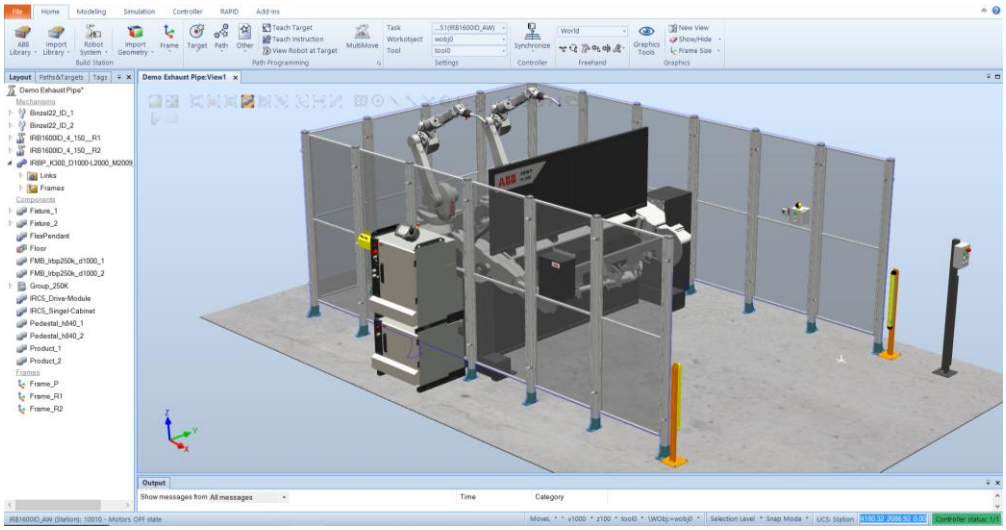
Bogdan MOCAN

Sanda TIMOFTEI Anca STAN Mircea FULEA

RobotStudio®

Simulation of industrial automation processes and offline programming of ABBs robots

- Practical guide for students -



U.T. PRESS
CLUJ-NAPOCA, 2017
ISBN 978-606-737-254-0

Editura U.T. PRESS
Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: Ing. Călin D. Câmpean

Recenzia: Prof.dr.ing. Stelian Brad
Conf.dr.ing. Emilia Brad

ISBN 978-606-737-254-0

Table of Contents

Workshop 1: Getting started with RobotStudio®

Necessary resources and knowledge

Resources

Microsoft Windows 7 SP1 (recommended) 64-bit edition

Microsoft Windows 10 (recommended) 64-bit edition

CPU: 2.0 GHz or faster processor, multiple cores recommended

Memory: 3 GB if running Windows 32-bit 8 GB or more if running Windows 64-bit (recommended)

Disk: 10+ GB free space, solid state drive (SSD)

Graphics card: High-performance, DirectX 11 compatible, gaming graphics card from any of the leading vendors. For the Advanced lightning mode Direct3D feature level 10_1 or higher is required

Knowledge

Basic knowledge about industrial robotics. Basic knowledge about using PC.

Take away lessons

- Offline programming is the best way to maximize return on investment for robot systems. RobotStudio® allows robot programming to be done on a PC in the office without shutting down production.

- RobotStudio® provides the tools to increase the profitability of a robot system by letting you perform tasks such as training, programming, and optimization without disturbing production.

- Offline programming of robotic systems facilitates:
  - Risk reduction
  - Quicker start-up
  - Shorter change-over
  - Increased productivity.

1.1. Introduction

Industrial robot (Figure 1.1) as defined by ISO 8373 is [1]: “An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications”.

Reprogrammable: whose programmed motions or auxiliary functions may be changed without physical alterations;

Multipurpose: capable of being adapted to a different application with physical alterations;

Physical alterations: alteration of the mechanical structure or control system except for changes of programming cassettes, ROMs, etc.

Axis: direction used to specify the robot motion in a linear or rotary mode

In the case of ABB industrial robots, the company has realized a software application available just for their robots - RobotStudio®.

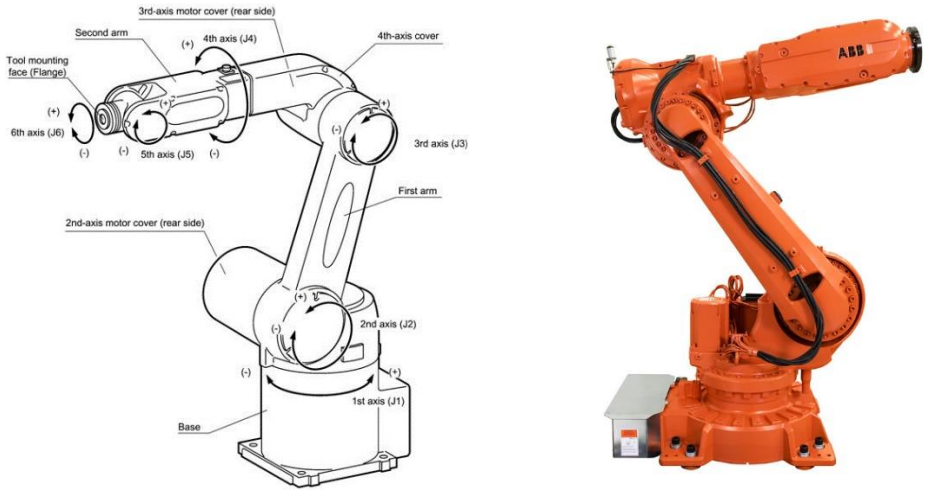


Figure 1.1. An example of industrial robot (robot manipulator)

RobotStudio® is a PC application for modelling, offline programming, and simulation of robotic cells. RobotStudio® allows you to work with an off-line controller, which is a virtual IRC5 controller running locally on your PC. This offline controller is also referred to as the virtual controller (VC).

RobotStudio® also allows you to work with the real physical IRC5 controller, which is simply referred to as the real controller [1].

When RobotStudio® is used with real controllers, it is referred to as the *online mode.* When working without being connected to a real controller, or while being connected to a virtual controller, RobotStudio® is said to be in *offline mode*.

Within this workbook, the focus will be on modelling robotic cells and offline programming with the help of RobotStudio® software application.

During installation process of the *RobotStudio®*, there are the following options [1]:

- Complete
- Custom, allowing user-customized contents and paths
- Minimal, allowing you to run RobotStudio® in online mode only.

The background of each program is a language programming specialized on the machine and the domain that is used, being impossible for it to be used for other aims. In the case of ABB robots programming, the *RAPID language* program was created. This language program is used to create different tasks for ABB industrial robots using all the information offered by the program.

Nowadays, each language programming is using English words because it is very easy to be understood by humans. RAPID is a high-level programming language [2], having at same time predefined data, instructions and so on. In this way, it is very easy to program in RAPID, even if it is an online or offline programming.

1.2. RobotStudio® - terms and concepts

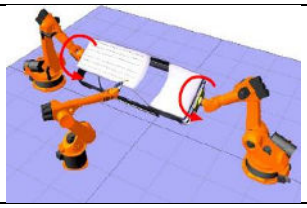
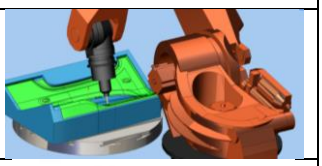
In a robotic cell, there are further hardware components that are used to work simultaneously in order to realize different tasks with an ABB industrial robot. Table 1.1. gives information about each component that is part of the robotic cell, in this case it is an IRC5 robotic cell.

Table 1.1. Standard hardware in an IRC5 robotic cell [1]

Hardware	Explanation	Examples
Robot manipulator	An ABB industrial robot.	
Control module	Contains the main computer that controls the motion of the manipulator. This includes RAPID execution and signal handling. One control module can be connected to 1-4 drive modules.	
Drive module	A module containing the electronics that power the motors of a manipulator. The drive module can contain up to nine drive units, each controlling one manipulator joint. Since the standard robot manipulators have six joints, you usually use one drive module per robot manipulator.	
FlexController	The controller cabinet for the IRC5 robots. It consists of one control module and one drive module for each robot manipulator in the system.	
FlexPendant	The programming pendant, connected to the control module. Programming on the FlexPendant is referred to as “online programming”.	
Tool	A device usually mounted on the robot manipulator to allow it to perform specific tasks, such as gripping, cutting or welding. The tool can also be stationary (not mounted on the robot; also, called “external tool”).	

To have a complete robotic cell, that will be programmed to realize a certain task that is desired to be done, further components are needed. These components, in the case of an IRC5 robotic cell, represent the optional hardware (see Table 1.2.)

Table 1.2. Optional hardware in an IRC5 robotic cell [1]

Hardware	Explanation	Examples
Track manipulator	A moving stand holding the robot manipulator to give it a larger work space. When the control module controls the motion of a track manipulator, it is referred to as a "Track External Axis".	 IRBT 4004
Positioner manipulator	A moving stand normally holding a work piece or a fixture. When the control module controls the motion of a positioned manipulator, it is referred to as an "External Axis".	 IRBP B
FlexPositioner	A second robot manipulator acting as a positioner manipulator. It is controlled by the same control module as the positioner manipulator.	
Stationary tool	A device that stands in a fixed location, the robot manipulator picks up the work piece and brings it to the device to perform specific tasks, such as gluing, grinding or welding.	
Work piece	The product being worked on.	
Fixture	A construction holding the work piece in a specific position so that the repeatability of the production can be maintained.	

1.3. RobotWare

RobotWare is a system that is used just in collaboration with RobotStudio®. This is the reason why there are further versions of the system, because it must be compatible with the RobotStudio® version [1]. *It must be mentioned that RobotWare must be installed before RobotStudio®.*

In the next table are presented all the components of a RobotWare system that are useful to work with RobotStudio® (Table 1.3.).

Table 1.3. RobotWare terminology and concepts [1]

Information	Explanation
RobotWare	As a concept, refers to both the software used to create a RobotWare System and the RobotWare systems themselves.
RobotWare installation	When installing RobotStudio®, only one version of RobotStudio® will be installed. To simulate a specific RobotWare system, the RobotWare version used for this particular RobotWare system must be installed on PC. RobotWare 5 is installed into the PC's program files folder using a standard PC installer. RobotWare 6 is automatically installed for the *Complete* installation option of RobotStudio®. Alternatively, use the RobotApps page in the Add-Ins tab to install RobotWare 6.
RobotWare key	Used when you create a new RobotWare system or upgrade an existing system. The RobotWare keys unlock the RobotWare options included in the system, and determine the RobotWare version from which the RobotWare system will be built. For IRC5 systems there are three types of RobotWare keys: ▪ The controller key (specifies the controller and software options). ▪ The drive keys, which specify the robots in the system. The system has one drive key for each robot it uses. ▪ Add-ins specify additional options, like positioned external axes.

Information	Explanation
	A virtual key allows you to select any RobotWare options you wish, but a RobotWare system created from a virtual key can only be used in a virtual environment such as RobotStudio®.
RobotWare system	A set of software files that, when loaded into a controller, enables all functions, configurations, data and programs controlling the robot system. RobotWare systems are created in the RobotStudio® software. The systems can be stored and saved on a PC, as well as on the control module. RobotWare systems can be edited by RobotStudio® or the FlexPendant.
RobotWare version	Each RobotWare is released with a major and a minor version number, separated by a dot. The RobotWare version for IRC5 is 6.xx, where xx identifies the minor version. When ABB releases a new robot model, a new RobotWare version will be released with a support for the new robot.
Mediapool	For RobotWare 5, the mediapool is a folder on the PC in which each RobotWare version is stored in a folder of its own. The files of the mediapool are used to create and implement all the different RobotWare options. Therefore, the correct RobotWare version must be installed in the mediapool when creating RobotWare systems or running them on virtual controllers.
RobotWare Add-In	A RobotWare add-in is a self-contained package that extends the functionality of a robot system. RobotWare add-ins are the RobotWare 6 equivalent of RobotWare 5 additional options.
Product	In the context of RobotWare 6, a product can be either a RobotWare version or a RobotWare add-in. Products can be free or licensed.
License	The license unlocks the options you can use in your robot system, for example robots and RobotWare options. If you wish to upgrade from RobotWare version 5.15 or earlier, you must replace the controller main computer

Information	Explanation
	and get RobotWare 6 licenses. Contact your ABB Robotics service representative at *www.abb.com/contacts*
Distribution package	A Distribution package may contain RobotWare and RobotWare add-ins. RobotWare 6 Distribution package also contains RobotWare Add-ins for Positioners and TrackMotion.

1.4. RAPID concepts

To use an ABB robot and to program it, as in all cases of industrial or automated equipment, a language programming must be known. In this case, it is about a particular programming language, used just for ABB robots, RAPID, whose structure is the general one, using the concepts presented in Table 1.4.

Table 1.4. RAPID concepts [1]

Concept	Explanation
Data declaration	Used to create instances of variables or data types, like *num* or *tooldata*.
Instruction	The actual code commands that make something happen, for example, setting data to a specific value or a robot motion. Instructions can only be created inside a routine.
Move instruction	Create the robot motions. They consist of a reference to a target specified in a data declaration along with parameters that set motion and process behaviour. If inline targets are used, the position is declared in the move instructions.
Action instruction	Instructions that perform other actions than moving the robot, such as setting data or sync properties.
Routine	Usually a set of data declarations followed by a set of instructions implementing a task. Routines can be divided into three categories: procedures, functions and trap routines.
Procedure	A set of instructions that does NOT RETURN a value.
Function	A set of instructions that that RETURN a value
Trap	A set of instructions that is triggered by an interrupt.

Concept	Explanation
Module	A set of data declarations followed by a set of routines. Modules can be saved, loaded and copied as files. Modules are divided into program modules and system modules.
Program module (.mod)	Can be loaded and unloaded during execution.
System module (.sys)	Used mainly for common system-specific data and routines, for example, an arcware system module that is common for all arc robots.
Program files (.pgf)	In IRC5 a RAPID program is a collection of modules files (.mod) and the program files (.pgf) that references all the modules files. When loading a program file, all old program modules are replaced by those referenced in the .pgf file. System modules are unaffected by program load.

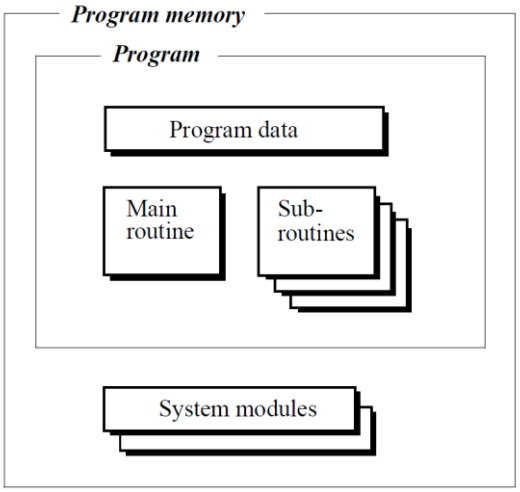


Figure 1.2. RAPID concepts and structure.

A program consists of instructions and data, programmed in the RAPID programming language (Figure 1.2), which control the robot and peripheral equipment in a specified way. The program is usually made up of three different parts:

- a main routine
- several subroutines
- program data.

In addition to this, the program memory contains system modules.

The main routine is the routine from which program execution starts.

Subroutines are used to divide the program up into smaller parts in order to obtain a modular program that is easy to read. They are "called" from the main routine or from some other routine. When a routine has been fully executed, program execution resumes at the next instruction in the calling routine.

Data is used to define positions, numeric values (registers, counters) and coordinate systems, etc. Data can be changed manually, but it can also be changed by the program; for example, to redefine a position, or to update a counter.

An *instruction* defines a specific action that is to take place when the instruction is executed; for instance, moving the robot, setting an output, changing data or jumping within the program. During program execution, the instructions are executed one at a time, in the order in which they were programmed.

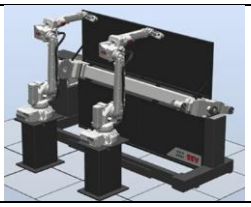
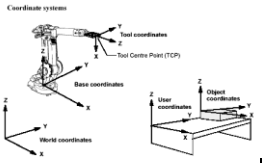
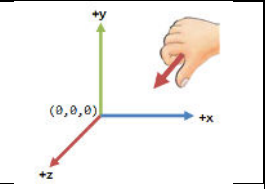
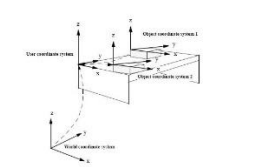
System modules are programs that are always present in the memory. Routines and data related to the installation rather than the program, such as tools and service routines, are stored in system modules.

1.5. Programming concepts

In the case of industrial robots from ABB, the programming of the robots can be realized in different ways. Types of programming and further programming concepts are presented in Table 1.5.

Table 1.5. Programming concepts [1]

Concept	Explanation	Examples
Online programming	Programming when connected to a real controller. This expression also implies using the robot to create positions and motion.	

Concept	Explanation	Examples
Offline programming	Programming without being connected to the robot or to the real controller.	
True offline programming	Refers to the ABB Robotics concept of connecting a simulation environment to a virtual controller. This enables not only program creation, but also program testing and optimizing offline.	
Virtual controller	A software that emulates a FlexController to allow the same software (the RobotWare system) that is controlling the robots to run on a PC. This gives the same behaviour of the robots offline as you get online.	
MultiMove	Running multiple robot manipulators with the same control module.	
Coordinate systems	Used to define positions and orientations. When programming a robot, you can take advantage of using different coordinate systems to more easily position objects relative to each other.	
Frame	A synonym for coordinate system.	
Workobject calibration	If all your targets refer to workobjects, you only need to calibrate the workobjects when deploying offline programs.	

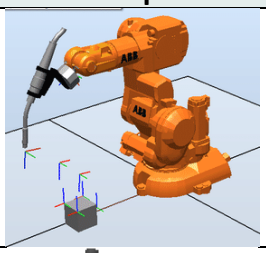
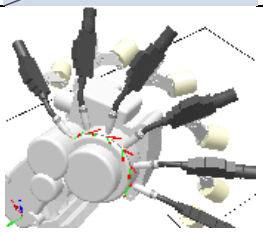
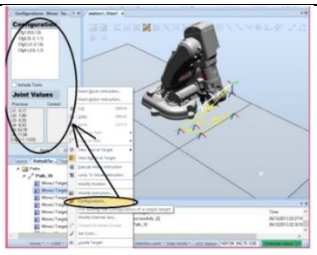
1.6. Paths and targets

In order to be able to program an industrial robot, it is necessary to know what the robot must do. For this, the ***targets*** (positions) and ***paths*** (sequences of move instructions to targets) must be known in order to be able to program the robot in RobotStudio® [1].

In RobotStudio®, there is an option to synchronize the RobotStudio® station to the virtual controller. It must be mentioned that the paths are realized in order to get RAPID programs. [1]

In RAPID programming, the **targets** are the points that must be reached by the robot. In this context, these must be saved in a data type that is recognized by the robot and then to be able to synchronize the robot with the virtual controller. A specific data must be used, and this is called *robtarget* [1]. In Table 1.6. are presented the characteristics of the targets.

Table 1.6. Targets characteristics [1]

Information	Explanation	Examples
Position	The position of the target, defined in a workobject coordinate system	
Orientation	The orientation of the target, relative to the orientation of the workobject. When the robot reaches the target, it will align the TCP's orientation with the target's orientation.	
Configuration	Configuration values that specify how the robot shall reach the target.	

The way in which the robot reached all the targets and how it moves from one point to another, represents a **path**. A path is a sequence of move instructions "used to make the robot move along a sequence of targets" [1]. Once the robot station is synchronized with the virtual controller, the paths convert into procedures [1].

In order to follow a path, the robot must move from one target to another. For this, **move instructions** are used. A move instruction is formed by:

- reference to target
- motion data (motion type, like speed and zone)
- reference to a *tooldata*
- *workobject* reference [1]

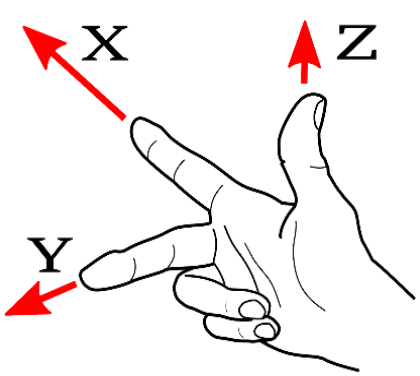
"**An action instruction** is a RAPID string that can be used for setting and changing parameters. Action instructions can be inserted before, after or between instruction targets in paths" [1].

1.7. Coordinate systems

A **coordinate system** is a system formed by one or more coordinates (numbers), to determine the position of a point or a geometric element, position, that is unique.

Generally, industrial robots use the "right hand" Cartesian coordinate system (Figure 1.3) that is, also, commonly used in manual and CNC machining and in most, if not all, CAD/CAM software applications.

In the following paragraphs, there are presented the coordinate systems used in RobotStudio®, to program the ABB robots in an offline manner. For offline programming, the systems that are already predefined (robots that are already in the RobotStudio® database) can be used: the coordinate systems are co-related hierarchically, the origin of each coordinate system is defined as a position in one of its ancestries. The following are the descriptions of the commonly used coordinate systems [1].



+x = index finger

+y = middle finger (and/or ring and pinky fingers) bent 90 degrees inward

+z = thumb sticking out

Figure 1.3. "Right hand" Cartesian coordinate system configuration

Tool Centre Point Coordinate system

The tool centre point coordinate system can also be called TCP and it represents the centre point of the tool. Different TCPs can be defined for one tool (multifunctional tool). It must be known that the implicit tool for each robot is *tool0*, which means that the tool's TCP is identical with the cartesian frame attached to the robot flange. When the robot is programmed, that means the robot moves the TCP from one point to another to reached all the programmed points. Because of this, before starting programming a robot, its TCP must be defined (Figure 1.4) [1].

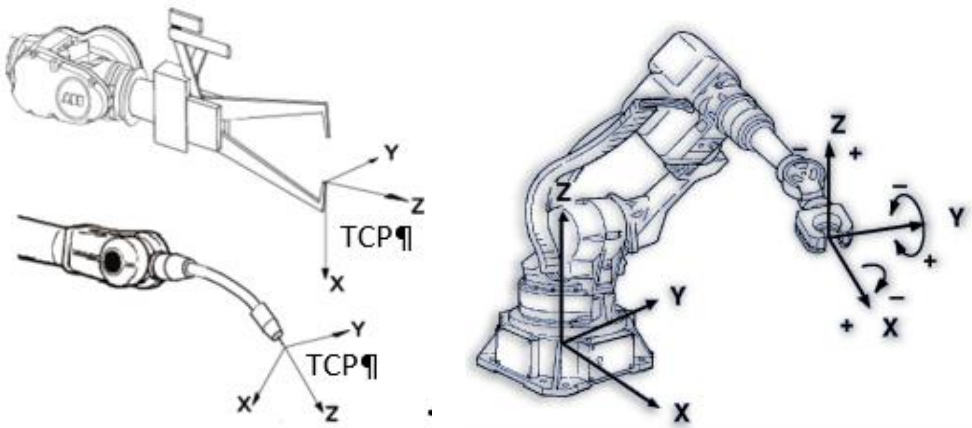


Figure 1.4. "Tool Center Point" associated to different tools

RobotStudio® World Coordinate system

When talking about the RobotStudio® world coordinate system, one refers to the entire station or robotic cell. This coordinate system is the reference for

the other coordinate systems, this being at the top of the hierarchy (when using RobotStudio®) [1].

Base Frame (BF)

The Base Frame (BF) is the base coordinate system and it has the origin at the base of the robot, whether this is about a robot in the real world or in RobotStudio® [1].

Task Frame (TF)

The origin of the robot controller world coordinate system (in RobotStudio®) represents the Task Frame. The differences between base frame and task frame are presented in Figure 1.5. In the first picture (left), the task frame and the base frame are located in the same position. In the other figure, the task frame has been moved to another position [1].

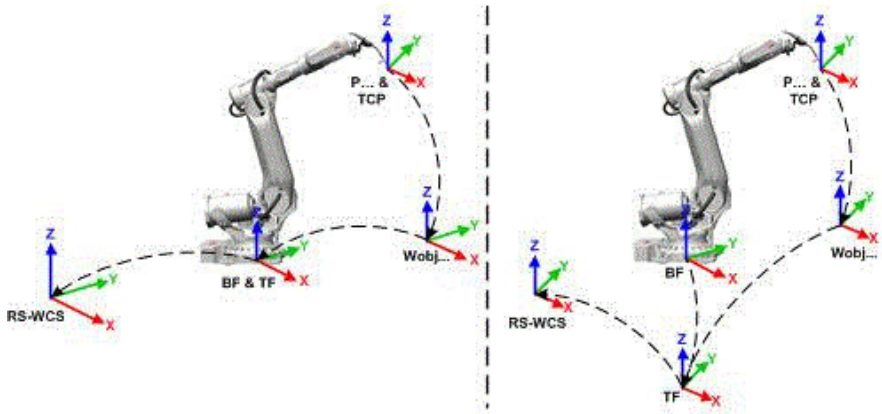


Figure 1.5. Representation for the Task Frame [1]

Figure 1.6 illustrates the mapping of the task frame in RobotStudio® to the robot controller coordinate system in the real world (e.g. on shop floor) [1].

In Table 1.7 there are presented the components' elements (regarding coordinate systems) of a station with a robot system.

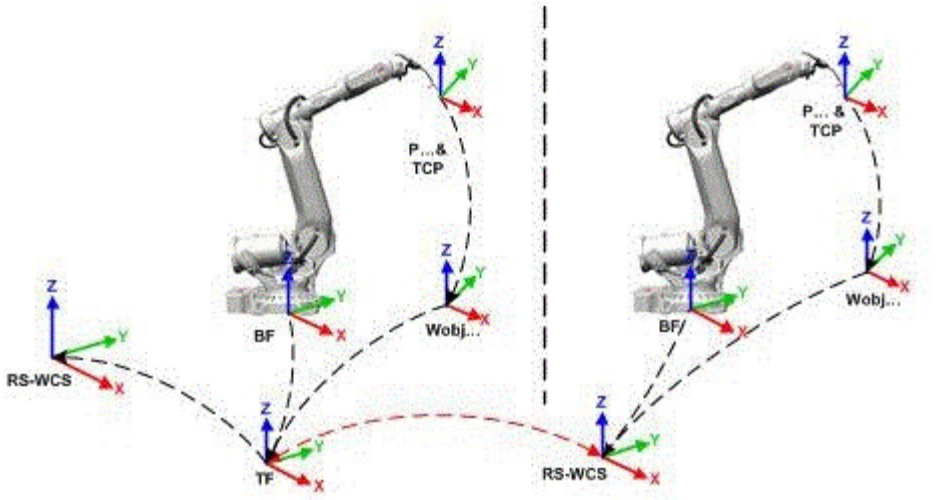


Figure 1.6. Mapping the Task Frame [1]

Table 1.7 Station with a robot systems [1]

RS-WCS	World coordinate system in RobotStudio®
RC-WCS	World coordinate system as defined in the robot controller. It corresponds to the task frame of RobotStudio®.
BF	Robot Base Frame
TCP	Tool Center Point
P	Robot target
TF	Task Frame
Wobj	Workobject

Stations with multiple robot systems

In the case of a single robot system, the task frame is the same with the robot controller world coordinate system. The presence of the task frame in the presence of several controllers, allows the connected robots to work in different coordinate systems. This means that each robot can be located independent having its own task frame (Figure 1.6) [1]. A station with multiple robot systems is presented in Figure 1.7.

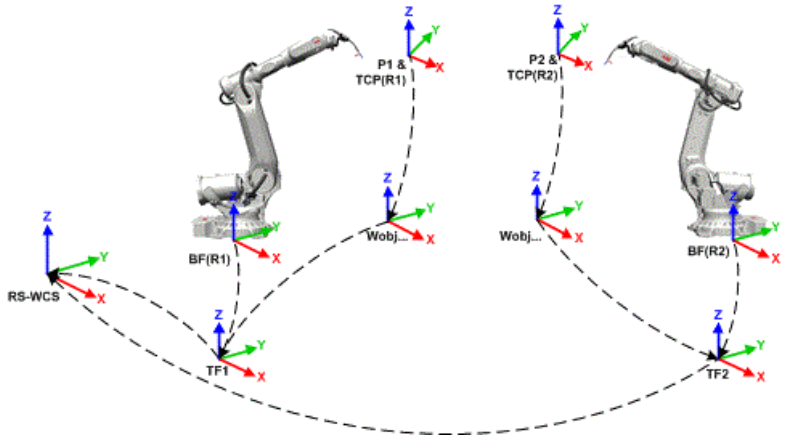


Figure 1.7. Station with multiple robot systems [1]

In Table 1.8 there are presented the components' elements (regarding coordinate systems) of a station with multiples robot systems.

Tabel 1.8 Stations with multiple robot systems [1]

RS-WCS	World coordinate system in RobotStudio®
TCP (R1)	Tool Center Point of robot 1
TCP (R2)	Tool Center Point of robot 2
BF (R1)	Robot Base Frame of robot system 1
BF (R2)	Robot Base Frame of robot system 2
P (R1)	Robot target 1
P (R2)	Robot target 2
TF (R1)	Task Frame of robot system 1
TF (R2)	Task Frame of robot system 2
Wobj	Workobject

1.8. MultiMove Coordinated systems

A function used in RobotStudio® is the MultiMove (Figures 1.8, 1.9, Table 1.9). This function helps to create and optimize programs for MultiMove systems. These types of systems consist of a robot or a position that holds the work piece and another robot that operates on it [1].

In the case a robot is using the RobotWare option MultiMove Coordinated, the robots must work in the same coordinate systems, because

“RobotStudio® does not allow task frames of the controller to be separated” [1].

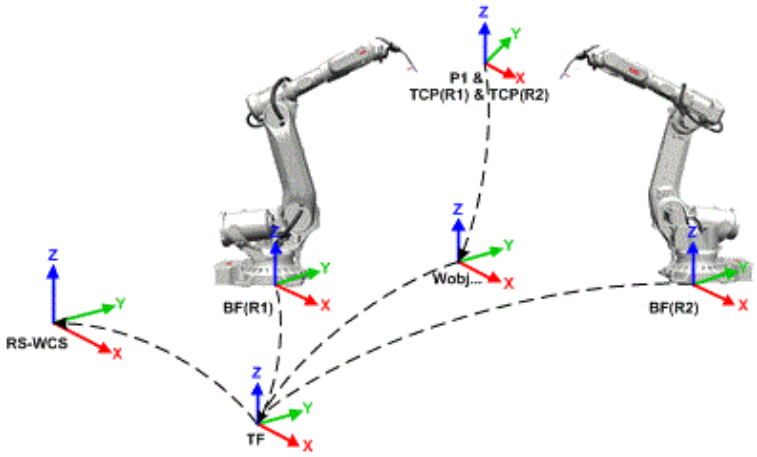


Figure 1.8. MultiMove Coordinate System (example 1) [1]

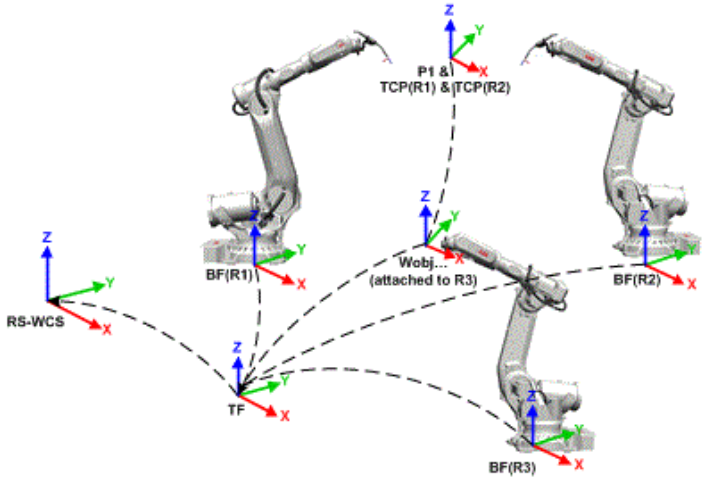


Figure 1.9. MultiMove Coordinate System (example 2) [1]

Tabel 1.9 MultiMove Coordinate System [1]

RS-WCS	World coordinate system in RobotStudio®
TCP (R1)	Tool Center Point of robot 1
TCP (R2)	Tool Center Point of robot 2
BF (R1)	Robot Base Frame of robot system 1
BF (R2)	Robot Base Frame of robot system 2
BF (R3)	Robot Base Frame of robot system 3
P1	Robot target 1

TF	Task Frame
Wobj	Workobject

MultiMove Independent systems

There is another option for a robot system with RobotWare and this is to use the MultiMove Independent option (Figure 1.10), where robots are working at the same time but independently, having the same controller [1].

The presence of one robot controller world coordinate system allows the robots to work individually in their own coordinate system. In this case, RoborStudio® offers the possibility to separate and set and independent position of the robots' task frames (Table 1.10) [1].

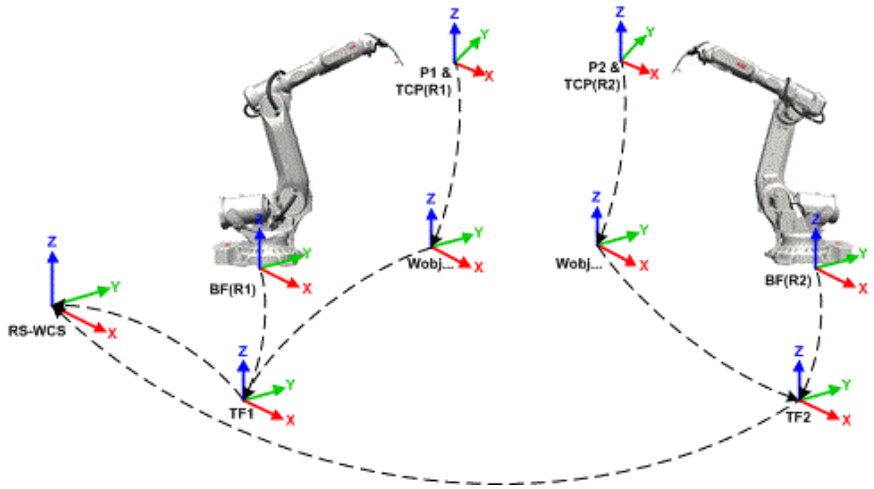


Figure 1.10. MultiMove Independent System [1]

Tabel 1.10 Stations with multiple robot systems [1]

RS-WCS	World coordinate system in RobotStudio®
TCP (R1)	Tool Center Point of robot 1
TCP (R2)	Tool Center Point of robot 2
BF (R1)	Robot Base Frame of robot system 1
BF (R2)	Robot Base Frame of robot system 2
P1	Robot target 1
P2	Robot target 2
TF (R1)	Task Frame of robot system 1
TF (R2)	Task Frame of robot system 2
Wobj	Workobject

Workobject coordinate system

The workobject is the work piece that will be moved or will be submitted to processing operations. It has two coordinate systems: User frame and Object frame, the last one being part of the first one [1].

If there is no workobject defined, the programmed targets (points) are related to a default object frame, called *wobj0*, which coincides with the base frame of the robot [1].

Defining your own workobject makes it much easier to adjust the programs from the robots, using just the offset option when the piece is moved from the initial position. In the case of offline programming, the solution is proper because even if the positions are not similar in the real world, they can be easily adjusted, particularly the workobject's position [1].

When a work piece is attached to a mechanical unit and the workobject is defined in accordance with the piece and the target is saved in accordance with the workobject, the target will be easily found in any position of the mechanical unit, just by specifying the workobject [1].

In Figure 1.11 the grey coordinate system is the world coordinate system, and the black ones are the object frame and the user frame of the *workobject*. Here the user frame is positioned at the table or fixture and the object frame of the workpiece [1].

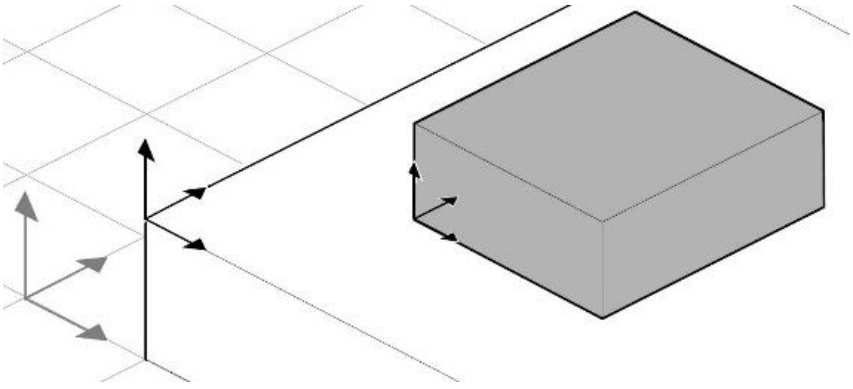


Figure 1.11. Work object system (wobj) [1]

1.9. Robot axis configurations

Axis configuration is a characteristic of the robot that defines the way in which a target is reached. When targets are saved, the action is done with respect to the workobject coordinate system. To reach a certain target, the controller is calculating the position of the robot's axis, finding, this way, different possibilities. For one target, there can exist different possible configurations (Figure 1.12). The proper configuration is defined by a value that represents the quadrant in which each axis must be located [1].

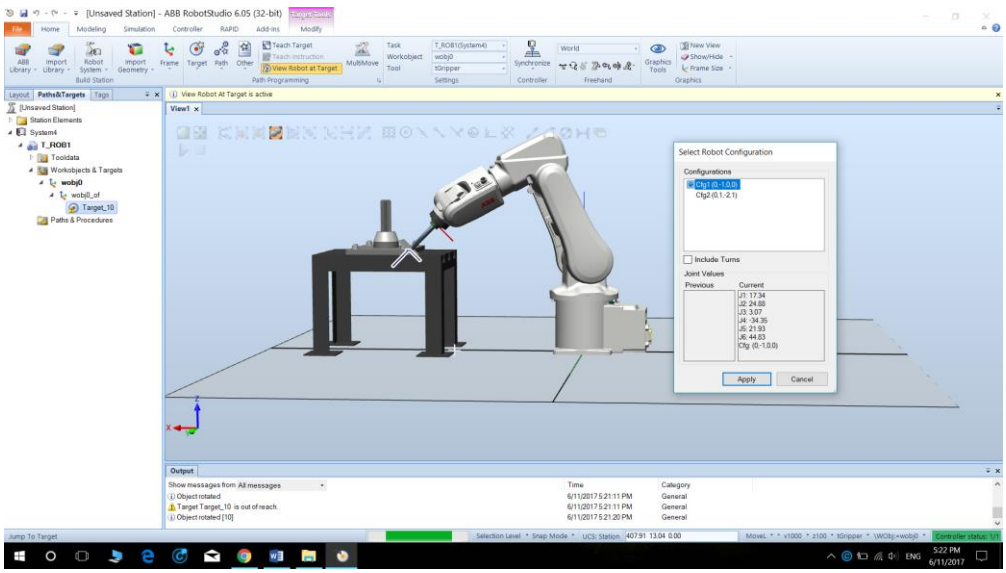


Figure 1.12. RobotStudio® - Selecting the desired robot configuration among the possible configuration

Storing axis configurations in targets

Once a configuration has been set, it is stored in the target. When it is saved, the default value is replaced with the good configuration. Default value (0,0,0,0) is invalid when a wanted target is reached [1].

Common problems related to robot axis configurations

When a configuration is created in ways other than jogging, there is a possibility that they cannot reach their default configuration [1].

In some cases, the targets in a path have validated configurations, but errors might appear when running the path. In other words, the robot cannot move from one target to another and this is because the axis shifts are greater than 90 degrees in the case of linear movements. This is possible in the case of moving targets even if the targets keep their configuration [1].

Common solutions for configuration problems

The problem described above can be solved if each configuration (Figure 1.13) is assigned to the target and it will be checked if the robot can move along the path. Another solution is to turn configuration monitoring off. In this case the configuration is done automatically. If this is not done correctly, there is the possibility to get unexpected results.

Another solution is to reposition the work piece, to reorient the targets or to add an external axis that allows the work piece or the robot to be replaced, increasing the reachability.

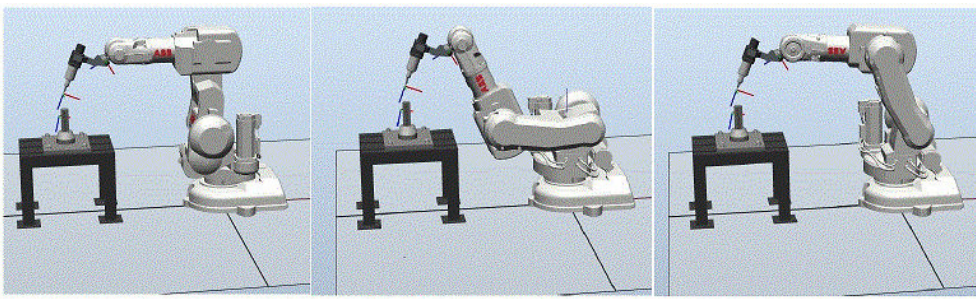


Figure 1.13. RobotStudio® - Different configurations for the same target [1]

How configurations are denoted (quaternions)

The robot's axis configurations are denoted by quaternions. A *quaternion* is a four-element vector that can be used to encode any rotation in a 3D coordinate system. Technically, a quaternion is composed of one real element and three complex elements, and it can be used for much more than rotations.

The general definition of a quaternion is given by (1):

$$Q = a + b * i + c * j + d * k = [a \;\; b \;\; c \;\; d] \qquad (1)$$

Quaternions representation (Figure 1.14): let's consider a vector $\vec{V}$ defined by 3 scalars (V_x, V_y and V_z) and θ an angle of rotation around $\vec{V}$:

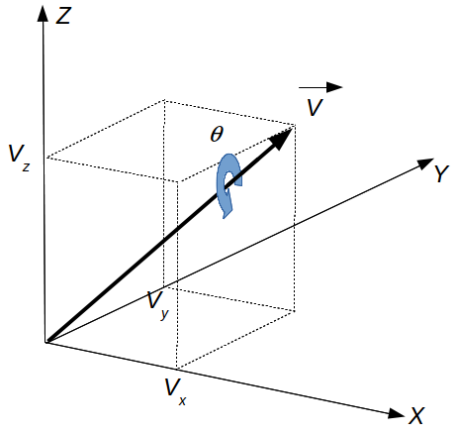


Figure 1.14. Rotation of a solids using quaternions

The quaternion associated to this transformation is given by (2):

$$Q = [\cos\frac{\theta}{2} - V_x \sin\frac{\theta}{2} - V_y \sin\frac{\theta}{2} - V_z \sin\frac{\theta}{2}] \quad (2)$$

Rotation around axes:

Based on formula 2, we can now calculate the quaternion defining a rotation around each axis:

Rotation around X (3)

$$Q_x = [\cos\frac{\theta}{2} - V_x \sin\frac{\theta}{2} \quad 0 \quad 0] \quad (3)$$

Rotation around Y (4)

$$Q = [\cos\frac{\theta}{2} \quad 0 \quad -\sin\frac{\theta}{2} \quad 0] \quad (4)$$

Rotation around Z (5)

$$Q = [\cos\frac{\theta}{2} \quad 0 \quad 0 \quad -\sin\frac{\theta}{2}] \quad (5)$$

Thus, the ABB robot's axis configurations are denoted by quaternions, a series of four integers, specifying in which quadrant of a full revolution significant axes are located. The quadrants are numbered from zero for positive (counter clockwise) rotation and from -1 for negative (clockwise) rotation.

For a linear axis, the integer specifies the range (in meters) from the neutral position in which the axis is located.

A configuration for a six-axis industrial robot (like IRB 140) may look like:

[0 -1 2 1]

- The first integer (0) specifies the position of axis 1: somewhere in the first positive quadrant (between 0 and 90 degrees' rotation).
- The second integer (-1) specifies the position of axis 4: somewhere in the first negative quadrant (between 0 and -90 degrees' rotation).
- The third integer (2) specifies the position of axis 6: somewhere in the third positive quadrant (between 180 and 270 degrees' rotation).
- The fourth integer (1) specifies the position of axis x, a virtual axis used for specifying the wrist centre in relation to other axes.

Configuration monitoring

When executing a robot program, you can choose whether to monitor configuration values or not. If configuration monitoring is turned off, configuration values stored with the targets are ignored, and the robot will use the configuration closest to its current configuration for reaching the target. If turned on, it will only use the specified configuration for reaching the targets.

Configuration monitoring can be turned off and on for joint and linear movements independently and is controlled by the *ConfJ* and *ConfL* action instructions.

Turning configuration monitoring off

Running a program without configuration monitoring may result in different configurations each time a cycle is executed: when the robot returns to the start position after completing a cycle, it may choose a different configuration then the original one.

For programs with linear move instructions this might cause a situation where the robot gets closer and closer to its joint limits and eventually will not be able to reach the target.

For programs with joint move instructions this might cause sweeping, unpredictable movements.

Turning configuration monitoring on

Running a program with configuration monitoring forces the robot to use the configurations stored with the targets. This results in predictable cycles and predictable motions. In some situations, however, like when the robot moves to a target from an unknown position, using configuration monitoring may limit the robot's reachability.

When programming offline, you must assign a configuration to each target if the program shall be executed with configuration monitoring.

Libraries, geometries and CAD files

In order to program or simulate in RobotStudio®, CAD models are needed to create the real robotic cell. These models can be imported from the libraries or geometries that exist in RobotStudio® or they can be imported as geometries. Another option is to create them in RobotStudio® [1].

Difference between geometries and libraries

The imported objects can be libraries (objects saved in RobotStudio® as external files) or geometries (CAD files, that once imported, are copied to the RobotStudio® station). When importing a library in RobotStudio®, a link is created between the station and library file. This is not happening in the same way for geometries. "For example, if a tool is saved as a library, the tool data is saved together with the CAD data" [1].

How geometries are constructed

Imported geometries are a body called Part that can be seen in the Layout browser. From RobotStudio®'s Modelling tab, each component that forms the part can be seen, even if it is solid, surface or curve [1].

In the case of **solid** bodies, it is about 3D objects realized by faces. A **surface** is a 2D object formed by just one face and a curved body is not formed by any child nodes [1].

The Modeling tab offers the possibility to edit the parts using further commands like adding, deleting unnecessary bodies, moving, rearranging or creating new bodies that will be grouped after [1].

Importing and converting CAD files

For importing geometries from single CAD files, you use RobotStudio®'s import function.

"RobotStudio® retains assembly structures in the imported CAD part. For parts with many entities, the import may take long. To work around this problem, in the Home tab, click **Import Geometry** and then select **Convert CAD geometry to single part**" [1].

Supported 3D formats

"The native 3D format of RobotStudio® is ACIS. RobotStudio® contains ACIS R25SP2 which supports later versions of its supported CAD formats. RobotStudio® also supports other formats for which you need an option. The following table shows the supported formats and the corresponding options" (Table 1.11) [1].

Table 1.11 Supported formats by RobotStudio®

Format	File extension	Option required
3DStudio	3ds	-
3DXML, reads version v4.3	.3dxml	CATIA V5

Format	File extension	Option required
ACIS, reads versions R1 - R25, writes versions V6, R10, R18 - R25	sat	-
CATIA V4, reads versions 4.1.9 to 4.2.4	model, exp	CATIA V4
CATIA V5/V6, reads versions R8 – R25 (V5 – 6 R2015), writes R16 – R25 (V5 – V6 R2015)	CATPart, CATProduct, .CGR	CATIA V5
COLLADA 1.4.1	dae	-
DXF/DWG, reads versions 2.5 - 2014	.dxf, .dwg	AutoCAD
IGES, reads up to version 5.3, writes version 5.3	igs, iges	IGES
Inventor, reads V6 – V2015	ipt	Inventor
JT, reads versions 8.0 - 9.5	.jt	JT
NX, reads versions 11 – NX 10	.prt	NT
OBJ	obj	-
Parasolid, reads versions 9.0.* – 27.0.*	.x_t, .xmt_txt, .x_b, .xmt_bin	Parasolid
Pro/E / Creo, reads versions 16 – Creo 3.0	prt, asm	Pro/ENGINEER
Solid Edge, reads versions V18 – ST7	.par, .asm, .psm	SolidEdge
SolidWorks, reads versions V18 – ST7	sldprt, .sldasm	SolidWorks
STEP, reads versions AP203 and AP214 (geometry only), writes version AP214	stp, step, p21	STEP
STL, ASCII STL supported (binary STL not supported)	stl	-

Format	File extension	Option required
VDA-FS, reads 1.0 and 2.0, writes 2.0	Vda, vdafs	VDA-FS
VRML, reads VRML2 (VRML1 not supported)	wrl, vrml, vrml2	-

1.10. Installing and licensing RobotStudio®

It is important to know that in order to install RobotStudio®, you should have administrator privileges [1].

To install RobotStudio®, further options are available:

- **Minimal** –just the features needed to program, monitor and configure a real controller that is using Ethernet connection are available
- **Complete** – all the features that are required to use all the functionalities of the program RobotStudio®. In this case, Basic and Premium functionality is available
- **Custom** – in this case just the options that the user needs are installed [1]

On a 64-bit operating system, setting installing option Complete, both 32 and 64-bit versions of RobotStudio® will be installed. Having 64-bit version, large CAD-models can be imported in order to create a robotic station. At same time, some limitations of this version are available [1]:

- "ScreenMaker, SafeMove Configurator, and EPS Wizard are not supported.
- Add-ins will be loaded from folder C:\Program Files (x86)\ABB Industrial IT\Robotics IT\RobotStudio® 6.03\Bin64\Addins" [1]

Activation the RobotStudio®

RobotStudio® has two feature levels [1]:

- Basic - offers the possibility to configure, program, and run a virtual and a real controller, the last one being connected via Ethernet;
- Premium – using this version, RobotStudio® has the functionality to simulate and program offline multiple robots. This contains Basic level,

but it requires activation that can be taken from the local ABB Robotics sales representative (www.abb.com/contacts).

To activate RobotStudio®, two types of licenses are available, a standalone license and a network license.

In the case of a **standalone license**, the activation is done using Activation Wizard. Having internet connection, the activation is done automatically, otherwise, it must be done manually. In order to do Activation Wizard, follow these steps [1]:

1. File tab > Help section
2. Manage Licenses (under Support) > Licensing options (in Options dialog)
3. Activation Wizard (under Licensing) → license options for RobotStudio®

When the computer has internet connection, an activation request is sent automatically to the ABB licensing server by the Activation Wizard. The license is installed automatically and then the program is ready to use, once the program is restarted after activation. If the computer is not connected to the internet, a manual activation must be done [1].

The activation of RobotStudio® can also be done using a **network license**, that means installing the license on a single server and not on an individual client machine. This type of license offers the possibility for further clients to use the program.

Network Licensing is done using these steps [1]:

1. Install the server for network licensing (See Installing the Network Licensing Server on page 45)
2. Activate the licenses for network licensing (See Using the SLP Server Web Interface on page 46)
3. Set up the client for network licensing (See Setting up Network Licensing in the client on page 48) [1].

Workshop 2: Introduction in RobotStudio® environment

Necessary knowledge

Workshop 1 completed.

Workshop 2 summary:

At the end of this workshop, the students should know:

- How to make the basic settings for RobotStudio® when it is launched
- How to import a robot from the library and how to change its position according to the world frame
- How to import the controller into the system and define it
- How to import a tool from the library and attach it to the robot flange

2.1. Aim of the workshop

The aim of this workshop is for the students to learn how to start using ABB RobotStudio® environment and the additional programs that it needs in order to function.

2.2. Theoretical notions

RobotStudio® is a PC application for offline programming, and simulation of robotic systems which integrates one or more robots and different auxiliary equipment. RobotStudio® allows you to work with an off-line controller, which is a virtual IRC5 controller, running locally on your PC. This offline controller is also referred to as the Virtual Controller (VC). RobotStudio® also allows you to work with the real physical IRC5 controller, which is simply referred to as the real controller.

When RobotStudio® is used with real controllers, it is referred to as the **online mode**. Working without being connected to a real controller, or while being connected to a virtual controller, RobotStudio® is said to be in **offline mode**.

RobotStudio® offers the following installation options:

- Complete
- Custom, allowing user-customized contents and paths
- Minimal, allowing you to run RobotStudio® in online mode only

All the necessary stages to create a given robot application are described, step-by-step in the next chapters and workshops. Several print screens from RobotStudio® help to understand in a better way the process of operating with RobotStudio® and of creating robot programs off-line.

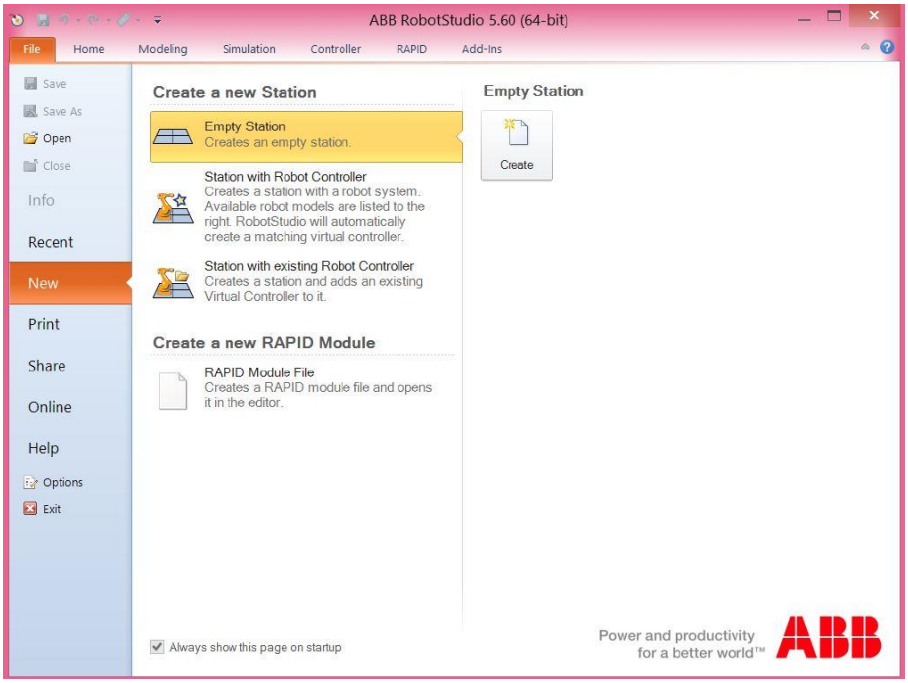


Figure 2.1. RobotStudio® - starting window

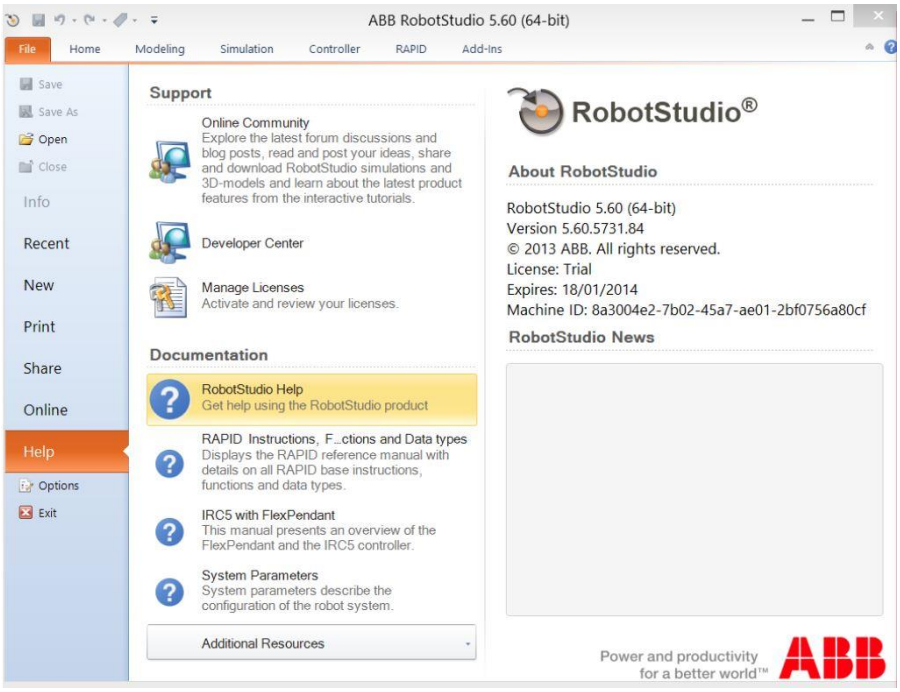


Figure 2.2. RobotStudio® - Help option

Figure 2.1 represents the window that will appear once ABB RobotStudio is launched. This window offers you the possibility to choose from further options: how to open or create a new station. Also, from this window, you have the Help option, which gives all the necessary information about RobotStudio® 5.60 version, RAPID language and the ABB teach pendant (FlexPendant) (Figure 2.2).

How to create a station in RobotStudio®?

From the starting window, using the New option, an Empty station can be created. This station only contains the working plane with a reference system (world reference system). Therefore, we need to include a robotic arm in the station. Just go to the ABB Library (Figure 2.3) and select the desired robot (robotic arm).

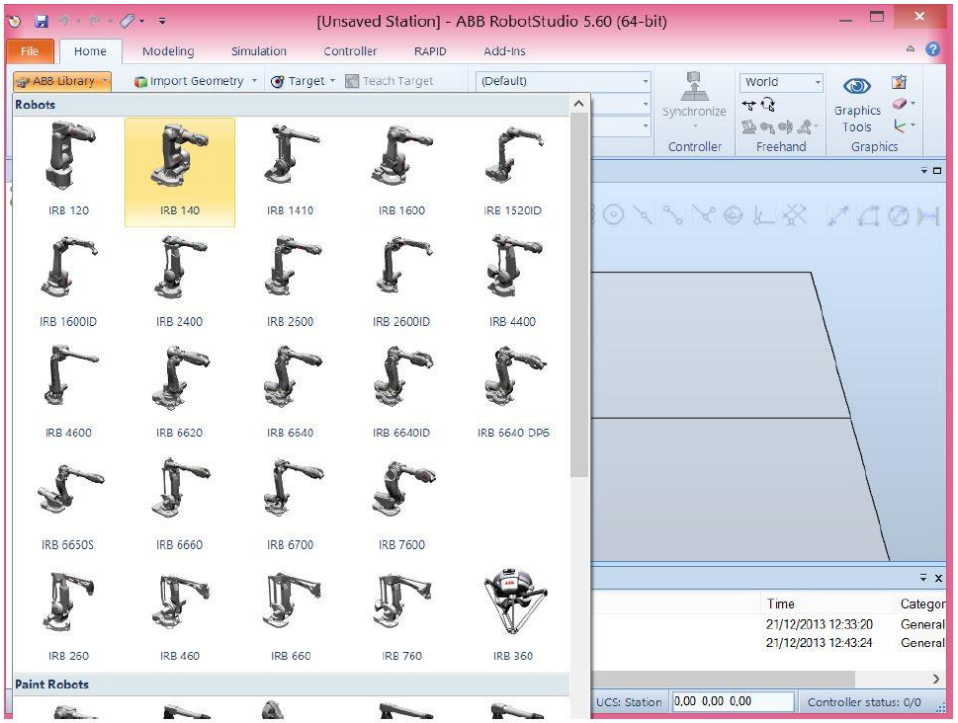


Figure 2.3. RobotStudio® – ABB Library

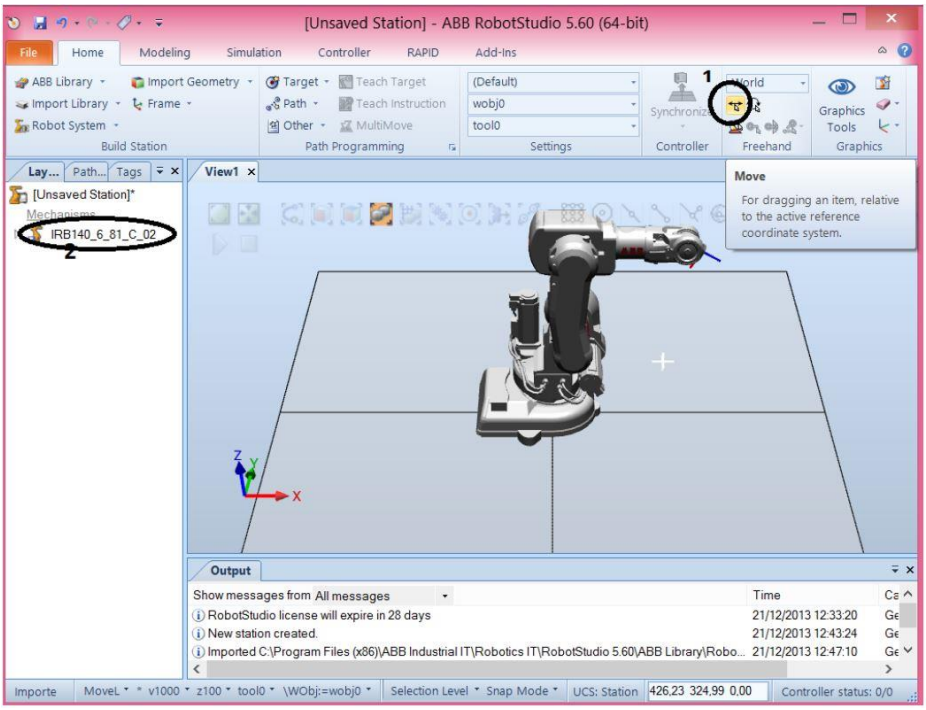


Figure 2.4. RobotStudio® - Move option of the robot

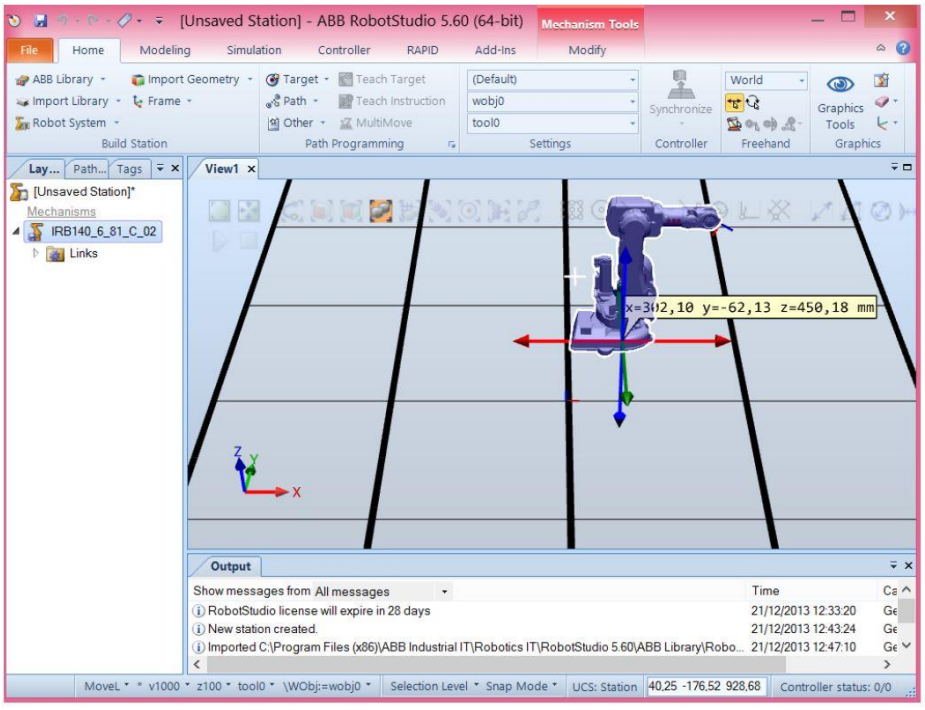


Figure 2.5. RobotStudio® - Move option of the robot (translation in the robot base)

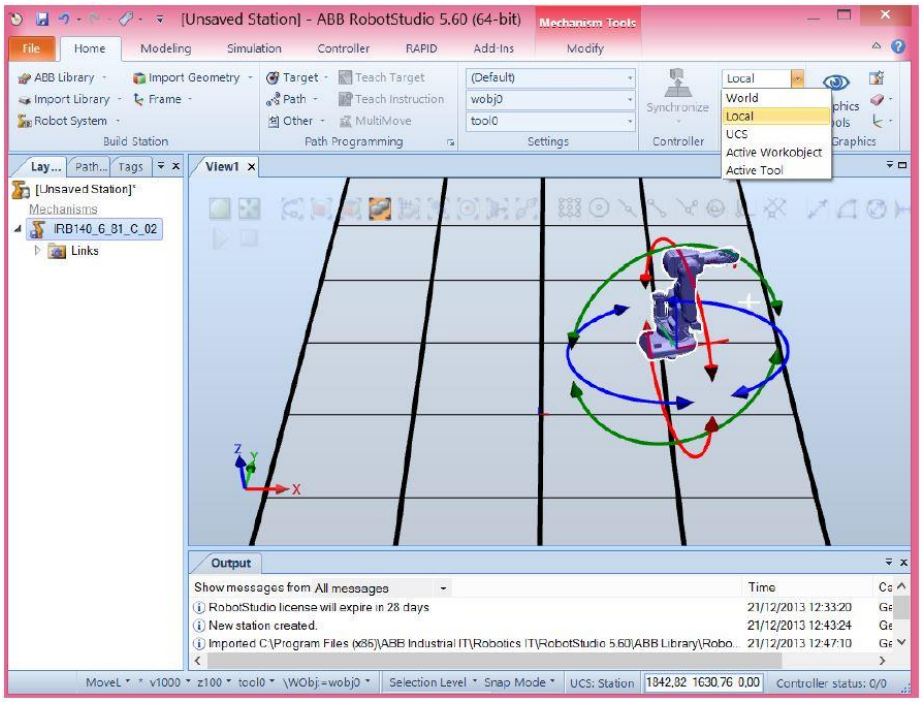


Figure 2.6. RobotStudio® - Move option of the robot (rotation in the robot base)

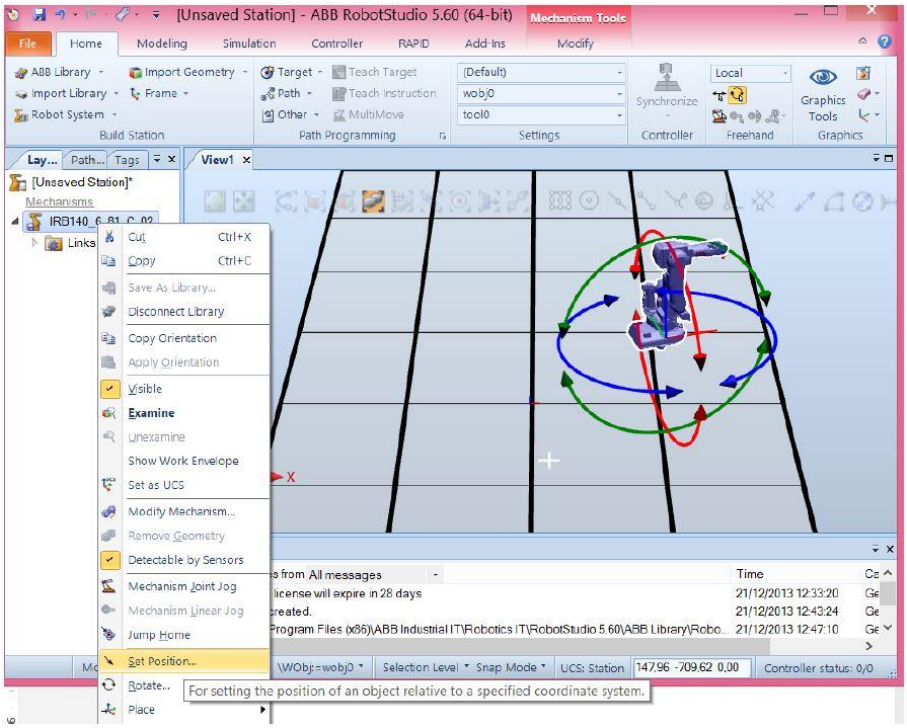


Figure 2.7. RobotStudio® - Move option of the robot with accuracy

It is now possible to move (translations + rotations) the robot in the working space. In addition, we can change the angle of each robot's joint. To do that, just click on the robot model (Figure 2.4, the highlighted area on the left). By clicking on the button move (highlighted in Figure 2.5) we can move the robot's base with the mouse (see Figure 2.6).

In Figure 2.7 it can be seen how the robot can be positioned with precision and in the next figure (Figure 2.8) the result of this movement can be observed.

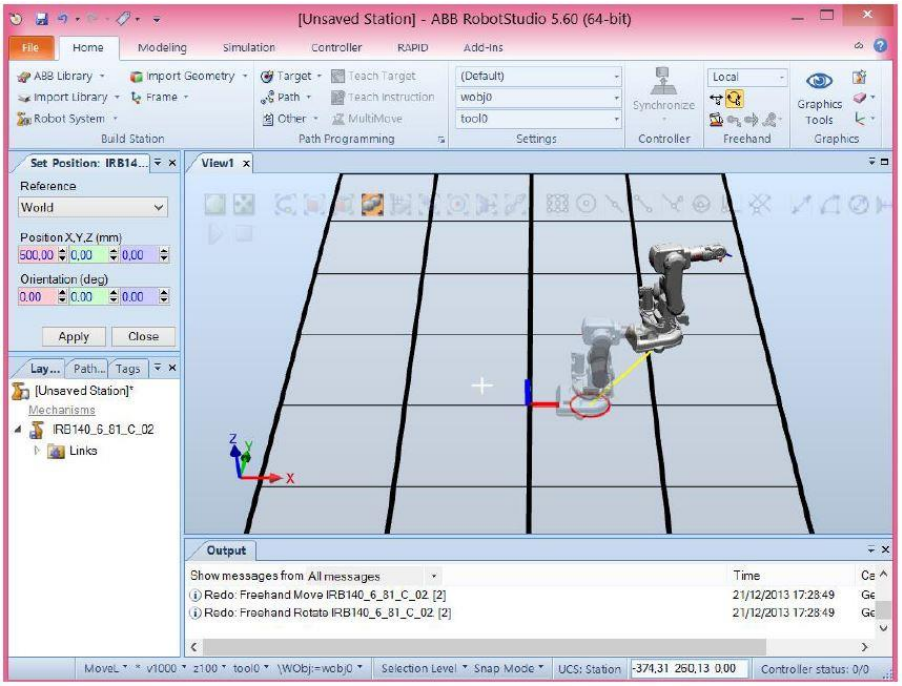


Figure 2.8. RobotStudio® - Move option of the robot with accuracy (result)

You can change the zoom of the working environment with the roller of the mouse.

Pressing CTRL and left click with the mouse you can move the scene.

Pressing CTRL and SHIFT and left click with the mouse you can rotate the scene.

An explanation on how you can change the robot's joint angles (Figure 2.9) can be observed further on, just right click on the robot model (Layout tab) and select Mechanism Joint Jog.

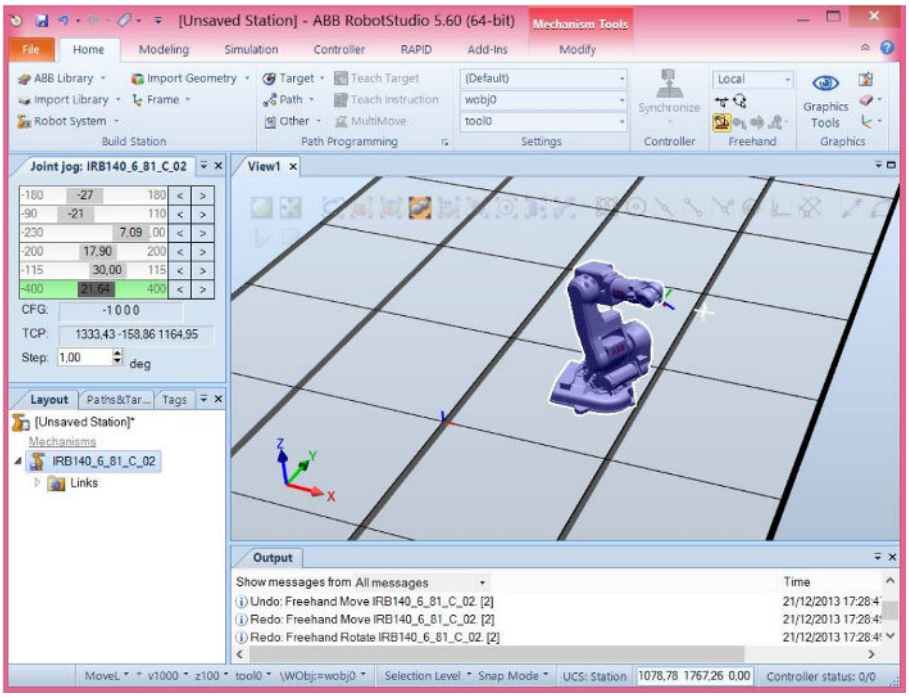


Figure 2.9. RobotStudio® - Joint jog of the robot

How to program a robot to work in RobotStudio®?

To make a robot move in RobotStudio®, like in real life, it has to be programmed. In order to program it, the robot has to have a "brain" which controls its movements. This way, it is about having a controller. When a robot is imported from library, there is no controller, just the robot as an object.

To import a controller into the system, press the Home menu button, from Robot System. From this list, you choose From Layout. The graphical view of these steps can be seen in Figure 2.10.

You can select a number of options for the controller, but, for now, the default configuration is considered to be ok. Continue by just pressing next, next and finish. Remember that this process may take a while until obtaining the green light (Figure 2.11).

The 3 buttons highlighted in the Figure 2.12 are active or can be accessed like the ones in Figure 2.13. At this moment, the virtual robot controller is ready to apply motion to the robot.

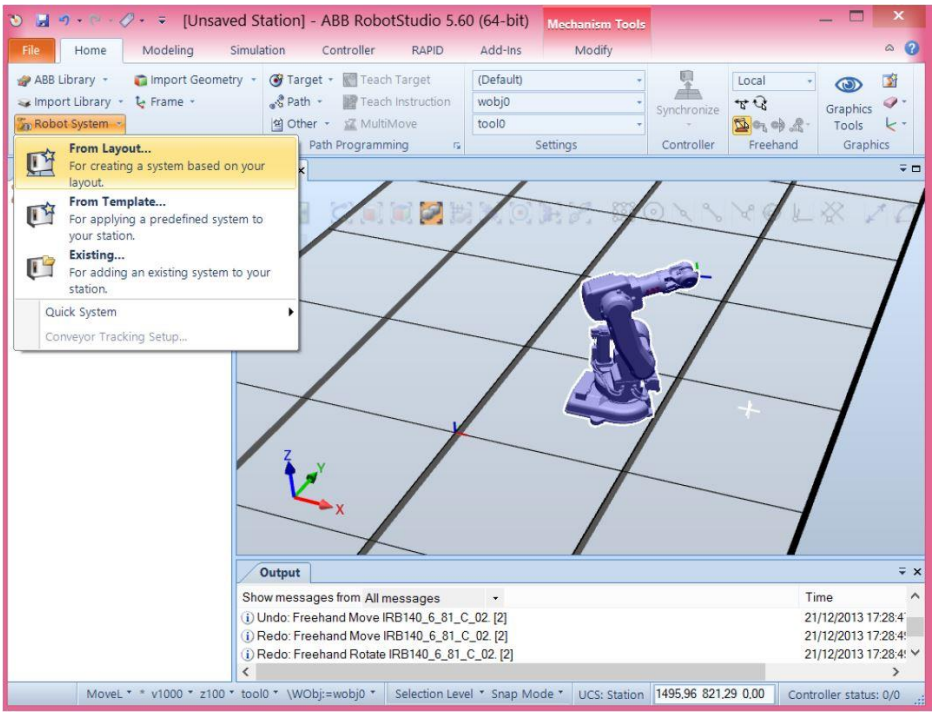


Figure 2.10. RobotStudio® - From Layout

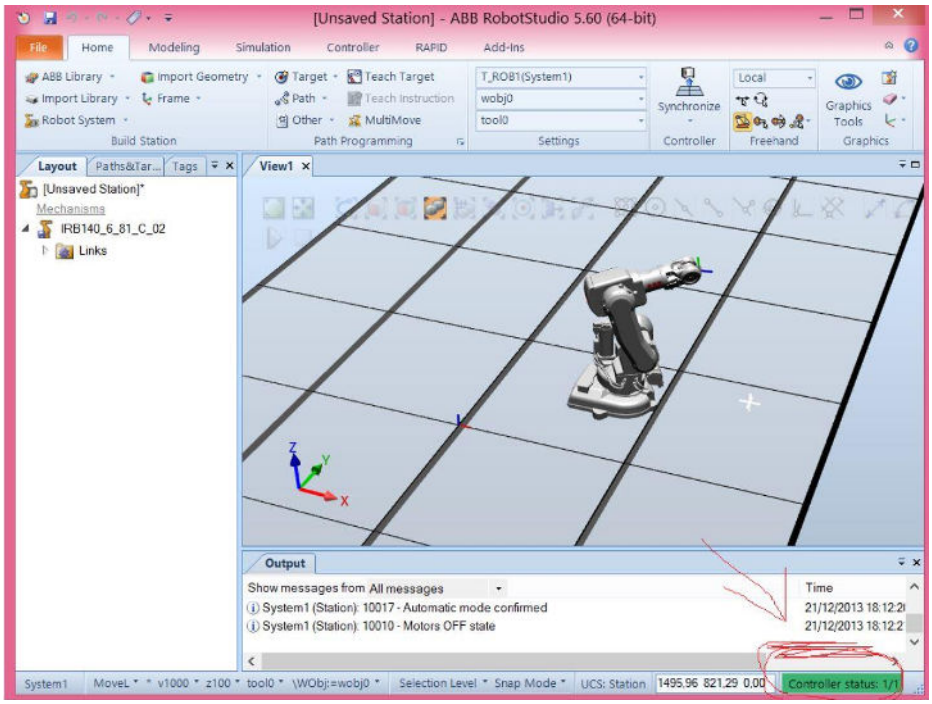


Figure 2.11. RobotStudio® – Robot controller is ready

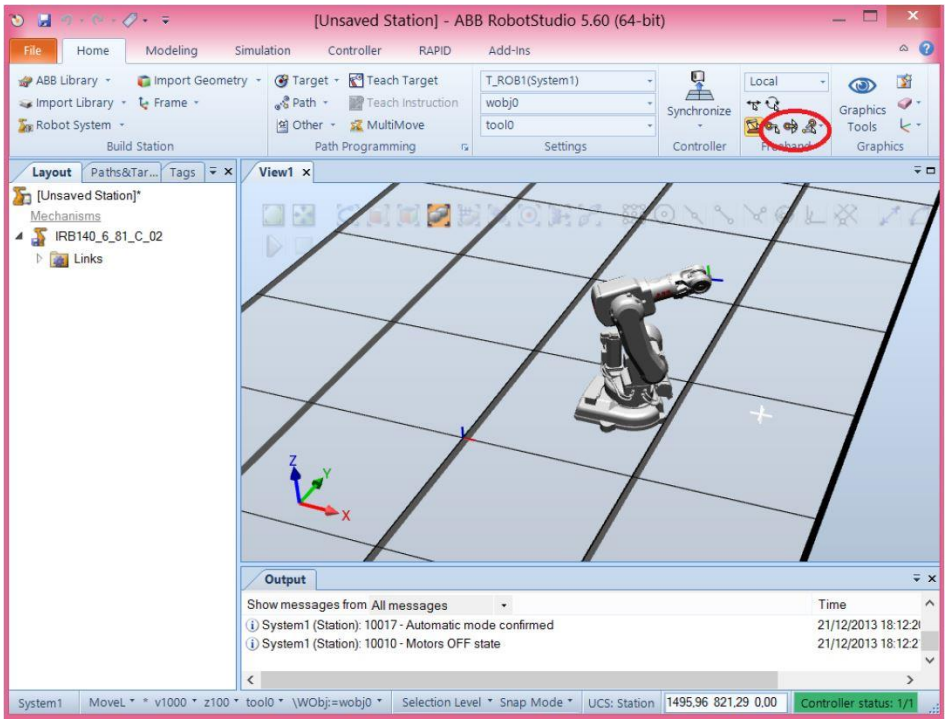


Figure 2.12. RobotStudio® – Move buttons are active

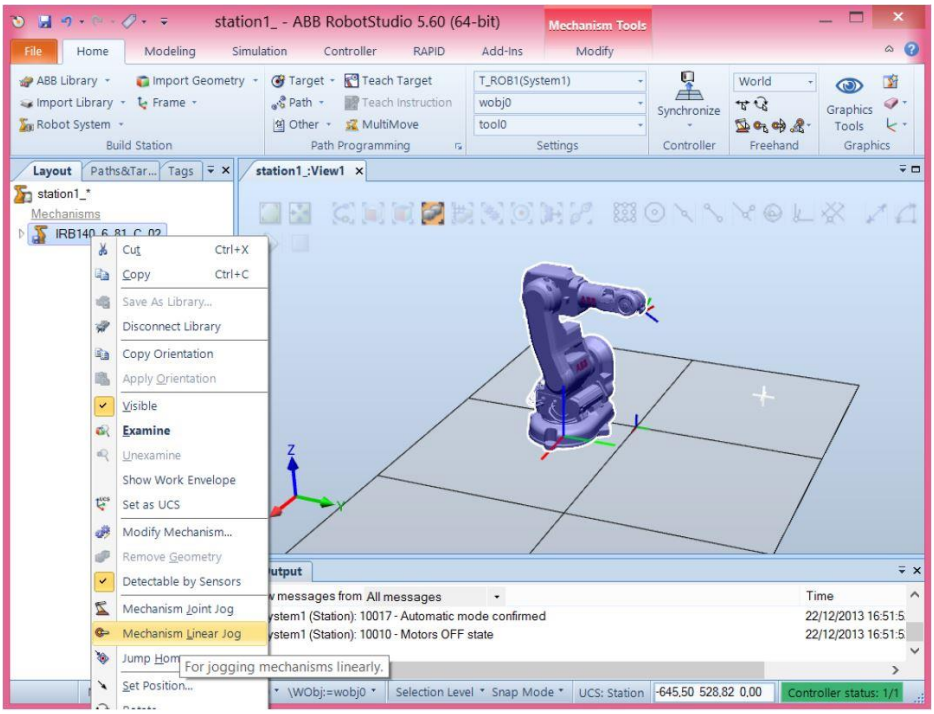


Figure 2.13. RobotStudio® – Another way to realize the robotic arm motion

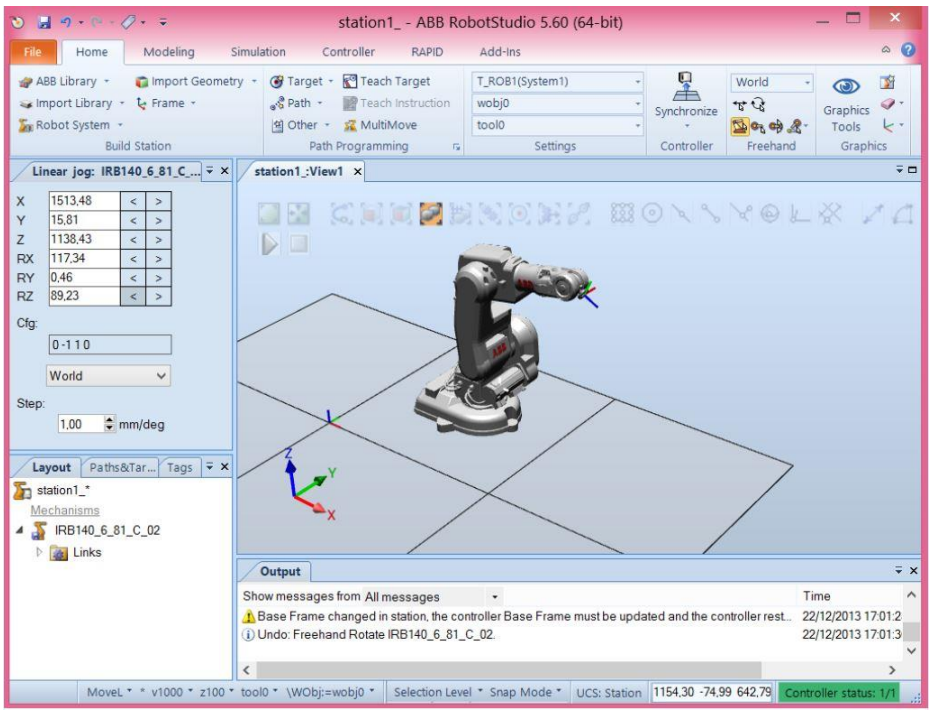


Figure 2.14. RobotStudio® – The result of the robotic arm motion

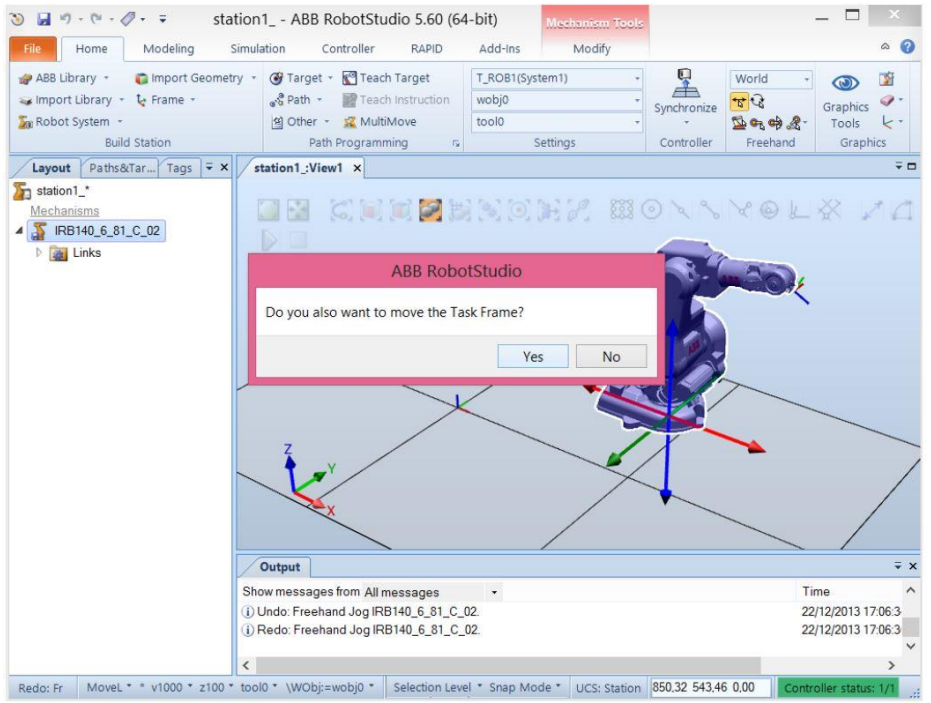


Figure 2.15. RobotStudio® – Task Frame associated to the base of the robot

Figures 2.14 and 2.15 show the result of the motion of the robotic arm and the associated task frame to the base of the robot.

Now, since we have a controller associated to the arm, if we want to move the robot's base, the software asks if we want to move the task frame associated to the base of the robot. The answer is yes.

How to import a tool in RobotStudio®?

The next step is to attach a tool to the robot's wrist. It can be imported from the library, just select, for example, a pre-defined tool named MyTool (Figure 2.16).

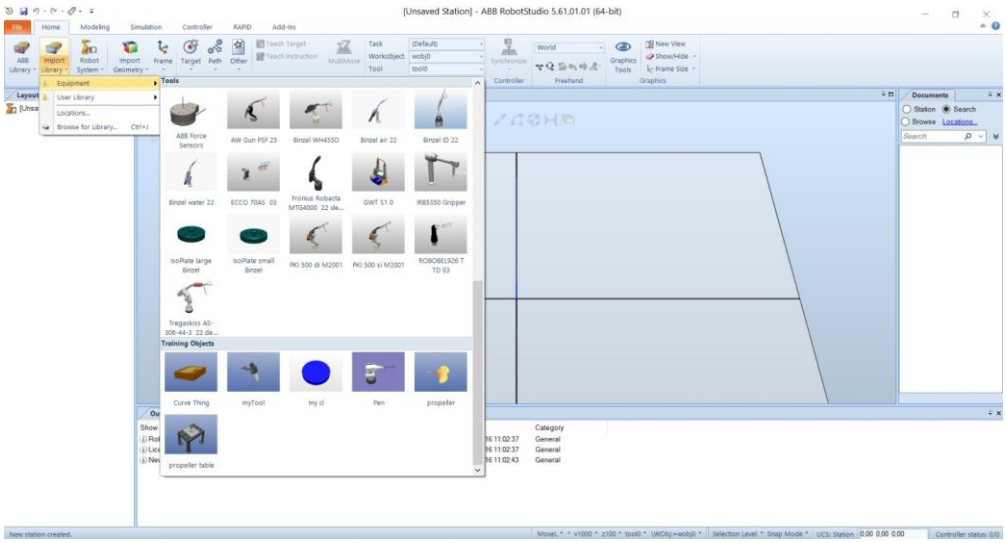


Figure 2.16. RobotStudio® – Import a tool

After selecting a tool, we have to attach that tool to the robot, and, in order to do this just drag it to the inside of the robot (Figure 2.17) or right click on the tool, in the third window, and select "Attached to" and then select the [name/type of the robot].

Now, it is possible to change the orientation of the tool while keeping the position of the "Tool Center Point" (TCP) (Figure 2.18).

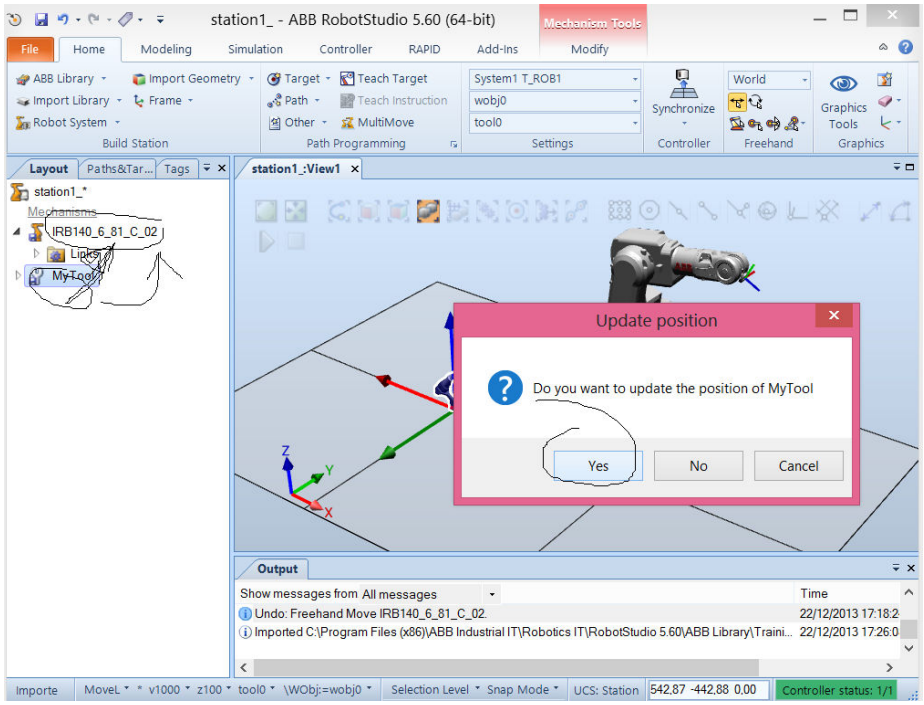


Figure 2.17. RobotStudio® – Tool position updated

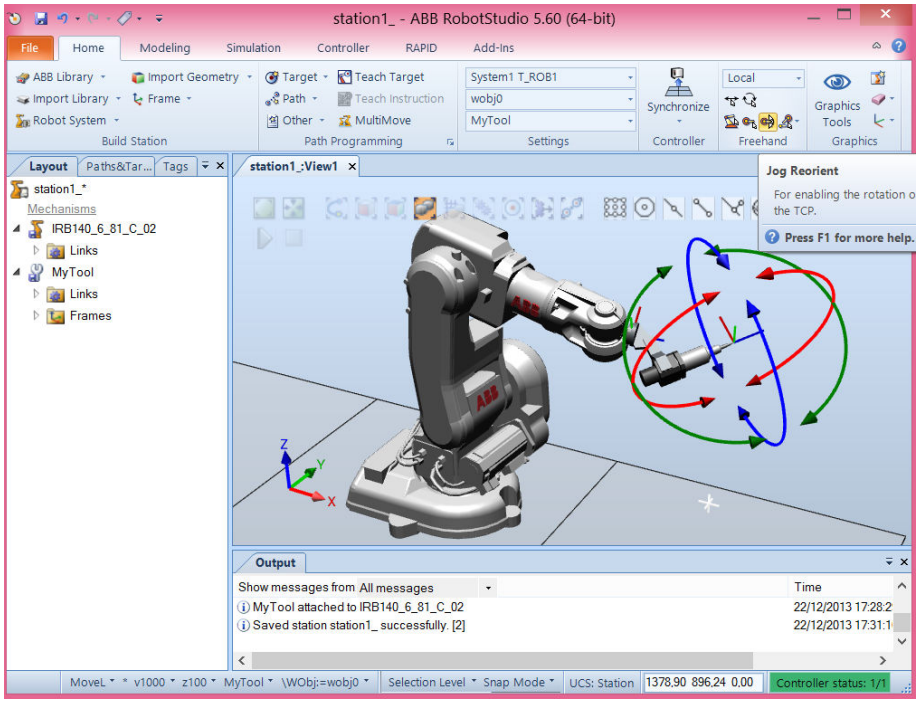


Figure 2.18. RobotStudio® – Change the orientation of the tool

Workshop 3: Define Targets and Paths (trajectories)

Necessary knowledge

Workshop 2 completed

Workshop 3 summary

At the end of this workshop, the students should know how to:

- Create and define robot targets
- Create a path with the existing targets and select the type of motion
- Modify the position and orientation of the tool in the defined targets
- Select and set the configuration of the robot in each target
- Simulate the operation that the robot will do (following a defined path)
- Save the robotic cell in order to use it on another computer

3.1. Aim of the workshop

The aim of this workshop is for the students to start learning how to program a robot to move in ABB RobotStudio®.

3.2. Robot Targets

In order to make a robot move in RobotStudio®, firstly, it has to know which points must be reached. The points that must be reached are called targets. In the next steps, you will learn how to define the *target points*. These points represent the base for the robot's *paths*.

You can create a new target manually, either by entering the position for the target in the **Create Target** dialog box, or by clicking in the graphics' window. The target will be created in the active workobject.

A ***workobject*** is a coordinate system used to describe the position of a work piece. The workobject consists of two frames: *a user frame and an object frame*. All programmed positions will be related to the object frame, which is related to the user frame, which is related to the world coordinate system.

A ***path*** is a sequence of targets (Figure 3.1) with move instructions that the robot follows. An empty path will be created in the active task.

If the work piece has curves or contours that correspond to the path to be created, you can create the paths automatically. The create the path from curve's command generate paths, complete it with targets and instructions along existing curves. The path will be created in the active task. The orientation of the targets that will be created will be according to the settings of the approach/travel vectors in the **Options** dialog box. To create a path from a curve, the curve must first be created in the station.

The points are created in relation to workobject_1 (Figure 3.2) by following the steps.

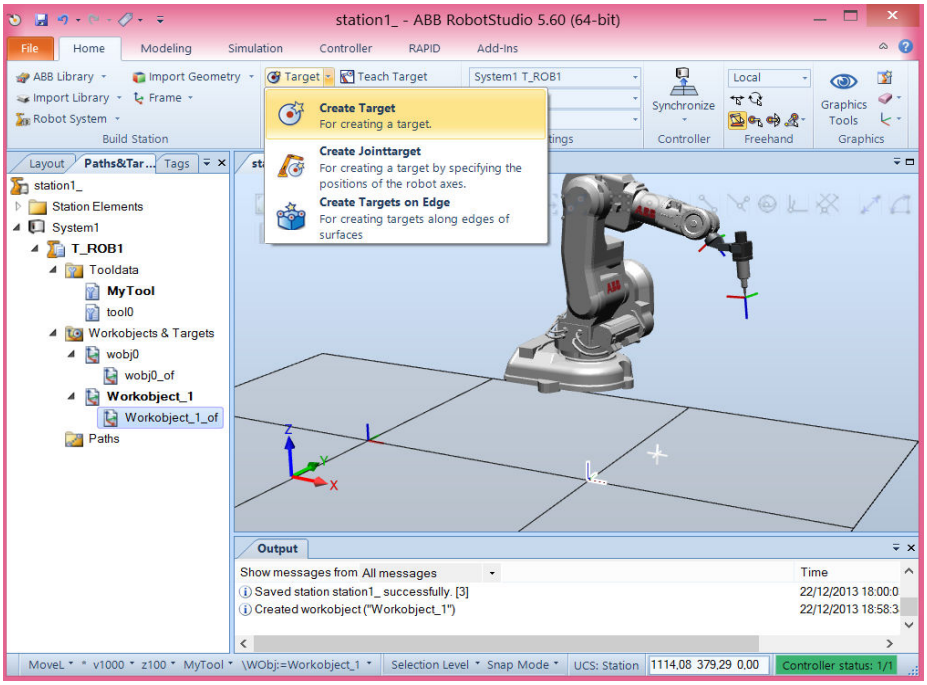


Figure 3.1. RobotStudio® - Create a target

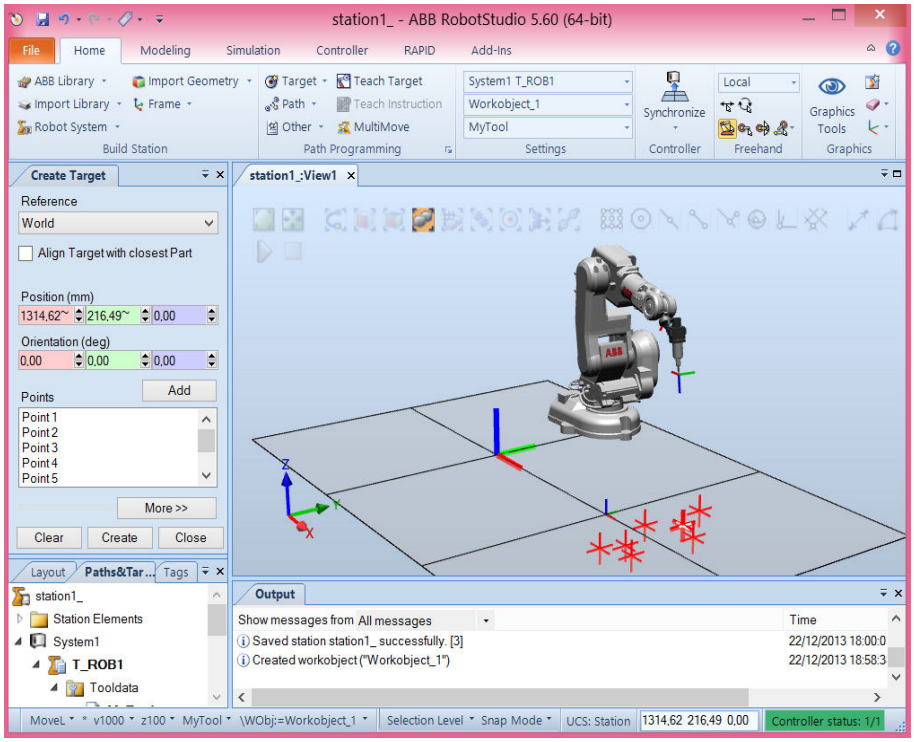


Figure 3.2. RobotStudio® - The frame in which the points are chosen

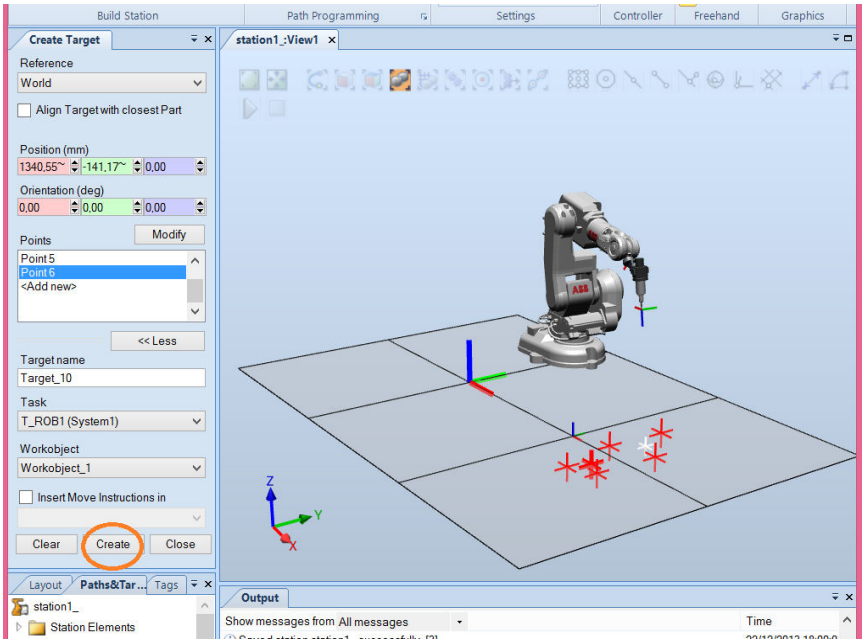


Figure 3.3. RobotStudio® - Select the targets

Now, it becomes easy to move the workobject_1 with the associated target points.

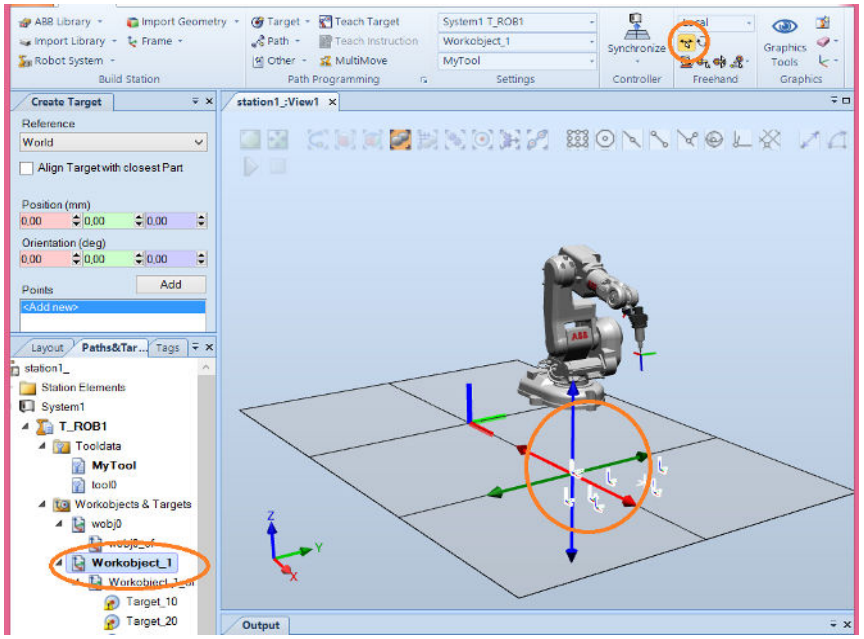


Figure 3.4. RobotStudio® - Correlation between working frame and targets

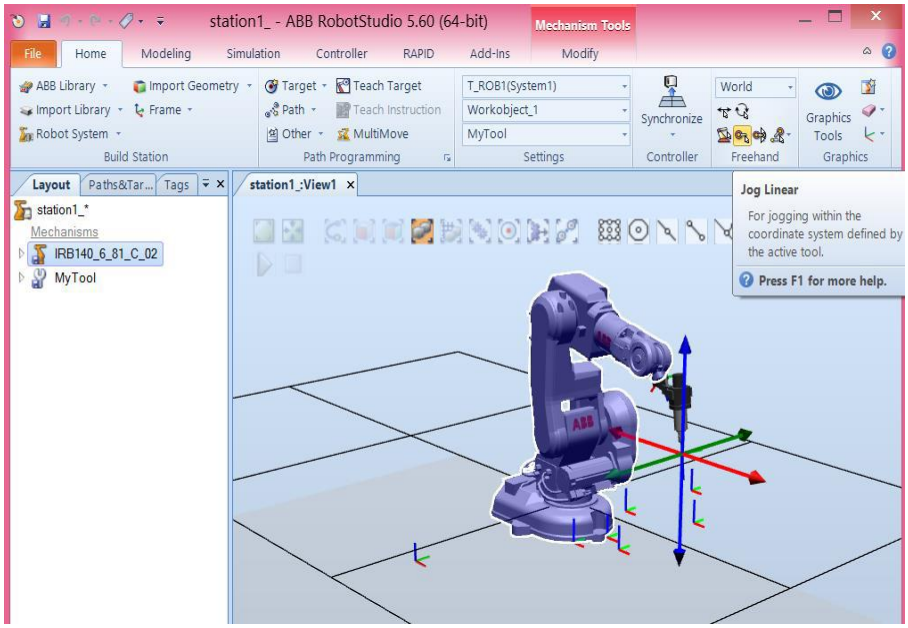


Figure 3.5. RobotStudio® - Move the robot in the desired position

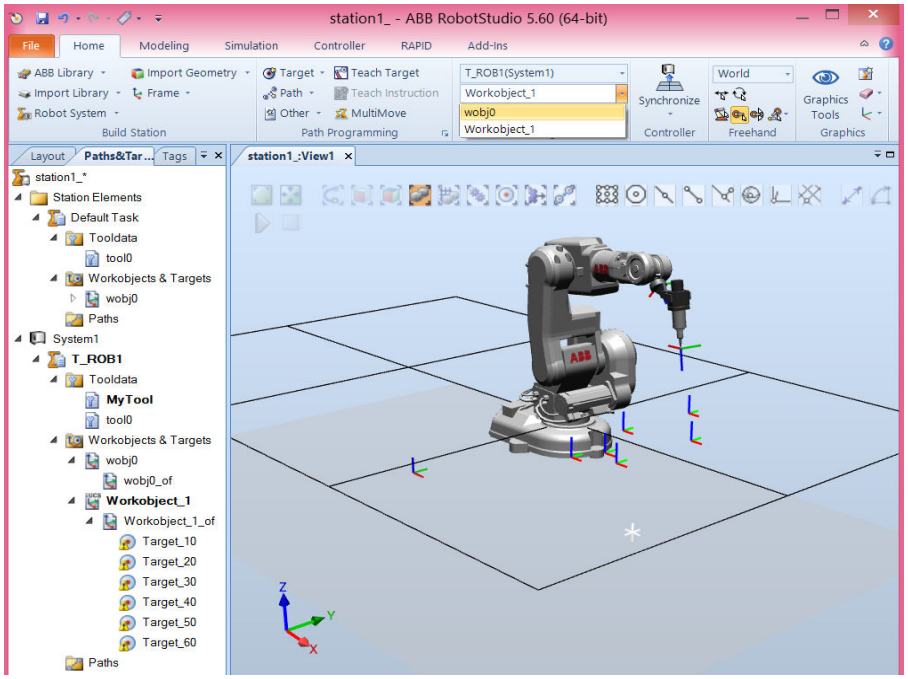


Figure 3.6. RobotStudio® - Select the workobject on which the robot is supposed to work

Answer yes (Figure 3.7) and the target is created (Figure 3.8):

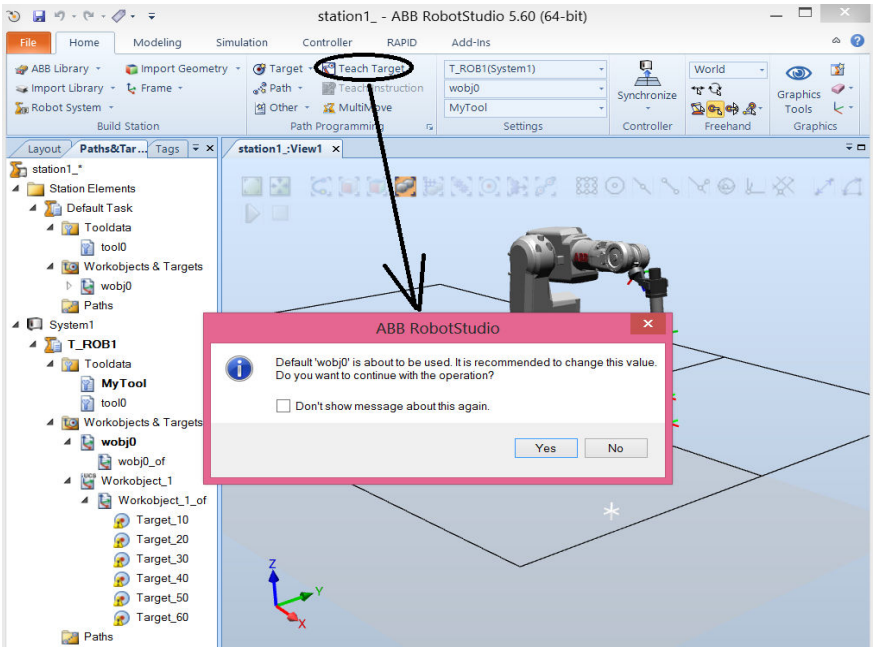


Figure 3.7. RobotStudio® - Save a target

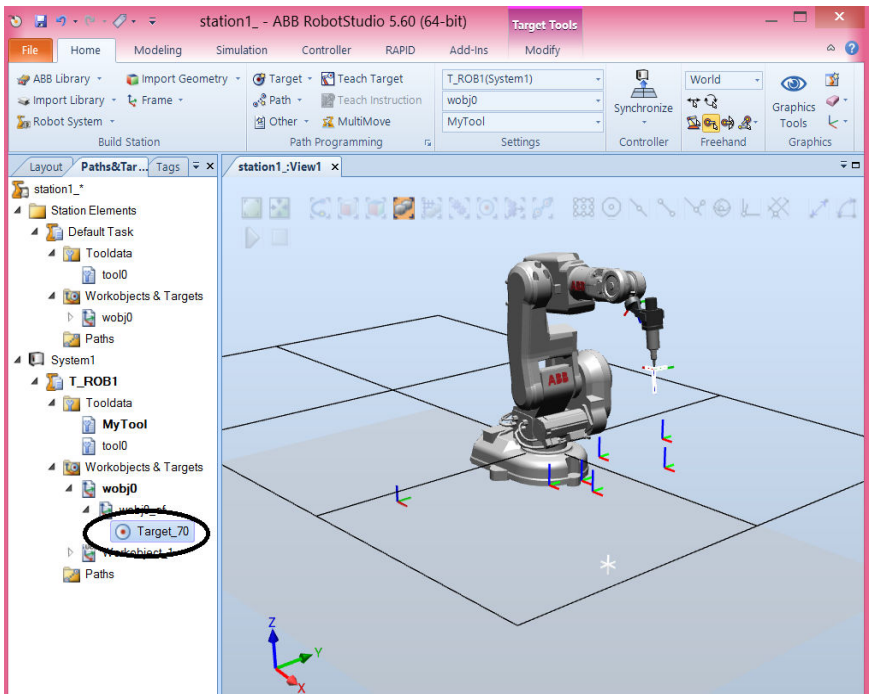


Figure 3.8. RobotStudio® - Save a target and the location where it is saved according to the chosen frame

At this moment, we don't have information on whether the robot can effectively reach the defined targets or not. However, in most of the robot

applications/ programs we usually define a home position for the robot in relation to the base of the robot, which in this case is wobj0. Therefore, you can move the robot to a desired home position (Figure 3.5), after selecting a target (Figure 3.3) and correlating the working frames to the targets (Figure 3.4).

Afterwards, create another type of target, a **Teach Target** in relation to wobj0 or another defined workobject (Figure 3.6). This creates a target according to the current position of the robot.

After this, we are going to check if the robot reaches or not the previously defined target point, starting with the following tool: *MyTool* (Figure 3.9).

It can easily be observed that the tool has the wrong orientation (Figure 3.10) and the robot is not able to reach that point with the desired orientation (Figure 3.12). The robot remains in the initial position until the orientation of the target point (Figure 3.13) is changed until a robot configuration is found, to allow the robot to reach the defined target point (Figure 3.14).

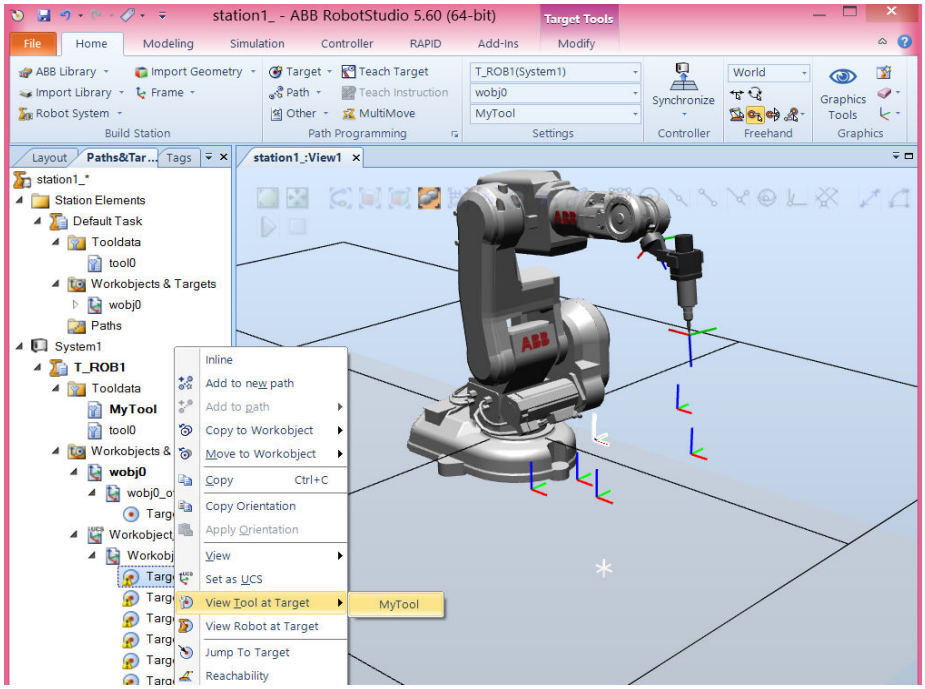


Figure 3.9. RobotStudio® - Check if the robot reaches the target

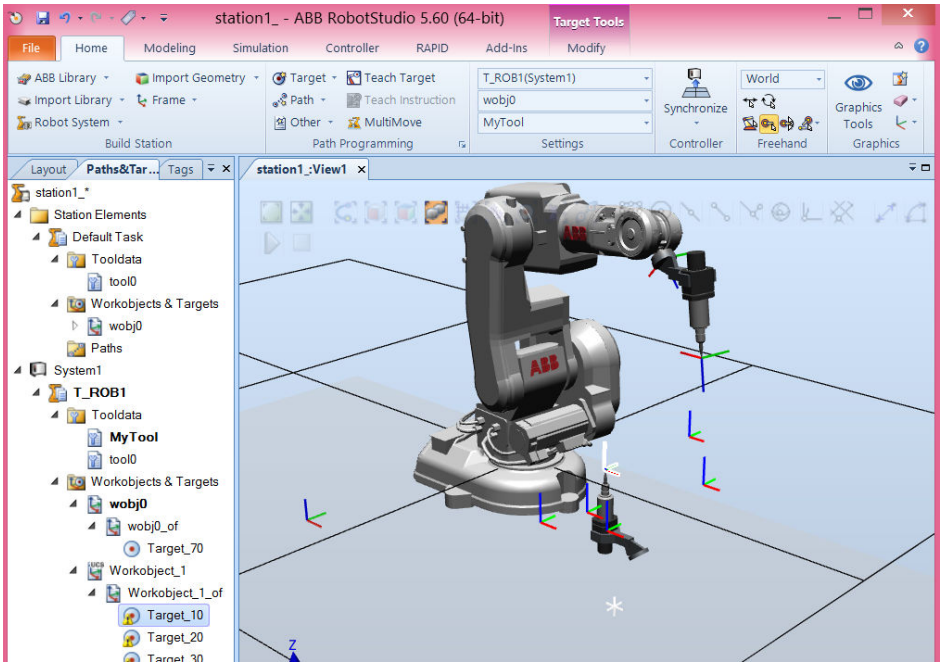


Figure 3.10. RobotStudio® - Check if the robot reaches the target (position of the tool)

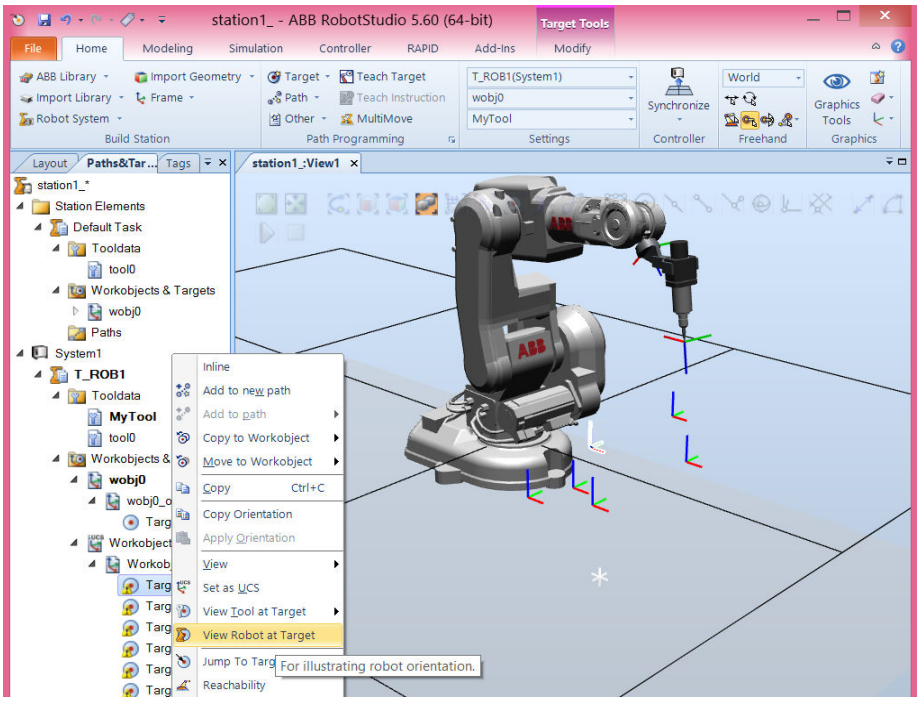


Figure 3.11. RobotStudio® - The position of the robot in the target

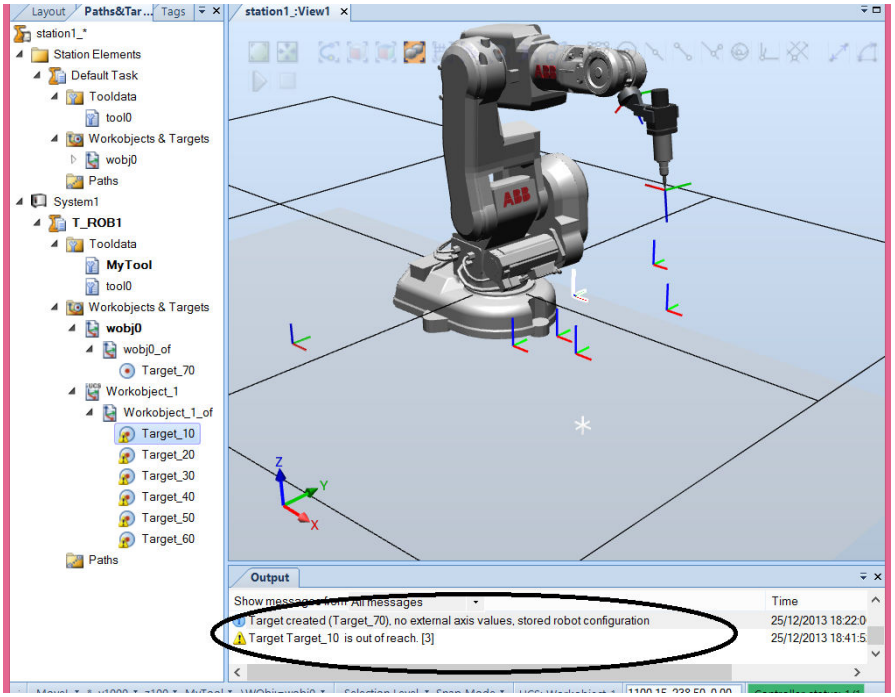


Figure 3.12. RobotStudio® - Robot cannot reach that point – an error in the Output window is listed

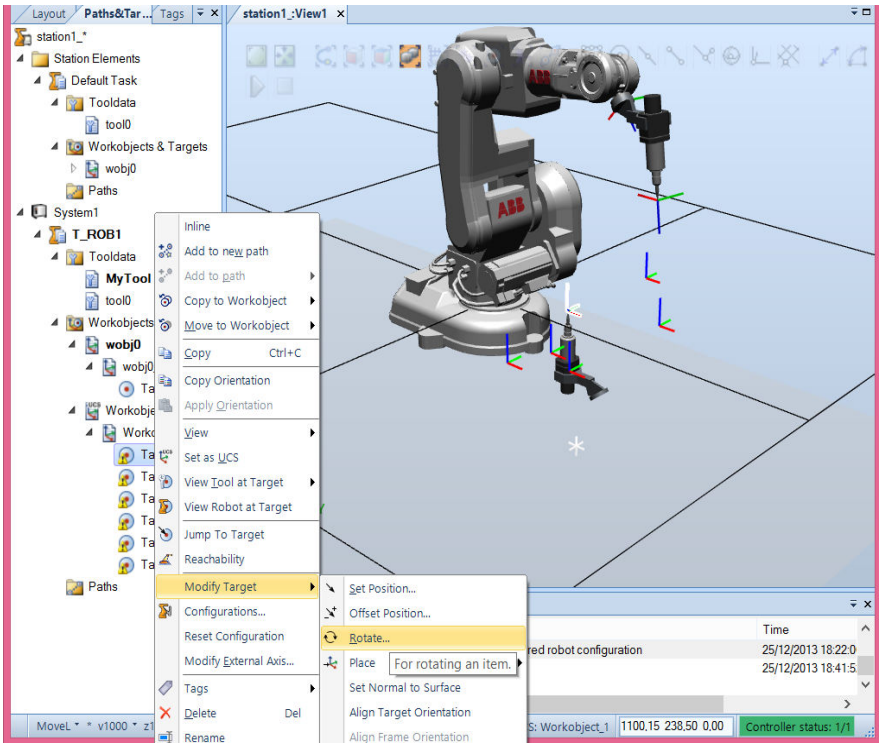


Figure 3.13. RobotStudio® - Changing the orientation of the tool

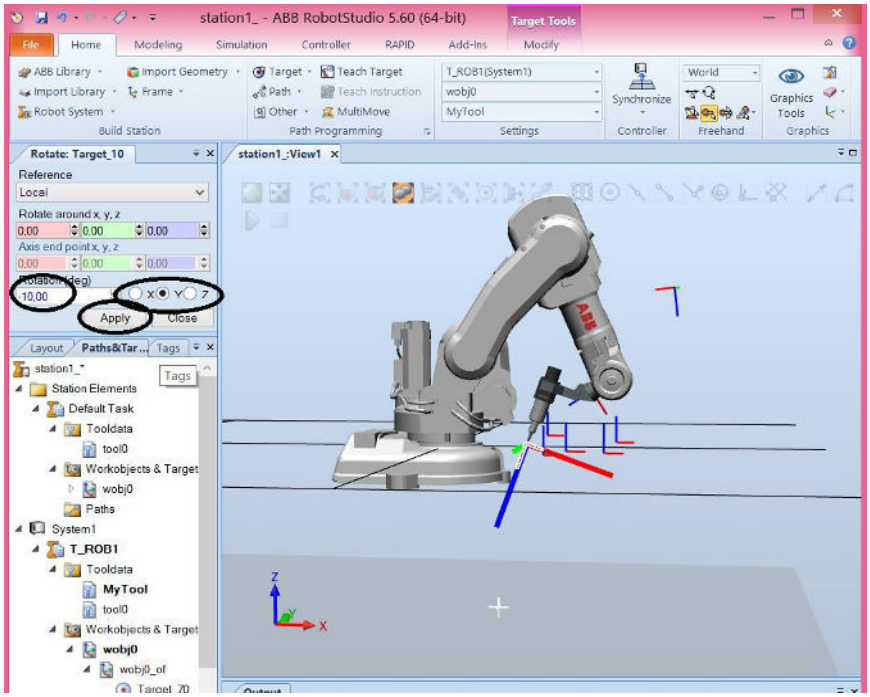


Figure 3.14. RobotStudio® - Changing the orientation of the tool according to Y axis

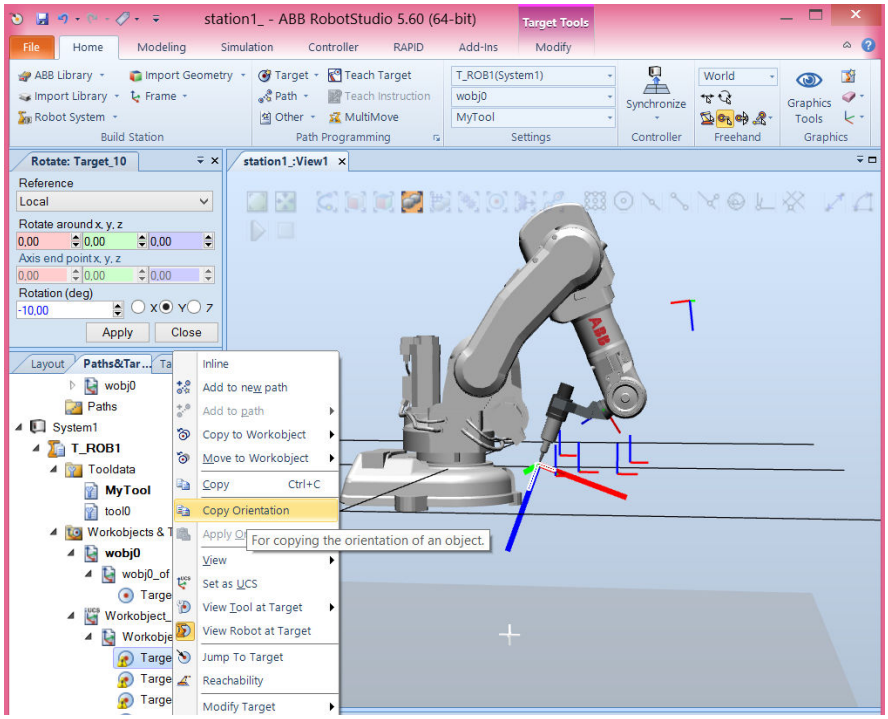


Figure 3.15. RobotStudio® - Copy the actual orientation of the tool

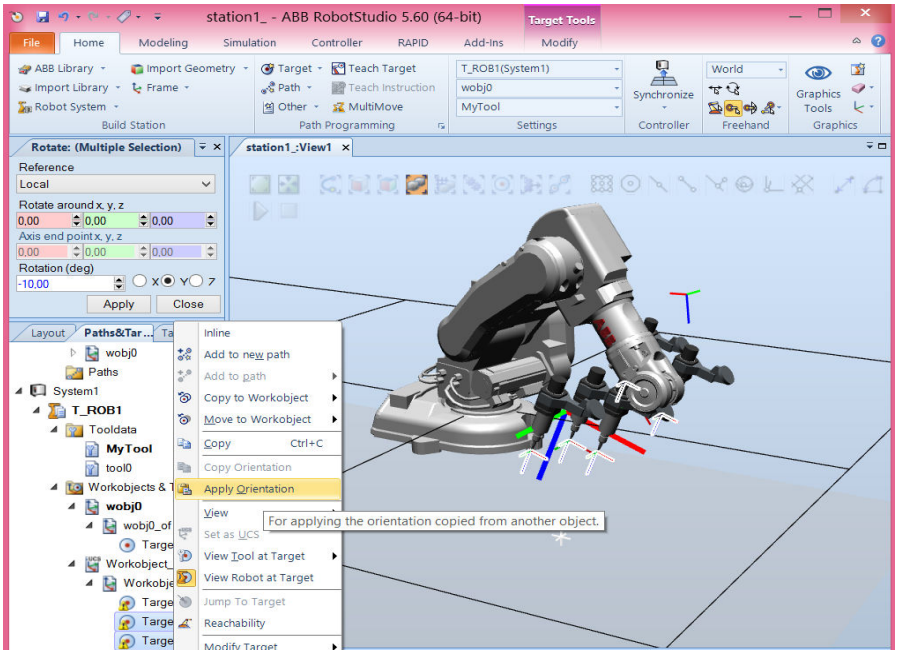


Figure 3.16. RobotStudio® - Copy the actual orientation of the tool in that target to the other targets

It can be observed nothing is happening (Figure 3.11) because the robot is not able to reach such a position. This way, RobotStudio® is giving us a warning. In order to fix this problem, we have to change the orientation of the tool: *MyTool* in that target point. The change is done for Target_10, but the other targets have the same problem. We can copy the orientation of this target (Figure 3.15) and apply that orientation to all the other targets. Select the other target points and apply orientation (Figure 3.16).

Now, by clicking on the targets we can see whether the robot reaches or not that targets. If the robot does not reach a target, we have to change that target point (position or orientation) using the command **Set Position** or **Rotate**.

Paths

Now the target points can be connected in order to create a working path (Figure 3.17) for the robot.

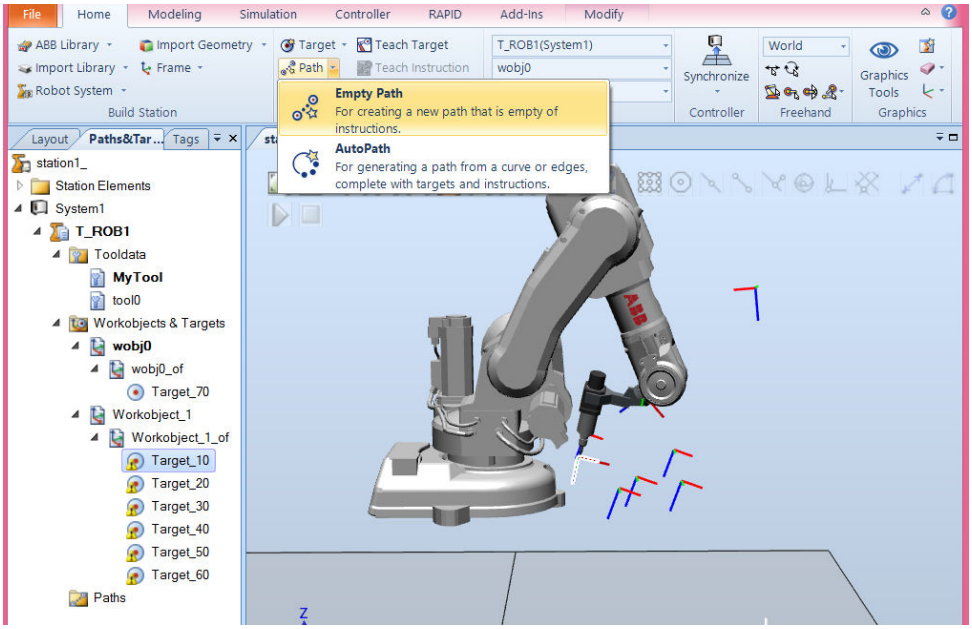


Figure 3.17. RobotStudio® - Create an empty path

And select MoveL (straight line motion between targets) (Figure 3.18).

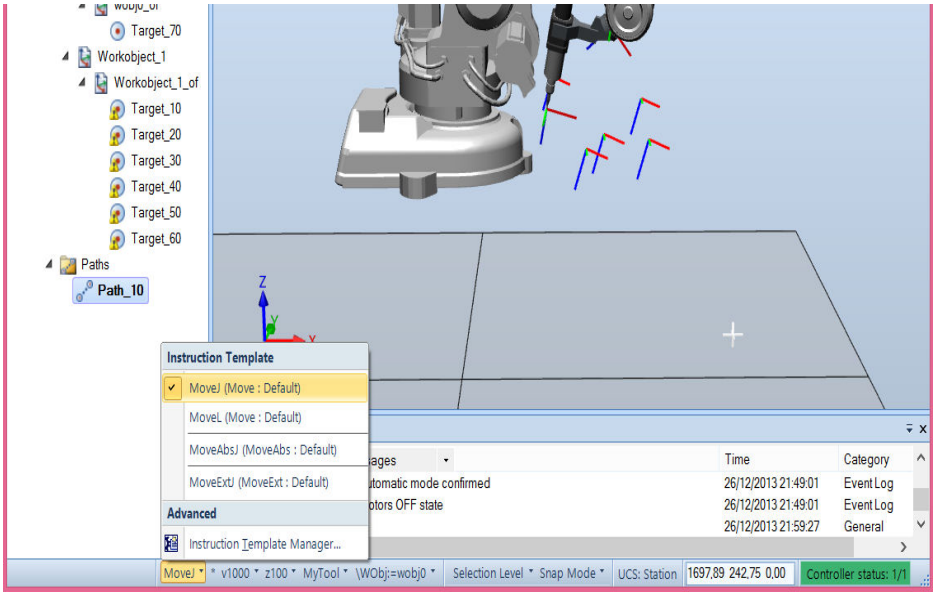


Figure 3.18. RobotStudio® - Type of motion between targets

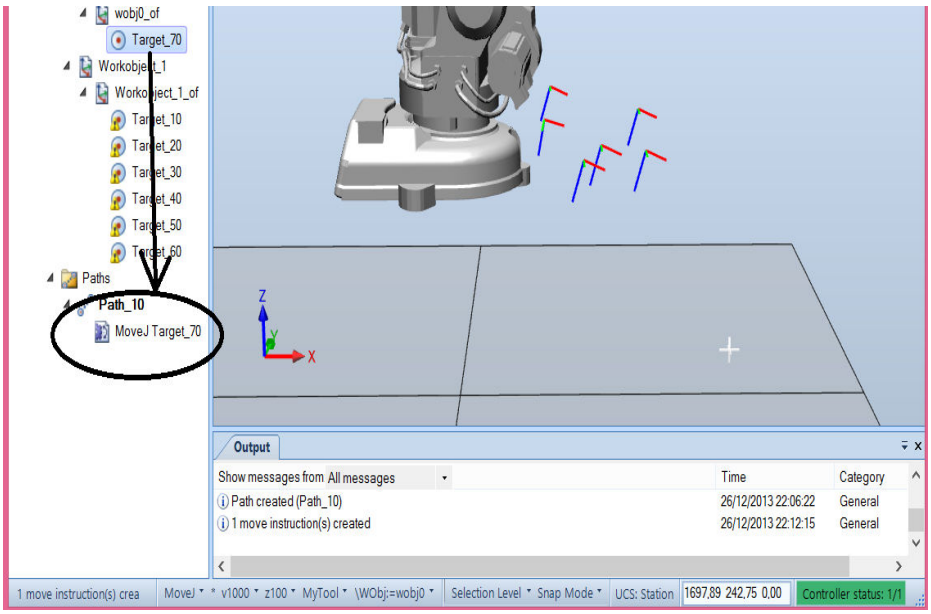


Figure 3.19. RobotStudio® - Create the path with drag and drop targets associated to wobj0

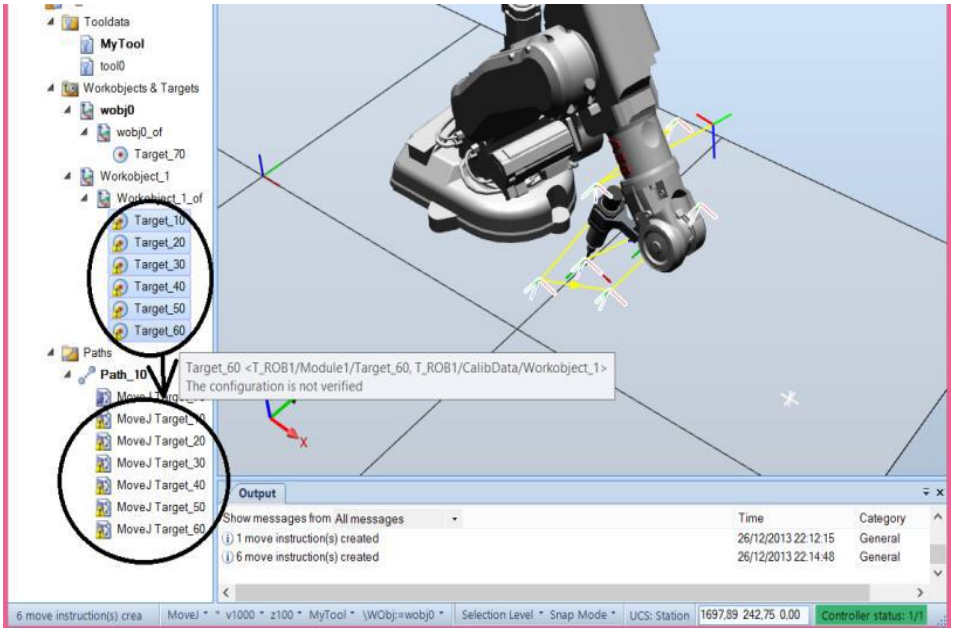


Figure 3.20. RobotStudio® - Create the path with drag and drop targets associated to workobject_1

After having created Path_10, drag the targets to Path_10 in the desired order (Figure 3.19, Figure 3.20). Having the targets connected, in order to eliminate the warnings, we have to define the robot configuration for each one. This is

happening because there are different ways to achieve the same position and orientation for the robot's tool (Figure 3.21).

The software is able to auto-configure the defined path (Figure 3.22). Or we are able to check and define the configuration for each target (Figure 3.23)

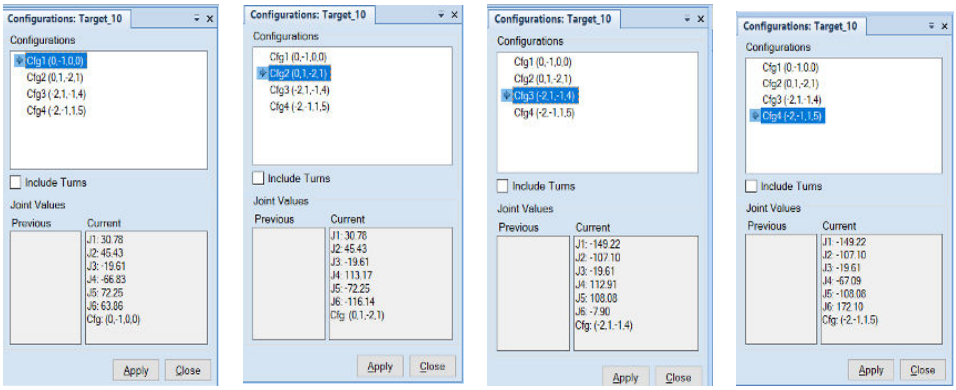


Figure 3.21. RobotStudio® - Configurations of a robot in the same point (Target_10)

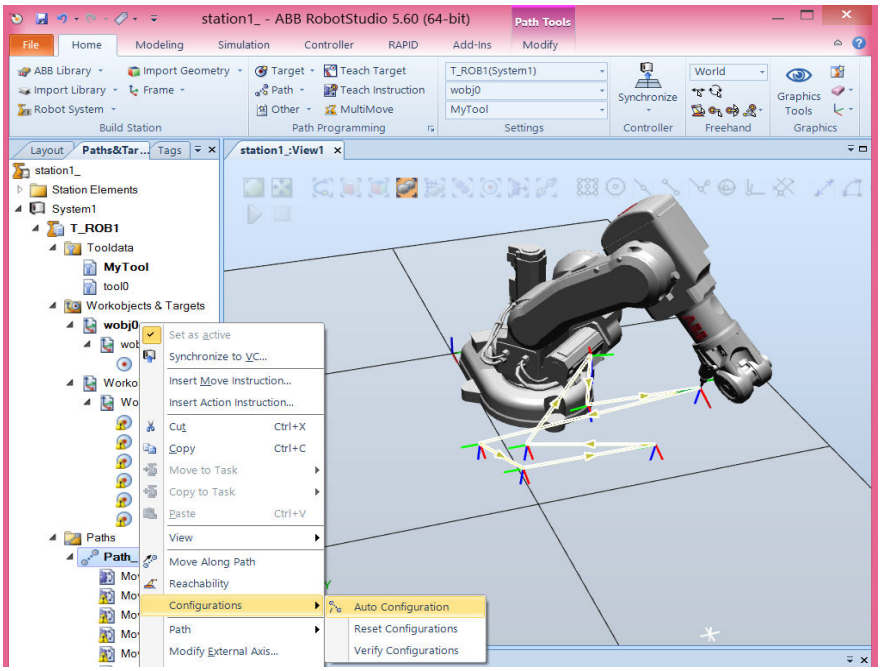


Figure 3.22. RobotStudio® - Auto-configuration of the robot

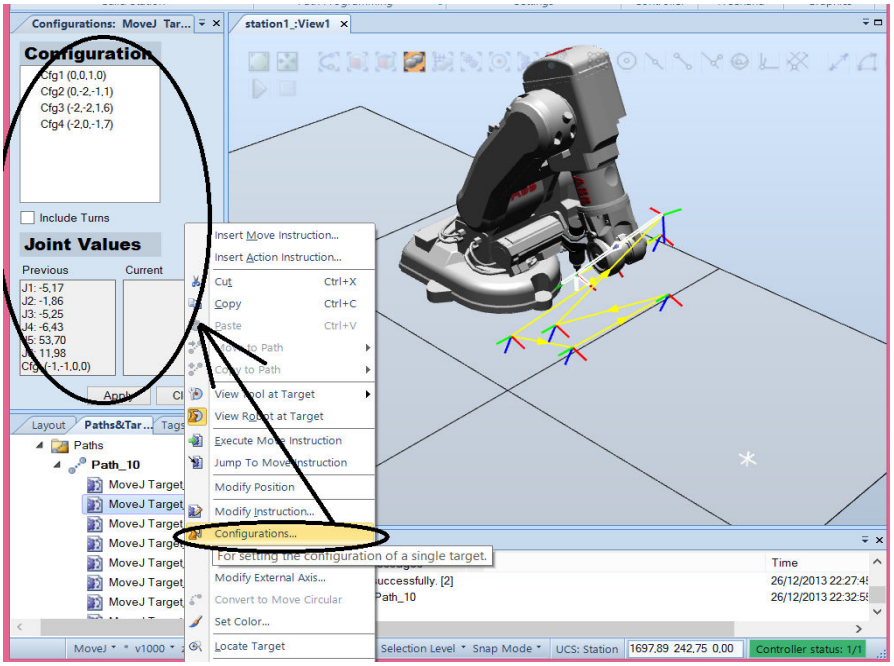


Figure 3.23. RobotStudio® - Select the configuration of the robot manually

Besides creating paths, there are several movements a robot can follow: reversing paths, rotating paths, translating paths etc. This section of the workshop will focus on defining and explaining some of them.

Reversing a path – changing the sequence of targets in which the robot moves, from last to first. One can reverse the entire motion process or just the target sequence.

Rotating a path – rotate the entire paths and move the targets that are used by the paths in accordance. Targets will lose their axis configuration if one was assigned. Before starting the rotate path command there must exist a frame or target to be able to rotate around.

Translating a path – move a path and all included targets.

Compensating paths for tool radius – compensate by offsetting a path. Targets will lose their axis configuration if one was assigned.

Interpolating a path – reorients the targets in order to have even distribution between the difference in orientation at start and end targets with the in-between ones. The interpolation can be either linear or absolute. The linear one assigns the difference in orientation equally, taking into consideration the targets' positions along the length of the path, while the absolute

interpolation assigns the difference in orientation equally, taking into consideration the targets' sequence in the path.

Simulation

It is time to simulate the robot movements based on the program we created. First, synchronize with the robot's virtual controller (VC) - this means that we have to upload the created program into the robot's controller (Figure 3.24, Figure 3.25).

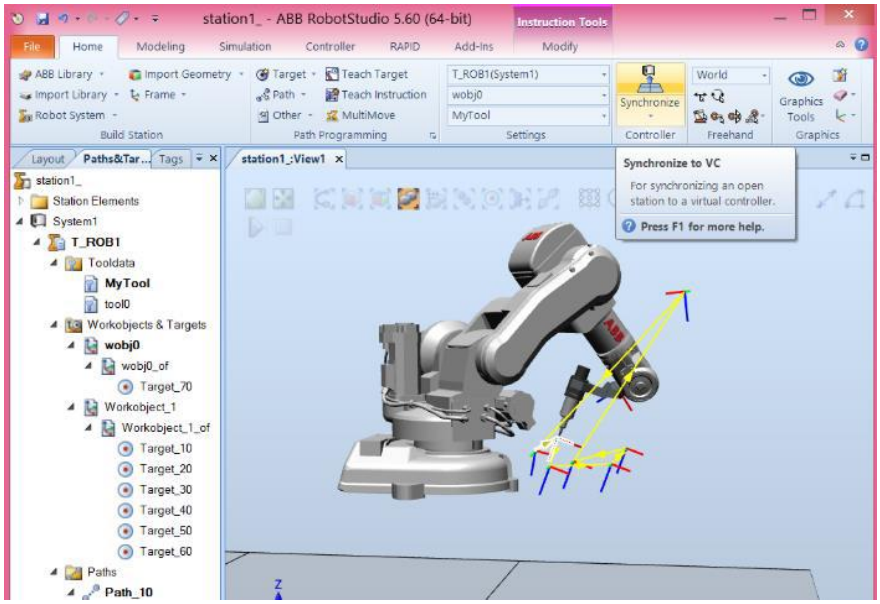


Figure 3.24. RobotStudio® - Synchronization with virtual controller (VC)

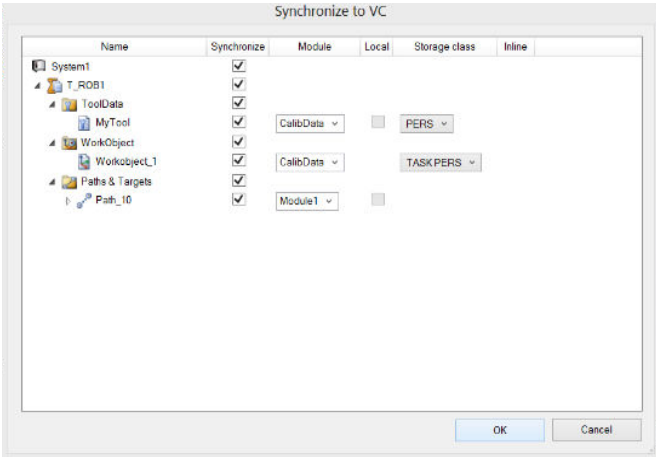


Figure 3.25. RobotStudio® - Synchronization with VC – select the equipment

Next, select what paths to simulate (Figure 3.26). In this case, we have only Path_10 to simulate.

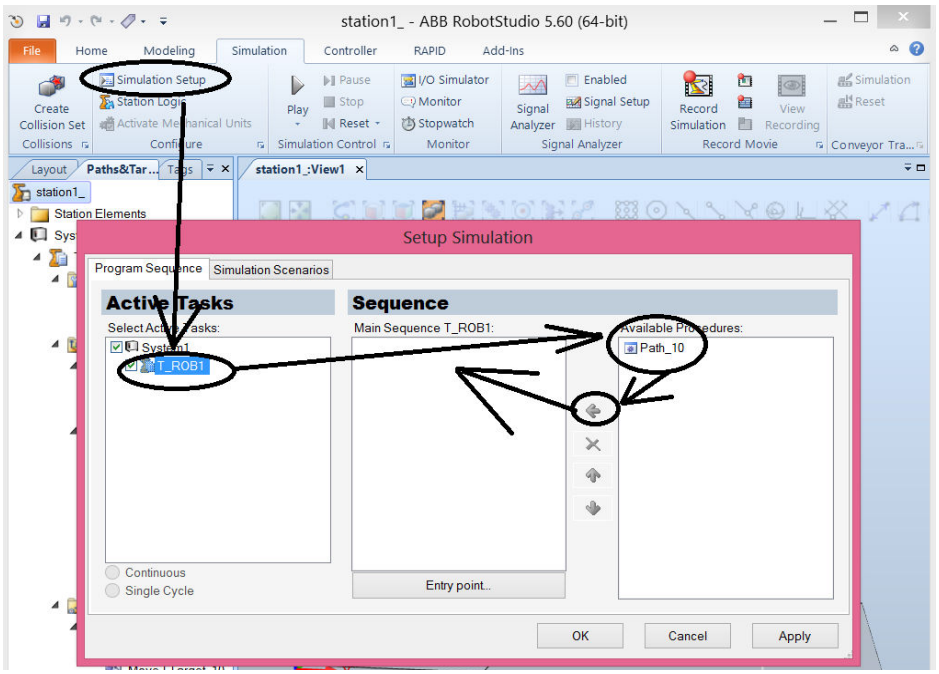


Figure 3.26. RobotStudio® - Select the desired path to be simulated

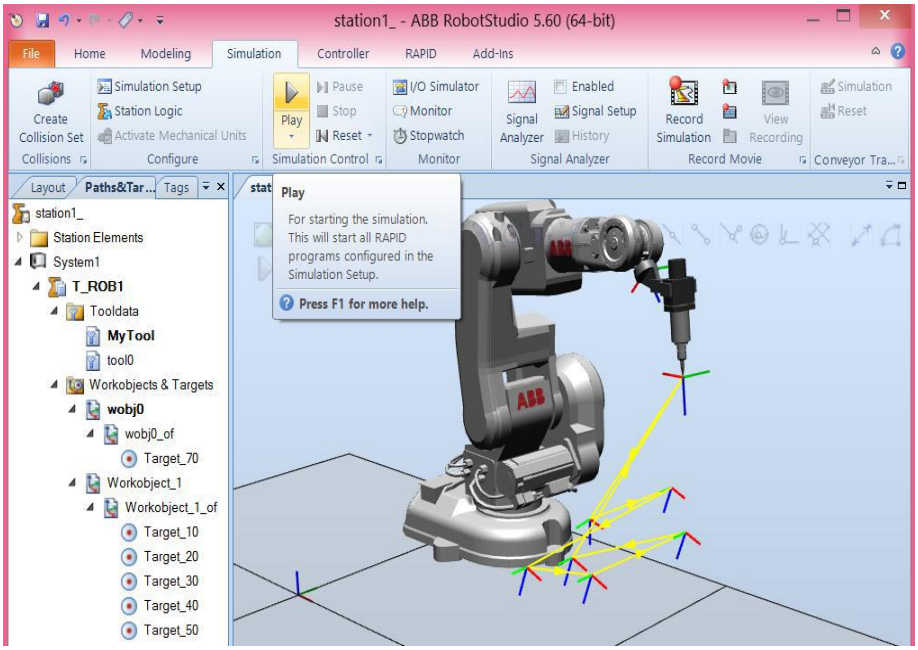


Figure 3.27. RobotStudio® - Run the simulation

And after that, select the Play button to start the simulation (the robot will follow the selected path (ex. Path_10) according to the generated program) (Figure 3.27).

After simulation, if you observe that something needs to be changed, like a target (orientation or position) for example, follow the explanation in Figure 3.28.

After taking this step, you have to synchronize again with the virtual controller. Sometimes, during simulation it is useful to view the angles of each joint of the robot (Figure 3.29).

If you want to save the robotic station (or robotic cell) you can do it in the usual way or by using “Pack and Go” function (Figure 3.30). This last option saves the entire project in a folder, so that you can open it on another computer.

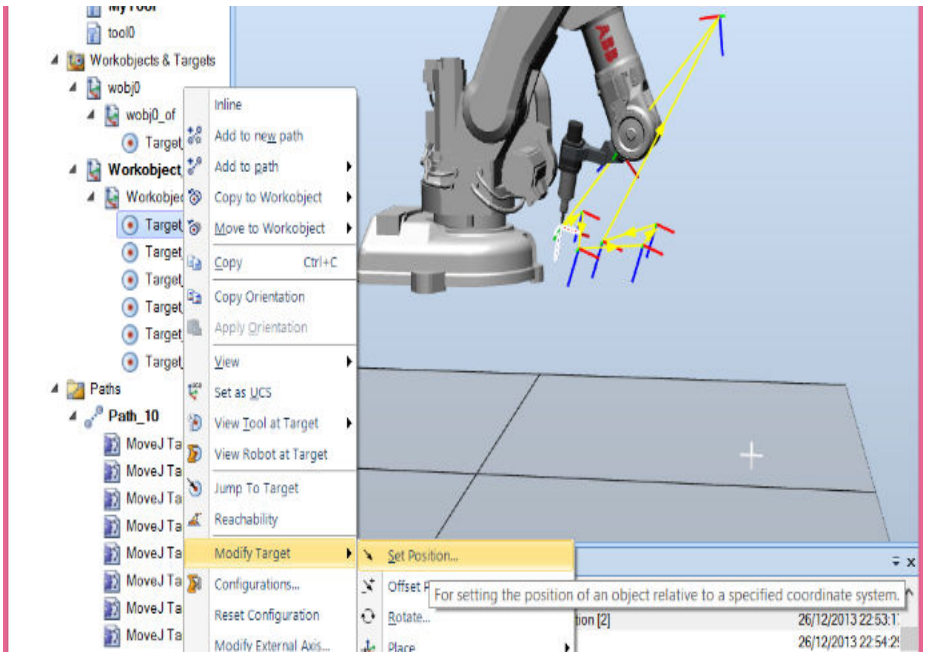


Figure 3.28. RobotStudio® - Changing the position of a known target

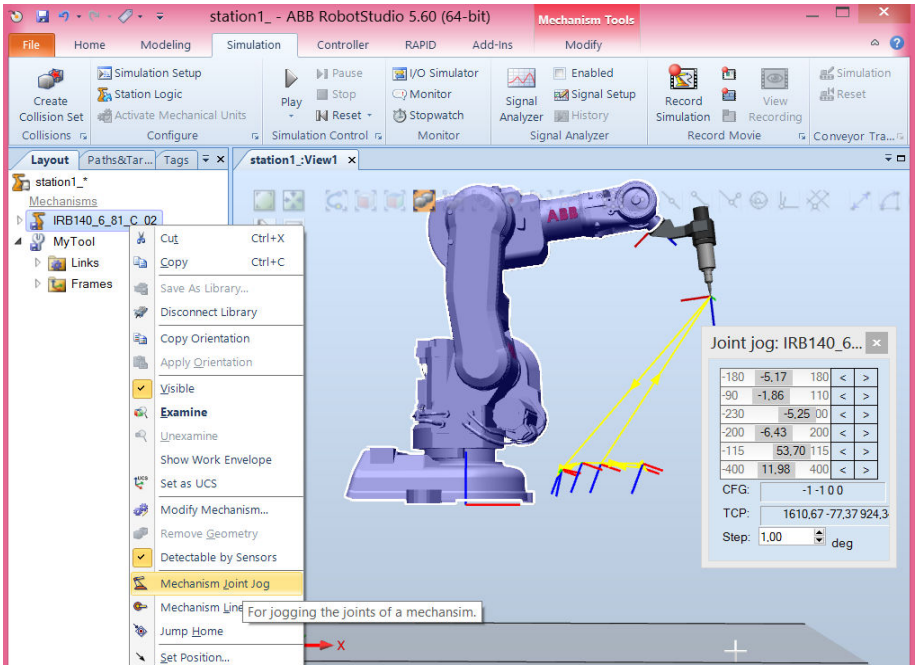


Figure 3.29. RobotStudio® - Activate the Joint Jog window for angles in joints

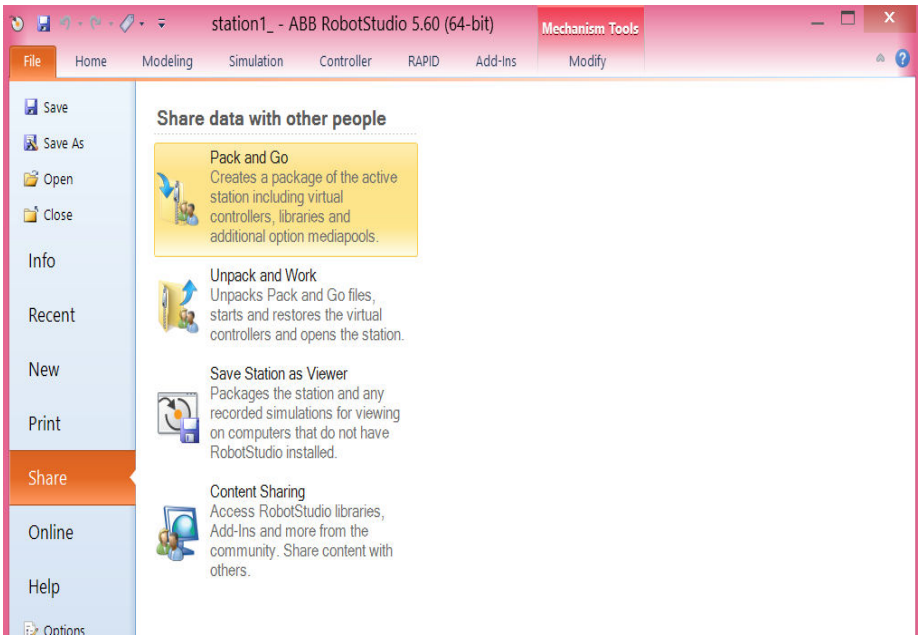


Figure 3.30. RobotStudio® - Pack and Go function

Workshop 4: Collision Control & Create a mechanism

Necessary knowledge

Workshop 3 completed.

Workshop 4 summary

At the end of this workshop, the students should know how to:

- Define and simulate a collision
- Import geometric parts (import 3D objects from another software application like solidworks, CATIA, etc.)
- Create and define a tool mechanism (create links, joints and define the tool)
- Save the created mechanism into Robotstudio® Library
- Define the TCP (Tool Center Point)

4.1. Aim of the workshop

The aim of this workshop is for the students to know the importance of collision study when referring to a robot programming. Furthermore, they must know how to create a mechanism for a tool in case they need a specific tool for their own application and they want to define it, in order to be recognized by the robot controller.

4.2. Collision Control

Sometimes, when a robotic cell is implemented, the fact that around the robot there are both other objects and an operator has to be taken into consideration. All these are considered obstacles and the contact between the robot and one of them is called a collision. This can be simulated in RobotStudio®. In the simulation tab press the **Create Collision** button (Figure 4.1).

Open the collision object and drag the tool to Objects A and the work pieces to Objects B. The software analyses collisions between objects, type A and B (Figure 4.2).

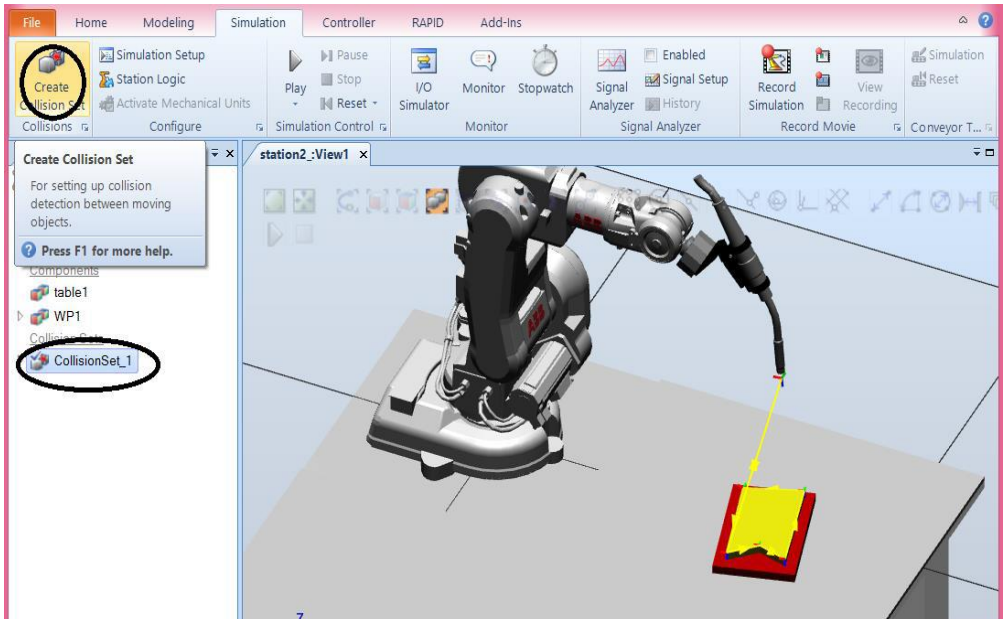


Figure 4.1. RobotStudio® - Create Collision

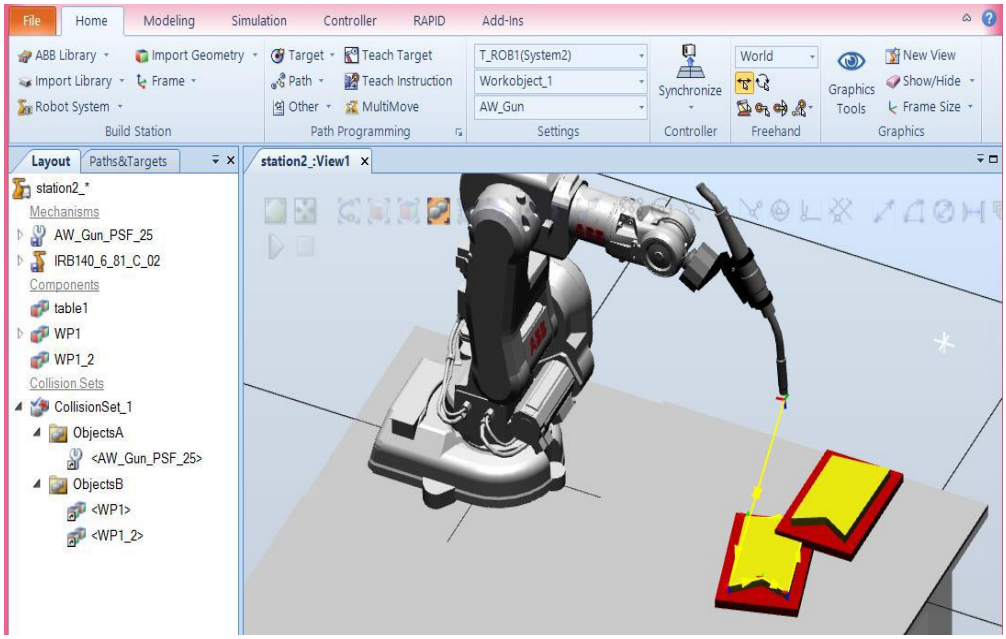


Figure 4.2. RobotStudio® - Specify the tool and the obstacle for collision

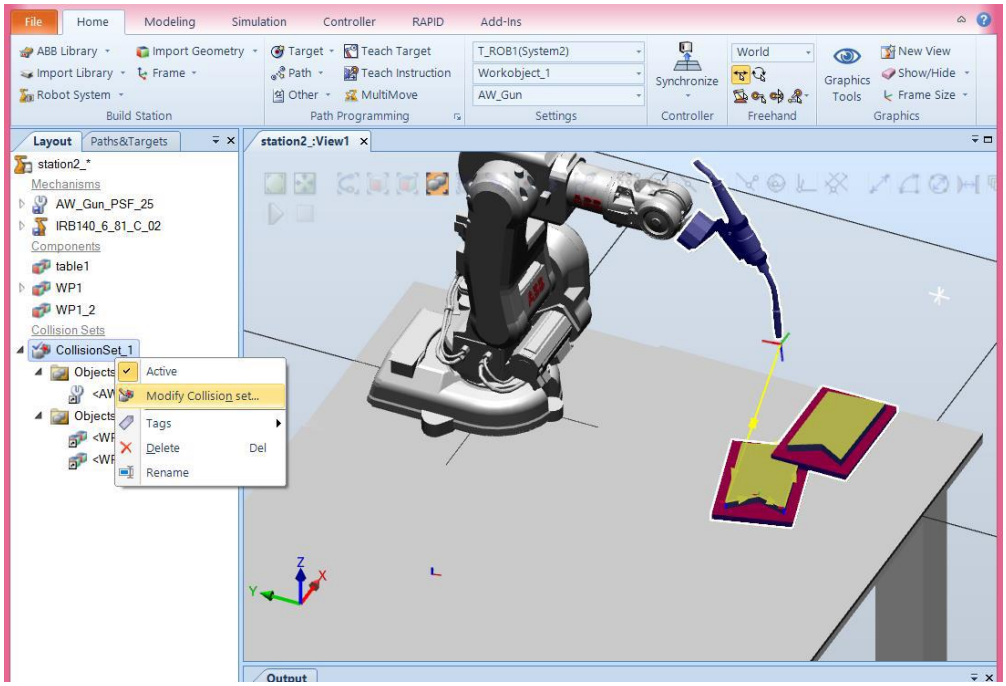


Figure 4.3. RobotStudio® - Modify Collision options

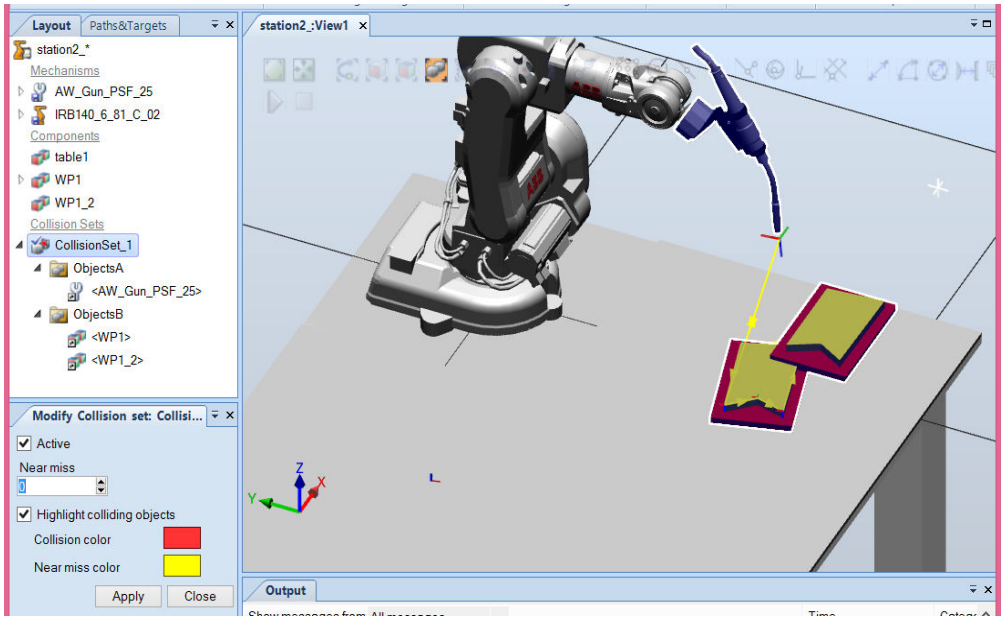


Figure 4.4. RobotStudio® - Set a certain collision distance

Also, we are able to modify the defined collision scenario (Figure 4.3) or to change the collision distance (Figure 4.4). Afterwards, we are able to simulate the collision and to analyse the results, as in Figure 4.5.

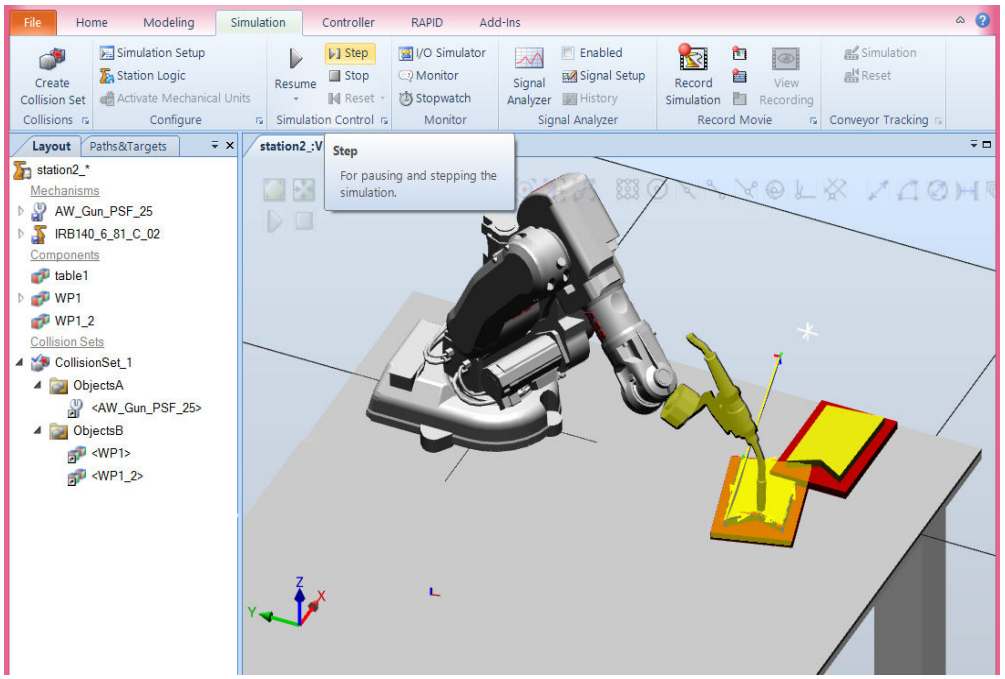


Figure 4.5. RobotStudio® - Simulate collision

4.3. Tool mechanism

To create and define a tool mechanism, all the 3D parts that form the tool are needed, with the **.sat* extension, created in CAD software.

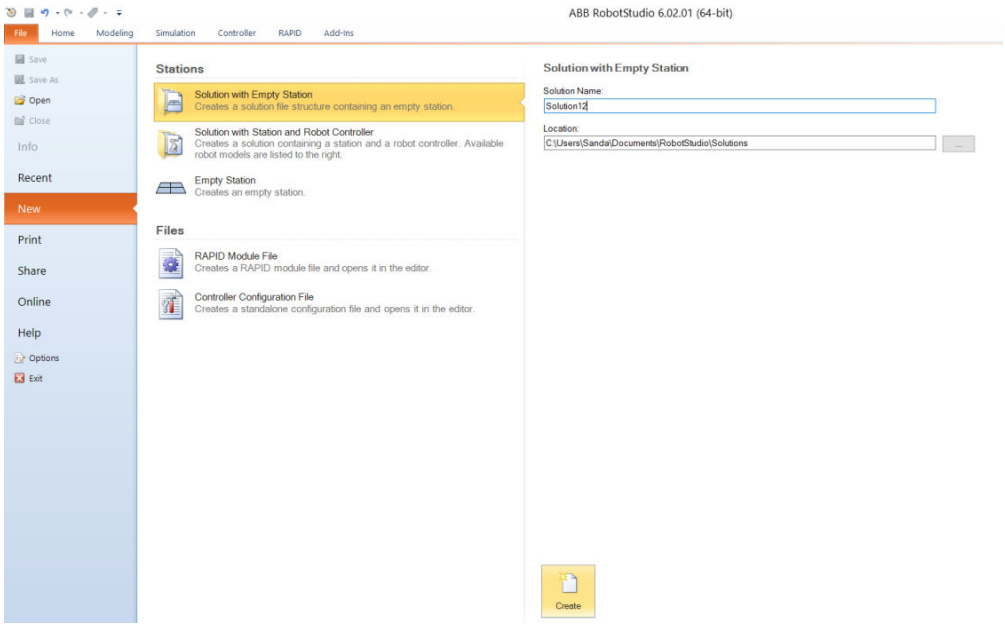


Figure 4.6. RobotStudio® - Create an empty station

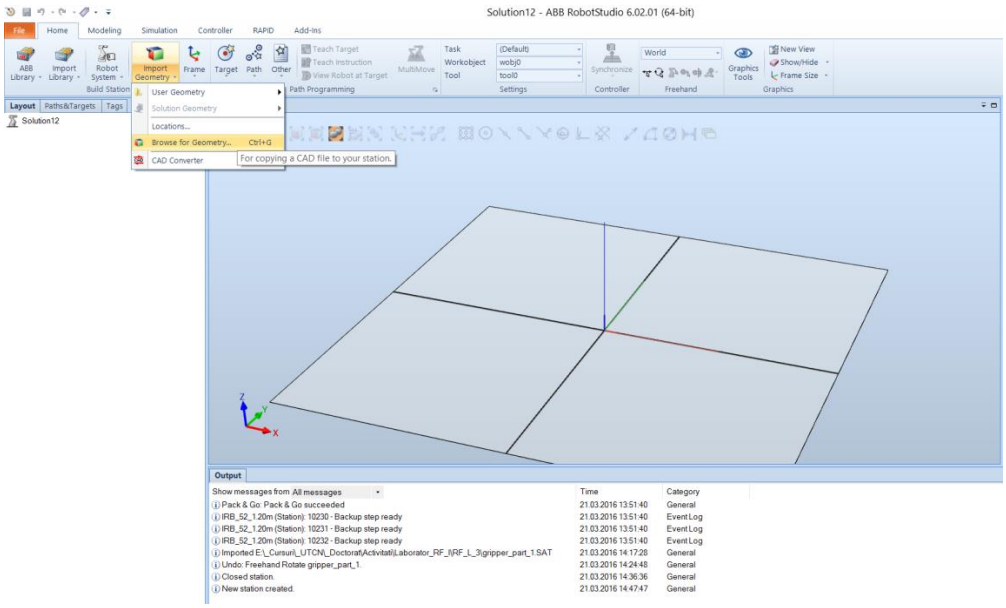


Figure 4.7. RobotStudio® - Import parts in RobotStudio®

Once all these 3D parts are created, in a specialized software application (ex. SolidWorks, Catia, etc.), they can be imported in RobotStudio® in order to make the tool and start defining it.

The first step is to create a new empty station (Figure 4.6). Once it is created, the 3D parts of the tool will be imported one by one (Figure 4.7). After we access the Geometry folder, the parts that form the gripper should be selected.

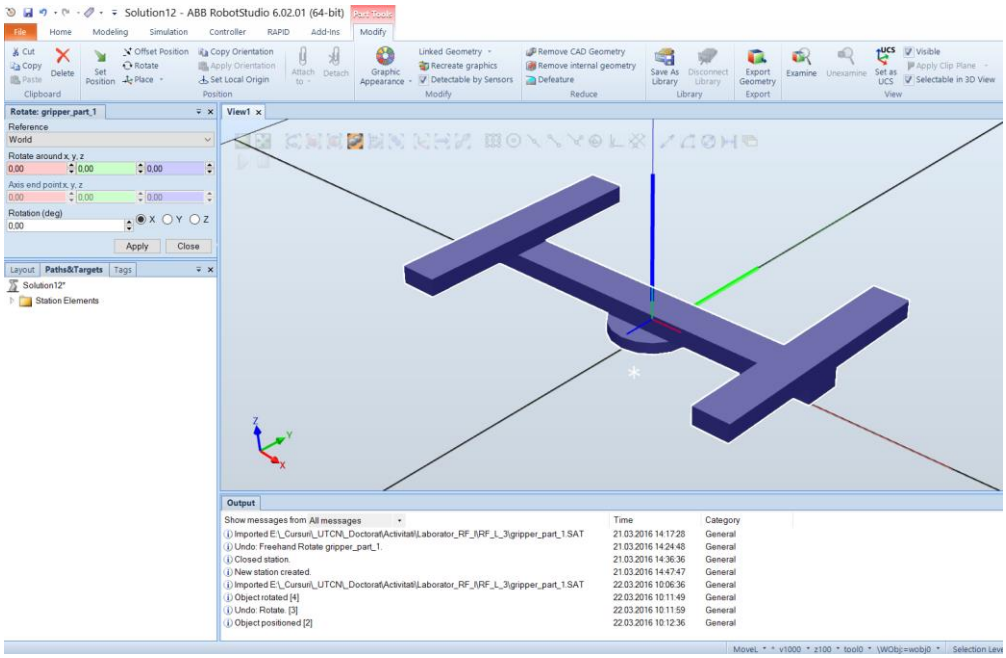


Figure 4.8. RobotStudio® - Positioning of the part which represent the base (rotation and translation)

Select the first 3D part, ex. *gripper_part_1* from the left list (Figure 4.7) and, instantly, the Modify menu is available. The 3D part is imported and displayed in the working window, like in Figure 4.8. The Rotation option is available and the position and the orientation of the *gripper_part_1* could be changed and so the base position of the gripper is changed (for exemplification, set 90° rotation according with X axis and with Z axis). PAY ATTENTION to the Reference system (cartesian system) according which the position of the gripper base is done. When you want to rotate or position it, it has to be done in accordance with the cartesian system established as Reference. In this case the position and orientation changes are done according to World Frame System (Figure 4.9).

From the Offset/ Set Position, set 20 mm along Z axis. After the desired position is achieved, the *Apply* button must be selected once and, the window can be closed. Also, from *Modify* menu, from Set Local Origin, one can set the origin of the gripper's base. Initially, the values are those that we used to set the position we wanted (Figure 4.10).

It can be noticed that the position of the gripper's base has not been modified.

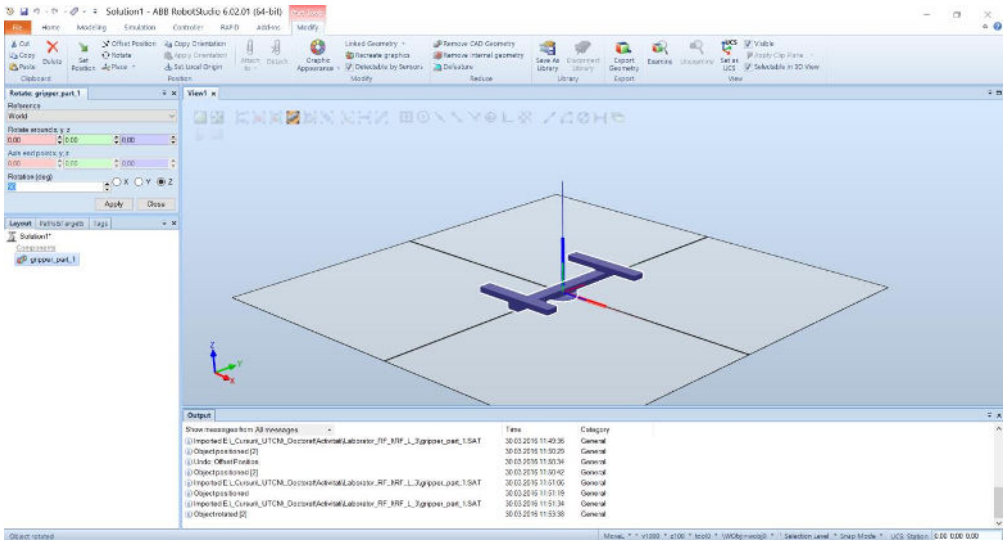


Figure 4.9. RobotStudio® - Set the desired position of the gripper base

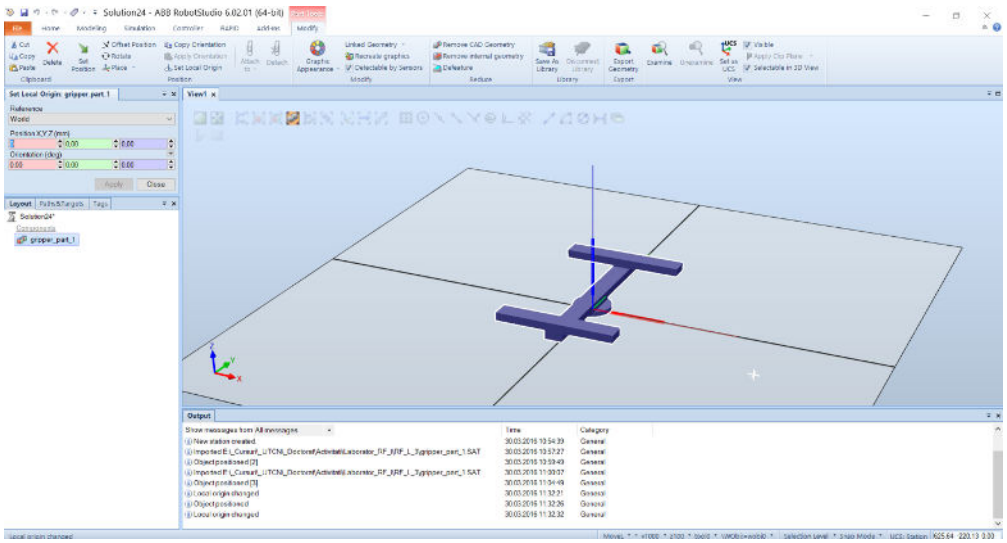


Figure 4.10. RobotStudio® - Set the origin of the gripper

The next step is to import, from the Browse for Geometry menu, the “fingers” of the gripper (in this case, the 4 fingers). From the Modify menu, each finger must be placed at the end of each “bar”, symmetrically, like in the Figure 4.11.

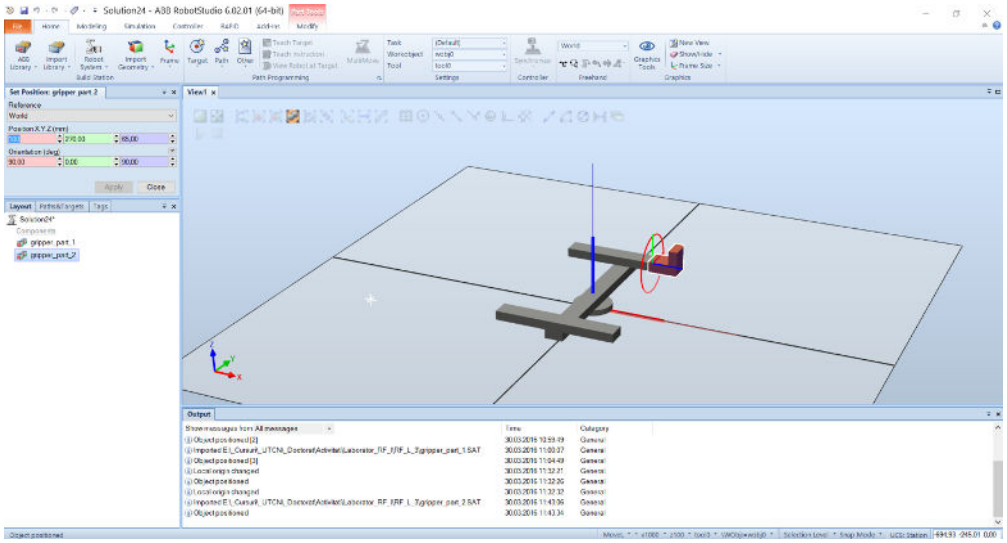


Figure 4.11. RobotStudio® - Set position of one finger

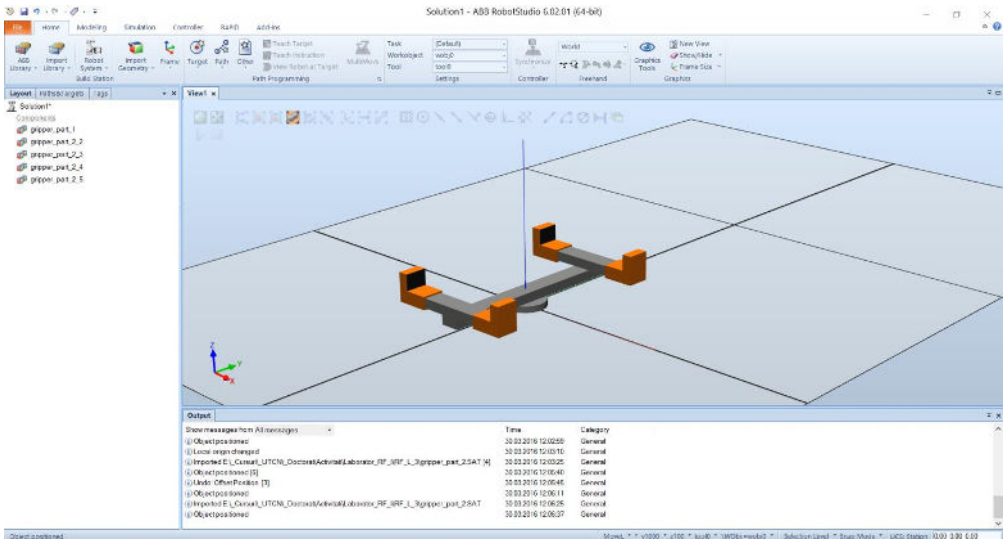


Figure 4.12. RobotStudio® - The result using the given values to set the position of each finger

One can use the below values for orientation and position the four fingers of the gripper:

Tx = 100, Ty=270, Tz = 65; Rx = 90°, Ry = 0°, Rz = 90°

Tx = -100, Ty=270, Tz = 65; Rx = 90°, Ry = 0°, Rz = -90°

Tx = 100, Ty=-270, Tz = 65; Rx = 90°, Ry = 0°, Rz = 90°

Tx = -100, Ty=-270, Tz = 65; Rx = 90°, Ry = 0°, Rz = -90°

The result of using these values can be seen in Figure 4.12. When all the parts are in the desired position, in order to function, a mechanism must be created. This it is done using the Modeling menu, Create Mechanism option (Figure 4.13).

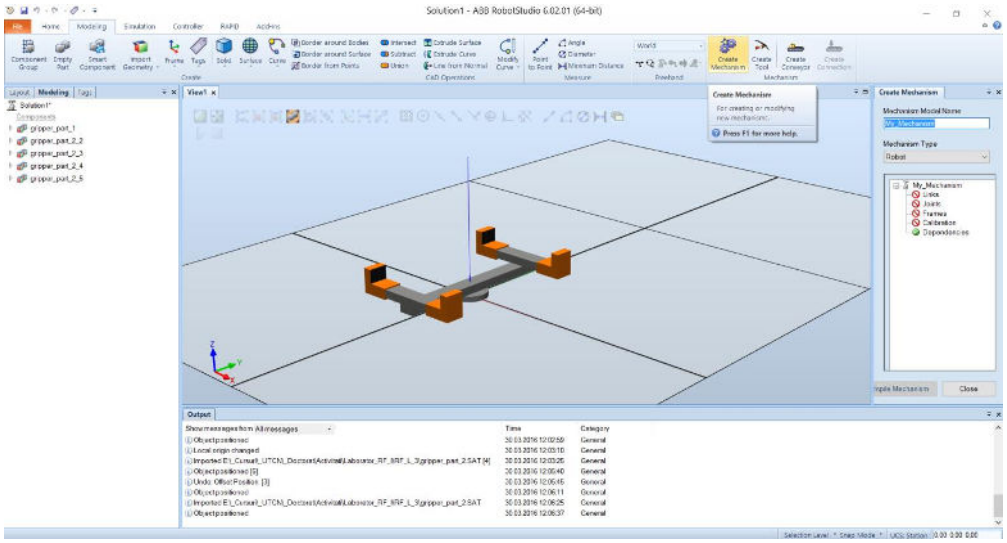


Figure 4.13. RobotStudio® - Create Mechanism option

After you click on Create Mechanism, a window will appear. In this window, you have to establish the name of the mechanism (should be an intuitive one: ex. Welding_gun), to set the type of the mechanism (tool, conveyor, etc.), in this case will be a mechanism Tool type. The next steps are to create joints between parts that are moving and to define the tool. Further on, define each part as a link: right click on Links and Add Link.

A new window (Figure 4.14) will be opened. In this window, each part (each finger) is defined as a link. The link's name is given automatically. The part you want to define must be set and then add it to the Added Parts list. Make sure

that the part: *gripper_part_1*; is Set as BaseLink. The other parts, like fingers, have to be defined like “Link” not as a BaseLink. After all the parts are defined, close the window.

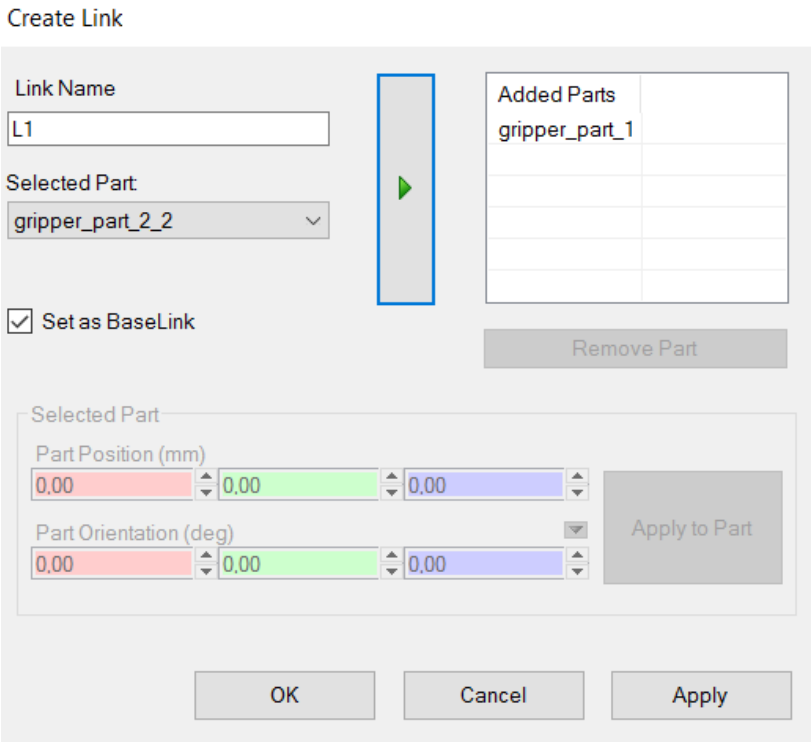


Figure 4.14. RobotStudio® - Set the links

The next step is to create the Joints between the parts, so right click on Joints and Add Joint and the window from Figure 4.15 will appear. The name is given automatically. The type of joint in this example is prismatic, for all the joints (be careful to set the Joint Type as Prismatic) and all joints are between the base and each “finger” (that changes each time from Child Link).

Based on the given example, regarding prismatic joints, the displacement length of the prismatic link will be set: for Second Position on X (red cell) on 100 [mm] value; Joints Limits: Min Limit will be 0 and Max Limit will be 100. The only value that changes for the 4 joints is the Second Position, which is 100 mm or -100 mm, depends on each of the four fingers is defined. To check if the fingers are moving in the right and logic position, use the slide from Jog Axis (Figure 4.15).

If everything is correct, based on the logic function of the gripper, set Apply and automatically go to the next joint. After all the joints are defined, click Cancel and the window closed.

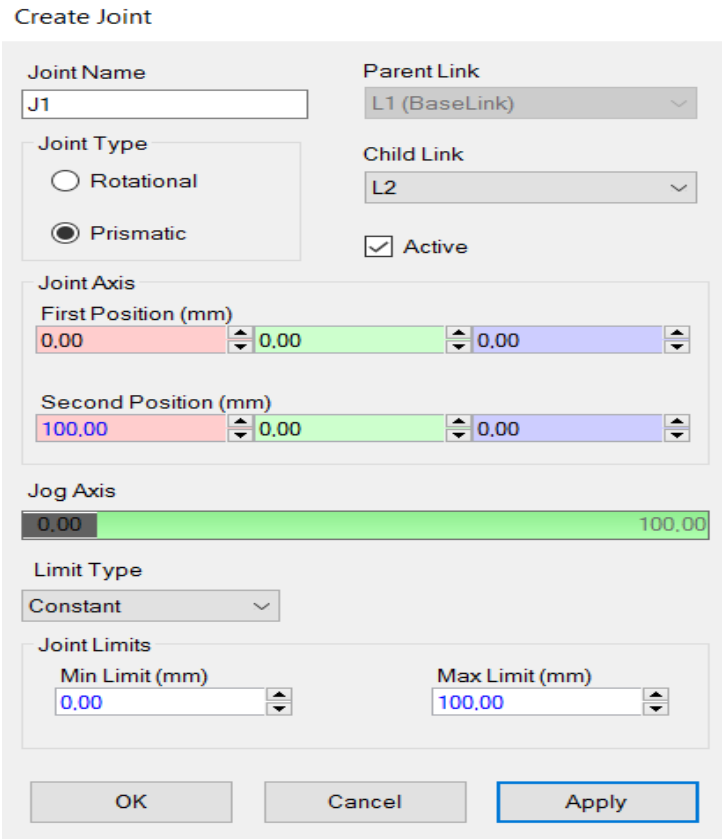


Figure 4.15. RobotStudio® - Create joints

Afterwards, the tool must be defined, precisely to set the Mass, the Center of Gravity and the Moments of Inertia. To set these, right click on Tooldata and Add Tooldata. The window from Figure 4.16 will open. The values that must be set, for our example, are presented in Figure 4.16. Make sure you set the base in the cell Belongs to Link.

The next step is to create dependencies between links. For this, right click on Dependencies and Add Dependency. The window from Figure 4.17 will appear. First dependency is between J2 and J1 (LeadJoint), then between J4 and J1 (LeadJoint) and, lastly, between J3 and J1 (LeadJoint). In all cases the Factor is 1. After all these dependencies have been created, click OK.

On the right side of the screen, notice that all the characteristics are indicated in green, highlighted in blue (Figure 4.18). The next step is to Compile Mechanism (Figure 4.18).

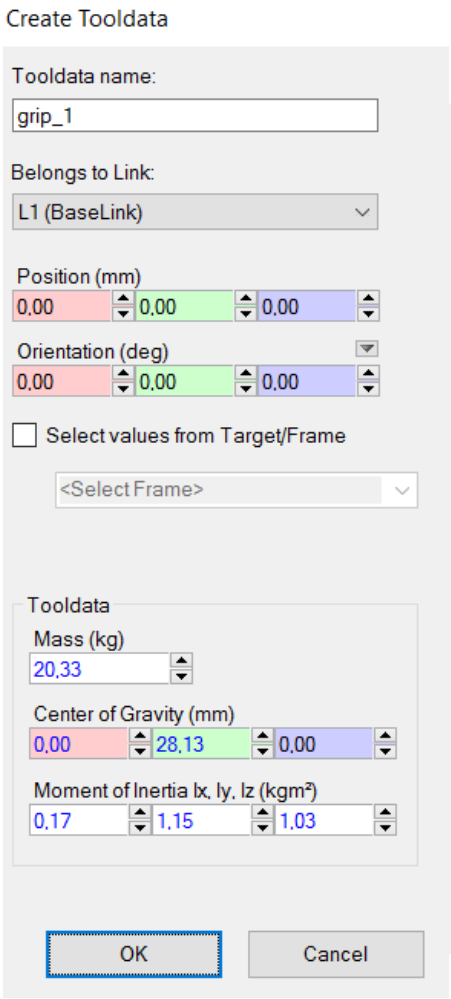


Figure 4.16. RobotStudio® - Define the tool

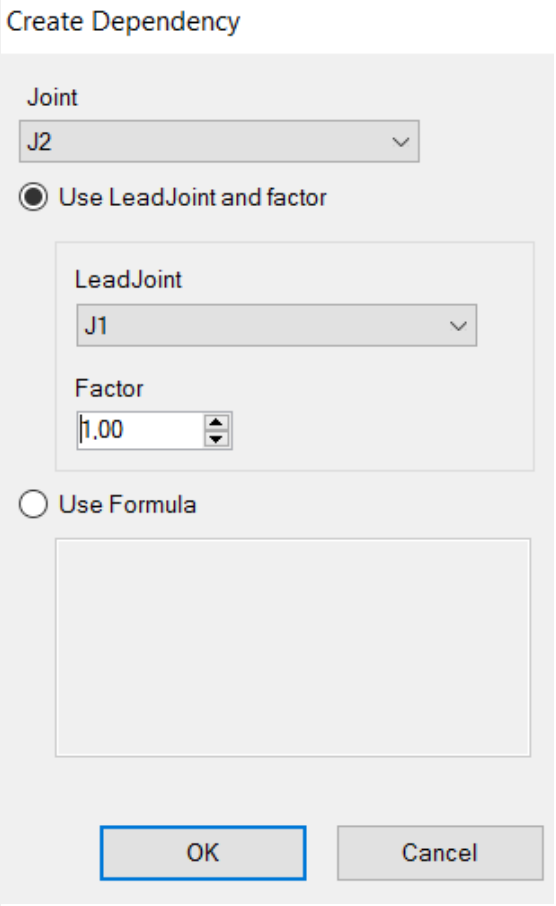


Figure 4.17. RobotStudio® - Create dependencies

Once the Compile Mechanism button is pushed, a new window appears. In that window, if in Joint Mapping the values are correct, select Set. At Poses establish the position of the gripper's fingers and give them specific names. To create a new one click, Add, to modify one click Edit and if you want to erase it, click Remove. Click on Add and create a HomePose and an Open position (see Figure 4.19 and Figure 4.20).

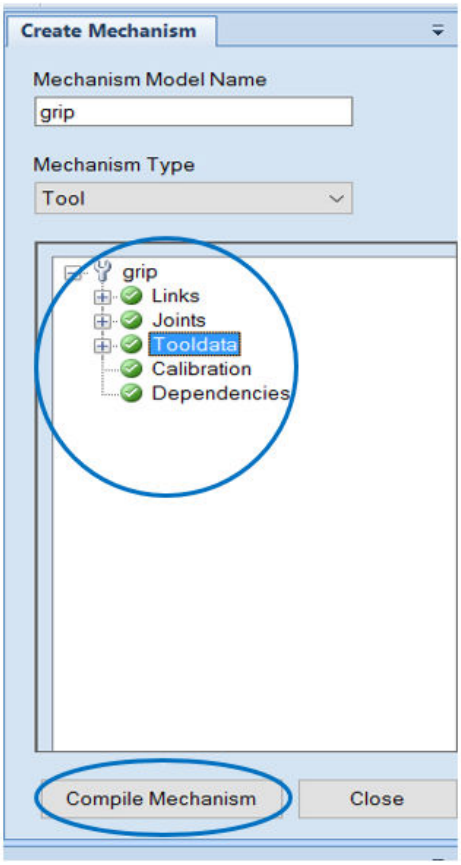


Figure 4.18. RobotStudio® - Compile Mechanism

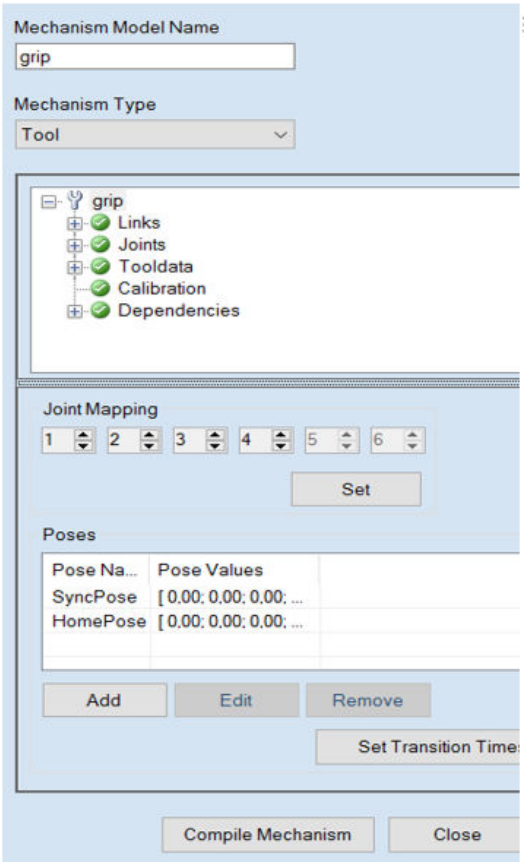


Figure 4.19. RobotStudio® - Joint Mapping and Poses

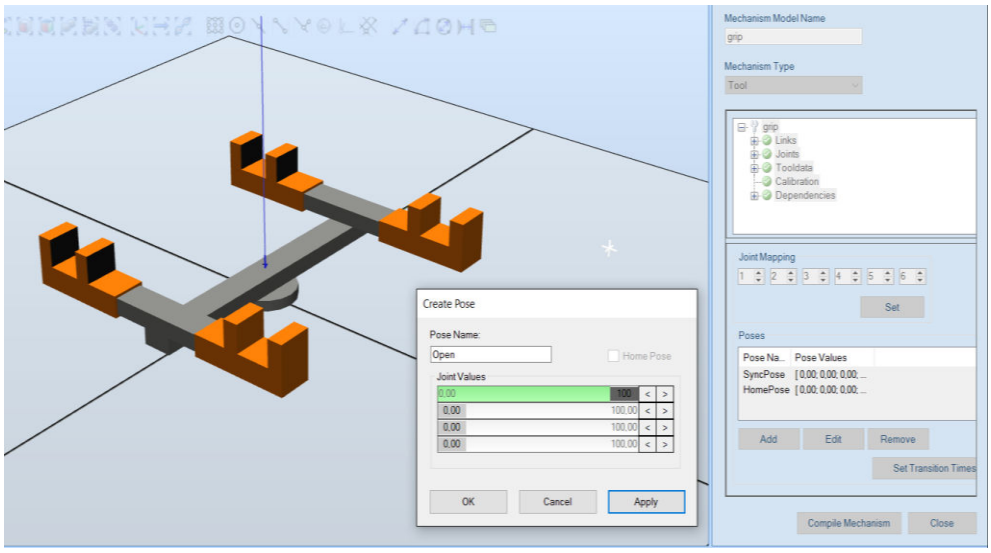


Figure 4.20. RobotStudio® - Values for Open position

Once all these are set, click on Set Transition Times and click OK and then Close. On the left side of the window, at Layout, the mechanism that was created can be seen.

Right click on its name and set Mechanism Joint Jog. Further slide bars are opened and if you move just the first one, which corresponds to the first joint, one can notice that all the fingers are moving at the same time. If the gripper is in Open position, then right click on its name and set Jump Home.

The next step is to learn how to save the mechanism in the RobotStudio® Library. For this, right click on the mechanism that was created, in the left list found on the screen and select Save As Library. Give it a name and Save it. To see if the mechanism has been saved, Import Library and User Library.

TCP definition

To set the TCP of a tool, follow these steps:

- click on Create Target,
- select the top of the tool,
- click Create and then Close.

If the tool has a mechanism like the one created in the above section, the TCP must be set at the center of the gripper (in-between the fingers) in a zone where the target point can be defined. To create/define a tool follow these aproach: Modeling -> Create Tool. Make sure to create a tool that is formed of a single body. If there are any other bodies, make just one using Union option from Modeling menu. Once the Create Tool option is activated, the window from Figure 4.21 will open.

Step 1: write a name for the tool

Step 2: at Select Part: Use Existing

Step 3: Center of Gravity is picked with Snap Center, then click Next

Step 4: Select TCP name (Figure 4.22)

Step 5: Select the TCP as being the Target_10 (Layout list) and then add it to TCP(s)

Step 6: Done

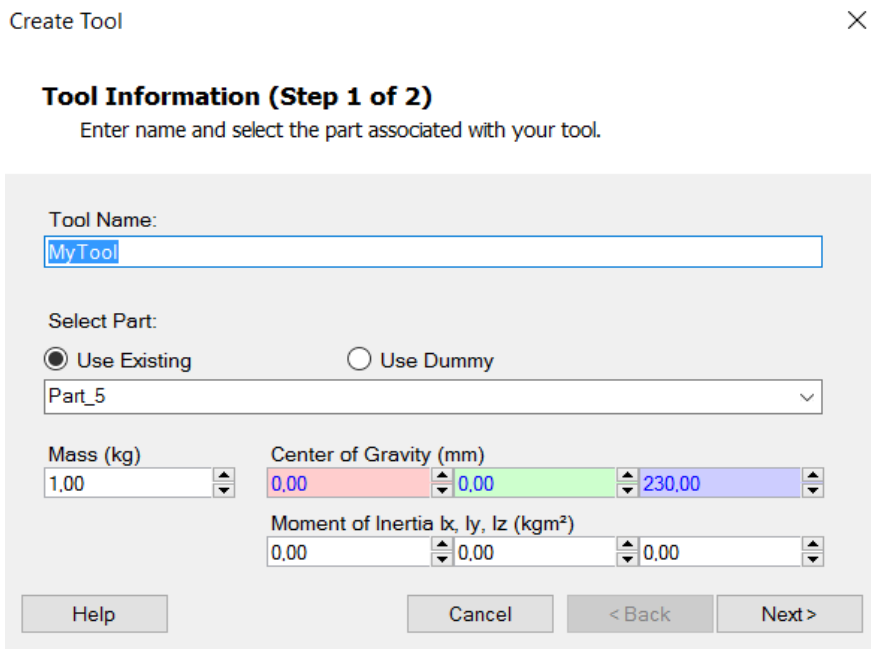


Figure 4.21. RobotStudio® - Set/define the TCP

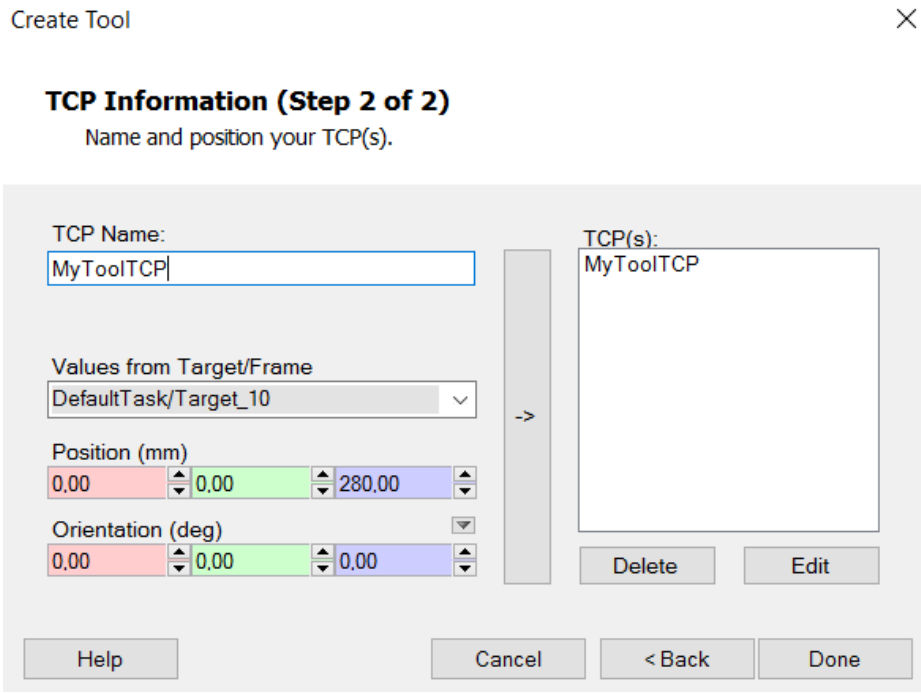


Figure 4.22. RobotStudio® - Set the TCP – attach the Target_10

Save the tool: right click on the mechanism that was created, in the left list of the screen and select Save As Library. Give it a name and Save. To see if the

mechanism has been saved, Import Library and User Library. If you followed the steps presented before, the tool must be there. To check if the tool is well defined, import an ABB robot, attach the tool and do like in Workshop 3 - Targets and Trajectories.

Note: For this section of the workshop, you can practice on the station found in the folder called “tool.rsstn”.

Workshop 5: Create the Conveyor's Mechanism and Programming MultiMove systems

Necessary knowledge

Workshop 4 completed

Workshop 5 summary

At the end of this workshop, the student should know how to:

- create a box and different 3D geometry in RobotStudio®
- create and define a conveyor mechanism from a 3D geometry in RobotStudio®

5.1. Aim of the workshop

The aim of this workshop is for the students to learn how to create a conveyor, which is a mechanism used to transfer the objects that are manipulated by the robots, from one point to another.

5.2. Create Conveyor Mechanism

Nowadays, the technology is at a high level. Because of this, more and more domains are automated or robotized. The time, as a resource, became even more appreciated by the companies, alongside with the quality of the products. Therefore, companies must manage these resources more efficiently and effectively.

A robot's working area is clearly defined in their data sheets even in their construction phase. Nevertheless, for some applications, this working area is too small and, therefore, must be extended. Both the extension of the working area and the improving of the production time can be done using conveyors or creating external axis for the robots.

Taking all these into consideration, the aim of this workshop is to create a conveyor mechanism in RobotStudio® to complete a virtual simulation of industrial robots in different situations. The first part of the workshop presents the theoretical part, while in the second part an application is conveyed.

In order to define the mechanism of a conveyor in RobotStudio® it is necessary to create an empty station. For this exercise, a box will be considered the conveyor. To create it, from Modeling menu, use the Solid option > Box. The box will have the following dimensions: length 5000 mm, width 400 mm and height 100 mm; for position, with y = -200 mm. Afterward, press Create and Close (Figure 5.1).

Like in the case of the defining a tool, a mechanism will also be created here, but the type of the mechanism will be different. From the Modeling menu, choose Create Mechanism (number 1 marked with red). Give a certain and intuitive name for the mechanism and at the Mechanism Type option select Conveyor (Figure 5.2).

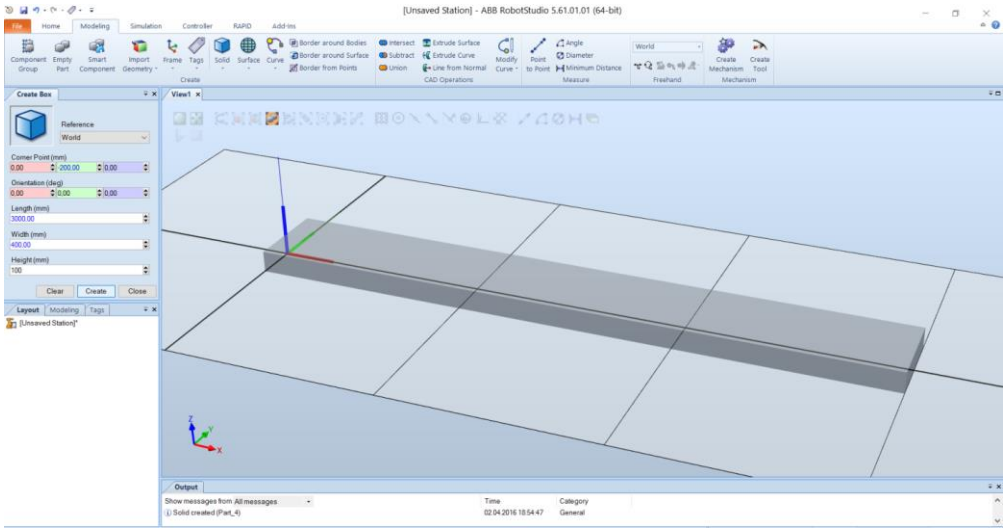


Figure 5.1. RobotStudio® - Create a box that will be considered the conveyor

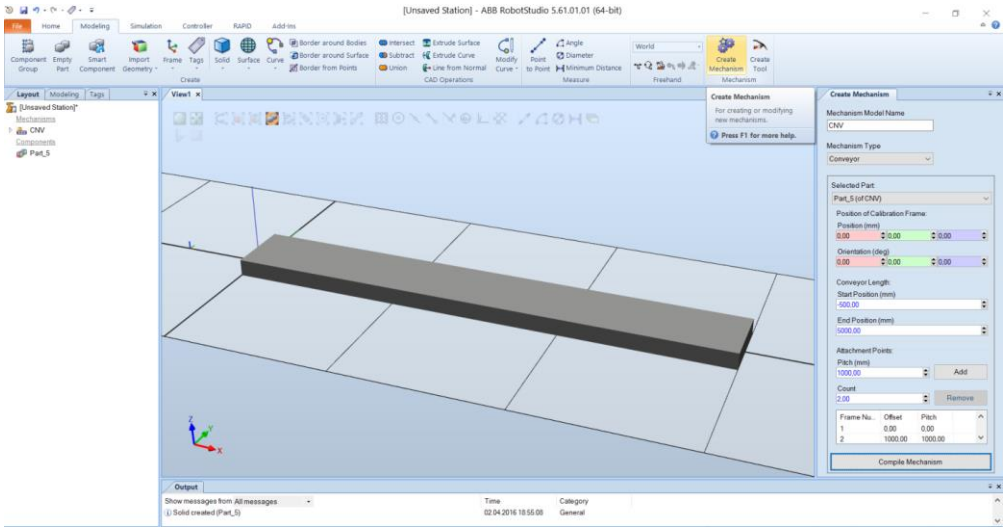


Figure 5.2. RobotStudio® - Define the conveyor mechanism

At Selected Part (2) choose the create box, chose a starting position and an end position). Choose a Pitch and a count and then Add (number 3 marked with red); data from the video's start position = -500 mm, end position 5000 mm, pitch 1000 mm and count = 2 (Figure 5.2).

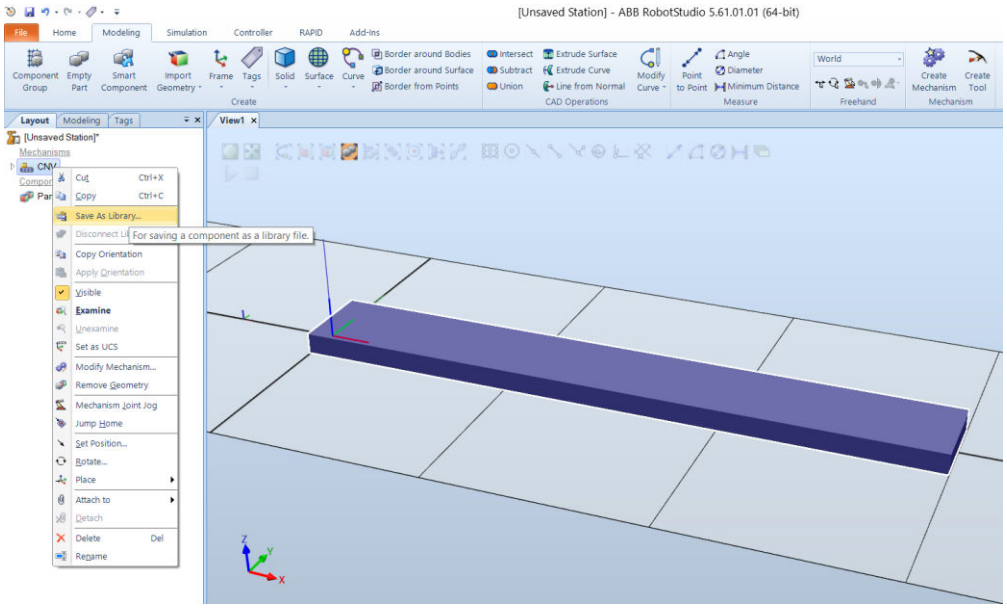


Figure 5.3. RobotStudio® - Save As Library

After all this data has been introduced, click on Compile Mechanism (number 4 marked with red) and your mechanism can be found in list on the left of the screen, in Layout (number 5 marked with red). From there you have to save it using Save As Library, where you have to choose a name (Figure 5.3). If all the steps have been correctly executed, the mechanism can be found in Import Library and User Library.

5.3. Programming/Setting up/Testing MultiMove systems

Programming MultiMove systems

If you want to develop or optimize programs for MultiMove systems you use MultiMove functions. This subchapter details the main workflow to program MultiMove systems with the help of RobotStudio®.

In oder to be able to use the MultiMove functions, one must possess the following [2]:

- A virtual controller that can run a MultiMove system
- The tools used by the system
- All coordinate systems

- All the paths the tool will move onward (these paths will be created in a workobject that pertain to a tool robot and that adhere to the work piece robot

If you want to create MultiMove programs utilizing the MultiMove function, you must complete the steps shown in Table 5.1.

Table 5.1. Typical and additional workflow for creating MultiMove programs [2]

Typical Action	Description
Set up the MultiMove	Select the robots and paths to use in the program
Test the MultiMove	Execute the motion instructions along the paths
Tune the motion behavior	Tune motion behavior, such as tolerances and constraints for TCP motions
Create the program	Generate the tasks for the robots
Additional Action	Description
Create Tasklists and Syncidents	The tasks and paths that shall be synchronized with each other
Add and update ID arguments to the instructions to synchronize	Add and update IDs for instructions in paths that already are synchronized. Add IDs to instructions in paths that have not yet been synchronized.
Add and adjust Sync instructions to the paths.	Add SyncMoveon/Off or WaitSyncTask instructions to the paths to synchronize and set their tasklist and Syncident parameters
Teach MultiMove instructions	It is also possible to jog all robots to the desired positions and then teach instructions to new synchronized paths.

Setting up MultiMove systems

In order to select the robots and paths in the station, that will be utilized for the MultiMove program, it is mandatory to make sure that all the robots of

the MultiMove program belong to the same system. After completing this step, follow items 1-10 shown below [2]:

1. Home tab → MultiMove → Setup tab below the MutliMove

work area

2. In the work area, press System config bar to expand the system configuration section
3. Select System box → select the system that contains the robots to program. The robots of the selected system are now displayed in the System grid (below the Select system box)
4. Select the check box in the Enable column (for each robot that will be used in the program)
5. For each robot specify whether it carries the tool or the work piece using the options in the Carrier column
6. Click the Path config bar for expanding the path configuration section (in the work area)
7. Select the Enable check box (for the tool robot) → press the expand button in order to display the robot's paths
8. Using Path name column select the order of the paths that are to be executed
9. Select the check box in the Enable column for each path that will be included in the program
10. Continue testing the MultiMove and, if necessary, tune the motion properties

Testing the MultiMove systems

This section refers to the motion instructions along the paths in accordance with the current setting on the setup of the MultiMove.

Basically, it refers to setting the robot's start position and testing its movements along the path.

In order to test the paths, one must [2]:

- Jog the robots to a good start position
- Home tab → MultiMove → Test tab (bottom of the MultiMove work area) – displays the test area

- (If wanted) press the Stop at end check box (this ensure that the simulation stop subsequently to moving along the paths). If the Stop and end is not pressed, the simulation will loopingly continue until clicking Pause
- In order to simulate the motions along the paths, click Play. If the motions are satisfactory, advance developing multimove paths. However, if the motions are not satisfactory, choose to do one of the following actions (Table 5.2):

Table 5.2. Actions to adjust motions [2]

Action	Description
Examine the robots' positions for critical targets	Press Pause and use the arrow buttons to move to one target at a time
Jog the robots to new start positions	The cause of changed motions are new start positions. Taking this into consideration, please avoid positions near the robots' joint limits
Go to the Motion Behavior tab and remove constraints	For the motion properties, the default setting is no constraints. If this has changed, there might exist limited motions.

Workshop 6: Create a smart component tool

Necessary knowledge

Workshop 5 completed

Workshop 6 summary

At the end of this workshop, the students should know how to:

- Create a Smart Component
- Add new components and signals used to define a Smart Component
- Make the connections between the added components and signals

6.1. Aim of the workshop

The aim of this workshop is for the students to know what a smart component is, how to define and how to use it. Furthermore, they will learn how to work with signals in RobotStudio® and how to make the connections between the tool's elements in order for them to work as a real tool.

6.2. The smart component's definition

Nowadays, more and more persons are interested in the new developed technology and is eager to know everything it has to offer. This interest also manifests itself in wanting to understand how each new device and newly developed gadget works. Therefore, this workshop aims to teach students how to program a tool in order for it to work as it does in reality. It aims at teaching how to connect all the tool's elements and how to define the necessary sensors.

This workshop has the aim to define a tool as a smart component that is working with vacuum. In the beginning, an industrial robot and the tool that will be defined are imported. One of the important aspects is for the tool to be saved as library.

The next step is to create the smart component. The Smart Component option can be found in the Modeling menu. Once the option is accessed, a window will open (Figure 6.1). In the Layout menu (indicated with red in Figure 6.1), using drag and drop, place the imported tool in the smart component object.

Once the tool is set to be a smart component, in the right window of the screen it can be seen as a Child component (Figure 6.2). Right click on the smart component and select Set as Role.

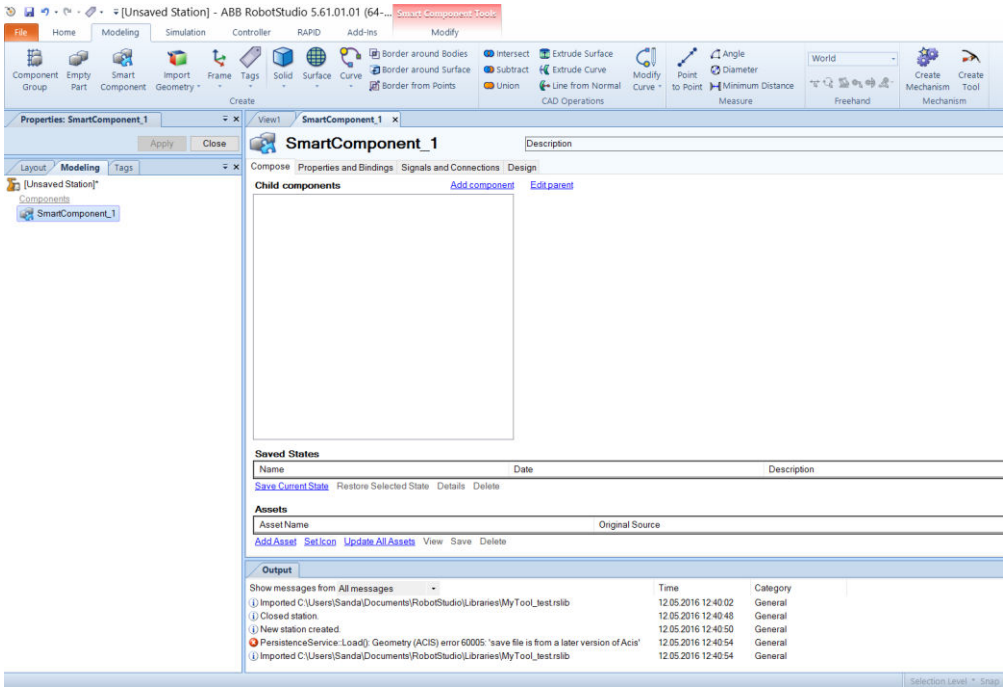


Figure 6.1. RobotStudio® - Smart Component option

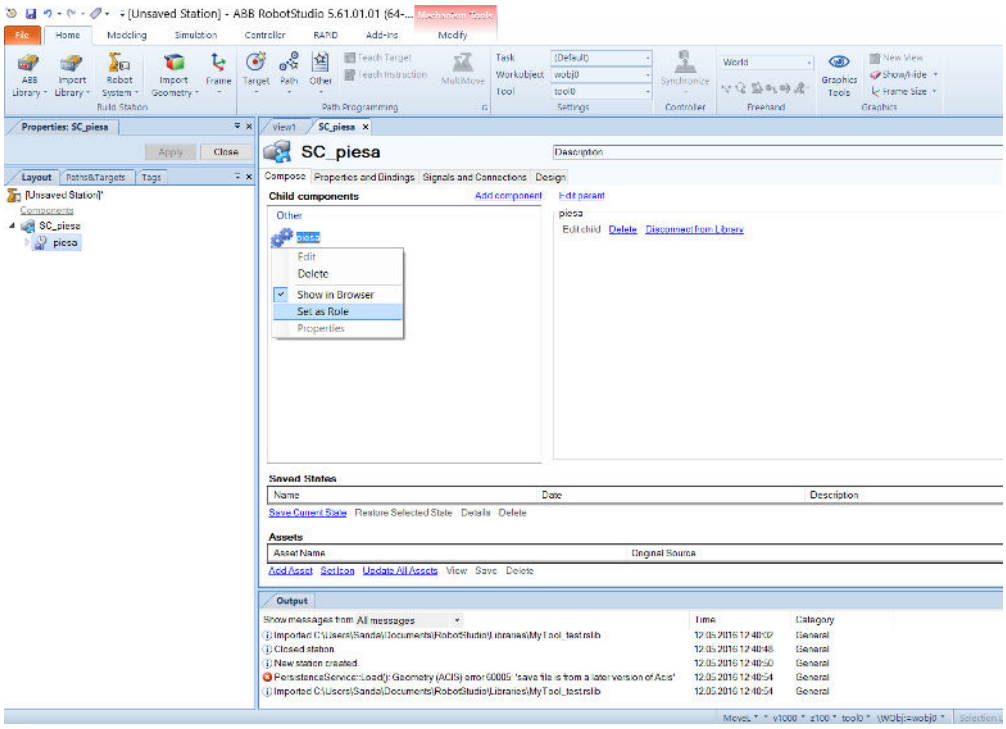


Figure 6.2. RobotStudio® - The Smart Component's definition

The next step is to add to the tool (that actually is an object) different components, in order for it to be defined as a smart component. The first component is called Line Sensor (click on Add component > sensors; see Figure 6.3). It is added in order to define a sensor inside the object, that will later be programmed. It is not enough to add it, you must also define it (Figure 6.4). Afterwards, click Apply.

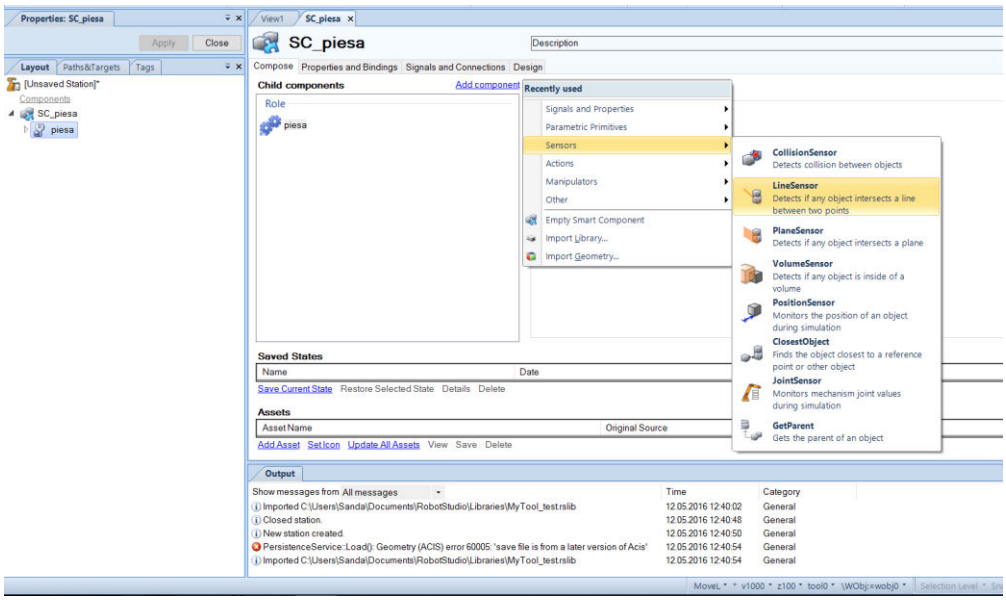


Figure 6.3. RobotStudio® - Add a Line Sensor

This tool is using the vacuum technology. This is why it needs to have the attach and detach functions. Taking this into consideration, the next steps are to add the two components, Attacher and Detacher. These two components can be found in the Add component menu, in Actions (Figure 6.5). The only option that must be set here, is for the Parent to be set from the tool list as a smart component. Then click Apply.

Attach function will work if it is connected to the sensor. In this case, the next step is to create a connection between the attach function and the Line sensor from Properties and Binding (Figure 6.5). Select Add Binding (Figure 6.6) and then set the characteristics market with blue and click OK. This connection can be observed in Property Bindings.

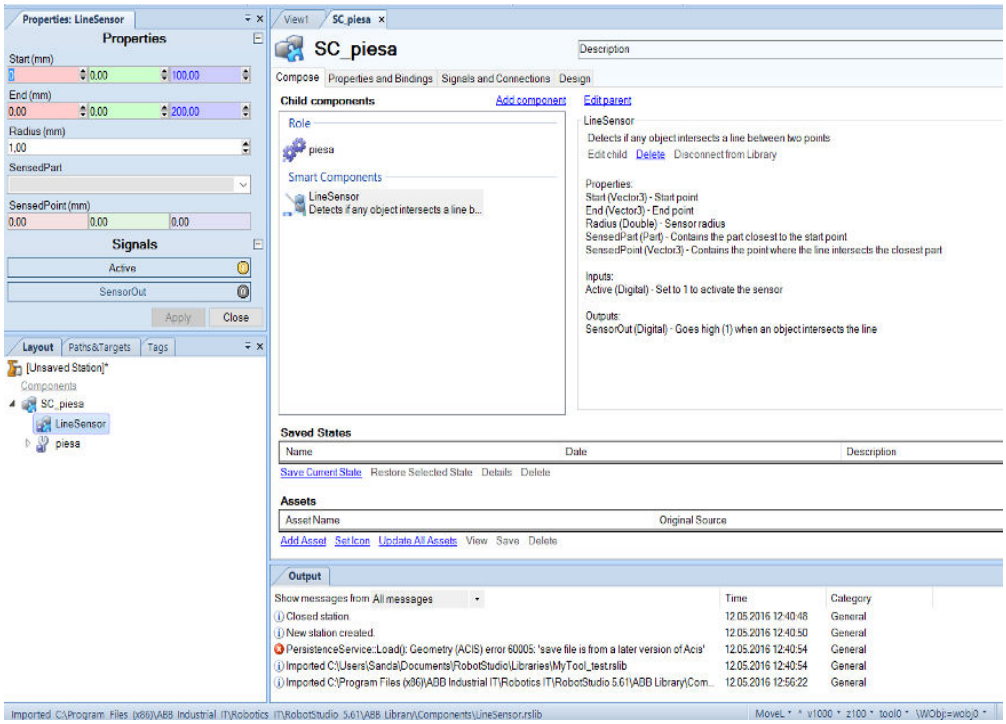


Figure 6.4. RobotStudio® - Define a Line Sensor

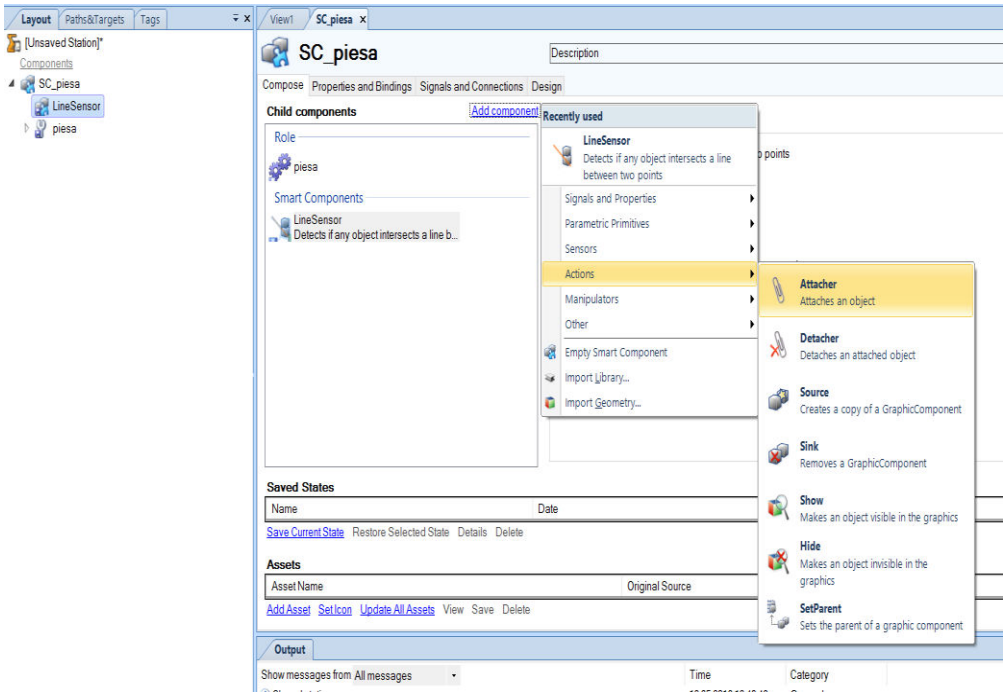


Figure 6.5. RobotStudio® - Attach and Detach components

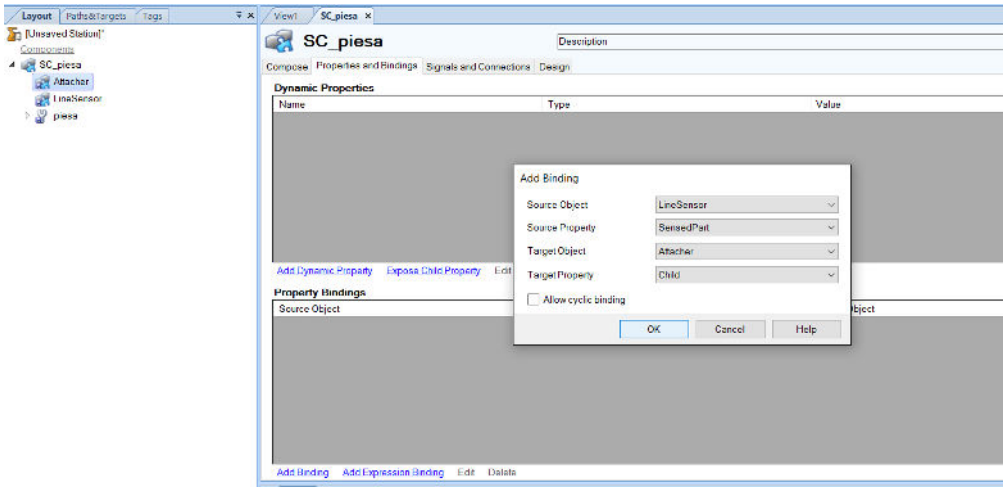


Figure 6.6. RobotStudio® - Add a binding for the attach function

Once the attach function was defined, the next step is to define de detach function, too. As already mentioned, the detach function will be created from the Compose window, Add component, and, in Actions, the component Detacher will be selected. Also, for this component a binding will be created. The characteristics are presented in Figure 6.7.

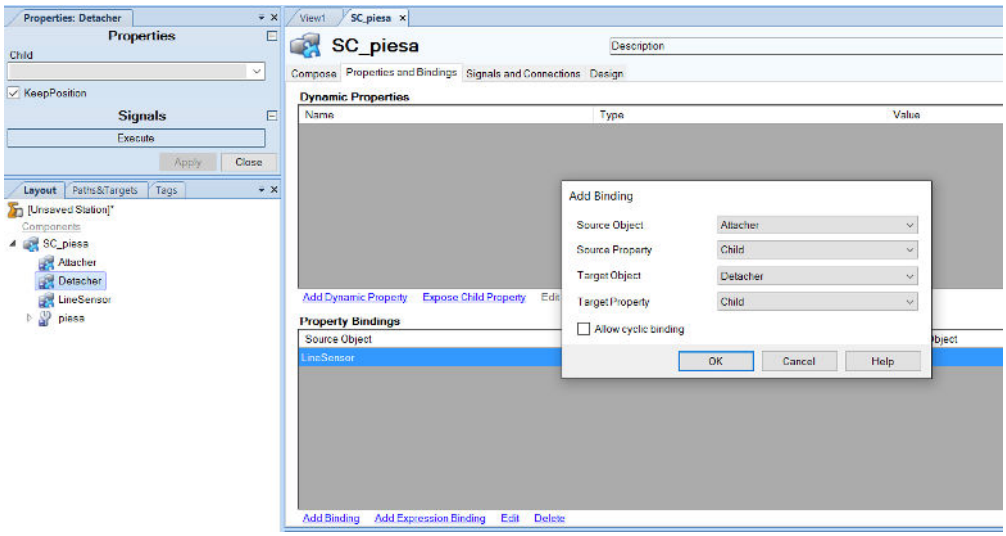


Figure 6.7. RobotStudio® - Add a binding for the detach function

The functions are needed to be defined. The next step is to create a link between the Line Sensor and these functions. For this, from the Signals and Connections menu, the needed signals will be added, in the beginning a digital

input signal (Figure 6.8) and then a digital output signal (Figure 6.14). From the Add I/O Connection menu the connections between the elements will be made (Figure 6.9, Figure 6.10). The steps are presented in the next figures.

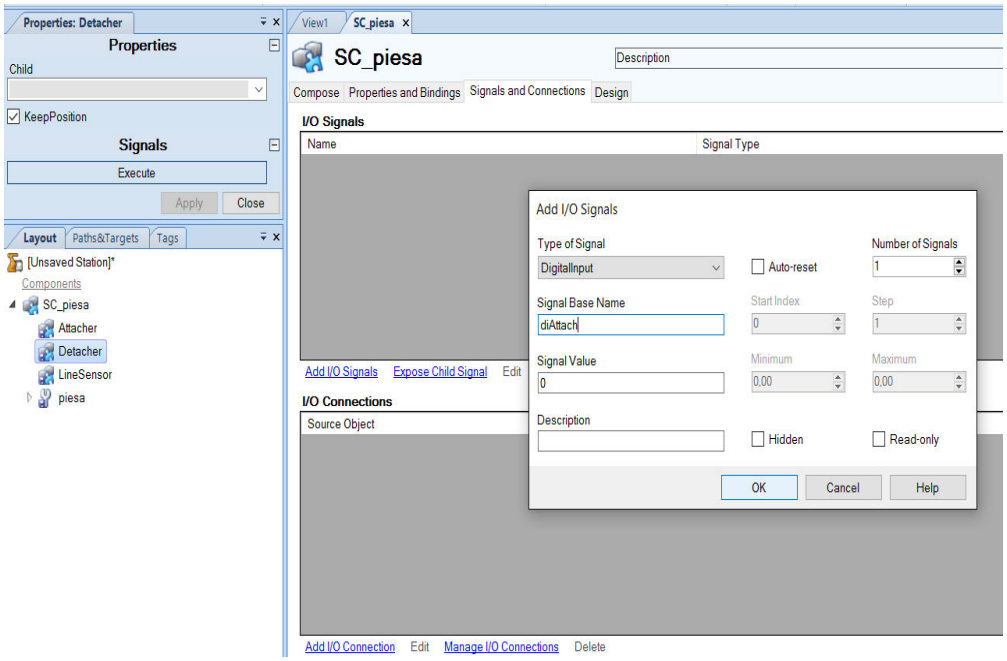


Figure 6.8. RobotStudio® - Add a digital input signal

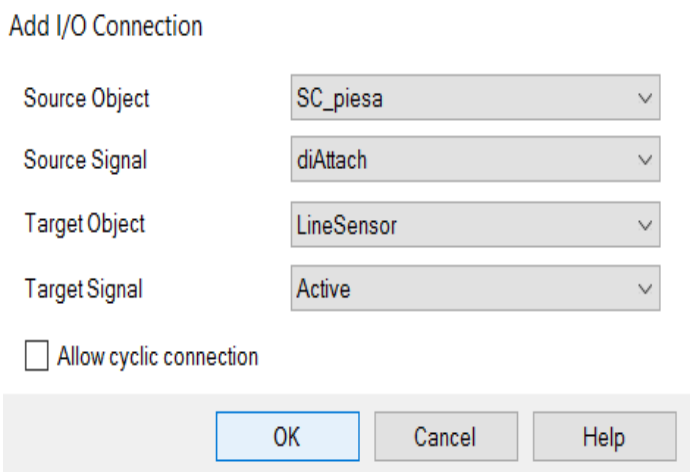


Figure 6.9. RobotStudio® - Connection between the tool that will be a smart component and the line sensor

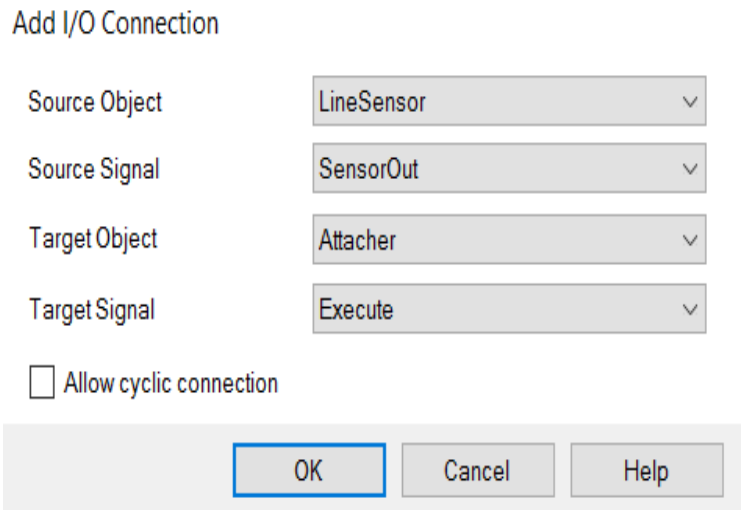


Figure 6.10. RobotStudio® - Connection between the line sensor and the attach function

From the Compose menu, a Logic Gate will be added. This can be found in the Add component menu, in Signals and Properties. This is a logic function that has certain properties shown in Figure 6.11. Once this option is defined, continue with defining the connections between the elements (Figures 6.12, 6.13, 6.1, 6.15, 6.16, 6.17).

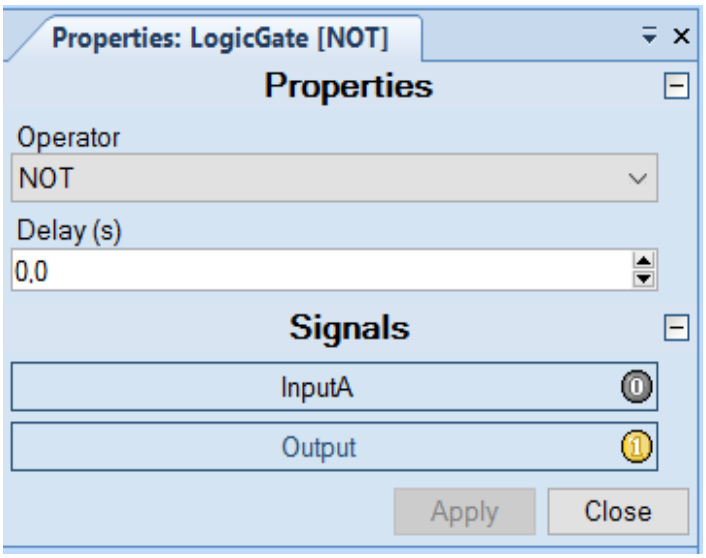


Figure 6.11. RobotStudio® - Logic Gate properties

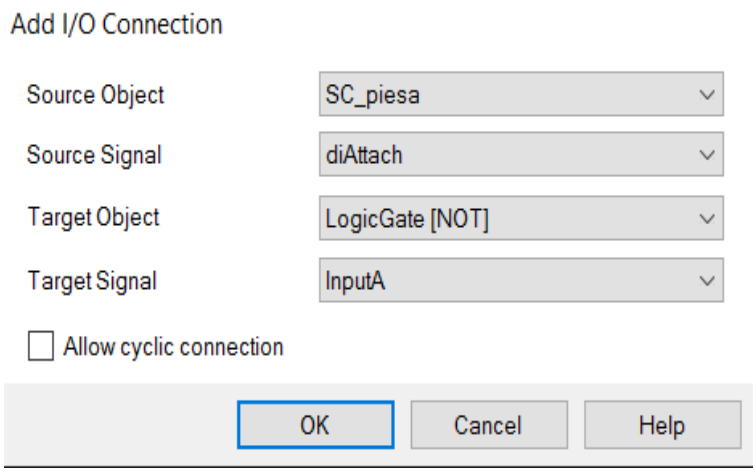


Figure 6.12. RobotStudio® - Connection between the tool and Logic Gate

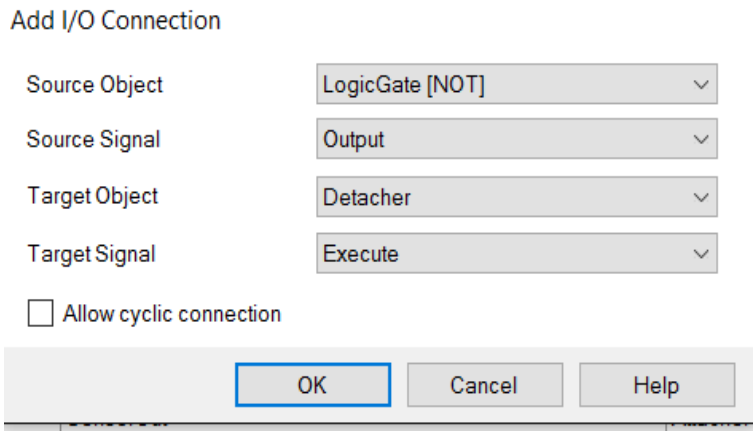


Figure 6.13. RobotStudio® - Connection between Logic Gate and the detach function

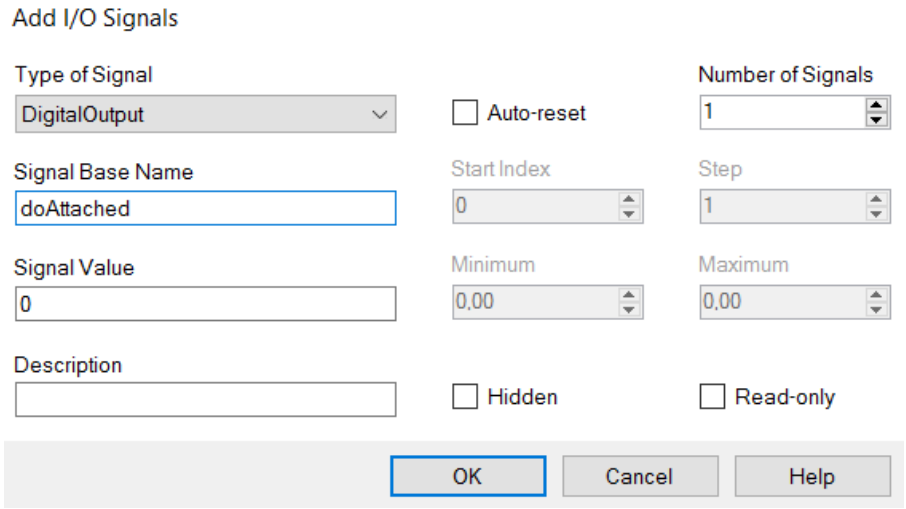


Figure 6.14. RobotStudio® - Add a digital output signal

It is a known fact that a sensor must be reset before any other operation starts. Knowing this, a logic component to set the reset will be added. This it is added from the Compose menu, Add component, Signals and Properties. This component is called LogicSRLatch. The connections' definitions are shown in the next figures.

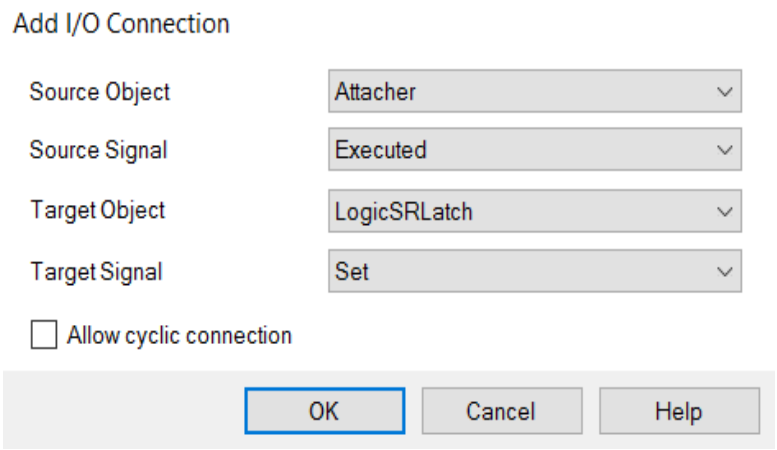


Figure 6.15. RobotStudio® - Connection between the attach function and LogicSRLatch

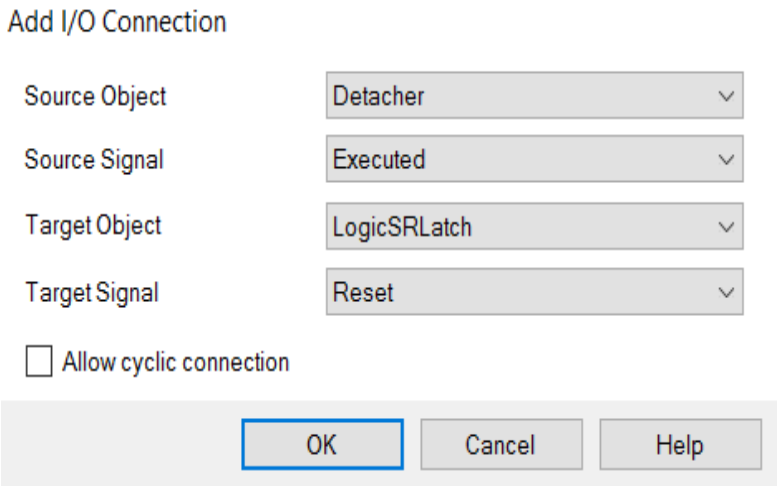


Figure 6.16. RobotStudio® - Connection between the detach function and LogicSRLatch

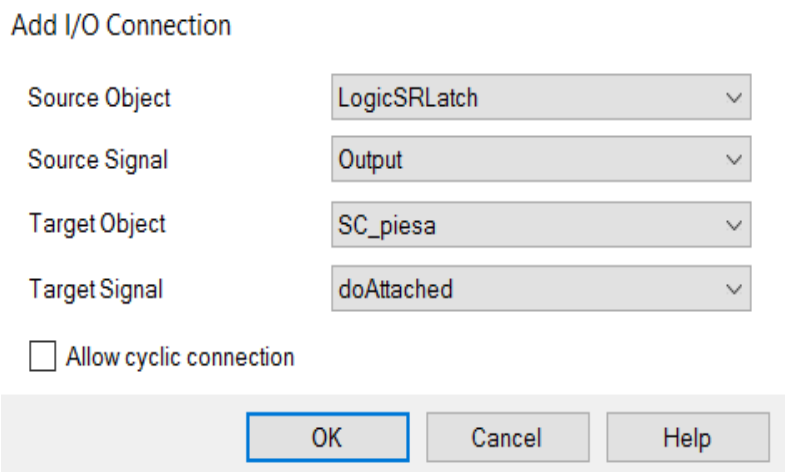


Figure 6.17. RobotStudio® - Connection between LogicSRLatch and the tool that will be a smart component

In the Smart Component window, in View/Design, all the connections that have been made between the created components and signals can be seen.

To check if the tool is well defined, go to the View window of Robot Studio, attach the tool to the robot, import or create a box from the Modeling menu and check if the vacuum function of the tool is working. With Jog Linear and Jog Reorient (Home menu, Freehand), position the robot with the tool on the object and on the left side of the window, in signals, set de digital input signal to make sure it is active. Once this is active, having the 1 value, the digital output signal also has the 1 value. This signifies that the tool has been well defined as a smart component and it can be used further on, in other applications.

Remember, this example of a smart component's definition is a particular example used just to define a vacuum gripper.

Workshop 7: Create a path from a curve

Necessary knowledge

Workshop 6 completed

Workshop 7 summary

At the end of this workshop, the student should know how to:

- Create an autopath
- Use the autoconfiguration command
- Use the RAPID editor

7.1. Aim of the workshop

One of the aims of this workshop is for the students know how to easily create a path that contains lines and curves. Furthermore, the student learns how to edit an already created program in RobotStudio®.

7.2. Defining an Auto path

For this application, it is necessary to create a station. This station must contain a robot, a tool to perform the operation and a part, an object on which the robot will work on. The first step is to select the surface and to create a border around that surface, that will serve as a trajectory to follow by the robot. These two steps are presented in Figure 7.1 and 7.2. Once the surface is selected, click Create the “Border around the Surface”.

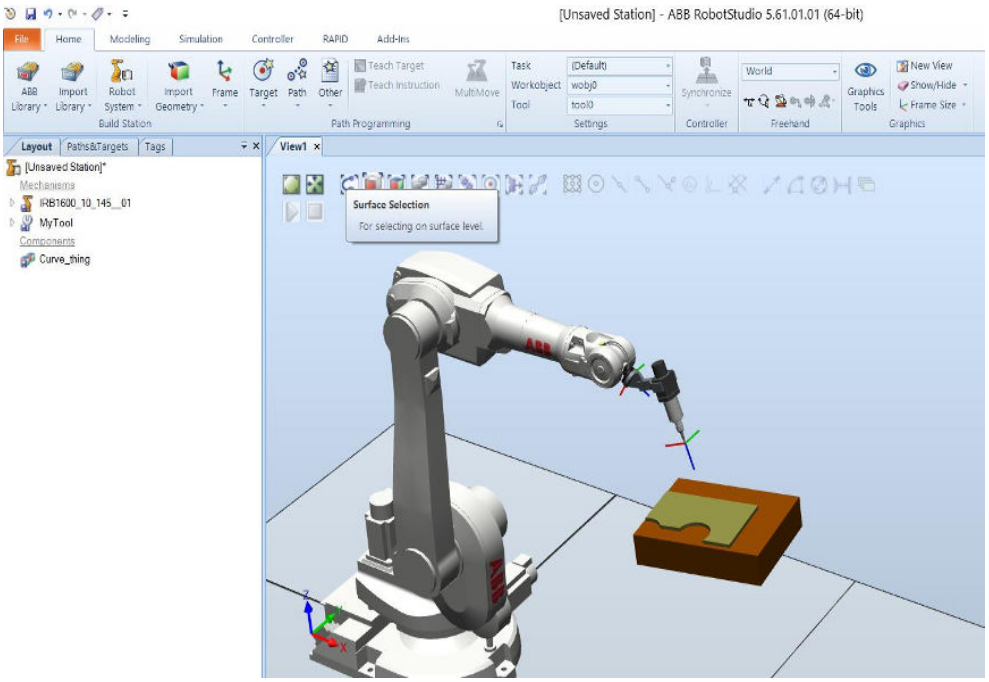


Figure 7.1. RobotStudio® - Surface selection

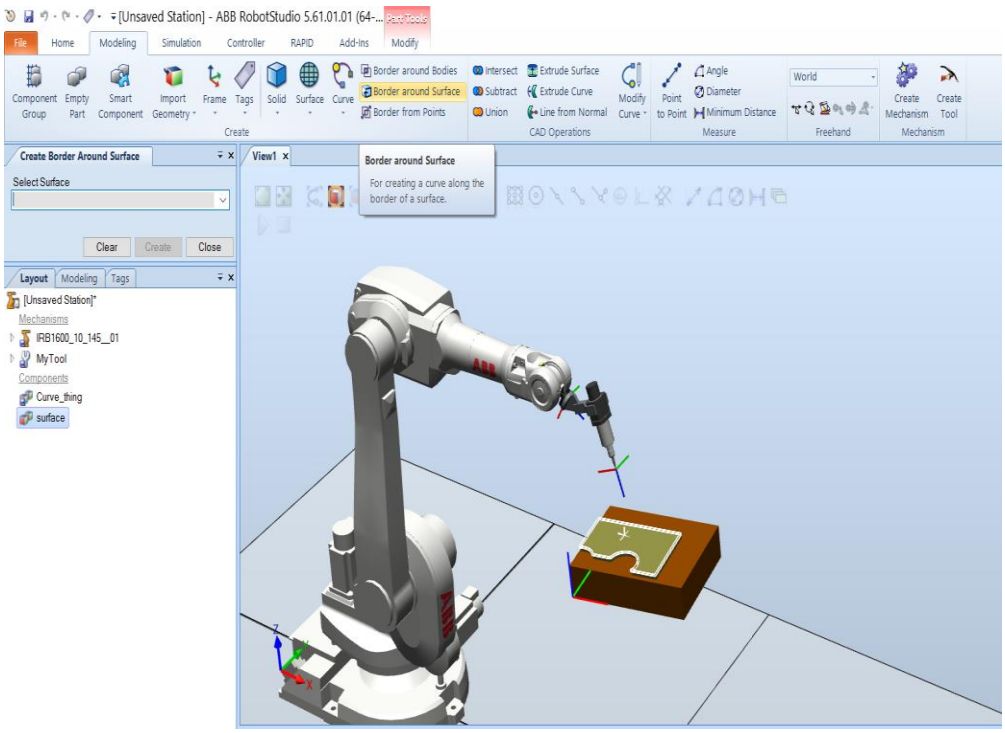


Figure 7.2. RobotStudio® - Border around surface

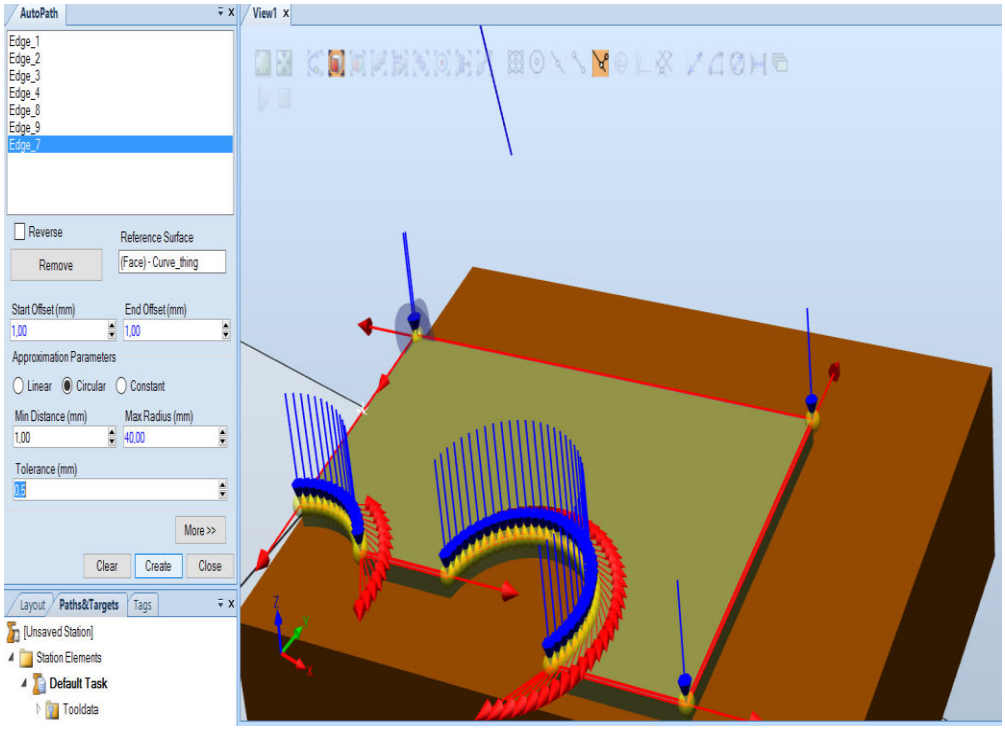


Figure 7.3. RobotStudio® - AutoPath function

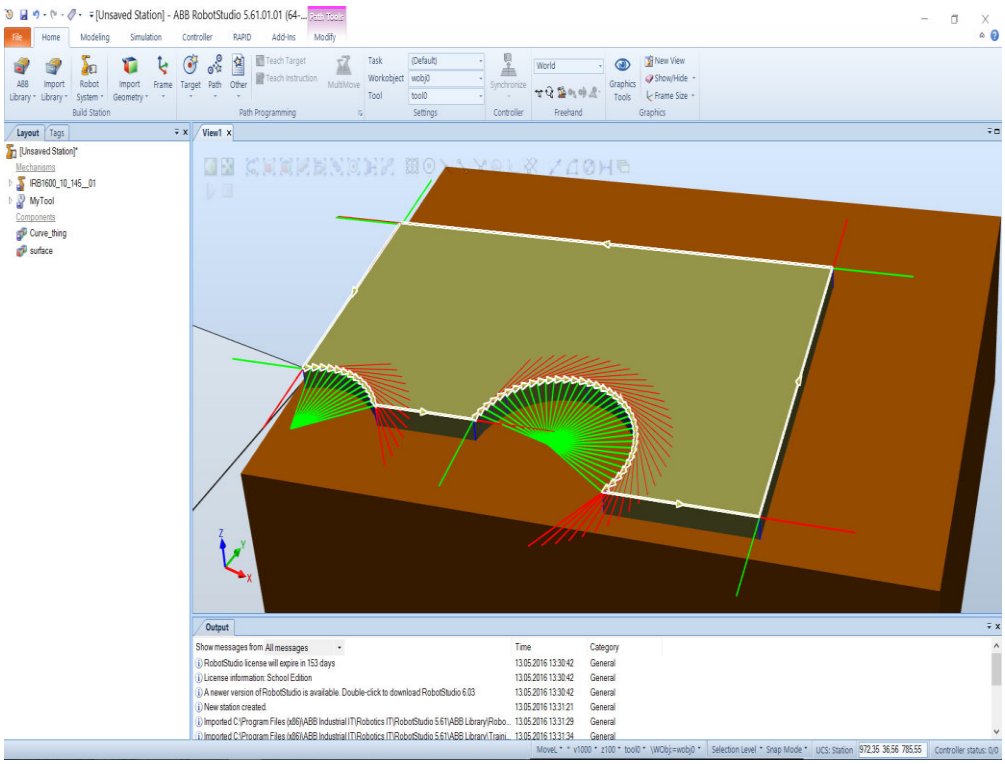


Figure 7.4. RobotStudio® - Frames of the targets that form the path

On the left part of the window, in the Layout menu a new part that represents the selected surface will be created. The next step is to create the path that will be automatically followed by the robot. This can be performed from the Home menu, Path, AutoPath. A new window will open (Figure 7.3). The trajectory that the robot will follow is also presented in this window. Once all the characteristics have been set, click Create. Once the path has been created, for each point that forms the path, its own reference frame will appear (Figure 7.4).

The next steps refer to the targets. These steps have been presented in the 2nd workshop and are in reference to position and orient the targets and defining their configurations or autoconfiguration.

7.3. Edit a RAPID program in RobotStudio®

RobotStudio® is the software through which offline programming for ABB robots can be performed. The program language used is called RAPID.

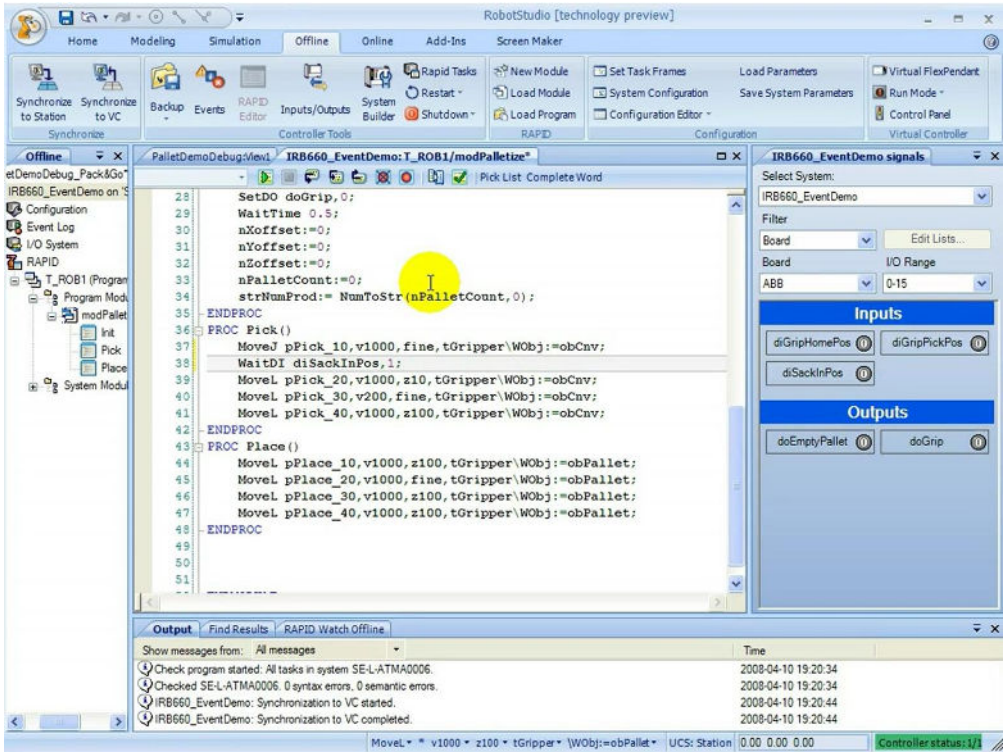


Figure 7.5. RobotStudio® - RAPID editor – example

It is very easy to program a robot, if you know the programming language. The programs can be written using the virtual teach pendant or the RAPID editor. Furthermore, you can edit any program you want to modify with it (Figure 7.5).

Workshop 8: Virtual FlexPendant from RobotStudio®

Necessary knowledge

Workshop 7 completed

Workshop 8 summary

At the end of this workshop, the student should know how to:

- Launch the virtual FlexPendant integrated in RobotStudio®
- Use FlexPendant menu

8.1. Aim of the workshop

The aim of this workshop is for the students to know how to use virtual FlexPendant from the RobotStudio®. Furthermore, the students will familiarise themselves with the virtual FlexPendant menus.

8.2. Virtual FlexPendant in RobotStudio®

ABB calls the teach pendant as FlexPendant. RobotStudio® provides us a virtual teach pendant. We are able to use the virtual FlexPendant in RobotStrudio® after defining a robotic system (robot arm and active controller). To launch the ABB virtual FlexPendant (Figure 8.2 and 8.3) go to Controller menu and select FlexPendant (Figure 8.1).

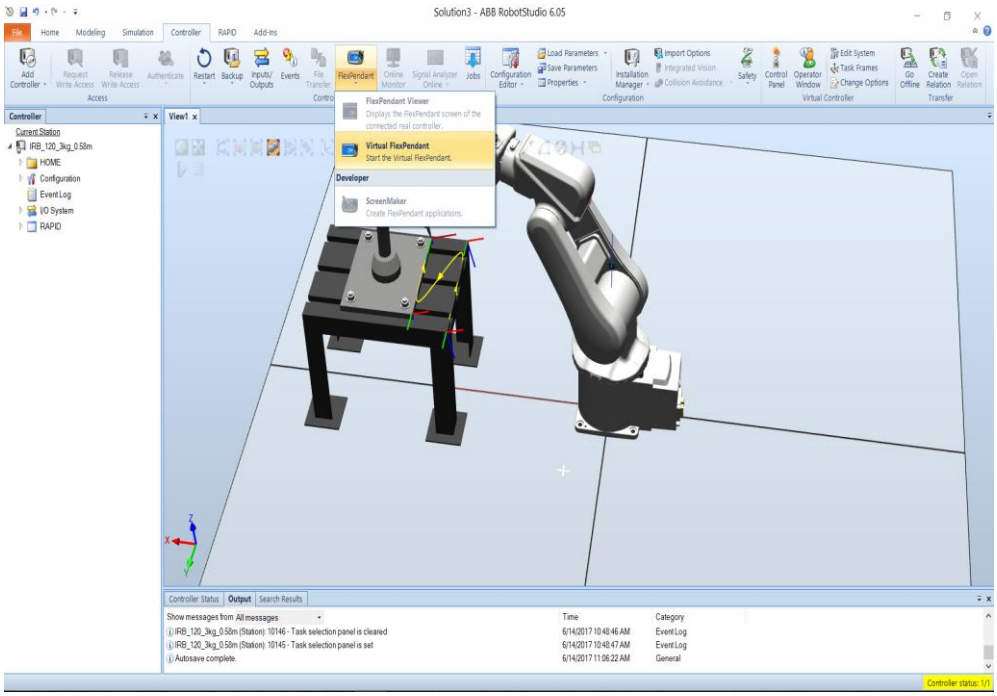


Figure 8.1. RobotStudio® - Launching the virtual FlexPendant in RobotStudio®

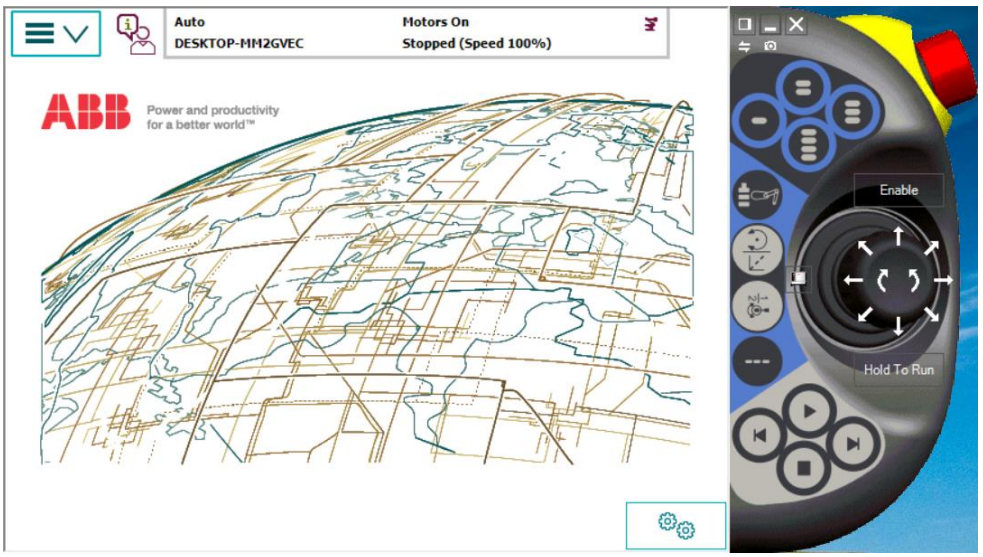


Figure 8.2. RobotStudio® - The FlexPendant starting window

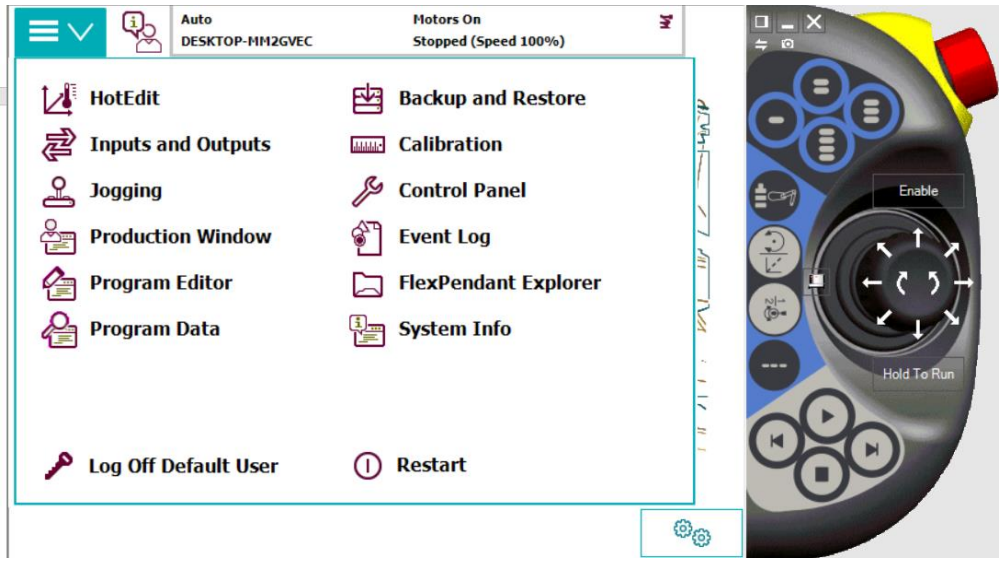


Figure 8.3. RobotStudio® - The FlexPendant menus illustration

HotEdit menu

HotEdit (Figure 8.4 and 8.5) is a function for tuning programmed positions. This can be done in all operating modes, even while the program is running. Both coordinates and orientation can be tuned. HotEdit can only be used for named positions of the defined robtarget. The functions available in HotEdit may be restricted by the user's authorization system (UAS).

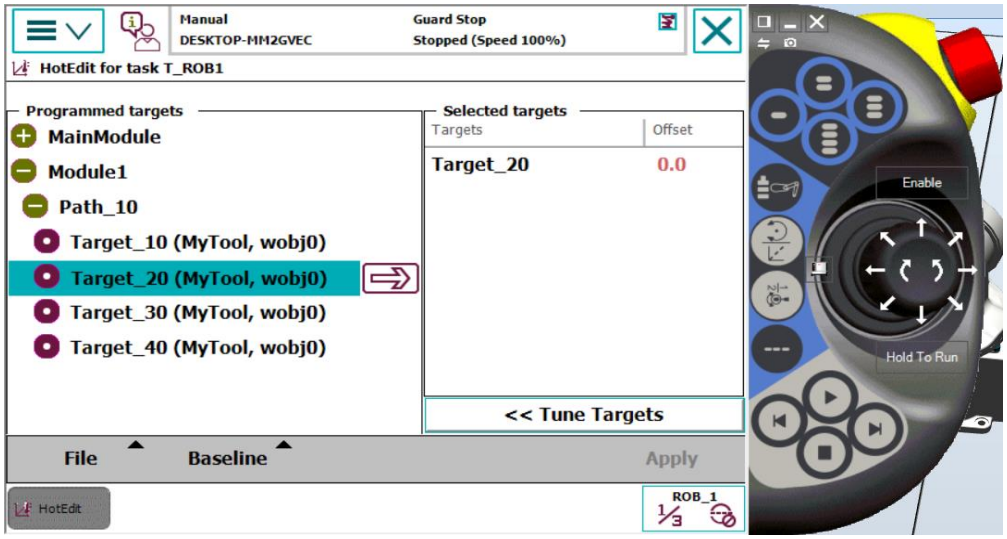


Figure 8.4. RobotStudio® - The HotEdit menu illustration

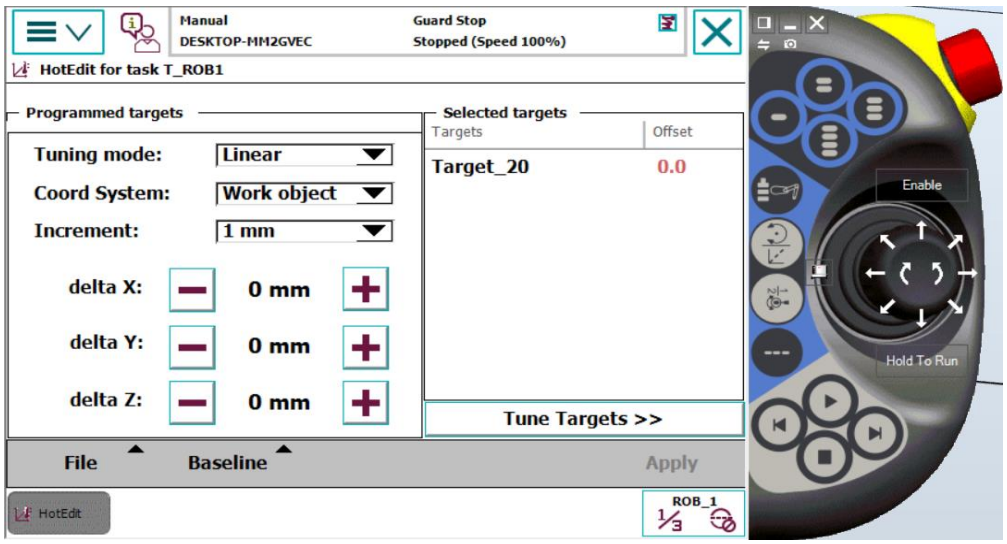


Figure 8.5. RobotStudio® - The HotEdit menu details – Tune Targets

The functions available in HotEdit menu are presented in Table 8.1.

Table 8.1. Functions in HotEdit

Target selections	Lists all named positions in a tree view. Select positions and add them to the section by tapping the arrow. Note that if a position is used in more than one routine, it will appear in all places used and any changes made to the offset will be the same for everywhere it is used.

Selected targets	Lists all selected positions and their current offset. Tap the trash can to the right of the position name to remove them from the selection
File	You can save and load selections of often used positions using the File menu. If your system uses UAS, this may be the only way to select positions for editing.
Baseline	The baseline menu is used to apply or reject changes to the baseline.
Tune targets	Tap Tune targets to display icons for editing the offset values (coordinates and orientation).
APPLY	Tap APPLY to apply changes made in the Tune targets menu. ***Note***: that this does not change the original values for the positions!

Inputs and outputs, I/O menu

Inputs and outputs, I/O, are signals used in the robot system. An I/O signal is the logical software representation of an I/O signal located on a fieldbus unit that is connected to a fieldbus within the controller. By specifying a signal, a logical representation of the real I/O signal is created. The signal configuration defines the specific system parameters for the signal that will control the behavior of the signal (Figure 8.6).

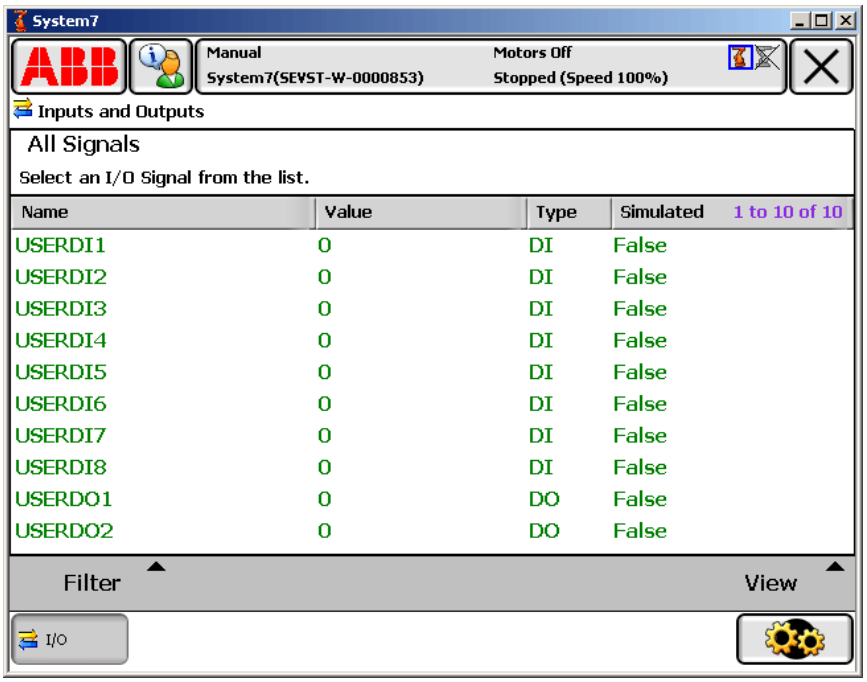


Figure 8.6. RobotStudio® - Inputs and outputs used in the robot system

Jogging menu

The Jogging functions are found in the Jogging window (Figure 8.7 and 8.8). The most commonly used are also available under the Quickset menu.

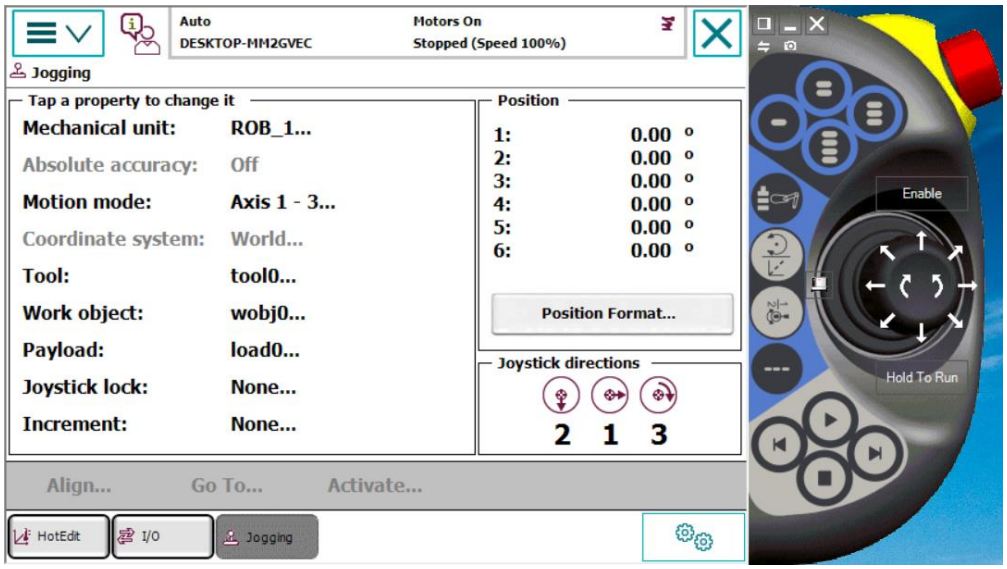


Figure 8.7. RobotStudio® - The Jogging menu illustration

The functions available in Jogging menu are presented below:

Table 8.2. Functions in Jogging menu

Mechanical unit	Select active mechanical unit
Absolute accuracy	**Absolute Accuracy: Off** is default. If the robot has the *Absolute accuracy* option, then **Absolute Accuracy: On** is displayed.
Motion mode (Figure 8.xz)	Select motion mode, described in section
Coordinate system	Select coordinate system
Tool	Select tool
Work object	Select work object
Payload	Select payload
Joystick lock	Select locking joystick directions,
Increment	Select movement increments,
Position	Displays each axis position in relation to the selected coordinate system.
Position format	Select position format
Joystick directions	Displays current joystick directions, depending on setting in Motion mode.

Align...	Align the current tool to a coordinate system
Go To...	Move the robot to a selected position/target.
Activate...	Activate a mechanical unit.

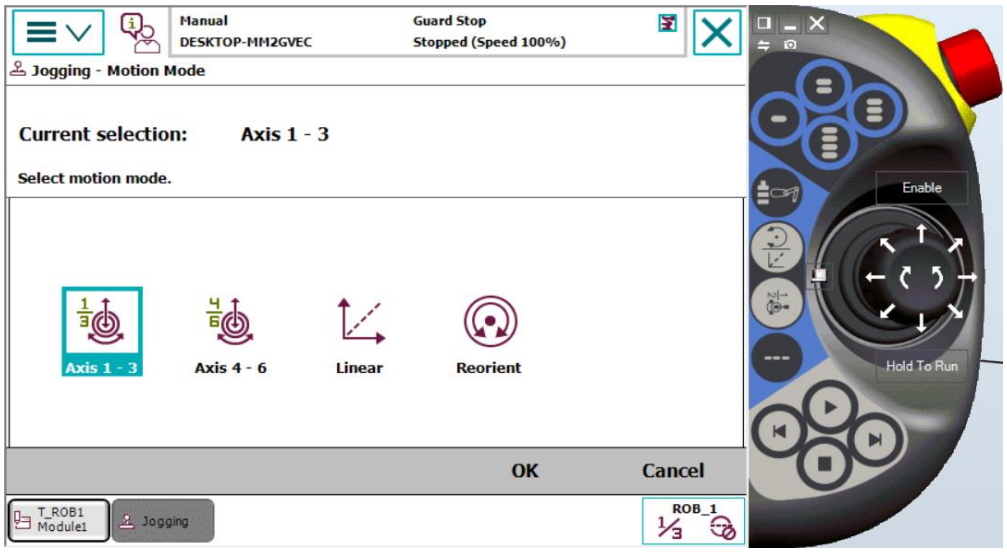


Figure 8.8. RobotStudio® - The Jogging menu – Motion Mode

Production window

The Production window (Figure 8.9) is used to view the program code while the program is running.

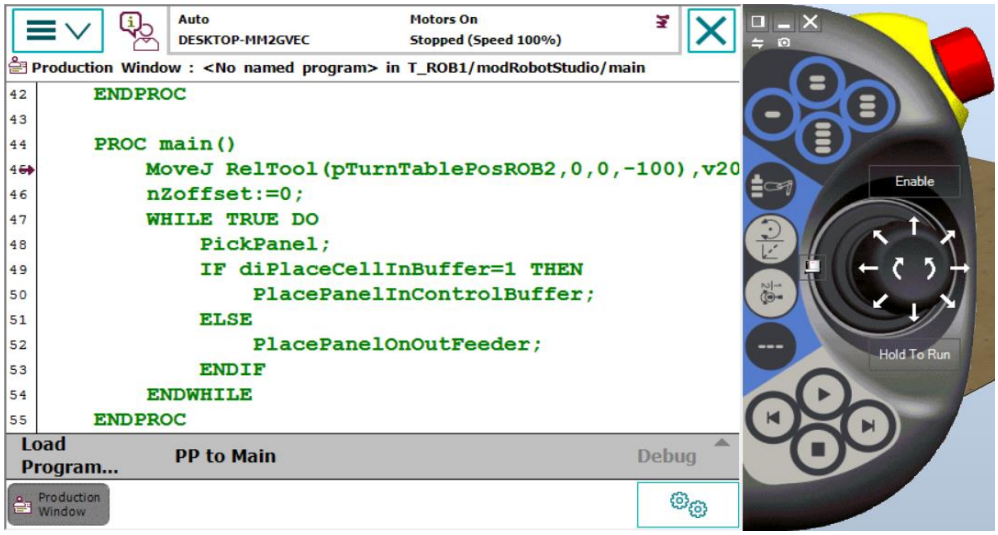


Figure 8.9. RobotStudio® - The Production menu illustration

The functions available in Production menu are presented below:

Table 8.3. Functions in Production menu

Load Program...	load a new program.
Move PP to Main	move the program pointer to the routine main
Debug	• **Modify Position** • **Show Motion Pointer** • **Show Program Pointer** • **Edit Program**. Debug is only available in manual mode.

Program editor

The Program editor (Figure 8.10) is where you create or modify programs. You can open more than one window of the Program editor, which can be useful when working with multitasking programs, for instance. The Program editor button in the task bar displays the name of the task.

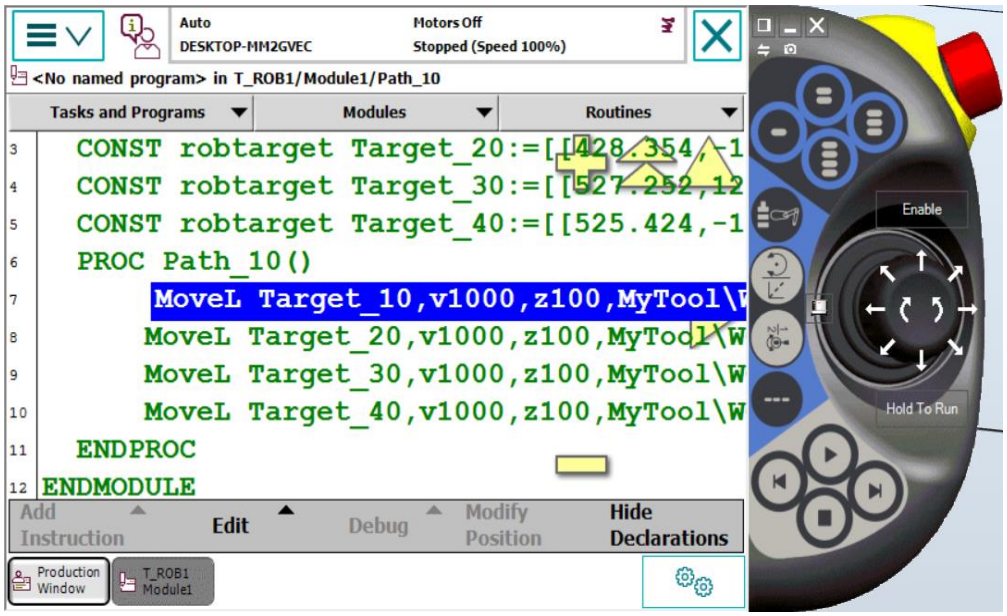


Figure 8.10. RobotStudio® - The Program Editor menu illustration

The functions available in Program Editor window are presented below:

Table 8.4. Functions in Editor window

Tasks and programs	Menu for program operations
Modules	Lists all modules,
Routines	Lists all routines,
Add instruction	Opens instruction menu,
Edit	Opens edit menu,
Debug	Functions for moving the program pointer,
Modify position	Modifying positions by jogging the robot to the new position
Hide declarations	Hide, for example, constant or variable declaration

Program data

The Program data view (Figure 8.11) contains functions for viewing and working with data types and instances. You can open more than one window of the Program data, which can be useful when working with many instances or data types.

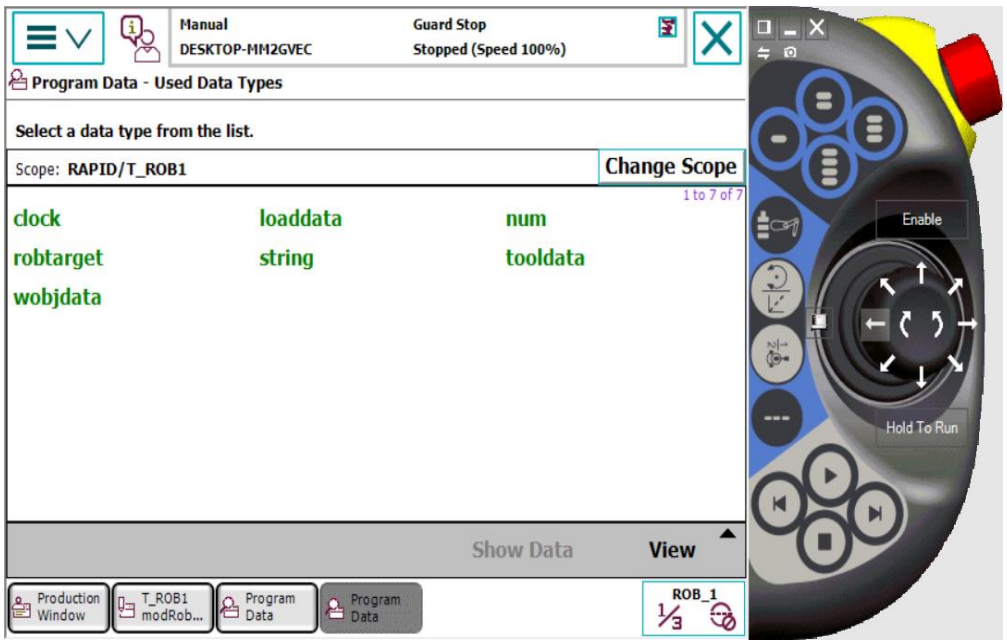


Figure 8.11. RobotStudio® - The Program Data menu illustration

The functions available in Program Data window are presented below:

Table 8.5. Functions in Program Data window

Change scope	changes scope of data types in the list
Show data	shows all instances of the selected data type
View	shows all or only used data types.

The Quickset menu

The QuickSet menu (Figure 8.12) provides a quicker way to change among other things jog properties rather than using the Jogging view. Each item of the menu uses a symbol to display the currently selected property value or setting. Tap the Quickset button to display available property values.

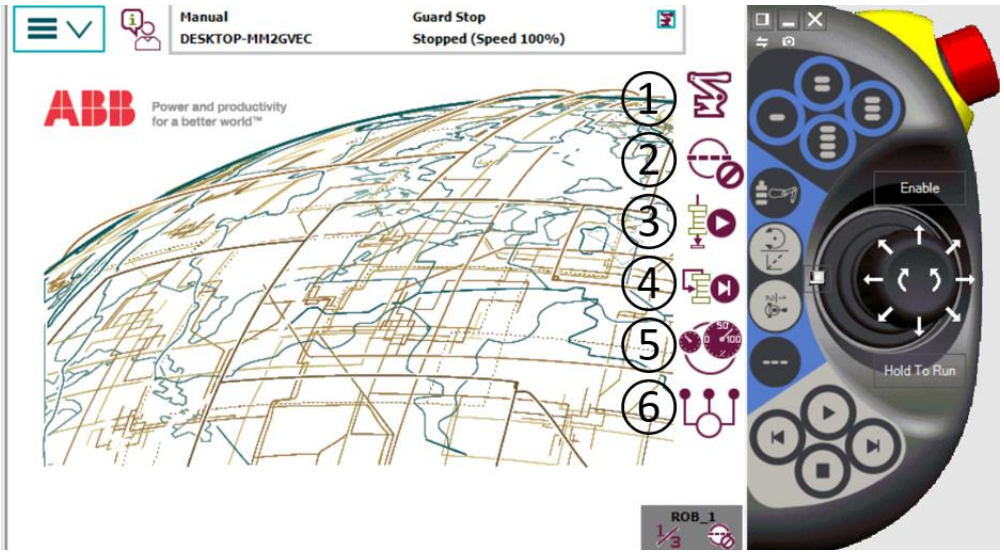


Figure 8.12. RobotStudio® - The buttons in the Quickset menu

The table 8.1 describes the buttons in the Quickset menu.

Table 8.6. The buttons in the QuickSet menu

① Mechanical unit,	④ Step Mode,
② Increment,	⑤ Speed,
③ Run Mode,	⑥ Tasks (to stop and Start)

Workshop 9: Creating a robotic station using RobotStudio®

Necessary knowledge

Workshops 1, 2, 3, 4, 5, 6, 7,8 completed

Workshop 9 summary

At the end of this workshop, the student should know how to:

- Design, simulate and program a robotic system using RobotStudio®

9.1. Aim of the workshop

The aim of this workshop is for the students to know how to develop and simulate a robotic cell/ system with one or more industrial ABB robots using RobotStudio®.

9.2. Creating a robotic station using RobotStudio®

When following the next steps, a person will be able to develop and simulate a robotic system using RobotStudio®:

Step 1: Select and integrate a robot from the RobotStudio® database and integrate it in the scene

Step 2: Select a tool (from the RobotStudio® database or a user defined tool) and attach it to the robot

Step 3: Import the needed auxiliary equipment (ex. the conveyor defined within the workshop 5 or a part positioner or other mechanism)

Step 4: Define the controller for the integrated robot and auxiliary equipment

Step 5: Generate some 3D models that will be used as workobjects within the robotic system using RobotStudio® facility (ex. 3D cubes)

Step 6: Define the target points (position and orientation of the tool in the targets) that the robot has to "touch"

Step 6: Orient the tool in each targetpoint and find a suitable configuration of the robot's structure within each targetpoint

Step 7: Generate the path(s) that the robot has to "follow"

Step 8: Simulate the movement of the robot along the generated path(s)

Step 9: View the generated RAPID program

Each of the eight above steps will be detailed by a set of figures (captured from RobotStudio®) for exemplification and better understanding.

Step 1: Select and integrate a robot from the RobotStudio® database and integrate it in the scene

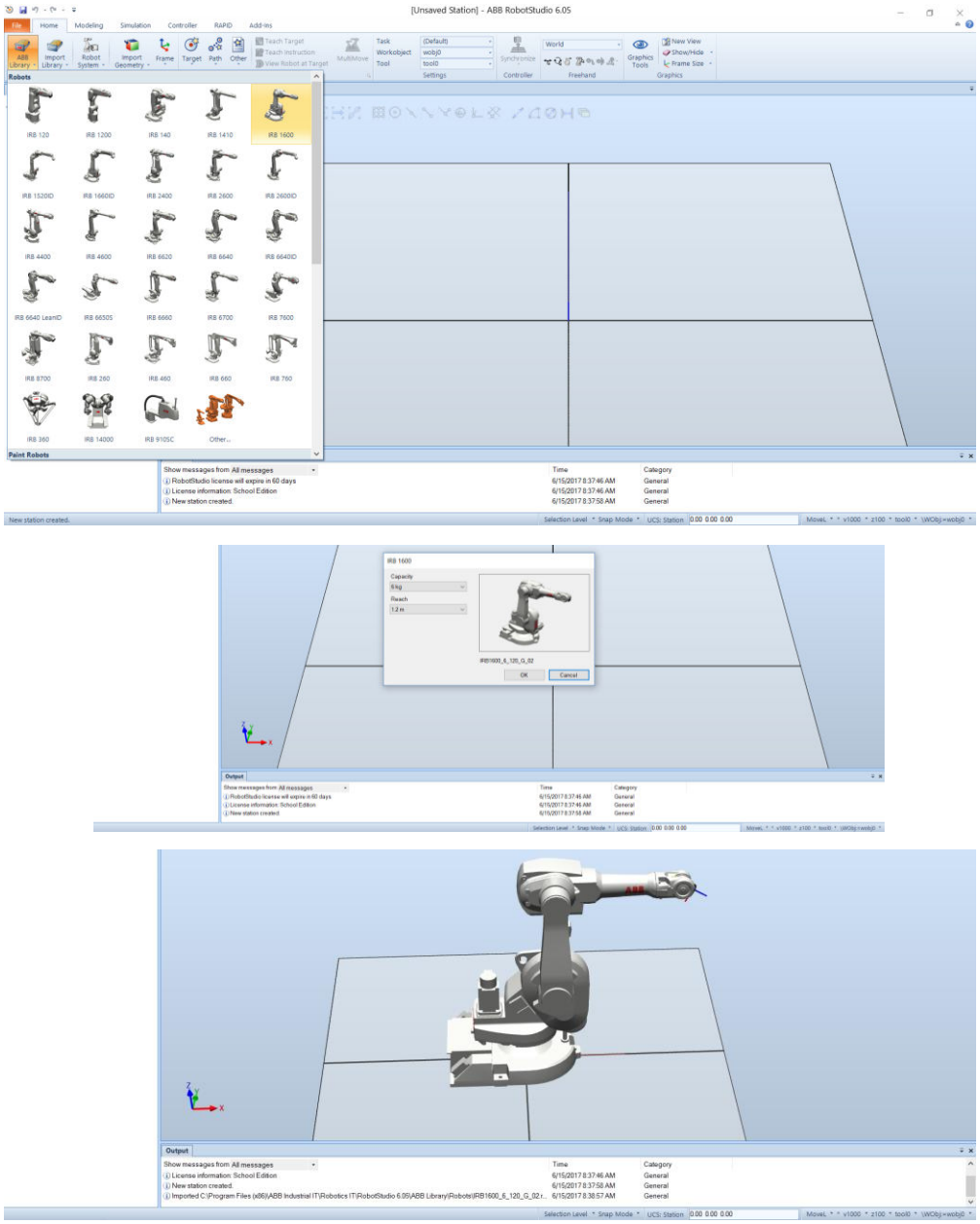


Figure 9.1. RobotStudio® - Select and integrate a robot in the scene

Step 2: Select a tool (from the RobotStudio® database or a user defined tool) and attach it to the robot

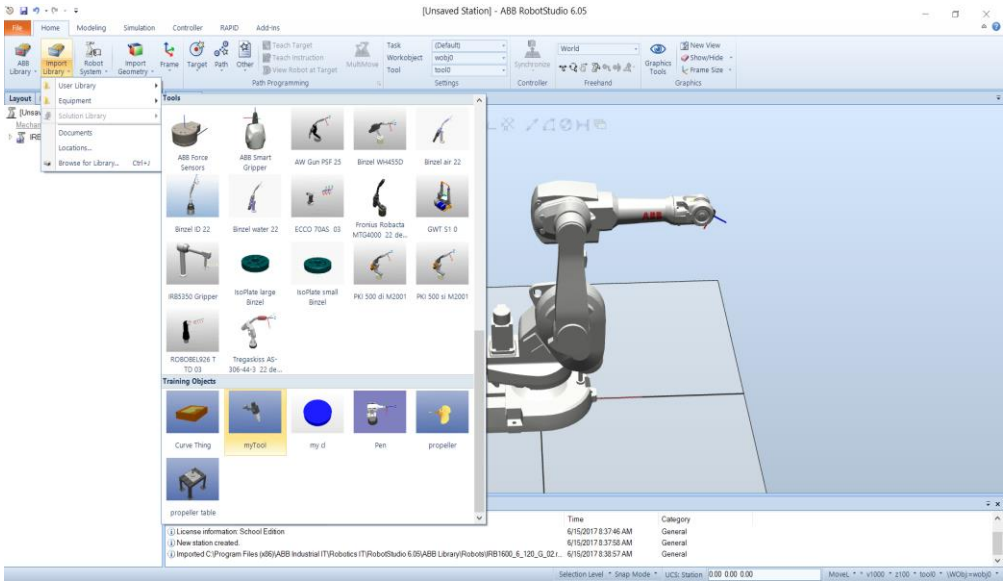


Figure 9.2. RobotStudio® - Select a tool for the robot

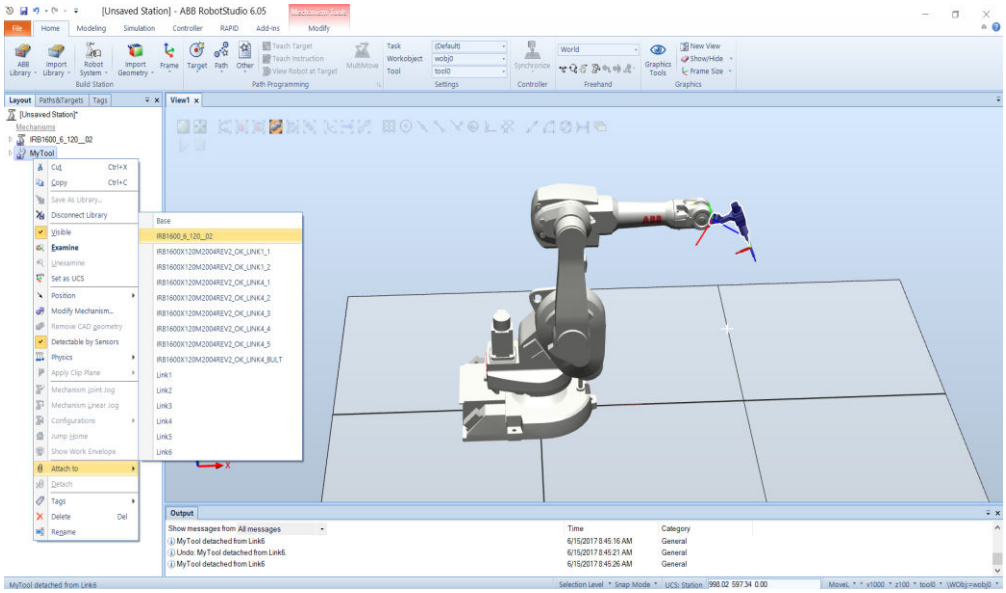


Figure 9.3. RobotStudio® - Attach the tool to the robot

Step 3: Import the needed auxiliary equipment (ex. the conveyor defined within the workshop 5 or a part positioner or other mechanism)

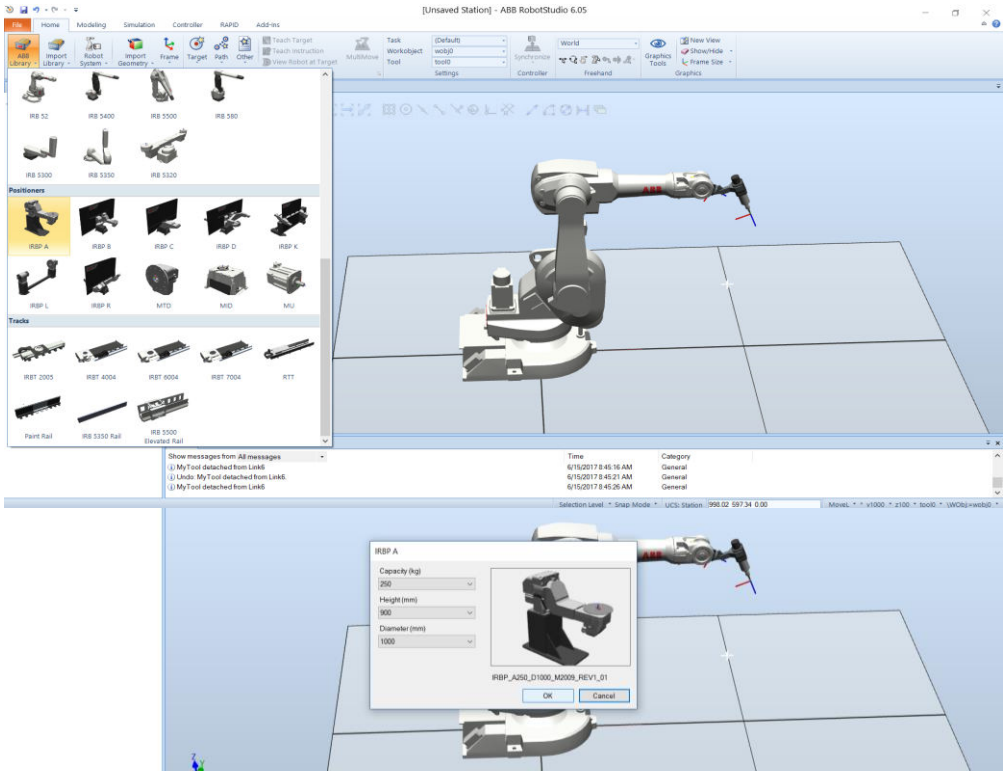


Figure 9.4. RobotStudio® - Import a part positioner form the ABB library

Or

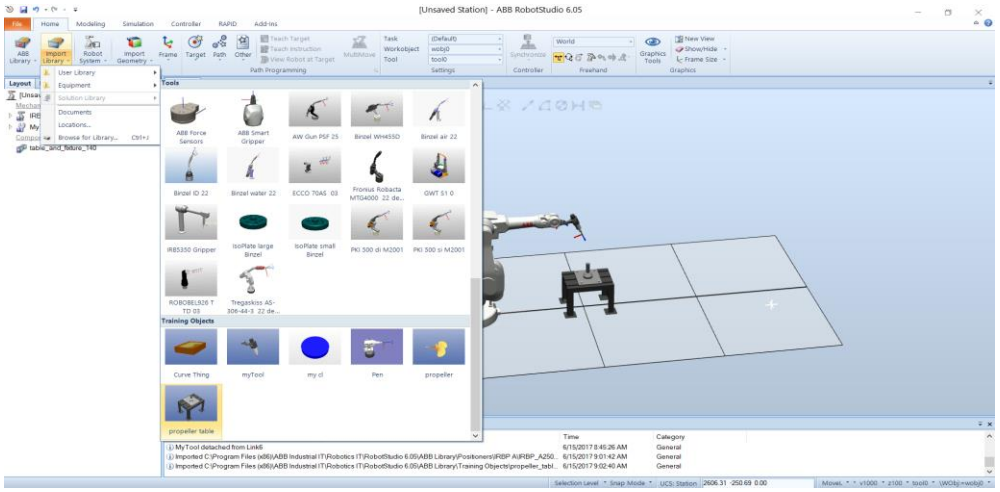


Figure 9.5. RobotStudio® - Import a workobject form the library

Step 4: Define the controller for the integrated robot and auxiliary equipment

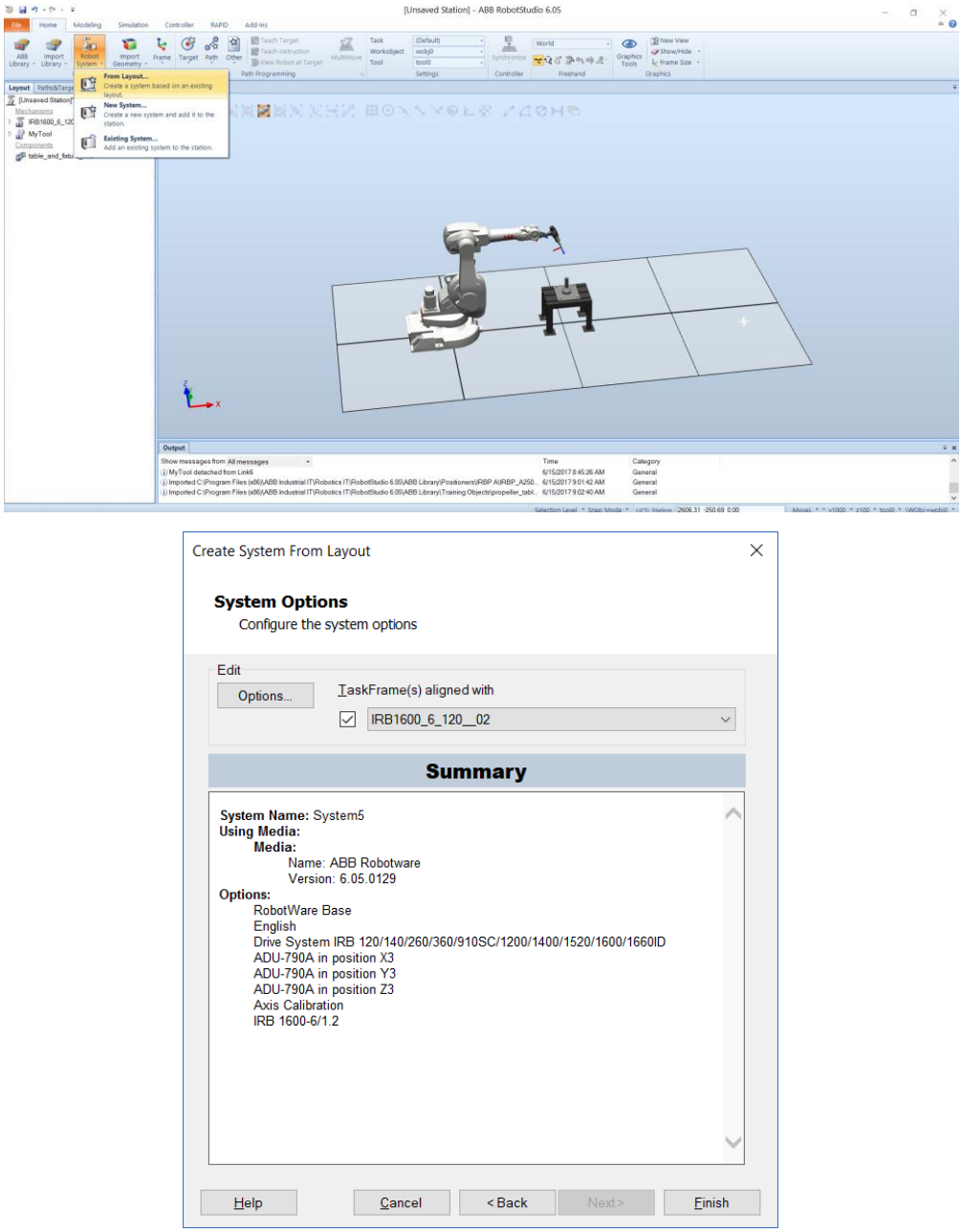


Figure 9.6. RobotStudio® - Define the controller for the integrated robot and auxiliary equipment

Step 5: Generate some 3D models that will be used as workobjects within the robotic system using RobotStudio® facility (ex. 3D cubes)

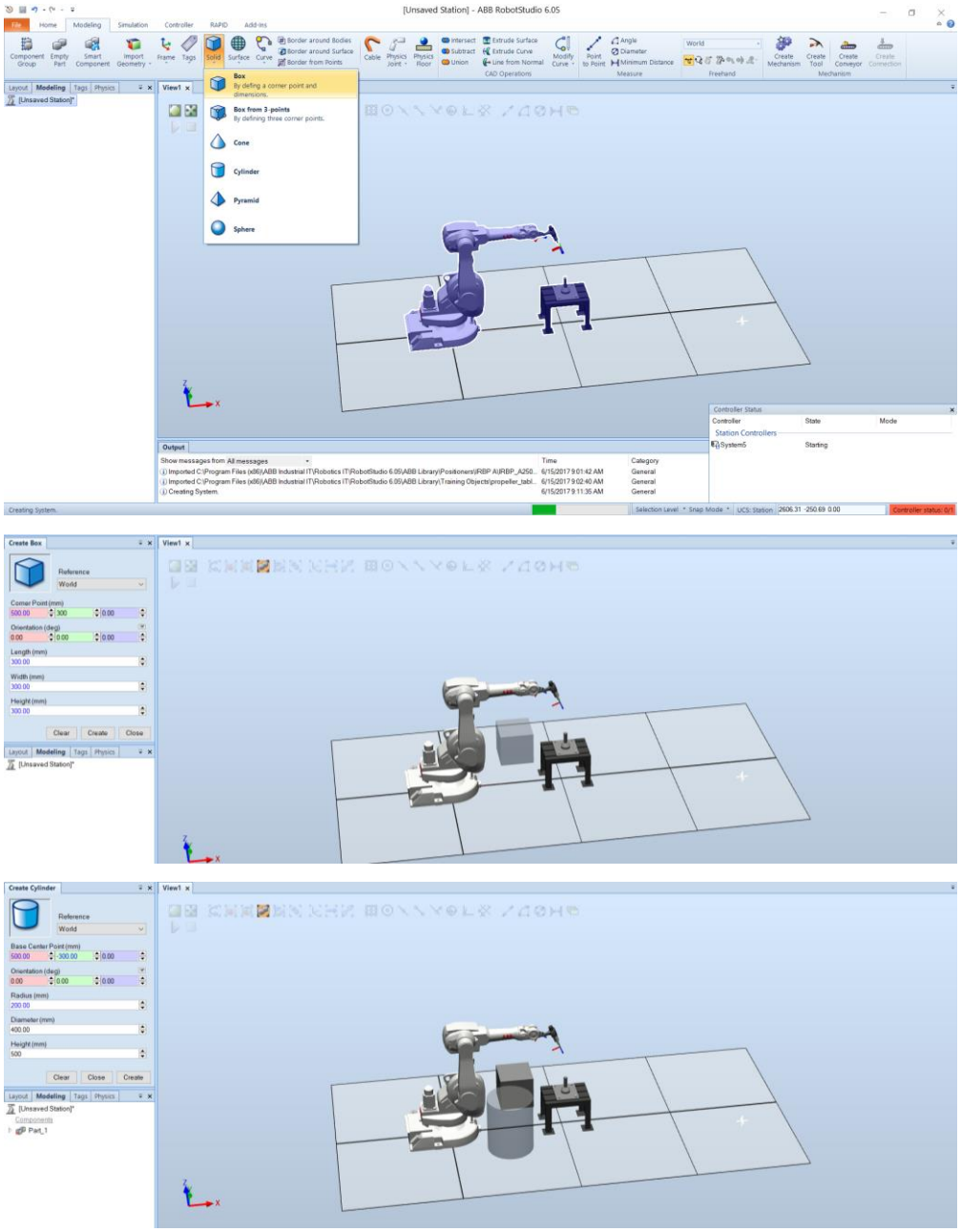


Figure 9.7. RobotStudio® - Generate two 3D models - a cube and a cylinder using RobotStudio facility

Step 6: Define the target points (position and orientation of the tool in the targets) that the robot has to "touch"

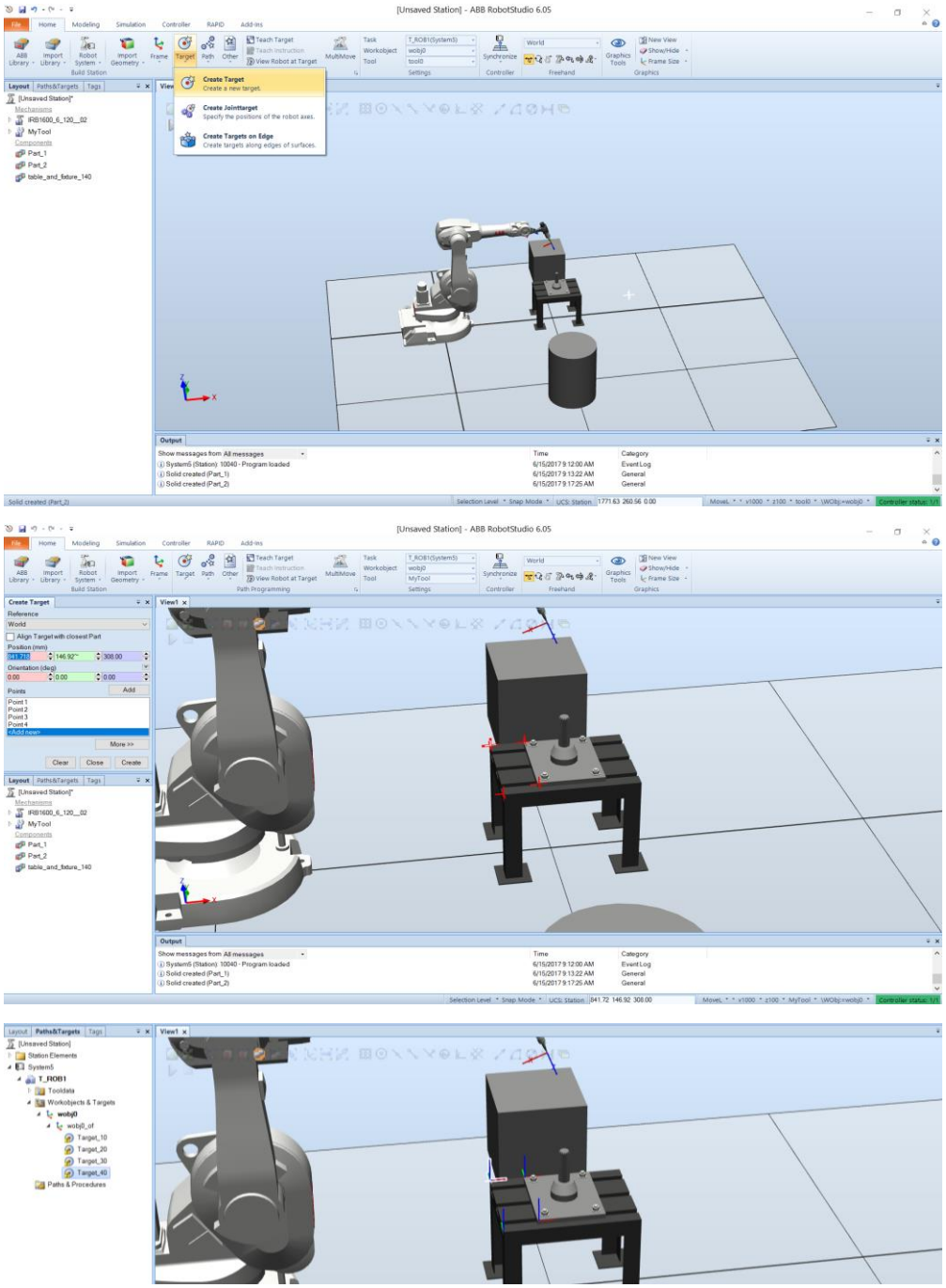


Figure 9.8. RobotStudio® - Define the robot targets points

Step 6: Orient the tool in each target point and find a suitable configuration of the robot structure within each target point

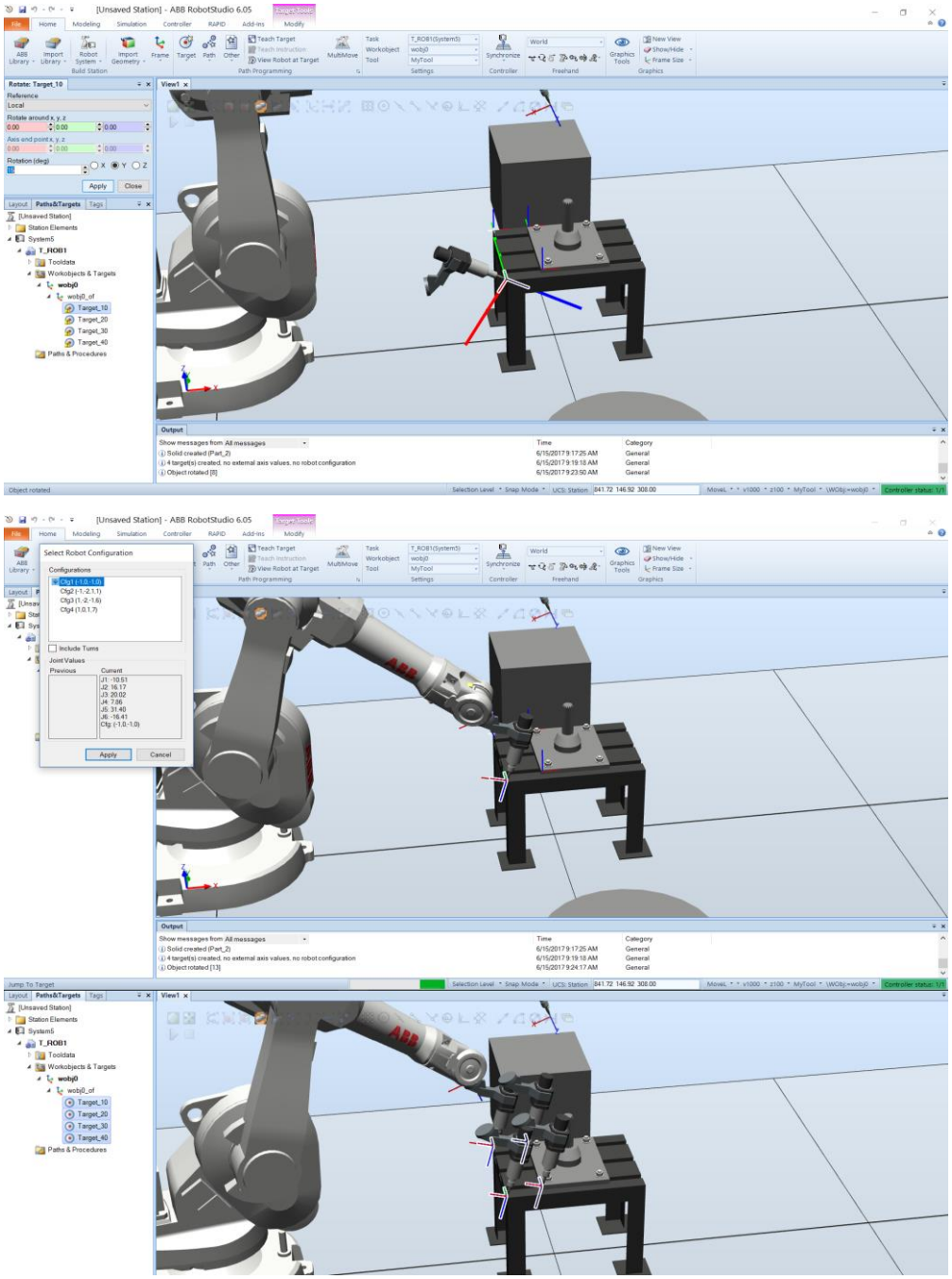


Figure 9.9. RobotStudio® - Define the orientation of the tool in each target point and the robot configuration

Step 7: Generate the path(s) that the robot has to "follow"

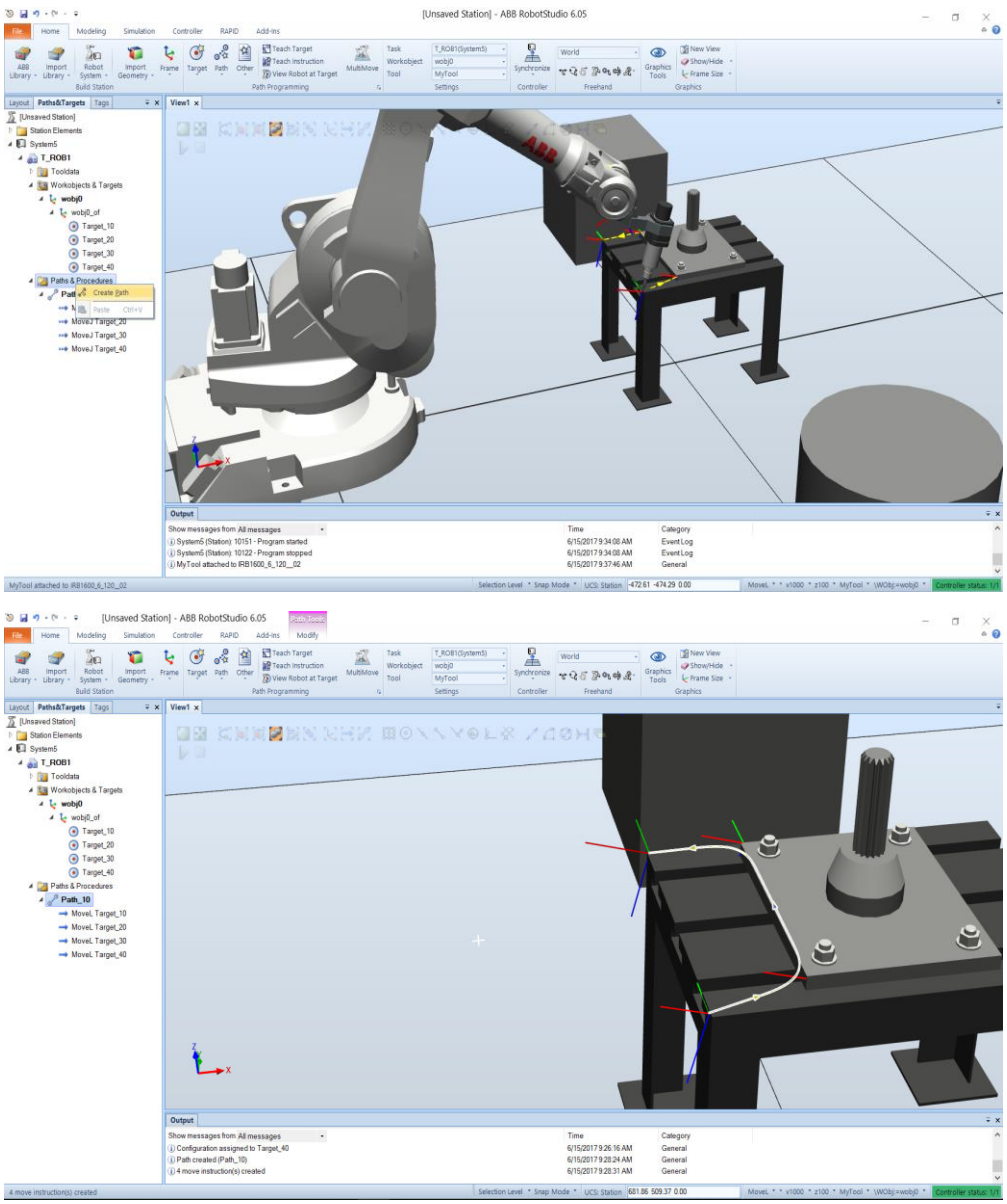


Figure 9.10. RobotStudio® - Define one or more path (trajectory) for the robot

Step 8: Simulate the movement of the robot along the generated path(s)

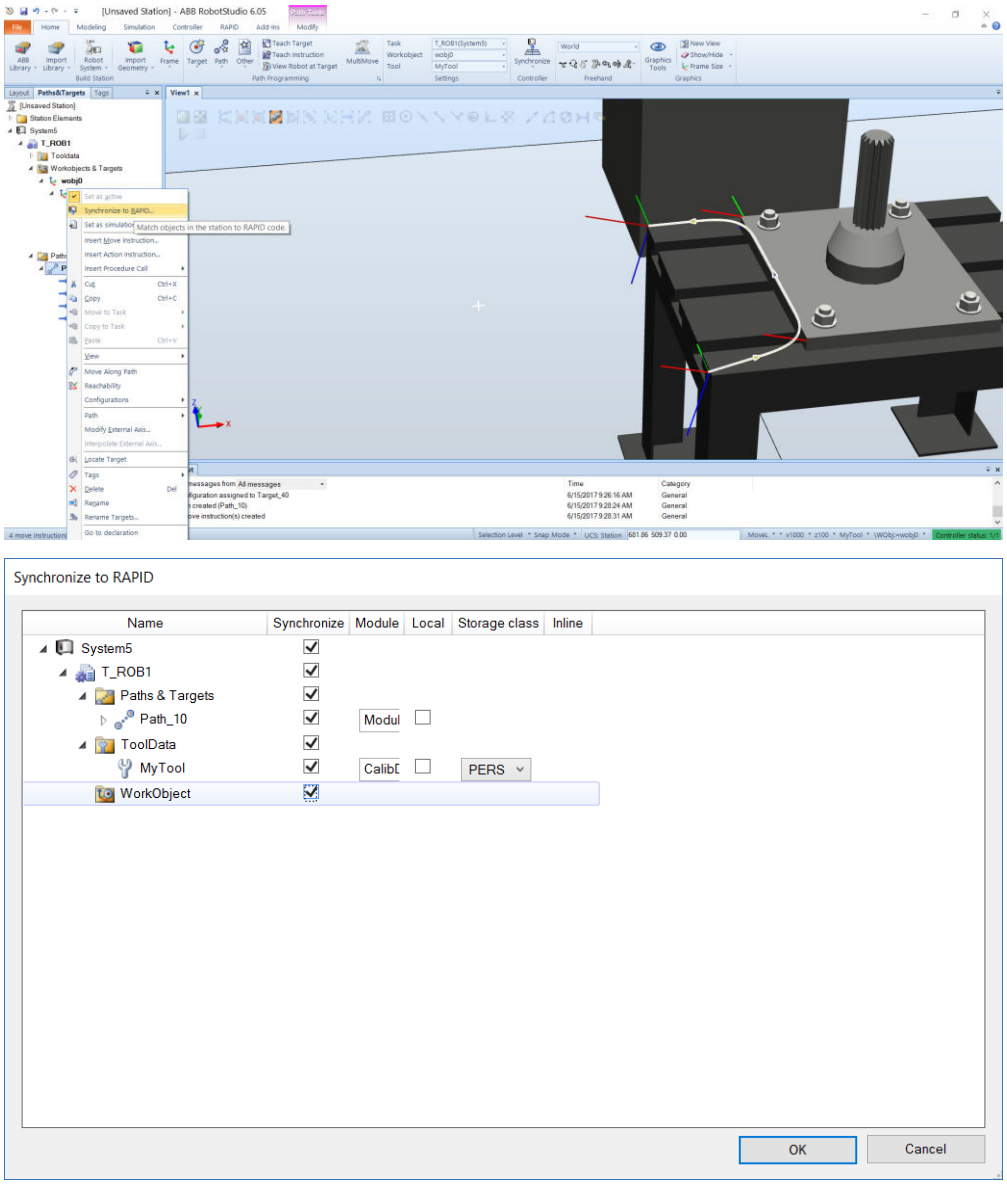


Figure 9.11. RobotStudio® - Simulate the movement of the robot along the generated path(s)

Step 9: View the generated RAPID program

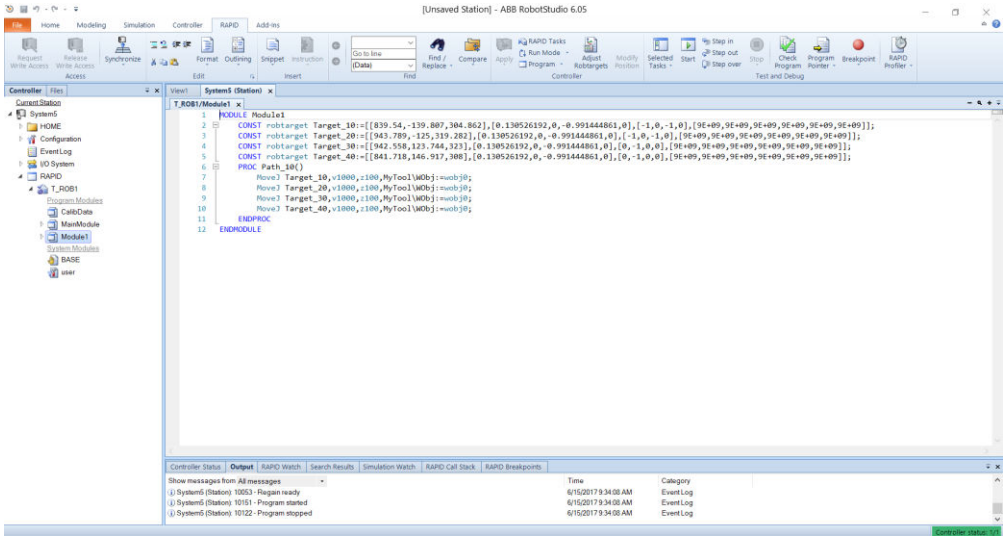


Figure 9.12. RobotStudio® - View the generated RAPID program

Workshop 10: Examples of robotic cells and RAPID programmes developed in RobotStudio®

10.1. Arc welding one robot cell overview

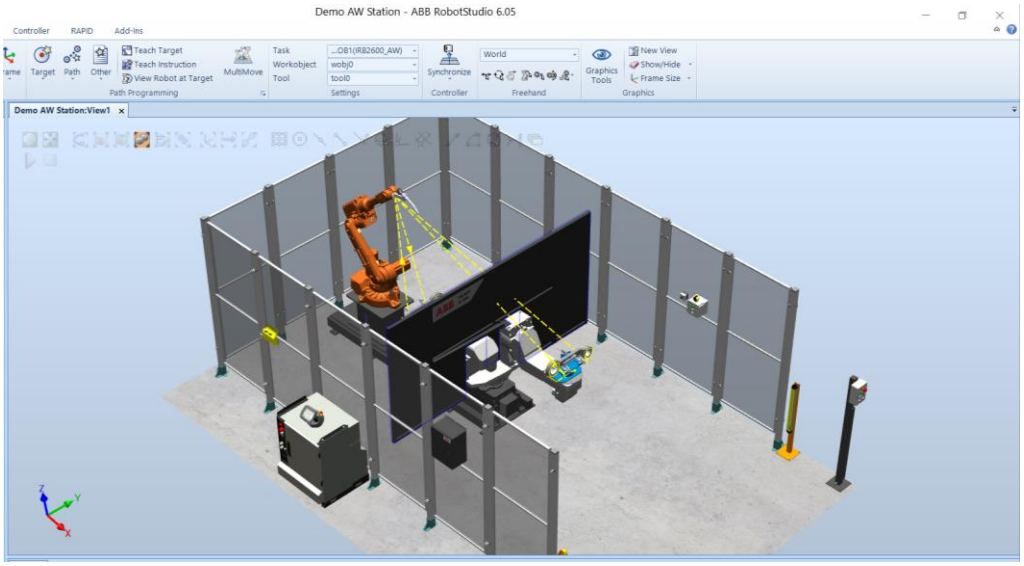


Figure 10.1. RobotStudio® - Arc welding one robot cell – view 1

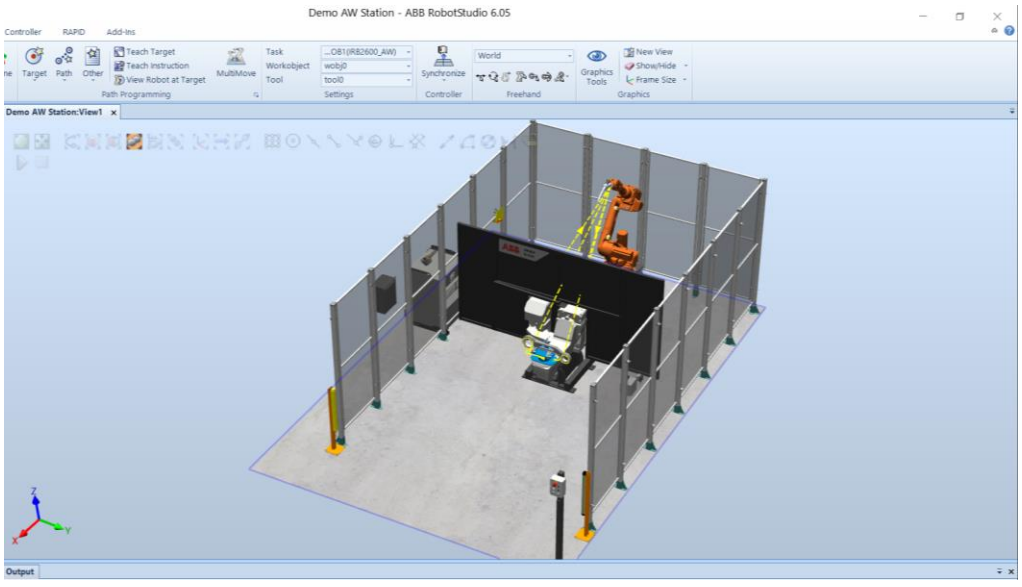


Figure 10.2. RobotStudio® - Arc welding one robot cell – view 2

10.2. RAPID program of the arc welding one robot cell

```
MODULE mPart_1
```

Examples of defining constants, targets and variables at the beginning of the program application - RAPID program

```
CONST jointtarget jt_1:=[[0,-40,0,0,30,0],[9E9,0,0,9E9,9E9,9E9]];
CONST jointtarget jt_2:=[[0,-40,0,0,30,0],[9E9,0,0,9E9,9E9,9E9]];
CONST jointtarget jt_1_2:=[[0,-40,0,0,30,0],[9E9,0,0,9E9,9E9,9E9]];
CONST jointtarget jt_2_2:=[[0,-40,0,0,30,0],[9E9,0,0,9E9,9E9,9E9]];
CONST robtarget p3:=[[153.054345246,19.898332596,243.218173048],
[0,0,-0.707106781,0.707106781],[-1,0,-1,0],[9E9,-90,-90,9E9,9E9,9E9]];
CONST robtarget p5:=[[230.532452302,19.898332596,279.670326794],
[0,0,-0.707106781,0.707106781],[-1,0,-1,0],[9E9,-90,-90,9E9,9E9,9E9]];
CONST robtarget p6:=[[308.054345246,19.898332596,194.999919243],
[0,0,0.707106781,-0.707106781],[-1,0,-1,0],[9E9,-90,-90,9E9,9E9,9E9]];
CONST robtarget p7:=[[261.673155788,19.898332596,119.279463141],
[0,0,-0.707106781,0.707106781],[-1,0,-1,0],[9E9,-90,-90,9E9,9E9,9E9]];
        ……..
```

Examples of robot moving instructions (ex. MoveJ, MoveAbsJ) and welding parameters (ex. ArcLStart, ArcCEnd) - RAPID program procedure

```
PROC Part_1_Pth_1()
    ActUnit STN1;
    MoveAbsJ jt_1,vmax,fine,tool0\WObj:=wobj0;
    MoveJ p1,v1000,z10,tWeldGun\WObj:=Workobject_1;
    ArcLStart p2,v1000,sm1,wd1,fine,tWeldGun\WObj:=Workobject_1\
    SeamName:="Part_1_Pth_1_Weld_1";
    ArcL p3,v100,sm1,wd1,z1,tWeldGun\WObj:=Workobject_1;
    ArcC p5,p6,v100,sm1,wd1,z1,tWeldGun\WObj:=Workobject_1;
    ArcCEnd p7,p8,v100,sm1,wd1,fine,tWeldGun\WObj:=Workobject_1;
    MoveL p4,v1000,z10,tWeldGun\WObj:=Workobject_1;
    MoveJ p17,v1000,z10,tWeldGun\WObj:=Workobject_1;
  ENDPROC
```

Examples of activating and deactivating the part positioner procedure – RAPID program procedure

```
PROC Intch_Pth_2()
      DeactUnit STN2;
      ActUnit INTERCH;
      MoveAbsJ jt_5,vmax,fine,tWeldGun\WObj:=wobj0;
      MoveAbsJ jt_6,vmax,fine,tWeldGun\WObj:=wobj0;
      DeactUnit INTERCH;
   ENDPROC
```

Examples of calling different procedures (ex. Intch_Pth_1) – RAPID program procedure

```
PROC Intch()
      Intch_Pth_1;
      Intch_Pth_2;
   ENDPROC
```

```
ENDMODULE
```

10.3. Arc welding two robots cell overview

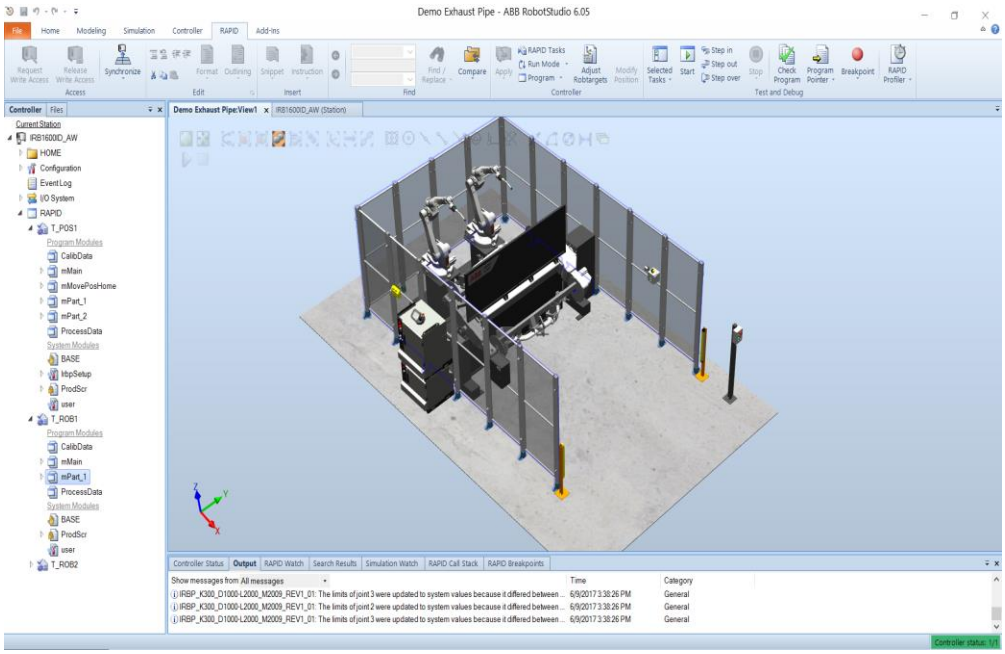


Figure 10.3. RobotStudio® - Arc welding two robots cell - view 1

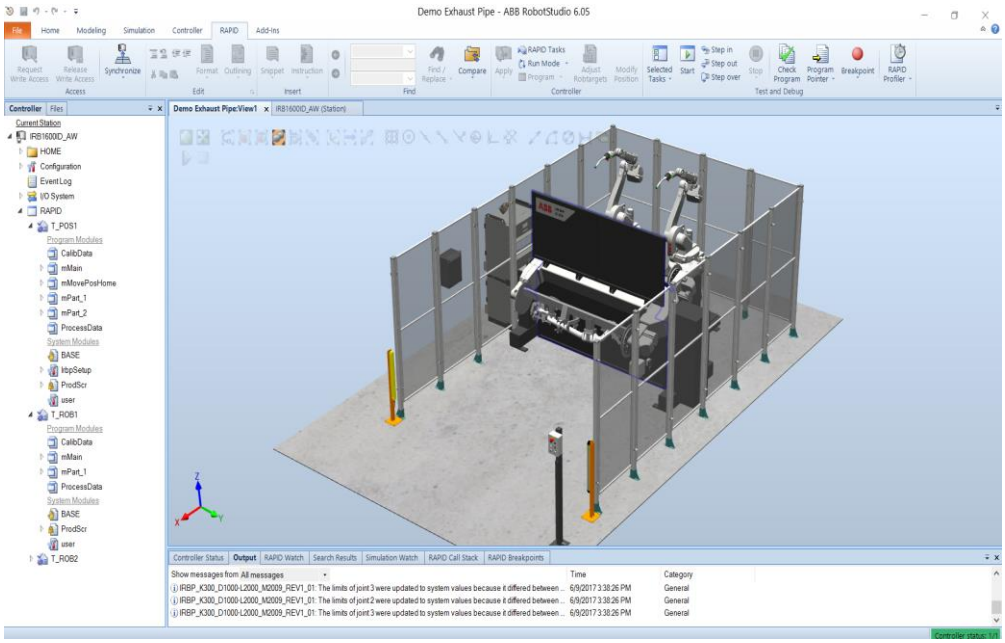


Figure 10.4. RobotStudio® - Arc welding two robots cell - view 2

10.4. RAPID program of the two robots arc welding cell

MODULE mPart_1

Examples of defining constants work object: constants, targets and variables definitions
CONST jointtarget jt_1:=[[0,-30,0,0,30,0],[9E+09,9E+09,9E+09,9E+09, 9E+09,9E+09]]; CONST jointtarget jt_2:=[[30.221811228,40.422424714,-36.801483469,-12.474192193,81.143214245,44.495178556],[9E+09,9E+09,9E+09,9E+09, 9E+09,9E+09]]; CONST jointtarget jt_3:=[[30.221811228,40.422424714,-36.801483469,-12.474192193,81.143214245,44.495178556],[9E+09,9E+09,9E+09,9E+09, 9E+09,9E+09]]; CONST jointtarget jt_4:=[[0,-30,0,0,30,0],[9E+09,9E+09,9E+09,9E+09, 9E+09,9E+09]]; VAR syncident s1; VAR syncident s2; VAR syncident s3; VAR syncident s4; VAR syncident s5;

```
        VAR syncident s6;
        VAR syncident s7;
        …….
```

Examples of calling different procedures (ex. Intch_Pth_1) – RAPID program procedure

```
PROC Part_1()
        Part_1_Pth_1;
        Part_1_Pth_2;
ENDPROC
```

Examples of defining the arc welding parameters – RAPID program procedure

```
MODULE ProcessData
        PERS tasks r1r2p1{3}:=[["T_ROB1"],["T_ROB2"],["T_POS1"]];
        TASK PERS seamdata sm1:=[0.2,0.05,[0,0,0,0,0,0,0,0,0,0],0,0,0,0,0,
        [0,0,0,0,0,0,0,0,0,0],0,0,[0,0,0,0,0,0,0,0,0,0],0.1,0,[0,0,0,0,0,0,0,0,0,0],0.
        05];
        TASK PERS welddata wd1:=[20,10,[0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0]];
ENDMODULE
```

Examples of robot moving instructions (ex. MoveJ, MoveAbsJ) and welding parameters (ex. ArcLStart, ArcCEnd) - RAPID program procedure

```
PROC Part_1_Pth_1()
        MoveAbsJ jt_1,v1000,fine,tool0\WObj:=wobj0;
        SyncMoveOn s1,r1r2p1;
        MoveJ p1\ID:=10,vmax,z10,tWeldGun\WObj:=r1_s1;
        ArcLStart p2\ID:=20,v1000,sm1,wd1,fine,tWeldGun\WObj:=r1_s1;
        ArcC p4,p3\ID:=30,v100,sm1,wd1,z1,tWeldGun\WObj:=r1_s1;
        ArcC p6,p5\ID:=40,v100,sm1,wd1,z1,tWeldGun\WObj:=r1_s1;
        ArcC p8,p7\ID:=50,v100,sm1,wd1,z1,tWeldGun\WObj:=r1_s1;
        ArcCEnd p10,p9\ID:=60,v100,sm1,wd1,fine,tWeldGun\
        WObj:=r1_s1;
        MoveL p11\ID:=70,v1000,fine,tWeldGun\WObj:=r1_s1;
        SyncMoveOff s2;
        MoveAbsJ jt_2,vmax,fine,tWeldGun\WObj:=wobj0;
        WaitSyncTask s3,r1r2p1;
        MoveAbsJ jt_3,vmax,fine,tWeldGun\WObj:=wobj0;
```

```
        SyncMoveOn s4,r1r2p1;
        MoveJ p12\ID:=10,vmax,z10,tWeldGun\WObj:=r1_s1;
        ArcLStart p13\ID:=20,v1000,sm1,wd1,fine,tWeldGun\
        WObj:=r1_s1\SeamName:="Part_1_Pth_1_Weld_2";
        ArcLEnd p14\ID:=30,v100,sm1,wd1,fine,tWeldGun\WObj:=r1_s1;
        MoveL p15\ID:=40,v1000,fine,tWeldGun\WObj:=r1_s1;
        SyncMoveOff s5;
        MoveAbsJ jt_4,vmax,fine,tWeldGun\WObj:=wobj0;
        WaitSyncTask s6,r1r2p1;
ENDPROC
```

```
ENDMODULE
```

10.5. Arc welding four robots cell overview

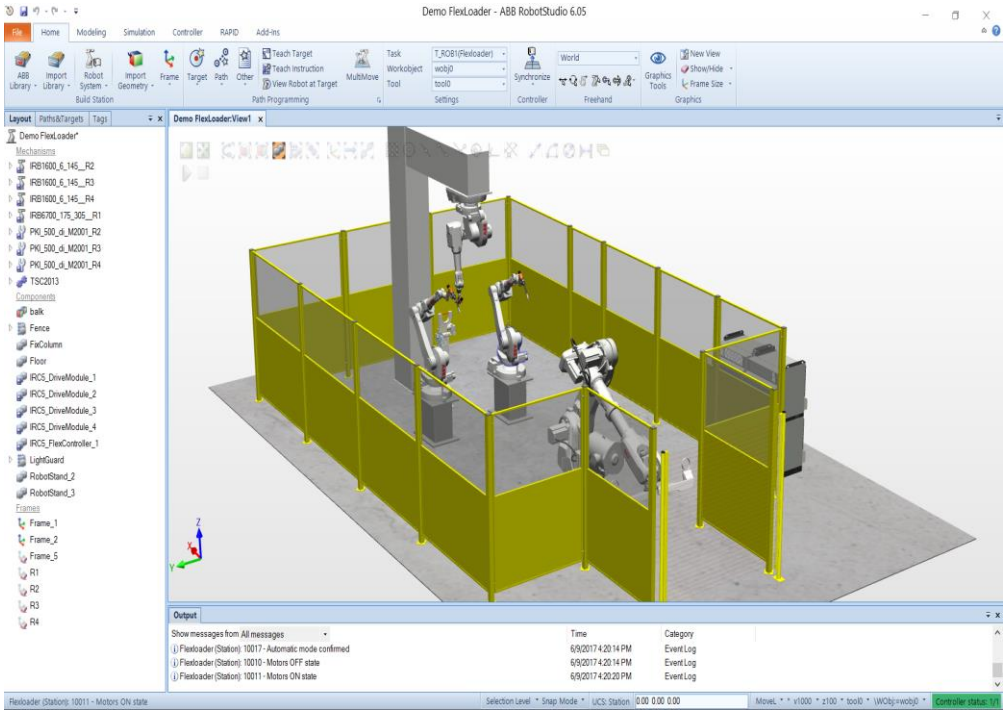


Figure 10.5. RobotStudio® - Arc welding four robots cell – view 1

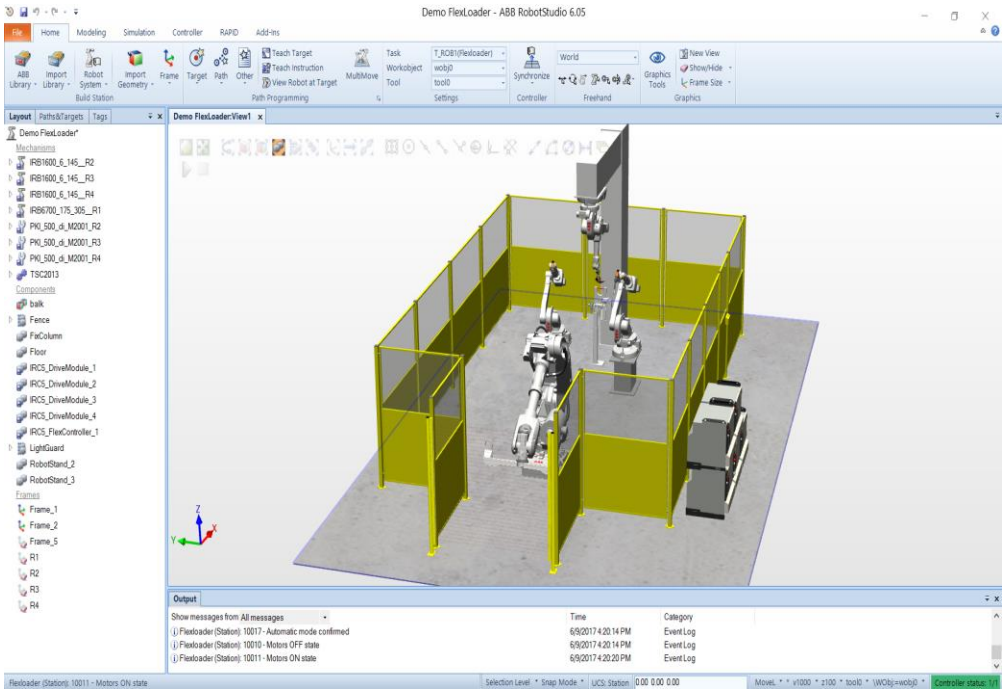


Figure 10.6. RobotStudio® - Arc welding four robots cell – view 2

10.6. RAPID program of the four robots arc welding cell

MODULE Module1

Examples of defining constants work object: constants, targets and variables definitions
VAR syncident ident1; PERS tasks task1{4}:=[["T_ROB1"],["T_ROB2"],["T_ROB3"],["T_ROB4"]]; VAR syncident ident2; VAR syncident ident3; VAR syncident ident4; VAR syncident ident5; VAR syncident ident6; PERS tasks task4{3}:=[["T_ROB1"],["T_ROB2"],["T_ROB3"]]; VAR syncident ident7; VAR syncident ident8; VAR syncident ident9; CONST robtarget pHome:=[[1369.51290546514, 37.4666443316968,1010.78350359101],[0.745592277204196,0.01481977

```
93186134,0.66466353261722,0.0457702820966485],[0,-1,0,1],
[9E9,9E9,9E9,9E9,9E9,9E9]];
        CONST robtarget pOver:=[[2078.6090717597,35.5392515079451,
1357.92304563372],[0.636059946803591,-0.0273127221326822,
0.771147462664317,-0.00365378000185044],[0,-
1,0,1],[9E9,9E9,9E9,9E9,9E9,9E9]];
        CONST robtarget pLoad:=[[-1606.02370263678,-
38.0457751002972,675.609388963257],[0.133106755464876,-
0.0305372750704237,-0.990630939577537,0.000638774305297796],[0,-
1,0,7],[9E9,9E9,9E9,9E9,9E9,9E9]];
        CONST robtarget pUnder:=[[2026.43017232093,
41.7456192089369,1534.29951708203],[0.91945264701403,-
0.0277764081076071,0.392048066563009,0.0115591762569867],[0,-
1,0,1],[9E9,9E9,9E9,9E9,9E9,9E9]];
        CONST robtarget pFront:=[[2530.56390764596,
41.3940929579109,1569.72454292973],[0.492426348032876,0.00122286
152930768,0.870261854609902,0.012613515910265],[0,0,0,0],[9E9,9E9,
9E9,9E9,9E9,9E9]];
```

Examples of calling different procedures (ex. Intch_Pth_1) – RAPID program procedure

```
PROC main()
        ToHome;
        Over;
        WaitSyncTask ident1,task1;
        WaitSyncTask ident2,task1;
        Under;
        WaitSyncTask ident3,task1;
        WaitSyncTask ident4,task1;
        WaitSyncTask ident5,task1;
        Front;
        WaitSyncTask ident6,task1;
        WaitSyncTask ident7,task1;
        FlipToBackside;
        WaitSyncTask ident8,task1;
        WaitSyncTask ident9,task1;
        FlipToFrontside;
        ToLoad;
```

```
ENDPROC
```

Examples of robot moving instructions (ex. MoveJ) - RAPID program procedure

```
PROC FlipToBackside()
        MoveJ pFlip_01,v1000,z100,tool0\WObj:=wobj0;
        MoveJ pFlip_02,v1000,z100,tool0\WObj:=wobj0;
        MoveJ pFlip_03,v1000,fine,tool0\WObj:=wobj0;
ENDPROC
```

Examples of robot moving instructions (ex. MoveJ) - RAPID program procedure

```
PROC FlipToFrontside()
        MoveJ pFlip_02,v1000,z100,tool0\WObj:=wobj0;
        MoveJ pFlip_01,v1000,z100,tool0\WObj:=wobj0;
        MoveJ pHome,v1000,z100,tool0\WObj:=wobj0;
ENDPROC
```

```
ENDMODULE
```

10.7. Assembly two robots cell overview

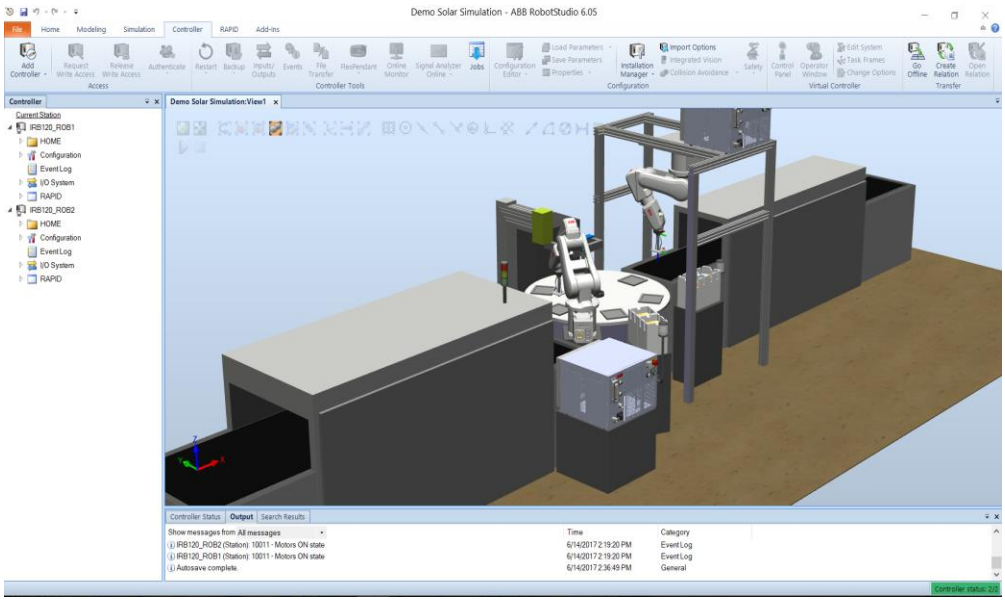


Figure 10.7. RobotStudio® - Assembly two robots cell – view 1

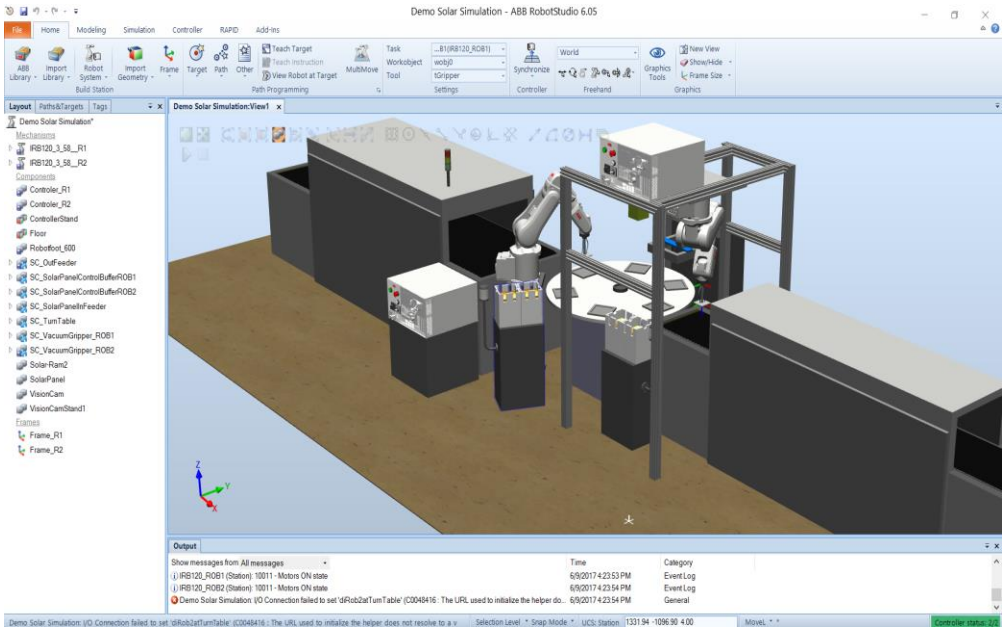


Figure 10.8. RobotStudio® - Assembly two robots cell – view 2

10.8. RAPID program of the two robots assembly cell

MODULE modRobotStudio

Examples of defining constants work object: constants, targets and variables definitions
CONST robtarget pTurnTablePosROB2:=[[0,4.89842541528954E-16,4],[-6.12303176911189E-17,1,-1.61279309130843E-47,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]]; CONST robtarget pOutFeederPos:=[[200.016009205003, 399.937244742038,6.99993896484363],[-2.01362528565985E-22,0.999999999994593,3.28860826072754E-06,-6.12303176907878E-17],[1,0,1,0],[9E9,9E9,9E9,9E9,9E9,9E9]]; CONST robtarget pBufferPosROB2:=[[224.500035230801, 100.00001157408,166.454416191376],[4.32963728535968E-17,0.707106781186547,0.707106781186548, 4.32963728535968E-17],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]]; PERS num nZoffset:=-4;

Examples of robot moving instructions (ex. MoveJ), conditional expressions (ex. IF, ELSE) and signals (ex. SetDO)- RAPID program procedure

```
PROC PlacePanelInControlBuffer()
        MoveJ RelTool(pBufferPosROB2,0,0,-
100),v1000,z10,tGripper\WObj:=wobjBuffer;
        MoveL RelTool(pBufferPosROB2,0,0,nZoffset),v100,fine,tGripper\
WObj:=wobjBuffer;
        IF diBufferFull=1 THEN
                nZoffset:=0;
        ELSE
                nZoffset:=nZoffset-4;
        ENDIF
        SetDO doVacuumOn,0;
        WaitDI diVacuum,0;
        MoveL RelTool(pBufferPosROB2,0,0,-
100),v1000,z10,tGripper\WObj:=wobjBuffer;
ENDPROC
```

Examples of robot moving instructions (ex. MoveJ) and signals (ex. SetDO)- RAPID program procedure

```
PROC PlacePanelOnOutFeeder()
        MoveJ RelTool(pOutFeederPos,0,0,-
100),v1000,z10,tGripper\WObj:=wobjOutFeeder;
        MoveL pOutFeederPos,v200,fine,tGripper\WObj:=wobjOutFeeder;
        SetDO doVacuumOn,0;
        WaitDI diVacuum,0;
        MoveL RelTool(pOutFeederPos,0,0,-
100),v1000,z10,tGripper\WObj:=wobjOutFeeder;
ENDPROC
```

Examples of robot moving instructions (ex. MoveJ) and conditional expressions (ex. IF, ELSE) - RAPID program procedure

```
PROC main()
        MoveJ RelTool(pTurnTablePosROB2,0,0,-
100),v200,fine,tGripper\WObj:=wobjTurnTableROB2;
        nZoffset:=0;
        WHILE TRUE DO
                PickPanel;
                IF diPlaceCellInBuffer=1 THEN
```

```
                PlacePanelInControlBuffer;
            ELSE
                PlacePanelOnOutFeeder;
            ENDIF
        ENDWHILE
ENDPROC
```

```
ENDMODULE
```

Bibliography

[1] ABB, *Technical reference manual RAPID Instructions, Functions and Data type*, 3HAC 16581-1, 2017

[2] ABB, *Operating Manual RoboStudio 6.05*, 3HAC032104-001 Revision: T, 2017

[3] ABB, *Operating Manual RoboStudio 5.61*, 3HAC032101-001 Revision: T, 2016

[4] RobotStudio® Help 6.05

[5] RobotStudio® software application versions 5.60, 5.61, 6.05