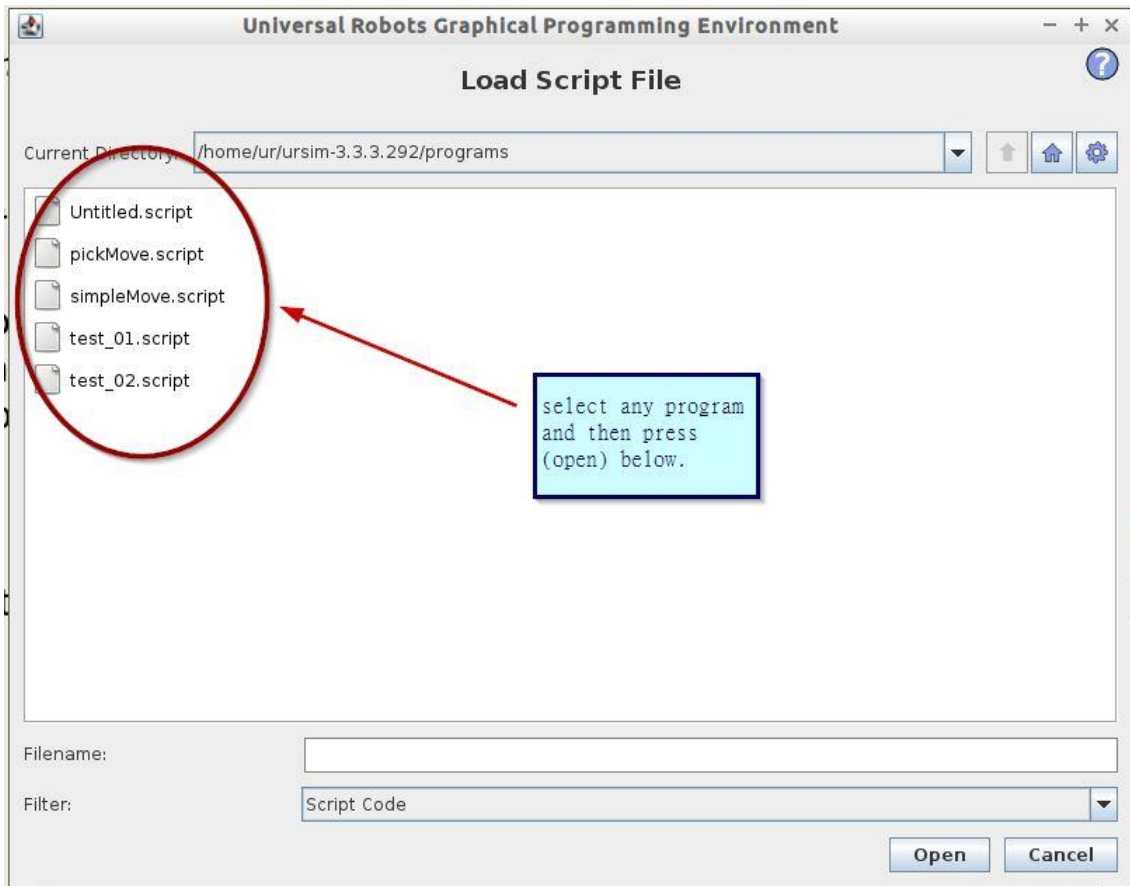
Then press the Edit button.



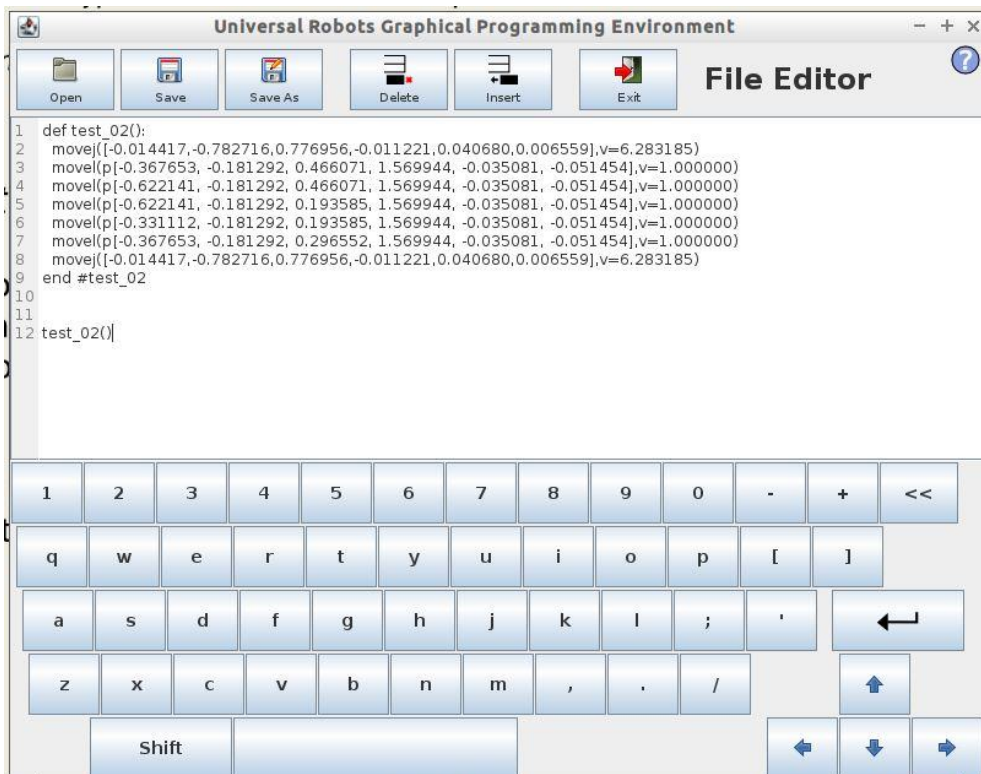
Then Press (Open)



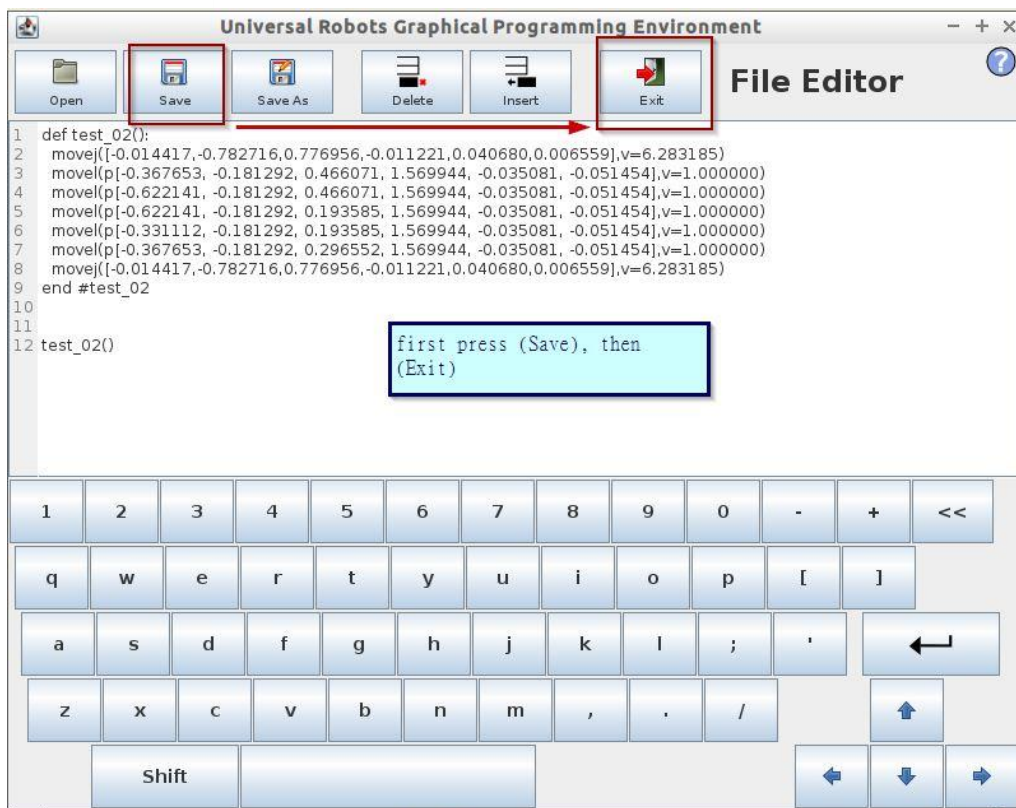
Now finally we need to select the imported robot program (post processed from VC Premium/Robotics) and load it to PolyScope.



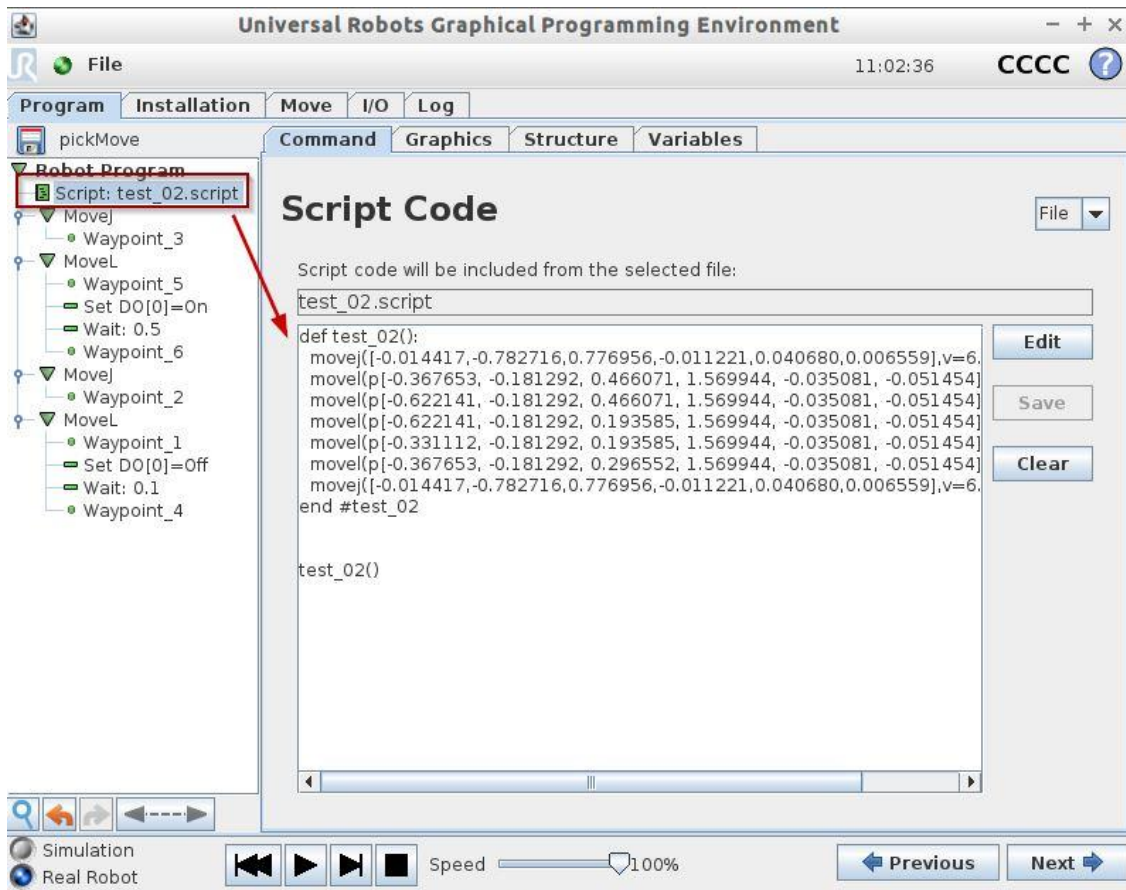
It will be read by the editor like below if reading is successful.



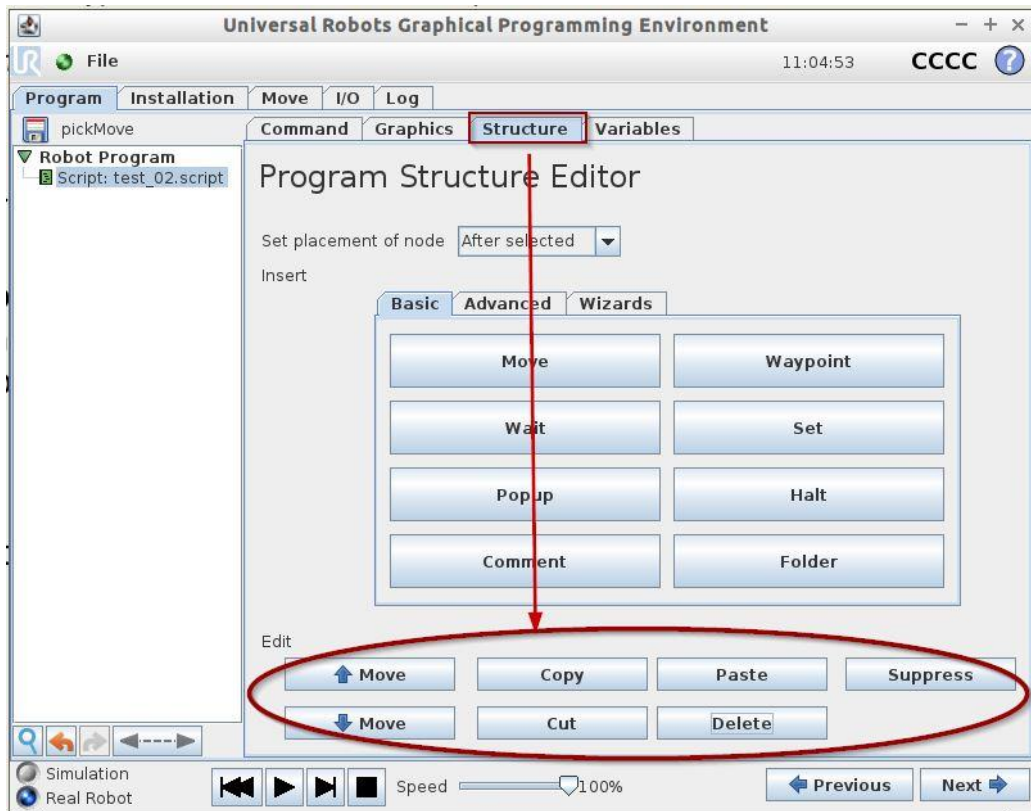
Now press save and then exit.



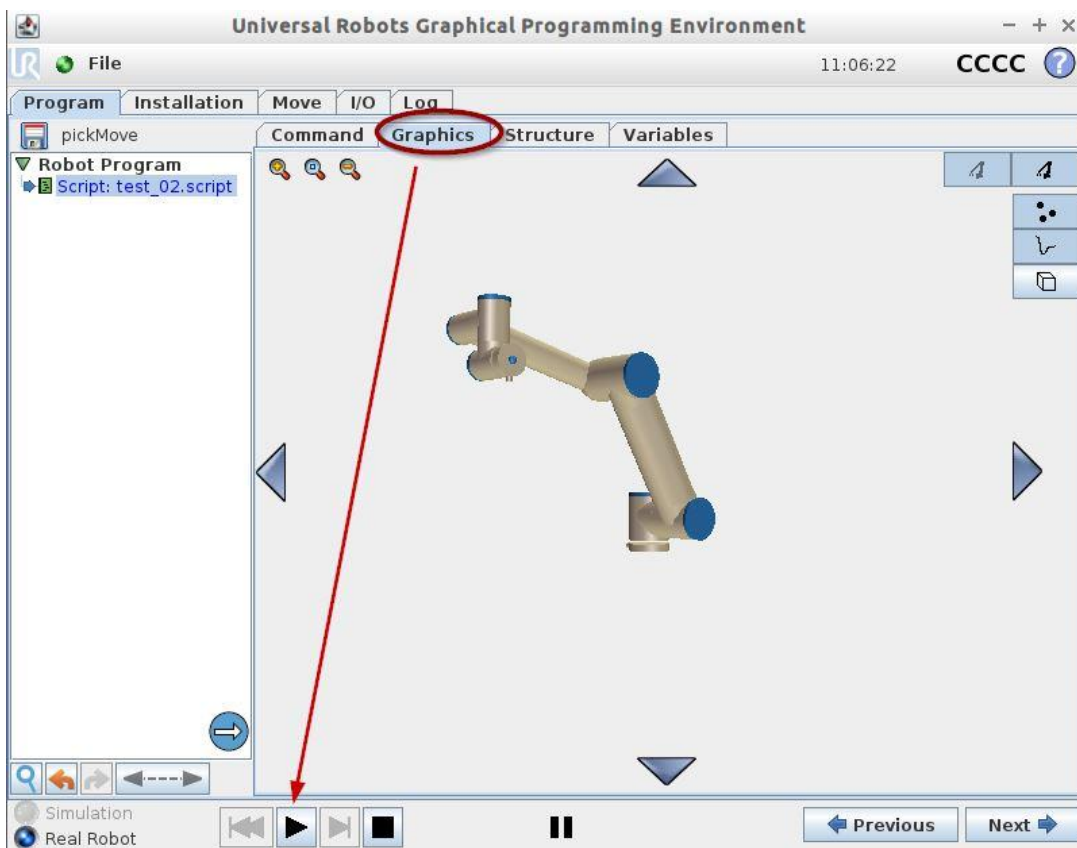
Now the user can see that loaded program as following.



Now the user can delete the rest of the Program from the Program Tree and keep only the Script program if he wishes.



Now user can go to Graphics > run the simulation.



Server-Client connection and emulation

Now we need to find the IP address of the Virtual Machine. We can do it in following way -

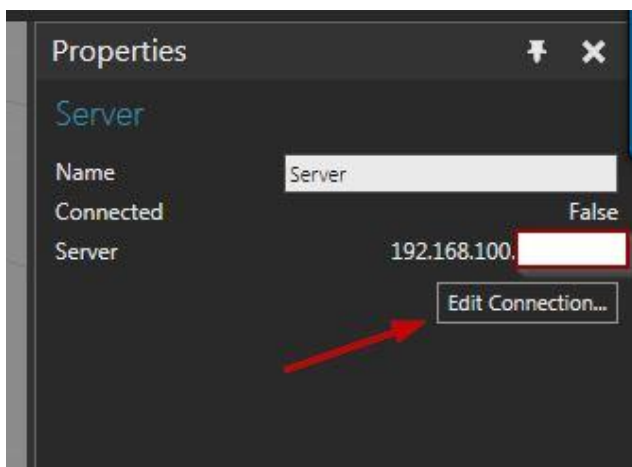
Open Terminal (In Linux [VMware])

Type >> `hostname -I`

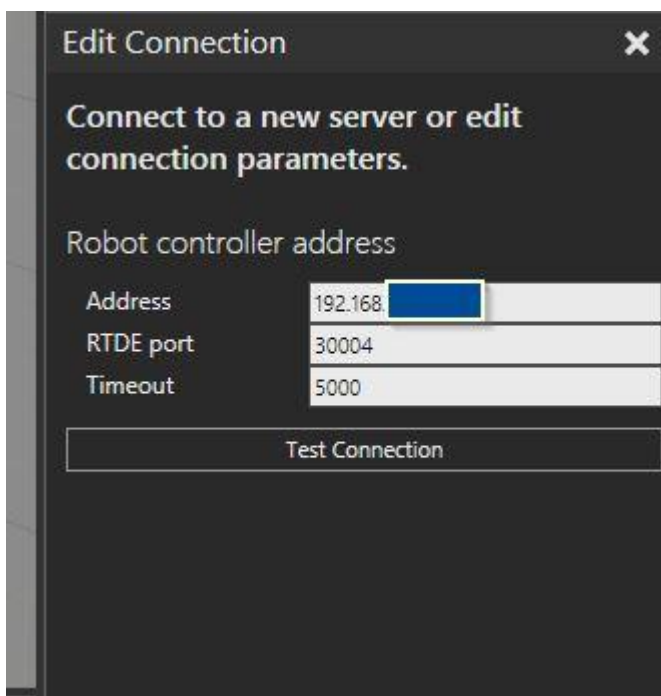
As an example we can assume we got the IP address of the VMware as

>> 192.168.100.XXX

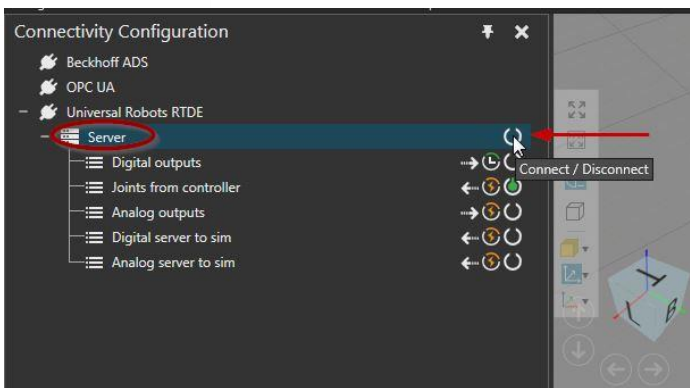
Go to Tab Connectivity > Universal Robots RTDE > Server > Edit Connection



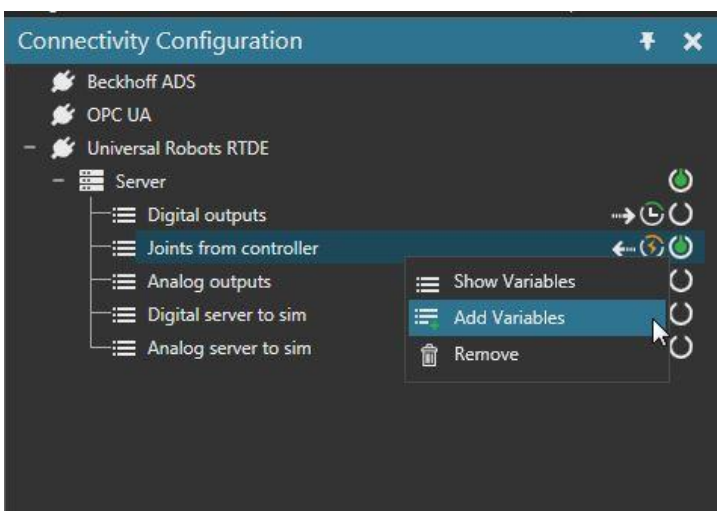
Then we set the IP address as we got from virtual machine. RTDE port number should be 30004. Then press [Test Connection], if succeeded press [Apply].



Now we need to connect the server.

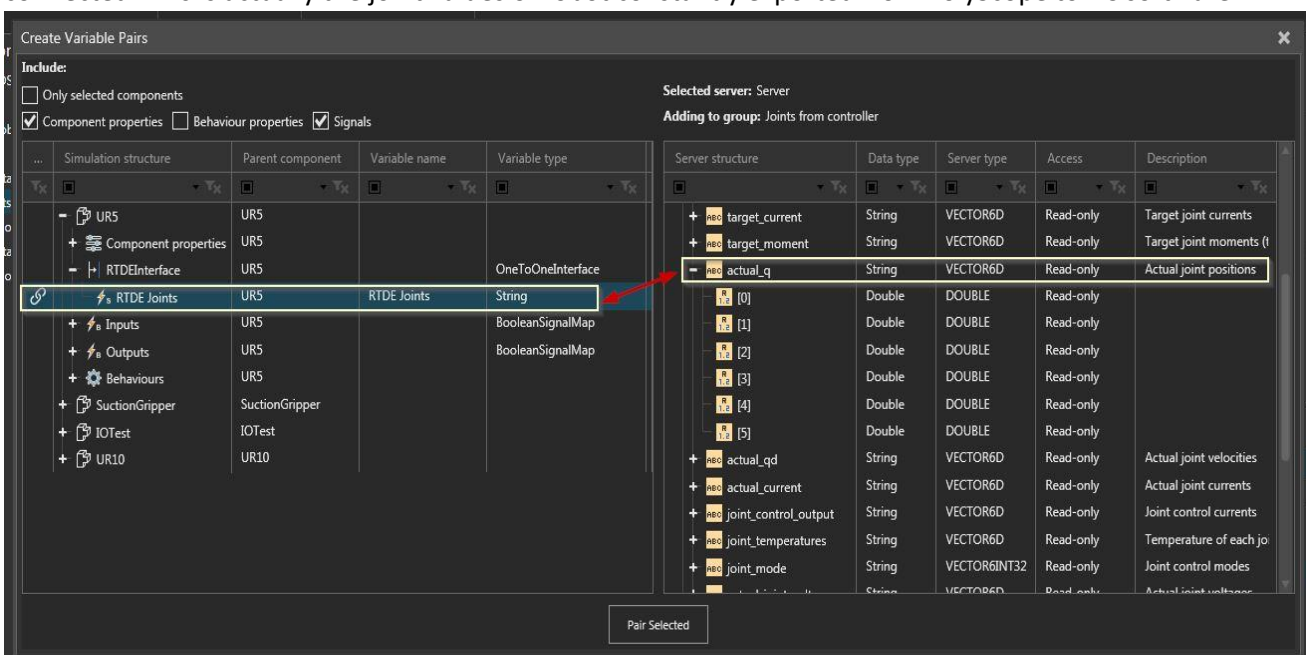


After connection established go to Server > Joints from controller > (right mouse button click) Add Variables



Now connect the Variables in between Client(VC software) and Server(PolyScope in VMware), the variables are -

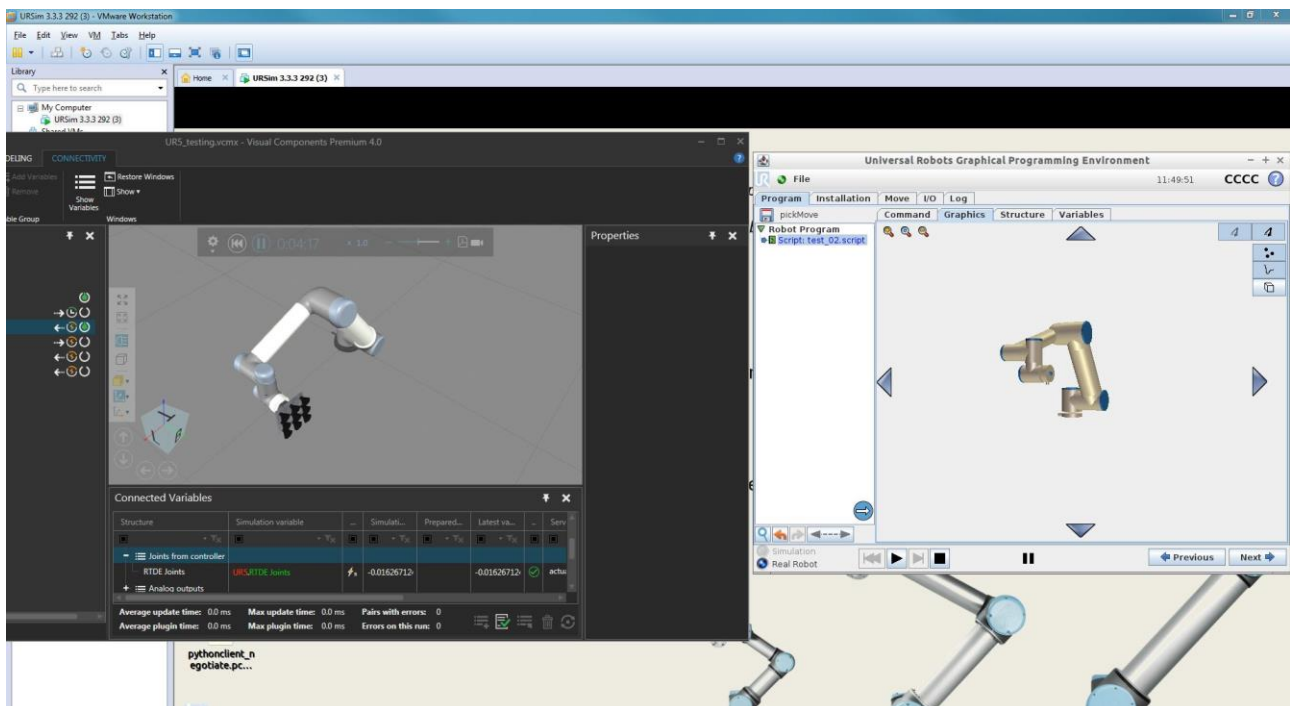
RTDE Joints(Client_String Variable) and # actual_q(Server_Vector Variable) which needs to be connected. This is actually the joint values of robot constantly exported from PolyScope to VC software.



The Connected Variables section looks like following

Connected Variables							
Structure	Simulation variable	...	Simulati...	Prepared...	Latest va...	...	Server variable
Server							
Digital outputs							
Joints from controller							
RTDE Joints	URS_RTDE Joints	⚡	-0.01626712		-0.01626712	✓	actual_q
Analog outputs							
Digital server to sim							
Analog server to sim							

Now user needs to press [Run] in both VC software and PolyScope. User will see the emulation in action.



Appendix

Universal Robots RTDE connection plugin

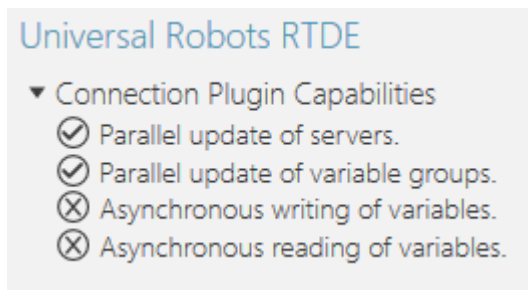
Can connect to Universal Robots' CB3-series robot controllers that have the RTDE interface. The RTDE (Real time data exchange) interface is available (and always enabled) in controllers with software version 3.2.19171 or newer*. The same RTDE interface is also available in the URSim robot controller simulator, which is provided free of charge as a virtual machine on UR support website.

*This specific control software version requirement is from UR's own sample RTDE client implementation. The support website article just says control software version 3.3 or newer.

More information about the RTDE interface:

<https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/real-time-data-exchange-rtde-guide-22229/>

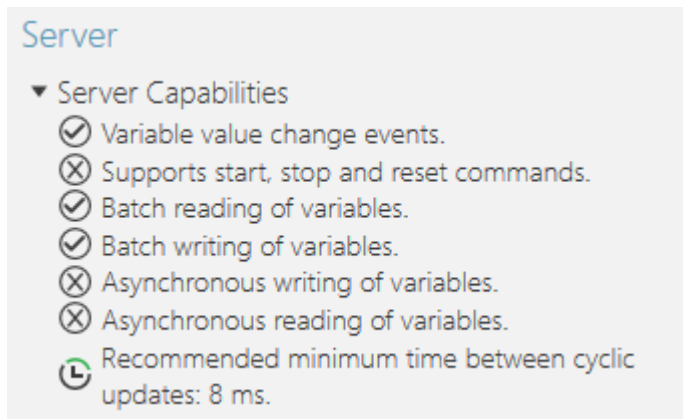
Plugin capabilities:



Universal Robots RTDE

- ▼ Connection Plugin Capabilities
 - ✓ Parallel update of servers.
 - ✓ Parallel update of variable groups.
 - ✗ Asynchronous writing of variables.
 - ✗ Asynchronous reading of variables.

Server capabilities (always the same within this plugin):



Server

- ▼ Server Capabilities
 - ✓ Variable value change events.
 - ✗ Supports start, stop and reset commands.
 - ✓ Batch reading of variables.
 - ✓ Batch writing of variables.
 - ✗ Asynchronous writing of variables.
 - ✗ Asynchronous reading of variables.
 - ⌚ Recommended minimum time between cyclic updates: 8 ms.

RTDE protocol operation principles

The RTDE interface provides a cyclic stream of value updates from the controller, and listens for inputs. The interface updates ie. sends and handles data packages at a fixed frequency. The RTDE interface 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 uses an automatically assigned (by .NET or possibly Windows) port for the socket.

The basic operation principle is that first the client configures with the server the data it wants to receive and data it wants to send. This is done in the "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 the rate it prefers. The "run" mode can also be paused by request of the client to return to the "configuration" mode.

The data packages the client and controller send to each other are defined in the “configuration” mode with input and output recipes. Input is data flow from client to controller, and 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.

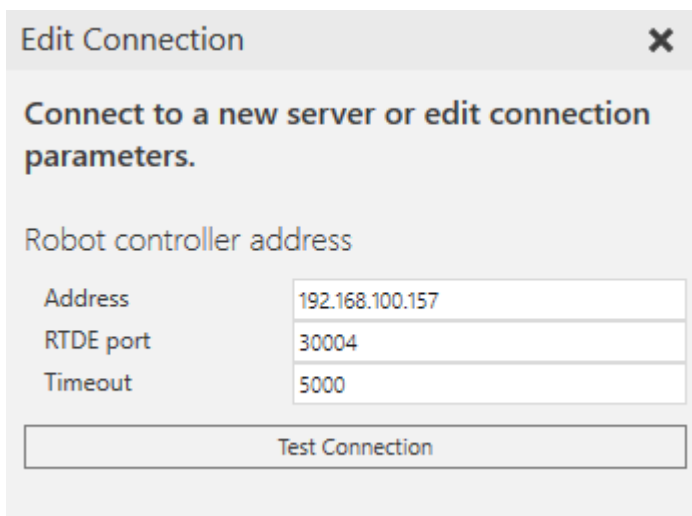
The current version of the RTDE protocol supports only a single output recipe per client, but up to 255 input recipes can be defined per client. Further, it is not possible to remove an input recipe without disconnecting, only to create new ones. This means that adding or removing a variable pair, or activating or deactivating a variable group always causes a recipe update. Since the recipe update can be only done in the “configuration” mode, the RTDE client implementation automatically requests pausing from the controller and then either redefines the output recipe shared between all variable groups or registers a new input recipe for the activated variable group.

The RTDE protocol doesn’t 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 doesn’t really work with the RTDE plugin. The times measured are only processing times to get data in or out of the cache, they don’t include the network delay or even how old the received output recipe data is when the cache is read using cyclic update mode.

Also note that since the controller handles the RTDE input recipe packages at a fixed rate (125 Hz nominal), it will miss values if sent faster than that from the simulation. The controller uses only the last received value for each variable if multiple values are received during the controller’s update cycle. The controller input variable values are retained so they don’t need to be sent at a fixed rate. Mask variables are an exception, they don’t retain their values between update cycles, see “Input mask variables”.

Connection settings

The connection settings include the robot controller’s IP address or host (dns) name, the RTDE port on the controller, and timeout. The port should always be the same default value 30004, it is included just for future proofing. The timeout (in milliseconds) is used for initial connect and synchronous operations including all configuration changes to recipes.



Robot controller address	
Address	192.168.100.157
RTDE port	30004
Timeout	5000

Test Connection

Controller address space

The RTDE interface specifies a **fixed set** of variables that each can be either read or written. The input variables of the controller can't be read by the client and output variables of the controller can't be written. However, there are some separate variables that enable reading and writing the same thing e.g. Digital outputs of the controller.

Example:

Input to controller (write-only in Connectivity feature): `standard_digital_output`

Output from controller (read-only in Connectivity feature): `actual_digital_output_bits`

The RTDE address space doesn't have hierarchy, and variables can only be read or written as a whole as part of recipe data packages. However, since the Connectivity core only handles basic data types, the RTDE connection plugin provides access to components of vector variables and single bits of integer variables. The single bit access is needed because the controller's digital inputs and outputs are only available as integers, not single Booleans. The connection plugin also provides read access (writing not supported) to vector output variables from the controller as strings. The vector's elements are converted to decimal strings, using full precision for floating point values and a period "." as decimal delimiter. The element strings are then joined with a single space character as delimiter.

The RTDE connection plugin uses a hardcoded information model of the server address space, which provides browsing capability. This hardcoded model corresponds to the UR controller version 3.3. The RTDE protocol doesn't include functionality for retrieving information about available variables on the controller. If new controller software versions add variables to the interface, those won't be accessible without updating the connection plugin.

The browsing functionality presents the variables in a more convenient hierarchical folder structure, and creates child variable nodes for vector components and bits in integers. Vectors can also contain integer variables, which can also be accessed bit by bit.

Vector variable browse example:

Server structure	Data type	Server type	Access	Description
<ul style="list-style-type: none"> Inputs Outputs <ul style="list-style-type: none"> timestamp actual_digital_input_bits actual_digital_output_bits Joint Variables <ul style="list-style-type: none"> target_q <ul style="list-style-type: none"> [0] [1] [2] [3] [4] [5] target_qd 	<ul style="list-style-type: none"> Double UInt64 UInt64 String Double Double Double Double Double Double String 	<ul style="list-style-type: none"> DOUBLE UINT64 UINT64 VECTOR6D DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE VECTOR6D 	<ul style="list-style-type: none"> Read-only Read-only Read-only Read-only Read-only Read-only Read-only Read-only Read-only Read-only Read-only 	<ul style="list-style-type: none"> Time elapsed since the controller was started [s] Current state of the digital inputs. Digital outputs Target joint positions Target joint velocities

Note that the robot RTDE interface variables use radians as unit of angle in variables such as current and target joint values. This means that they can't be directly paired to Value properties of DOF objects in the UR robot Components. As of 27.1.2017, Jamal should have updated the UR robot components in eCat to include a Python script, a string signal and an interface for updating the component's joint values from the actual_q variable on the robot controller. The script receives events from the signal, parses the value and uses immediate move to jump the robot to the received pose.

Input mask variables

Four inputs to controller (see below) also have separate mask variables. These masks act as a filter for applying the values sent by the client. The idea is that the RTDE client doesn't necessarily want to overwrite all bits in e.g. the variable that controls the robot's "standard digital outputs". This allows controlling some bits of a variable from the robot program on the controller and some from RTDE clients. However, the **RTDE interface doesn't allow more than one client to access the same controller input variable**. These masks create some limitations for using the input variables in Visual Components.

Selected server: Server				
Adding to group: Analog outputs				
Server structure	Data type	Server type	Access	Description
mask				
Inputs				
speed_slider_mask	UInt32	UINT32	Write-only	0 = don't change speed slider with this input 1 = use speed_slider_fraction to set speed slider value
standard_digital_output_mask	Byte	UINT8	Write-only	Standard digital output bit mask
configurable_digital_output_mask	Byte	UINT8	Write-only	Configurable digital output bit mask
standard_analog_output_mask	Byte	UINT8	Write-only	Standard analog output mask. Bits 0-1: standard_analog_output_0 standard_analog_output_1

Example: Conditions for modifying standard digital output state on the controller.

1. Variables standard_digital_output (UINT8) and standard_digital_output_mask (UINT8) must be registered in the same input recipe.
2. For each bit to apply (set to high or low state) in standard_digital_output, the value of standard_digital_output_mask must have a 1 in the corresponding bit. These must arrive in the same input recipe data package, meaning that setting the mask earlier or later won't result in the digital outputs actually changing on the controller. It seems that the mask is always reset to zero at the controller after processing each input data package.

Effects of those conditions for the Connectivity feature are:

1. Controller input variables with masks must have the variable and mask variable in the same variable group.
2. If event-based update mode is used, the mask should be set to a constant value, otherwise value changes could get lost due to the changes being transmitted in different input data packages.

Supported data types

The Universal Robots RTDE connection plugin supports access to all available variables. All signed and unsigned integer variables can be both paired directly or the individual bits paired as Booleans. Vectors can be read as strings, but cannot be written as a whole.

Table 1: Supported data types of the RTDE interface

RTDE type	.NET type	Remarks
UINT8	System.Byte	
UINT16	System.UInt16	
INT32	System.Int32	
UINT32	System.UInt32	VC integers are only signed 32 bit
UINT64	System.UInt64	VC integers are only signed 32 bit
DOUBLE	System.Double	
VECTOR3D	System.String	Converted to a single string by the connection plugin when read from the controller. Elements (3 x DOUBLE) can be read and written separately.
VECTOR6D	System.String	Converted to a single string by the connection plugin when read from the controller. Elements (6 x DOUBLE) can be read and written separately.
VECTOR6INT32	System.String	Converted to a single string by the connection plugin when read from the controller. Elements (3 x INT32) can be read and written separately.
VECTOR6UINT32	System.String	Converted to a single string by the connection plugin when read from the controller. Elements (6 x UINT32) can be read and written separately.

Performance

The communication delay is mostly defined by the UR controller's fixed update rate. Note that the timing functionality of the Connectivity core is not accurate with this plugin, see 0 for more info.

Using event-based update mode is recommended for variable groups with data flow from server to simulation unless the update rate is too high for the simulation to handle.

For simulation to server data flow, cyclic update mode is recommended, but setting update rate higher than 125 Hz doesn't provide any benefit. The controller uses only the last received value for each variable if multiple values are received during the controller's update cycle. Note that event-based update mode from simulation to server may send lots of updates that get lost due to the controller's update rate, and mask variables may not work as expected due to the mask and the masked value arriving in different input recipe value update packets.

Q&A

01. CONNECT POLYSCOPE TO 4.0 PRODUCT

- Does this only work with VC Premium/Robotics? (Yes/No, but with limitations)
- Only Premium and Robotics support Path statements.
- What limitations would there be for Essentials and Professional, which don't support Path statement?

Ans : The data connection works with any product that has the Connectivity feature.

02. JOGGING ROBOT

- Can you use Move tab in Polyscope to jog robot in 3D world in real-time? (Yes/No)

Ans : Yes, but the simulation needs to be running in VC for automatic updates because that's how the Connectivity feature works.

- Can you use Jog mode in 3D world to update robot in Polyscope in real-time? (Yes/No)

Ans : Not directly. A workaround would be to send joint values to general purpose registers on Polyscope and have a robot program running that drives the robot to that position. Again the simulation needs to be running on VC side for automatic updates.

03. IMPORT AND SIMULATE POLYSCOPE PROGRAM WITH 3D ROBOT

- File formats? (XML,JSON or something else)
- Table specifying what Polyscope statements are converted to in 3D robot program.
- Spec would need to account for differences in 4.0 products
- Are digital I/O imported? what happens to the analog I/O?
- I would expect a conflict with signal mappings defined in Action Script.
- If you import and export that same program, would it work the same in Polyscope or would there be lost data?

Ans : The whole point of this project is emulating UR Virtual Controller into VC Premium/Robotics/Robotics, this question is out of consideration. To my knowledge the conversion is one way only VC -> Polyscope.

04. EXPORT AND ANIMATE 3D ROBOT PROGRAM IN POLYSCOPE

- File formats? (XML,JSON or something else)

*Ans : File format is plain text URScript Programming Language. File extension *.script ; The binary *.urp format is not supported.*

- Table specifying what statements are supported and converted to in Polyscope program.
- Does it really support conversion of Set statements since in Polyscope you would always update the payload of the EOAT after grasp or release action?

Virtual Robot Controller (VRC)

The Virtual Robot Controller (VRC) Interface, that has been developed by automotive companies, robot and simulator manufacturers, enables to integrate the software of robot controllers into simulation systems via a standard interface. By this, the VRC-Interface enables controller simulation with high precision and a crosswise coupling of any controller software with any simulation system.

Offline Robot Programming (OLP)

Off-line programming (OLP) is a robot programming method where the robot program is created independent from the actual robot cell. The robot program is then uploaded to the real industrial robot for execution. In off-line programming, the robot cell is represented through a graphical 3D model in a simulator. Nowadays OLP and robotics simulator tools help robot integrators create the optimal program paths for the robot to perform a specific task. Robot movements, reachability analysis, collision and near-miss detection and cycle time reporting can be included when simulating the robot program. OLP does not interfere with production as the program for the robot is created outside the production process on an external computer. This method contradicts to the traditional on-line programming of industrial robots where the robot teach pendant is used for programming the robot manually. The time for the adoption of new programs can be cut from weeks to a single day, enabling the robotization of short-run production.

Post Processor for robot

A Post Processor defines how robot programs must be generated for a specific robot controller. A Post Processor is a key component of simulation and offline programming of industrial robots. The Post-Processor works with the off-line programming software to make sure the robot motion statement/output or program is correct for a specific robot build. The Post Processor controls the format and syntax of the program that is generated for the controller that controls a specific robot.