SOFTWARE MANUAL

Version: 1.3.0 EN01

Kassow Robots ApS
Kajakvej 2
DK-2770 Kastrup

info@kassowrobots.com
kassowrobots.com

Kassow Robots ApS
Kajakvej 2
DK-2770 Kastrup

info@kassowrobots.com
kassowrobots.com

1 Introduction

1.1 Product Description

The robot consists of the robot arm, a controller cabinet, a teach pendant and the connection cables between those.

Figure 1: Picture of the Robot arm

1. The robot arm consists of 7 geared servo motors connected mechanically by machined extrusions and castings and connected electrically by a 48V power supply bus and a serial communication bus.

2. The controller cabinet house various power supplies: the computer which coordinated the movements of the 7 geared servo motors; the I/O board, which has a range of safety related

functions and allow various electrical connections; the relays on the robot arm power supply bus; connectors for connecting to wall socket, robot arm and teach pendant and an E-stop button, a Protective stop button and a play/pause/resume toggle button.

3. The teach pendant / HMI which allows the user to program the robot system and which house two buttons, which when pushed allow the user to manually manipulate the robot arm, and an E-stop button, a Protective stop button and a play/pause/resume toggle button.

The wires connecting the system components are: Two wires with connector in both ends between robot arm and controller cabinet; one wire fixed to the teach pendant and with a connector in the other end, to connect teach pendant to controller cabinet and one wire with a 3P CEE female connector in one end and male connector in the other end, to connect wall socket with controller cabinet.

2 Safety

2.1 General

	Before the starting to program and operate this robot this instruction manual must be thoroughly read and understood!
	This warning symbol indicates that special precautionary measures must be taken. If the safety precautions are not observed, it may lead to hazardous conditions and result in personal injury or damage to property.
	This symbol indicates that the following information is important.

2.1.1 Operation

Always ensure safe operation of the unit. It is the responsibility of the owner to ensure that the personnel, who is operating the unit is familiar with the stopping procedure of the unit. The unit must only be used when all safety precautions have been observed, which means that guards, grids, covers etc. are to be firmly bolted on, etc. All safety prescriptions mentioned in the manual must be observed. Always ensure that the unit is well maintained and in good mechanical condition before any operation. Always inspect the unit before use. Ensure that all safety precautions are meet, before operating the unit; hence, that guard, grids, covers among others are secured and firmly fixed and not damaged. Make sure that no foreign objects (metal parts etc.) enter the unit during operation. Never open electrical cabinet door during operation. The unit generates noise as specified in the specifications. Hearing protection may be used, where it is considered necessary, taking into account factors such as the installation of the unit, the conditions of use of the unit, the characteristics of the workplace (such as, for example, noise absorption, the scattering of noise, noise reflections), noise emissions from other sources (such as, for example, from other machinery), the position of persons with respect to the sources of noise, the duration of exposure und the use of personal protective equipment (hearing protectors).

2.1.2 Training and instructions

For the personnel to protect themselves from machine and workplace hazards, each discipline - operators, cleaning and service personnel etc. - must receive instructions that suit their particular subject area.

- Operators/programmers: Instruction in the safe operation of the system is required and the manual must be accessible at all times.
- Cleaning and Service Personnel: The necessary safe instruction needed to carry out maintenance work, basic technical knowledge.

The unit must be maintained according to the instructions in the instruction manual.

Only trained/instructed and if required also qualified personnel may conduct cleaning, maintenance and repair of the unit. It is the responsibility of the owner to ensure that the personnel, who is conducting the cleaning, maintenance and repair is familiar with the stopping procedure of the unit.

Never attempt to make any repairs, adjustments, or inspections while the unit is operating/ running. Before any maintenance or repair works or cleaning, follow the lock out, tag out (LOTO) instructions below:

- Empty the equipment for material.
- Stop unit and disconnect the supply disconnecting device.
- Tag out the disconnecting device stating “ATTENTION! DO NOT CONNECT. PERSONNEL CARRYING OUT WORK ON THE UNIT”.
- Release the pressure inside the unit (where relevant).
- Ensure that no hazardous energies has been build up.
- Before any work on electric installations ensure that the power is disconnected.

Check the supply disconnecting device or power source and make sure the power is always off before starting any electrical work. Never remove nor disassemble electrical components and/or wires while the unit is running and/or connected to the power grid. Lock the supply disconnecting device if possible or if cannot, make sure everyone in the area knows it is turned off for a reason and leave an easily noticeable note that work on the electricity is ongoing. All electrical installations must be made by a qualified electrician/technician and in compliance with local regulations. The electrical installation must comply with the requirements for the motor and any other preinstalled components. See data sheets. Danger of injuries by rotating elements.

The owner must ensure at least 300 lux are present in the area where normal supervision takes place (front of the unit). During assembly, disassembly, installation, service, maintenance and the like it is advisable to establish temporary extra lighting. Only operate the unit while on the outside with all guards in place. Ensure to consider ergonomic hazards while tightening elements with a relatively high torque value.

2.1.3 Cleaning

Only trained/instructed personnel may conduct cleaning of the unit. It is the responsibility of the owner to ensure that the personnel will receive the training and knowledge they need, in order to become familiar with the unit and able to protect themselves from workplace hazards. It is the responsibility of the owner to ensure that the personnel, who is conducting the cleaning is familiar with the stopping procedure of the unit. Always wear long-sleeved rubber gloves when cleaning the unit. Cleaning may ONLY be done following LOTO instructions in this manual. Kassow Robots ApS will not be held liable for damage due to wrong or inappropriate cleaning methods or use of too harsh chemicals. Be careful not to damage the

surfaces nor functionality, when cleaning the equipment. Use only appropriate cleaning methods that do not deteriorate, corrode or discolour the surfaces of the equipment. Ensure that the detergent used does not deteriorate, corrode or discolour the surfaces of the equipment. If in doubt what suitable detergents to use, please consult your supplier of detergents and chemicals for cleaning purposes. Ensure to rinse all detergent residues of the equipment.

2.1.4 Instructions for safe operation

Only the operator that holds the Teach Pendant are allowed to be near the robot when using the Teach Pendant. All other persons shall be an distance of 3m.

2.1.5 Special Situations

If the operator is trapped by the robot arm the operator shall push the robot arm away to release themselves.

Liability

The content panel displays the tool selected via tool bar. There are two content panels (left and right) allowing the user to use two same or different tools at the same time.

The device described in this document is either an industrial robot or a component thereof.

- Manipulator
- Robot Controller
- Teach pendant
- Connecting cables
- Software

The cooperative robot is built using state-of-the-art technology and in accordance with the recognized safety rules. Nevertheless, misuse of the industrial robot may constitute a risk to life and limb or cause damage to the industrial robot and to other material property.

Terms used

Term	Description
RC	**Robot Controller** is the main controlling unit required for each Kassow Robot manipulator.
TP	**Teach Pendant** stands for the portable programming and manual platform providing all necessary user interface and safety controls.
TPUI	The teach pendant **User Interface** is the software user environment running on the manual platform (TP). This provides all software tools necessary for the robot programming, teaching and accessing extension modules.

ESTOP	A stop initiated by any active component of the robot (Joints, IO Board, Tool IO, Controller computer). Executive parts of the robot are immediately halted and cut loose of power anytime the ESTOP condition is identified within the system.
PSTOP	A stop initiated by any active component of the robot (Joints, IO Board, Tool IO, Controller computer). The executive parts become limited and monitored for energy supplies anytime the PSTOP condition is identified within the system.
Safety mode	Safe and Reduced modes limit individual robot axis moving at maximum speeds of 0.25m/s and 1.0m/s. The Normal mode doesn't provide any limitation upon the link and joint motion.
Back-drive	The KR free drive mode allowing to move the robot manually. The function is controlled by the teaching buttons present at the back side of the TP or at the Tool IO flange (located at the end of arm).

2.2 Safety Functions

Safety functions evaluate external and internal signals of the whole system which can act immediately to halt the robot or cut him loose from power if necessary.

2.2.1 Emergency STOP

The emergency STOP buttons are present at both Teach pendant and Robot cabinet. These are used to initiate a robot motion or other potentially hazardous situation.

While Emergency buttons provide immediate interaction and safety for the operator, the internal variables are also continuously monitored to avoid any internal damage to system.

The integrity guards check for thr basic life conditions of the system, including temperatures, voltage or currents. Further safety checks keep track with the sensor data consistency.

The emergency STOP always provoke immediate halt of the robot, followed by the power cut to all executive parts of the robot.

2.2.2 Protective STOP

The Protective STOP can be used interactively by the operator to pause and continue the running program.

Internally, the system can also provoke the Protective STOP to apply limit energy guards upon the system when some conditions are met and warn the user about the case. Depending on the type of diverging values, the program can continue its normal operation when the Protective STOP is released.

2.3 Applied norms and regulations

The safety function complies with EN ISO 13849-1:2015.

The emergency stop function is constructed as a category 3, performance level d according to EN 10218-1:2011

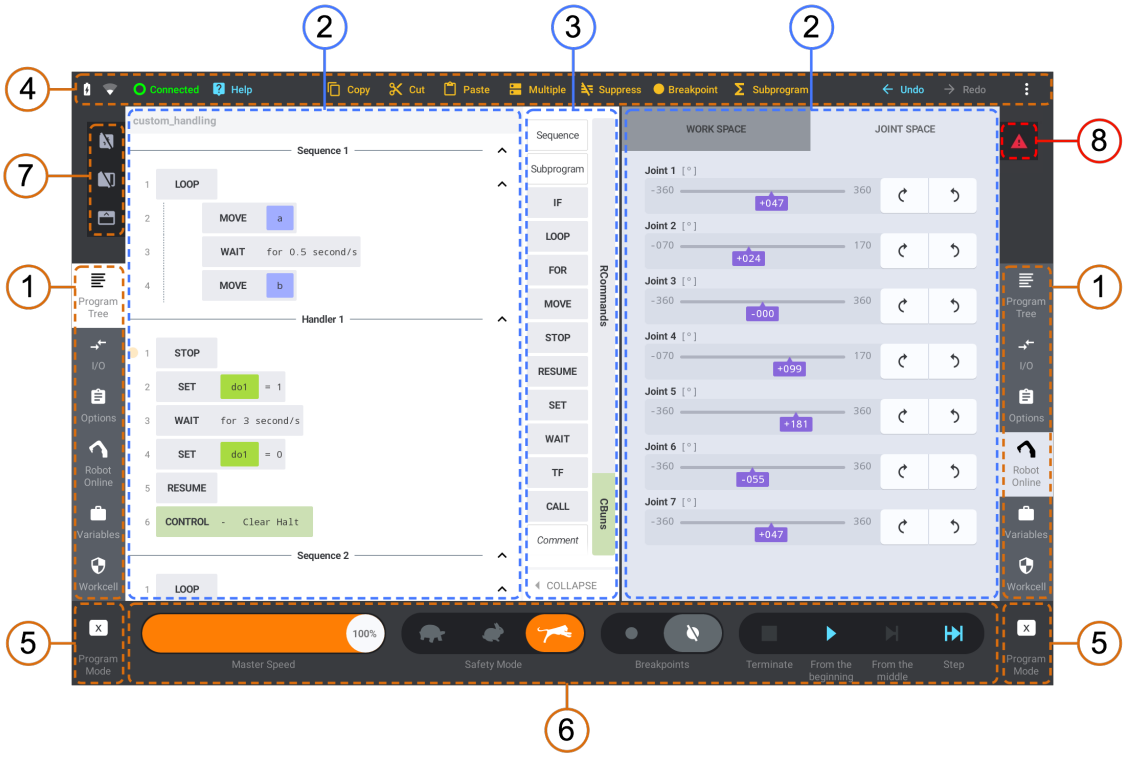
Additional emergency stops added to the robot, must also be constructed as performance level D according to EN 10218-1:2011

3 User Interface

Kassow Robots graphical user interface is intended to be intuitive, since it comes with familiar design of tablet applications. This intuitive touch-based interface allows users to control robot arm and HW equipment, build and run programs and configure robot installation setup.

3.1 Main Screen

Main screen represents the most important part of the user interface, which is based on the double pane layout. This design makes it easy to combine complementary or independent tools within one display.



Item	Description
1	**Tool Bar** [3.1.1, page 16] Provides tool selection and switches content of Content Panel.
2	**Content Panel** [3.1.2, page 16] Displays the selected tool on the screen. Both the left and right half of the screen can be used.
3	**Command Box** [3.1.3, page **Error! Bookmark not defined.**] Provides list of available program command.

4	**Action Bar** [3.1.4, page 17] Displays status icon and allows program edit.
5	**Switch Mode Button** Swaps Bottom Bar content.
6	**Bottom Bar** [3.1.5, page 18] Displays list of variables or program control.
7	**Display Options** Adjusts the program tree view, used in older versions. (Please note this area is used as a quick reference spot with recent software versions)
8	**Status Icon** [3.1.6, page 19] Indicates various system issues. Clicking the icon shows the issue description.

3.1.1 Tool Bar

The tool bar allows the user to select one of the available tools to be displayed in the related content panel. Two tool bars (left and right) are available to provide access to two same or different tools at the same time.

Tool	Description
Program Tree	The interactive view of the actual program listing providing the visual and responsive program alterations [3.2, page 20].
I/O	Immediate access to available I/O interfaces [3.3, page 22].
Options	Displays the settings of the selected program command or variable [3.4, page 29].
Robot Online	The robot jogging interface (Workspace/frame, joint, self-motion) [3.5, page 29].
Variables	Provides the list of custom (user defined) and system (mapped) variables and allows the user to create new ones [3.6, page 31].
Workcell	Provides robot installation setup [3.7, page 32].

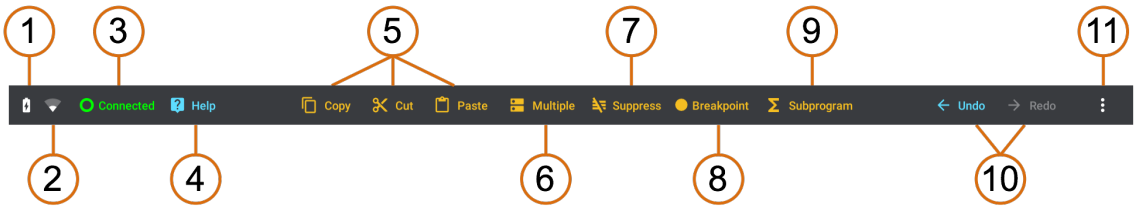
3.1.2 Content Panel

The content panel displays the tool selected via tool bar. There are two content panels (left and right) allowing the user to use two same or different tools at the same time.

3.1.3 Command Box

The command box provides an instant access to basic program building blocks, extensions loaded from the Cbuns or subroutines defined by the user. Those commands can be dragged and dropped into the program tree. The command box is available only when the program tree is shown at least in one of the content panels. Once the command box is available, it can be collapsed to provide more space for the content panels. See the detailed description of Commands in [Ch. 6, page 56].

3.1.4 Action Bar

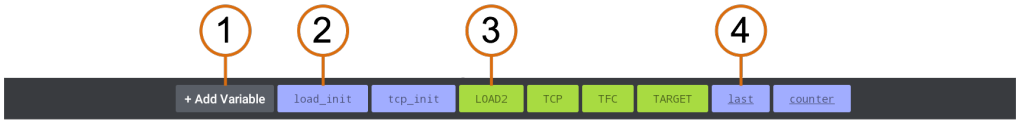


The action bar provides tools for program editing such as copy/cut/paste and undo/redo. Moreover, the action bar contains system status indicators (battery, connection) and allows the user to open main settings.

Item	Description
1	**Battery Status** Displays the battery status (level and charging state).
2	**WiFi Connection Status** Indicates if the device is connected to Wi-Fi and what is the signal strength.
3	**Robot Connection Status** Shows the connection status (unreachable, connected, not connected) and allows the user to connect or disconnect manually.
4	**Help mode** When in the interactive help mode, the user can view the relevant help document simply by touching the element on the screen.
5	**Copy/Cut/Paste** Provides the standard copy/cut/paste operations on program commands and variables. Copy/Cut/Paste controls are enabled only when command or variable is selected. Paste button also requires previous Copy/Cut action to be enabled. All the actions are supported also for multiple command selection.
6	**Multiple Selection** Once the switch is on (orange background), the user can select multiple program commands or variables.
7	**Suppress** The suppress button allows the user to suppress selected command or commands in the program tree. Those commands will not be executed. The suppress button is enabled only when a command is selected.
8	**Breakpoint** Places a breakpoint to all selected command/s. If the breakpoints are enabled (see Control Mode), the breakpoint pauses the program execution.

9	**Subprogram** The subprogram button converts selected command/s into a new Subprogram and replaces them with a single CALL command.
10	**Undo/Redo** Undo reverses the latest user action. Redo can restore the Undo changes.
11	**Settings** Opens the settings popup menu.

3.1.5 Bottom Bar



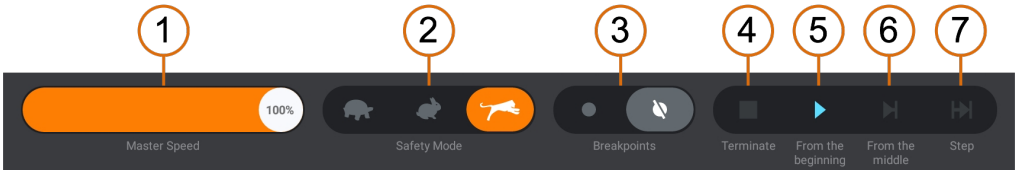
The bottom bar provides two different modes (Program Mode and Control Mode). The user can swap between these two modes via clicking the Switch Mode Button.

Program Mode

The program mode is recommended during the editation of a program, since it contains list of all custom and system variables. These variables can be dragged and dropped into special fields (options) of selected command or variable. Except the list of variables, the program mode contains also the Add Variable button, which displays the "*Create New Variable*" dialog.

Item	Description
1	**Add Variable Button** Allows the user to create new custom variable.
2	**Custom Variable** Example of custom (user defined) variable – blue colour.
3	**System Variable** Example of system (mapped variable) – green colour.
4	**Persistent Variable** Example of persistent (user defined) variable – blue colour, underline.

Control Mode



The control mode mode allows the user to launch the program and control the execution process (incl. Debugging). Therefore the control mode is recommended during the execution of program.

Item	Description

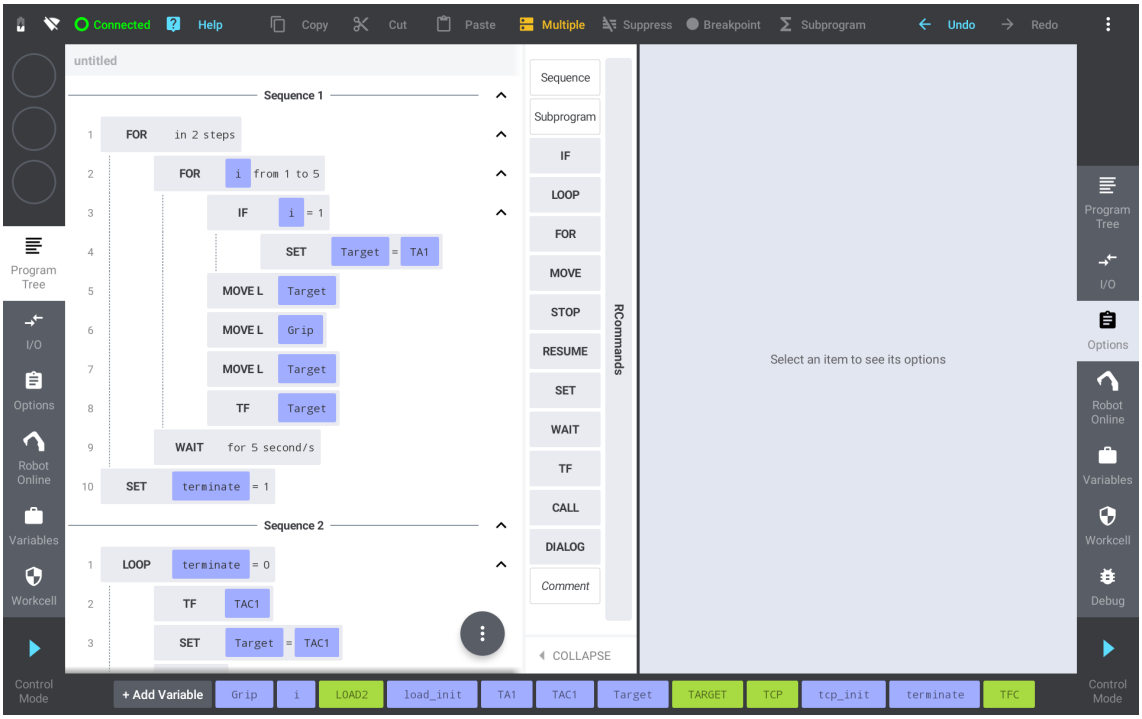
1	**Master Speed Slider** Sets the speed of robot movement during the program execution [7.1.1, page 89].
2	**Safety Mode Switch** Sets safety mode: Safe (Turtle), Reduced (Rabbit) or Unlimited (Cheetah) [7.1.2, page 89].
3	**Breakpoints Switch** Enables/disables the program debugging [7.6, page 93].
4	**Terminate Button** Terminates program execution.
5	**Play/Pause Button** Launches, pauses or resumes program [7.2, page 90].
6	**From the Middle Button** Not supported yet.
7	**Step Button** Provides an execution of a single program command [7.6.3, page 94].

3.1.6 Status Icon

The status icon (warning sign) is shown whenever some system issue occurs, or the program cannot be executed for some reason. Clicking the icon makes the ”System Info“ dialog appears. This dialog provides the recent issue description.

3.2 Program Tree

The Program Tree view allows you to build the robot program in an intuitive graphical way. To code, drag the available commands from the Command Box and drop them at the desired position in the Program Tree. The drop position determines whether the commands are siblings or parent and child/children. To modify command argument, the Options pane [3.4, page 29] will be open automatically (if not, click the one at the side bar). If you want to move the commands within the program tree, long click the command and use drag-and-drop. Select the command and click the **Cut** button to permanently remove it.



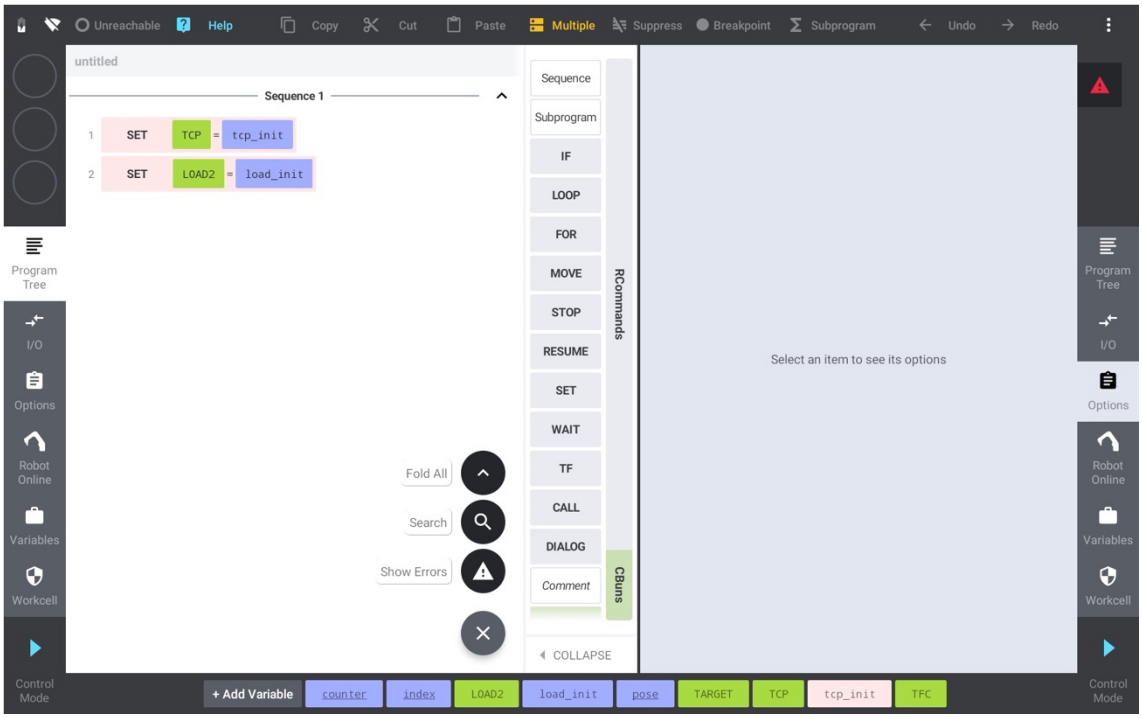
An essential concept of each program tree is Sequence of commands. Each program tree consists of one or more (max 10) Sequences that are executed simultaneously and asynchronously. The commands within Sequence are executed sequentially. See Variables [Ch. 5, page 42] and Commands [Ch. 6 , page 56] to learn more about the robot programming.

It is important to understand the distinction between the open program and the active program. The Program Tree tools display the open program, whereas the active program refers to the one that is currently being executed by the robot. Opening another program or making modifications to an existing one does not immediately impact the ongoing program execution. Changes to the program will only take effect once you initiate a relaunch of the robot program cluster, see Program Control for the details [Ch. 7, page 89].

3.2.1 Program Context Menu

The Program Tree tool is equipped with a context menu that provides convenient shortcuts for program-related actions. The contents of the menu dynamically change depending on the current state of the system. For instance, the **Show Errors** shortcut is accessible only when the program tree contains invalid commands. Conversely, the **Autofocus** shortcut is available solely during the execution of a program.

Moreover, when the program tree contains invalid commands, the menu button is accompanied by an error icon, providing a visual cue for users to identify and address errors efficiently.

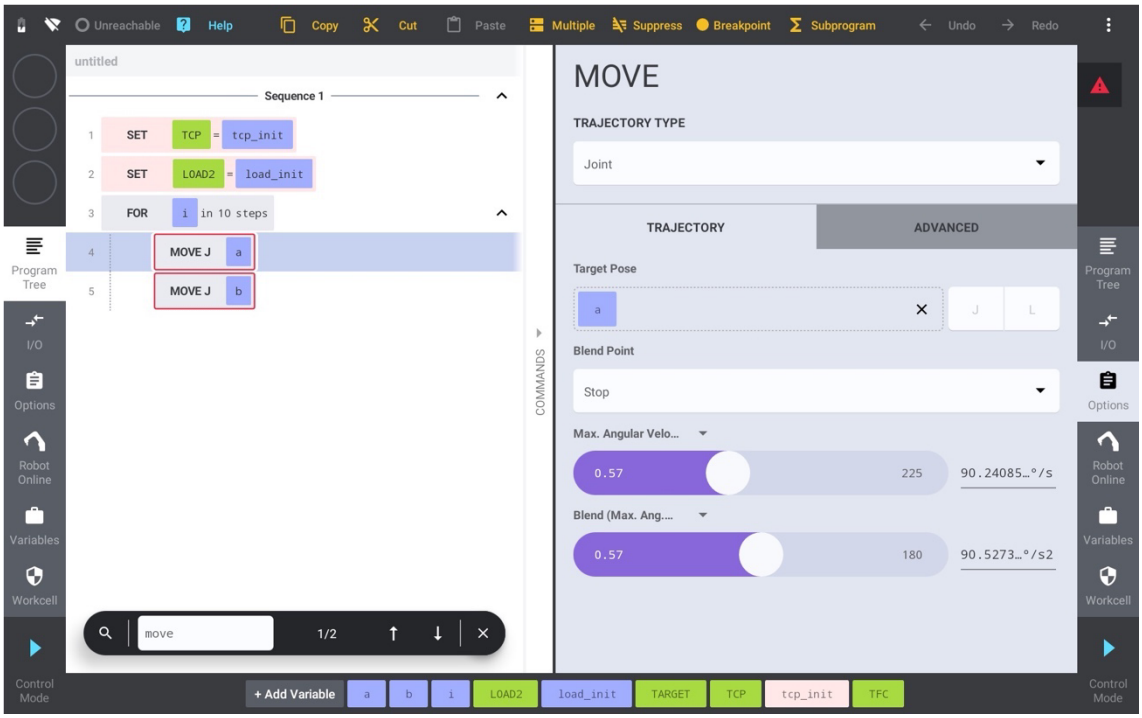


Item	Description
Fold All	The **Fold All** feature allows users to collapse all sequences withing the Program Tree, providing and improved overview of the entire program structure. If certain sequences were already collapsed, they will remain in their folded state even after the user clicks on the **Unfold** option.
Autofocus	If the **Autofocus** feature is enabled, the Program Tree automatically scrolls to the active (currently executed) command. The tracking of active commands is functional and accurate only when the open and the active programs are equal.
Search	The **Search** feature allows comprehensive full text search within Program Tree commands and provides intuitive navigation through search occurrences.
Show Errors	The **Show Errors** feature highlights all invalid commands for easy identification of errors and offers intuitive navigation through the affected commands. The Program Tree automatically scrolls as the user navigates through previous or next error occurrences.

Program Search

The Search feature allows comprehensive full text search within Program Tree commands and provides intuitive navigation through search occurrences. User can enter multiple keywords, separated by a space, or utilize the drag-and-drop feature for commands, variables or Cbun devices. Notably, the search function

is case-insensitive, ensuring a seamless search experience regardless of letter case.

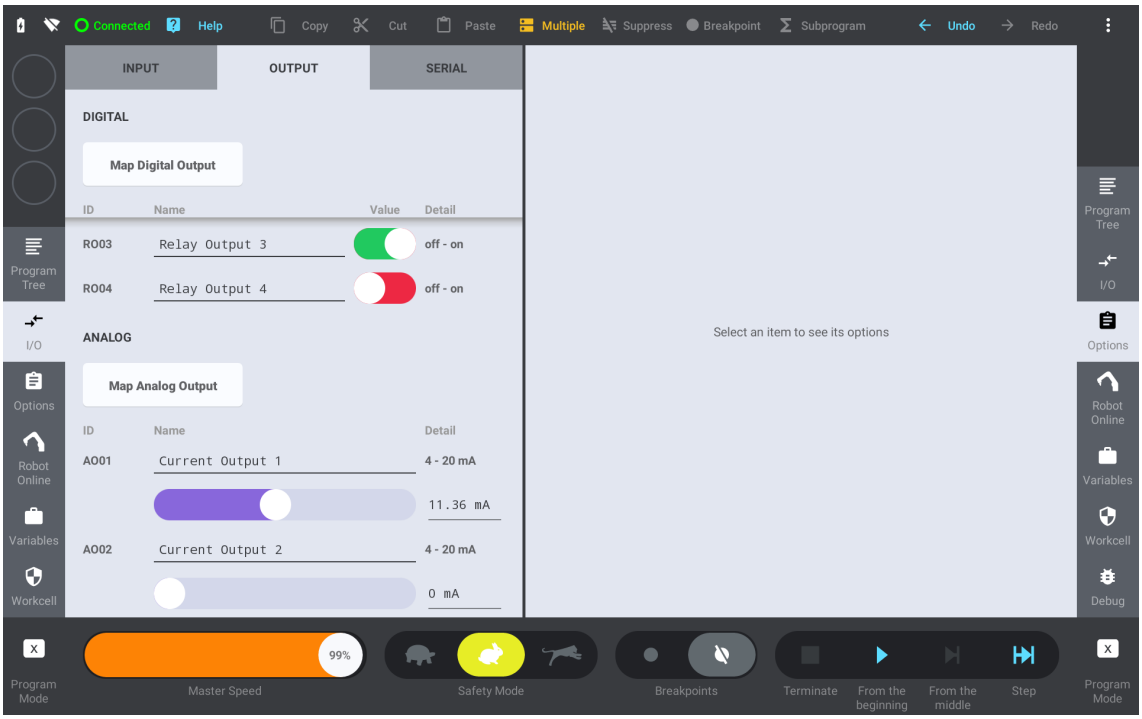


Item	Description
Command Label	The command matches the search, if the command label (such as IF, MOVE, WAIT, etc.) contains the requested search text.
Variable Label	The command matches the search, if the label of some of the involved variables contains the requested search text.
Dialog Content	The DIALOG command matches the search, if the title or message contains the given search text.
Comment	The Comment command matches the search, if the comment contains the given search text.
Cbun Device	The DEVICE command matches the search, if the selected device action label contains the given search text.
MOVE Params	The MOVE command matches the search, if the trajectory type (linear, joint, arc, spline or shape) contains the given search text.

3.3 I/O Interface

The **I/O** pan provides you with an interface for configuration, real-time monitoring and direct control of the robot's input/output signals (I/Os). Moreover, it offers the capability to read from and write to the robot's serial ports.

The I/O tool serves as an invaluable asset, particularly during the robot programming phase, as it allows you to map selected I/O signals into the system variables. Additionally, during the debugging process, the I/O controls play a vital role by allowing you to monitor the current I/O values in real time. The monitoring capability assists in identifying potential issues and ensures the smooth operation of the robot system by providing insights into the flow of data and signals.



Kassow Robots are equipped with two set of input/output ports: one located at the RC cabinet (**IO Board**) and the other at the end of arm tool flange (**Tool IO**). Both groups can be controlled and monitored from the robot programs, Cbun modules or the I/O tool. The configuration of the I/Os is possible only from the I/O tool or Cbun modules.

The robot's ports are organized into three groups: INPUT, OUTPUT and SERIAL.

3.3.1 Input

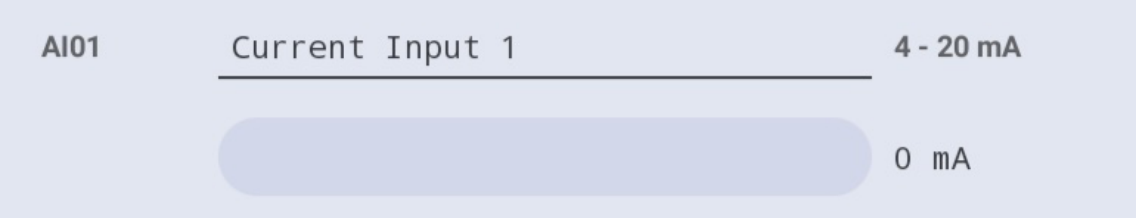
The Input tab provides you with an interface to access signal input ports of the I/O-board and the Tool-I/O. The input ports are organized into four sections: Digital, Analog, Quadrature and Configurable. Each section contains list of related input ports, along with the **Map Input** button for system variable initialization. It's important to note that the content of the tab may vary depending on the specific version of the robot.

Digital Input

The digital input port item provides quick access to the current input state. The red indicator signifies that the input is inactive, whereas the green indicator indicates that the input is active. Additionally, you can verify the supported input range for each port.

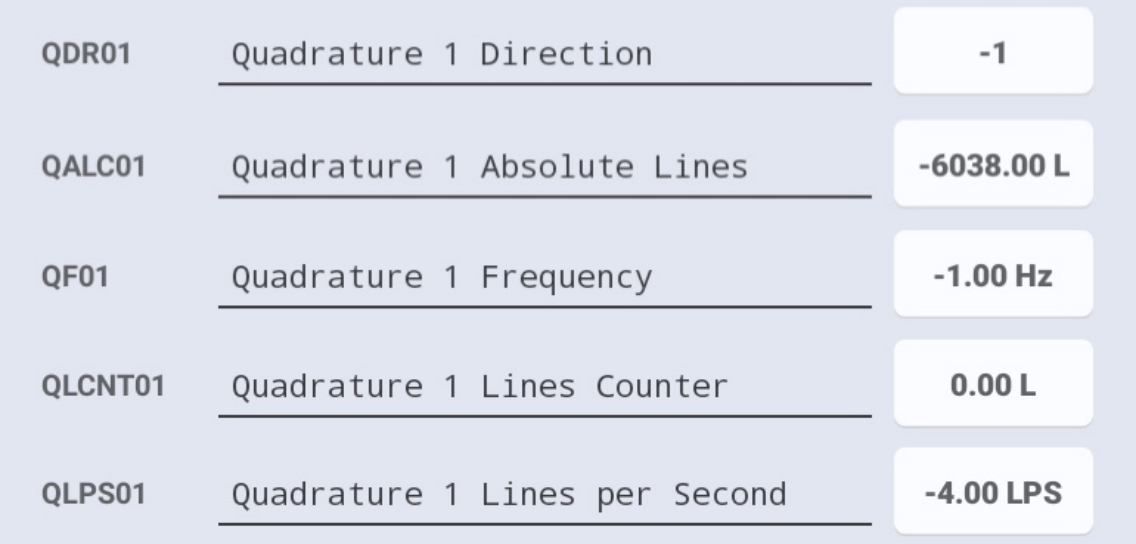
Analog Input

The analog input port item provides quick access to the current input value. You can monitor the value either in form of the indicator bar, which shows the value within the port value range, or as a floating-point number along with its physical units. Additionally, you can verify the supported input range for each port.



Quadrature Input

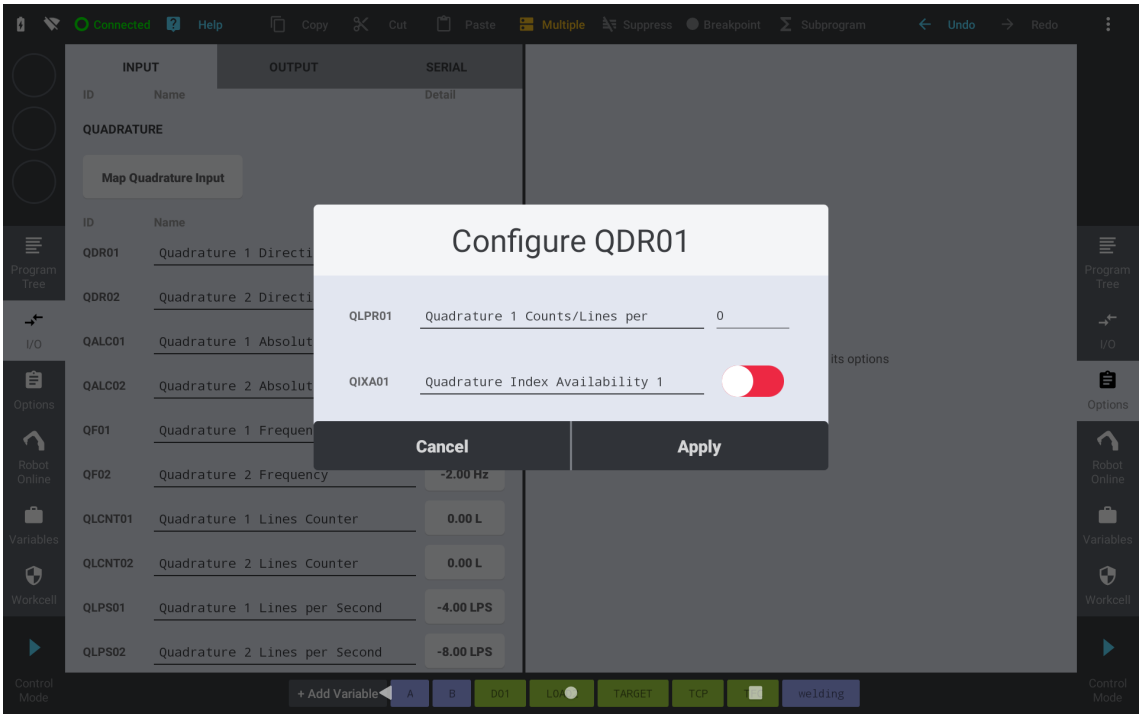
The quadrature interface allows you to equip the robot with quadrature encoder. Whereas each physical quadrature interface is based on four digital inputs, the UI representation consists of five input items.



Item	Description
Direction	The direction input denotes the direction of encoder's shaft rotation, represented by values of 1 and -1.
Absolute Lines	The absolute lines input represents the cumulative encoder position, including the direction, from the start of the measurement.
Frequency	The frequency input provides a measure of the rotational speed of the encoder's shaft, and it is calculated as the lines per second divided by 4.

Lines Counter	The lines counter reflects the count of the encoder's lines crossed within the encoder's shaft revolution (i.e., full revolution resets the counter).
Liner per Second	The lines per second input indicates how many signal transitions occur in one second for both A and B channels.

Accurate functioning of the quadrature Lines Counter input requires the correct quadrature interface configuration. To initiate the configuration process, simply click on the measured value of any quadrature item. This action will open the Configure Quadrature dialog, allowing for the necessary adjustments.



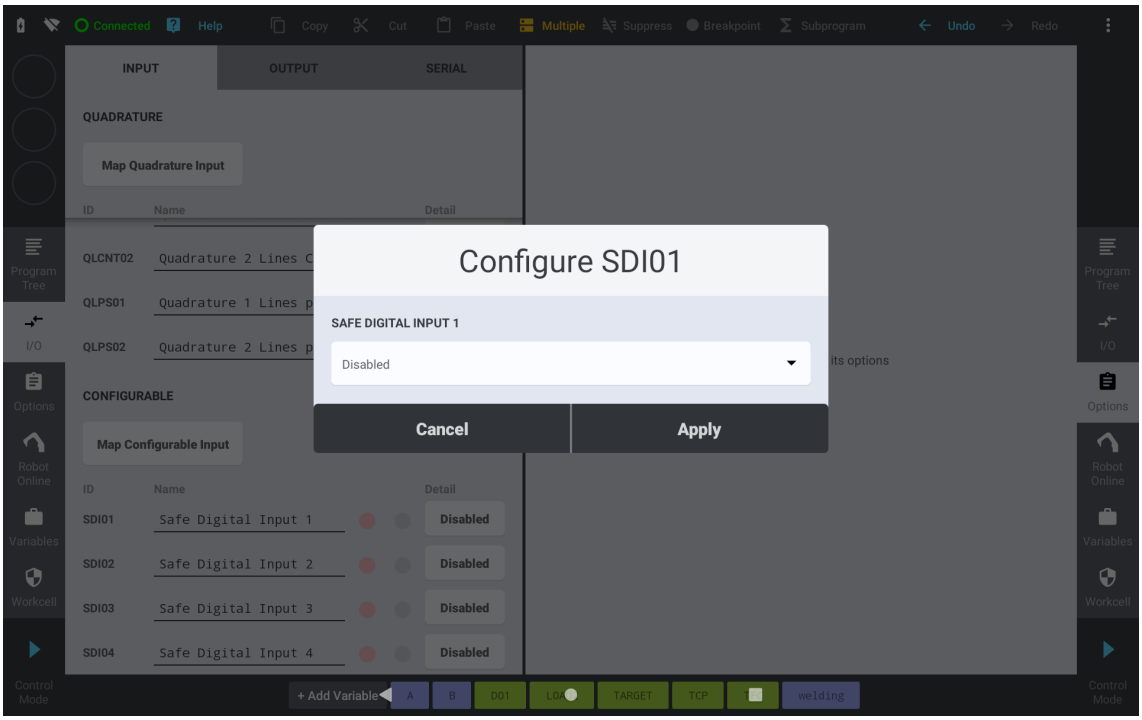
Item	Description
Lines per Revolution	The lines per revolution specifies the number of lines required for a complete revolution. When the rotation reaches a full revolution as defined by this setting, the **Lines Counter** is automatically reset.
Index Avail.	The index availability determines whether the encoder has an index signal available. If the Index Availability is enabled and the index signal is present, it is utilized for the direct reset of the **Lines Counter**. In other cases the counter range is curbed by the **Lines per Revolution** field.

Configurable Input

A configurable input port is a port whose behaviour can be adjusted or customized by the robot user. The selected configuration is displayed as a detail of the corresponding input port item, ensuring visibility and easy access to the current settings. Depending on the port type and its configuration, the configurable input may function similarly to a digital or analog input.

For instance, the Safe Digital Input acts as a digital input and its functionality can be enabled or disabled.

To initiate the configuration process, simply click on the selected settings (detail). This action will open the Configure Port dialog, allowing for the necessary adjustments.



3.3.2 Output

The Output tab provides you with an interface to access signal output ports of the I/O-board and the Tool-I/O. The output ports are organized into three sections: Digital, Analog and Configurable. Each section contains list of related output ports, along with the **Map Output** button for system variable initialization. It's important to note that the content of the tab may vary depending on the specific version of the robot.

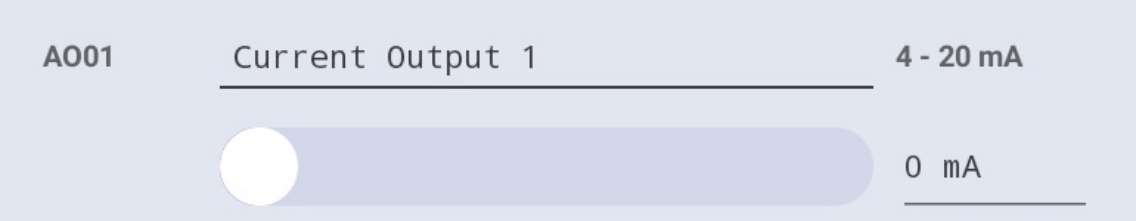
Digital Output

The digital output port item provides quick access to both setting and checking the current output state. The red switch state signifies that the output is inactive, whereas the green switch state indicates that the output is active. Furthermore, you can verify the supported output range for each port.

Analog Output

The analog output port item provides quick access to both setting and checking the current output value.

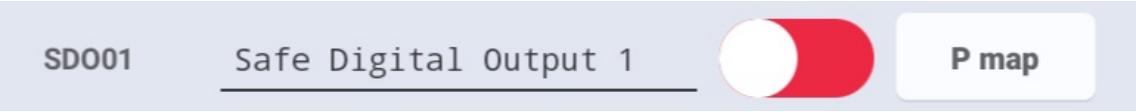
You can set the value either in form of the slider bar, which continuously adjusts the value within the port value range, or as a floating-point number. Additionally, you can verify the output type (voltage vs current) and the supported output range for each port.



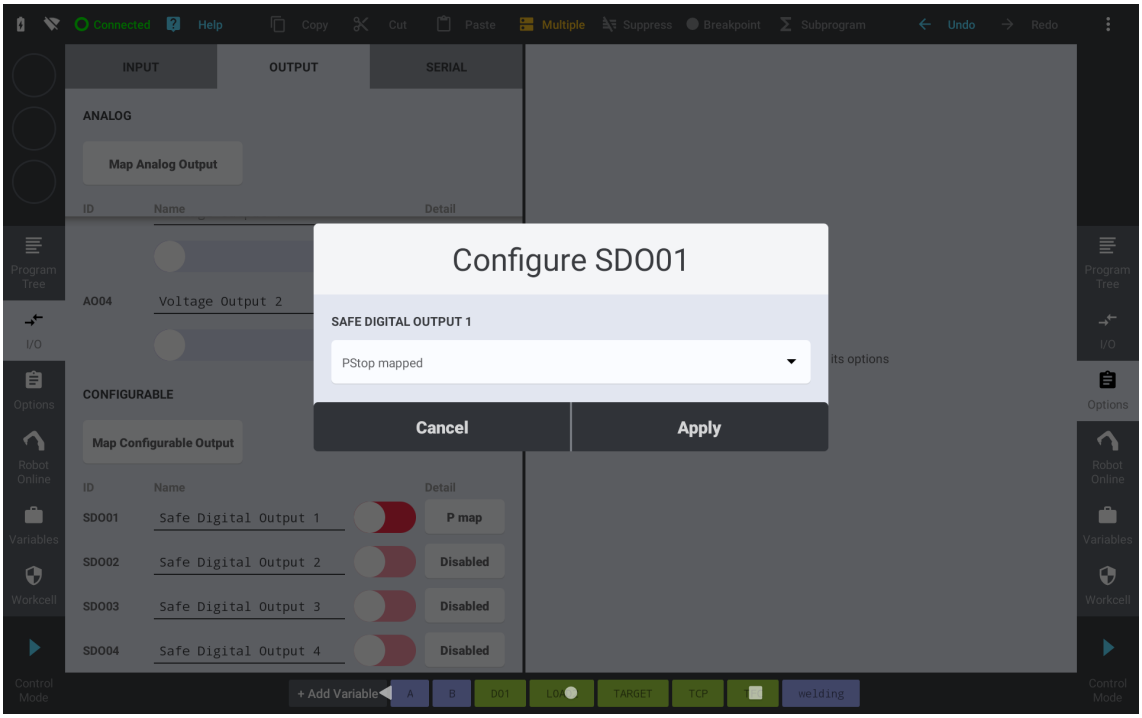
Configurable Output

A configurable output port is a port whose behaviour can be adjusted or customized by the robot user. The selected configuration is displayed as a detail of the corresponding output port item, ensuring visibility and easy access to the current settings. Depending on the port type and its configuration, the configurable output may function similarly to a digital or analog output.

For instance, the Safe Digital Output acts as a digital output and its functionality can be disable, enabled or mapped to P-Stop, E-Stop or both.



To initiate the configuration process, simply click on the selected settings (detail). This action will open the Configure Port dialog, allowing for the necessary adjustments.



3.3.3 Serial

The Serial tab offers an interface for accessing the serial communication ports of both the IO Board and Tool IO. Please note that the contents of the tab may vary depending on the specific robot version.

The serial communication port item provides convenient options for both writing to and reading from the communication port. To write a data, click the upper (output) box, enter the desired message content and click the arrow button. All received data are displayed in the bottom (input) box. To clear the input box, click the cross button.

Additionally, you can configure various parameters of the serial communication port, including baud rate and number of data bits, among others. To adjust the settings, click the desired setting and select the requested option. Configuring the serial communication port is crucial to ensure that the sender and receiver devices operate at the compatible protocol.

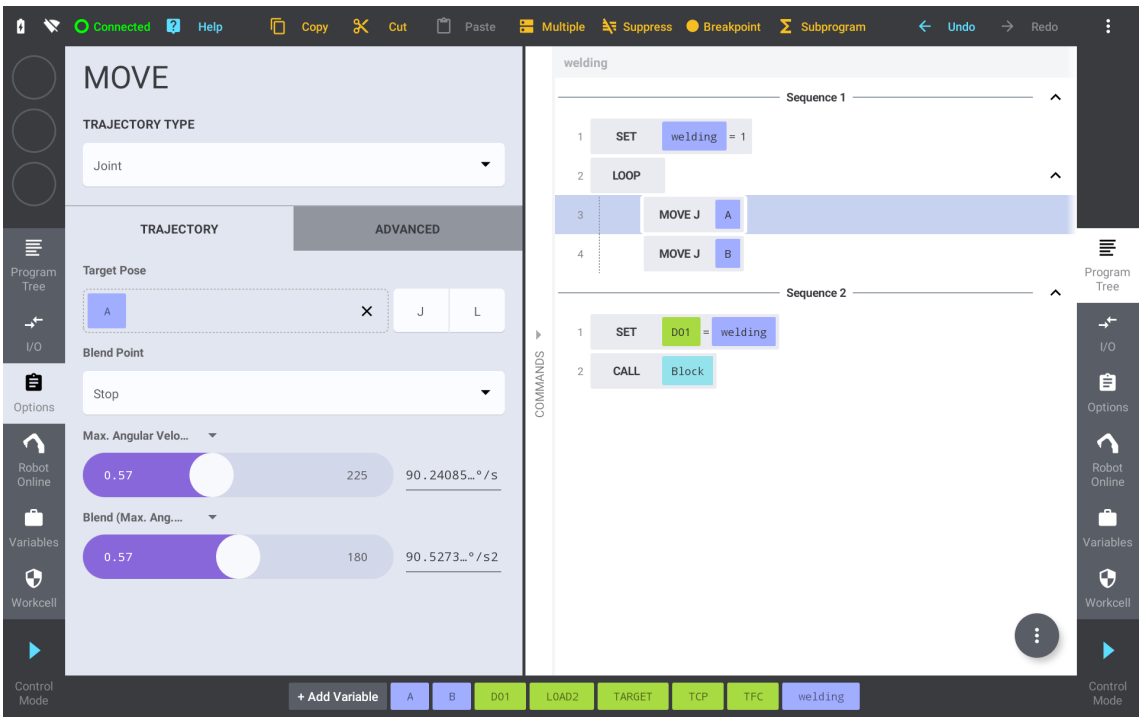
Item	Description
Baud Rate	The baud rate setting refers to the rate at which data is transmitted over the communication channel. It signifies the number of signal/symbol changes that occur per second.
Data Bits	The data bits configuration specifies the number of bits used to transmit each character. The setting determines the size of the data packet being sent or received.
Stop Bits	The stop bits settings refer to the number of bits sent at the end o each data packet to signal the end of a transmission.
Parity	The parity configuration specifies an error-checking process used to ensure the accuracy of transmitted data.

Please note that the writing to and reading from the serial port is exclusively accessible through the IO API from your own Cbun module. Unlike previous inputs and outputs, the serial communication port cannot be

directly accessed via system variable from the robot program. To do so, use the Serial Port interface of System Utils Cbun [Appendix D, page 144].

3.4 Options

The Options tool displays settings of selected program command, workcell element or variable, providing a distinct interface for each element type. If none or multiple elements are selected, the Options tool remains empty. The Options tool is typically opened automatically upon selecting a program or workcell element. However, in certain scenarios, such as when Program Tree tool is open on both sides of the screen, you may need to manually open the Options panel.



Options panes of various program and workcell elements are described in details later in Variables [page 42], Commands [page 56] and Workcell chapters [page 96].

3.5 Robot Online

The Robot Online tool provides an interface for jogging the robot either by moving the tool linearly within the robot's Workspace or by moving individual robot's joints. During the robot programming process, the Robot Online tool plays a vital role by allowing you to move the robot within its Workspace in real-time.

By default, the robot jogging speed is affected by the Master Speed settings. However, users have the option to disable this behaviour by navigating to *Settings -> Advanced -> Jogging Master Speed*.

Please note that the robot jogging feature is disabled when using the Dashboard CbunX App, which provides 3D visualization of the robot and its environment.

3.5.1 Control Modes

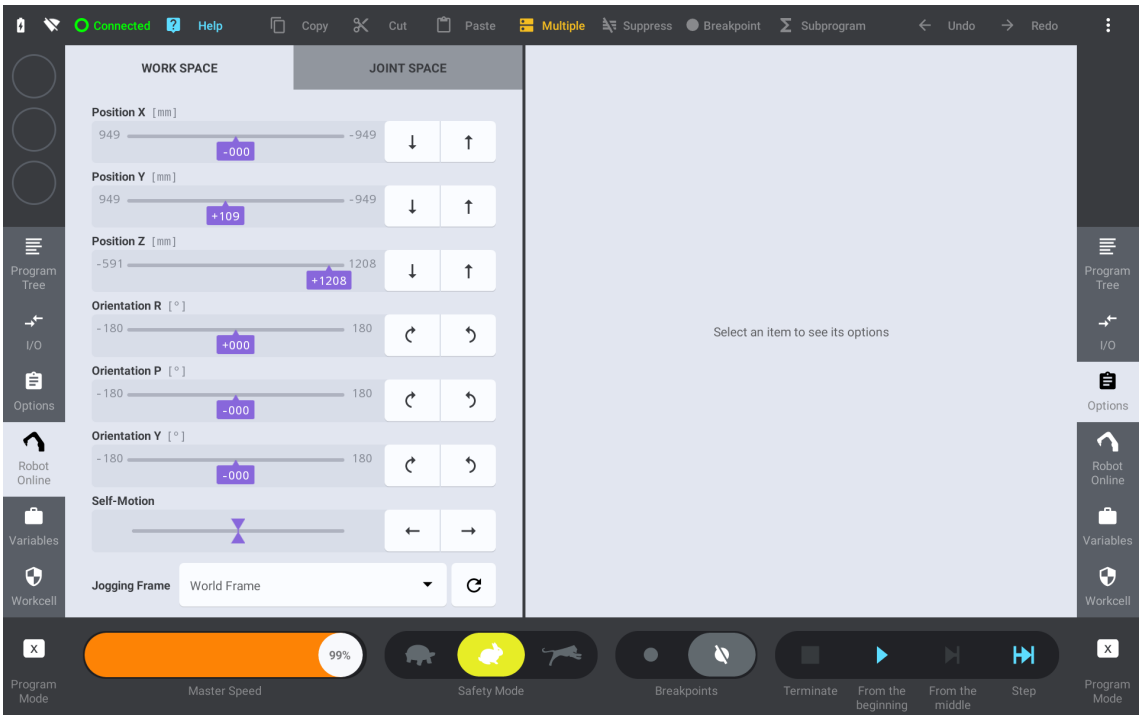
All Robot Online buttons, excluding Self-Motion, support three control modes. A short button tap (indicated by 1 vibration) leads to a micro fine tuning (0.1 mm in Workspace, 0.1° in Jointspace), while a prolonged button click (2 vibrations) enables a macro fine tuning (1 mm in Workspace, 1° in Jointspace). Pressing and holding a button initiates continuous robot movement (indicated by 3 vibrations). In this mode, the robot moves along the selected axis if the button is held down.

3.5.2 Workspace

The Workspace tab allows users to jog the robot within the Workspace, translating and rotating the tool center point (TCP) of the robot along the XYZ axes of the selected jogging frame. Additionally, the Self-Motion control provides a way to adjust the position of the robot's elbow while maintaining the position and orientation of the tool.

By default, the Workspace jogging frame is set to the World Frame, but users have the option to select any available pose variable. The coordinates of the chosen pose variable at the time of selection define the new jogging frame. This frame remains static and does not reflect any updates of the source pose variable. To synchronize the frame from the new pose variable coordinates, simply click the **refresh** button.

The limits of the individual position coordinates are obtained from the robot's Workspace bounding box. However, it's important to note that the robot may not be able to reach the individual these limits due to the different shape of the real robot's Workspace.



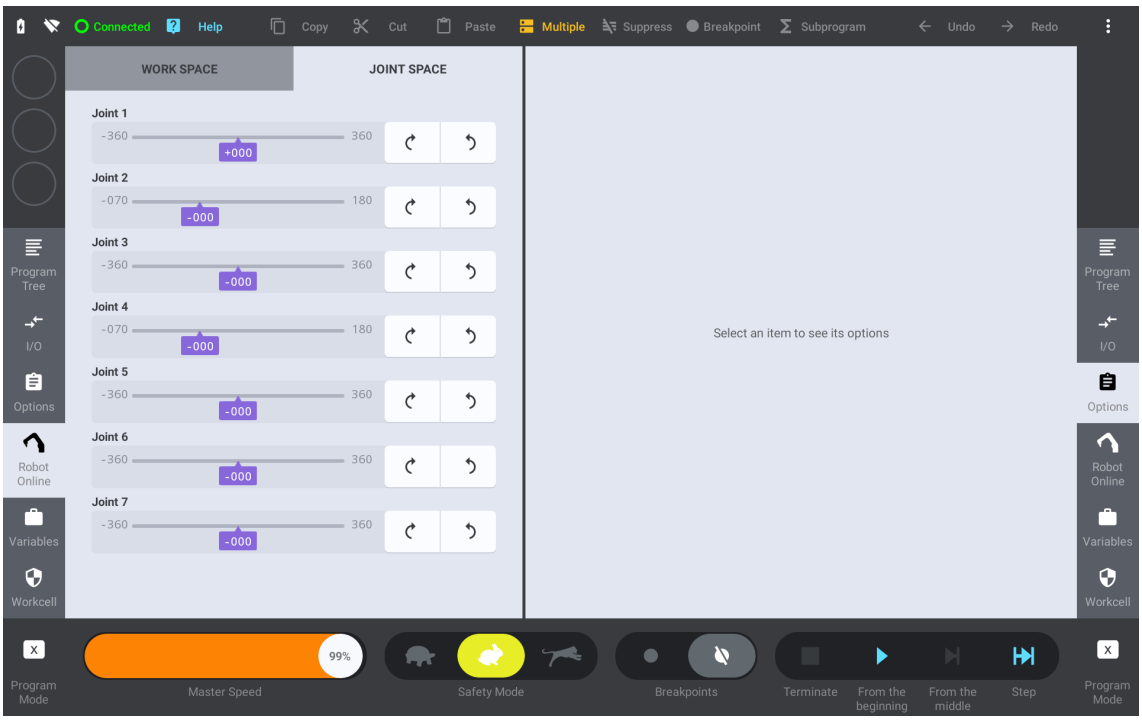
Although translating the robot along any axis directly affects the corresponding position coordinate, the behaviour differs slightly when it comes to rotation. You may notice that rotating the robot around one axis can lead to changes in a different orientation coordinate/s.

The discrepancy arises because click the button always rotates the tool around the selected axis of the static jogging frame, but the roll, pitch and yaw (RPY) coordinates express the achieved orientation of the TCP within the selected jogging frame.

Thanks to this, rotating the robot tool is intuitive and the final orientation coordinates are well defined.

3.5.3 Jointspace

The Jointspace tab allows users to jog individual robot joints around their corresponding axes. The limits of the individual joints are determined by the physical capabilities of the robot's joints. When a joint limit is reached, a warning dialog appears to notify the user.



3.6 Variables

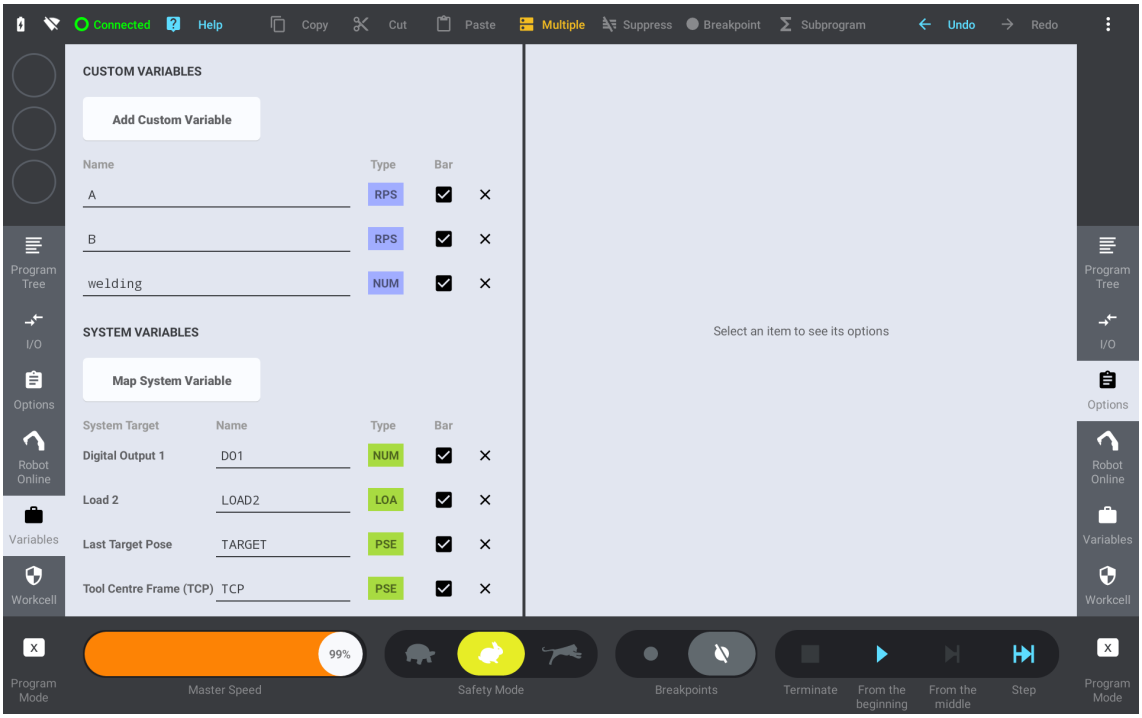
The Variables tool displays the complete list of all variables within the open program tree. During the robot programming phase, the tool consists of two sections: Custom and System Variables. When the program is being executed, the third, Runtime, section is added to allow users to monitor variables within the active robot program [7.6.5, page 95].

Individual variable items provide users with a quick access to the variable label and data type (see below). The shortcut checkbox determines whether the variable will be listed in the bottom bar.

- NUM: number
- PSE/RPS: pose/robot pose
- LOA: load
- PTR: grid pattern

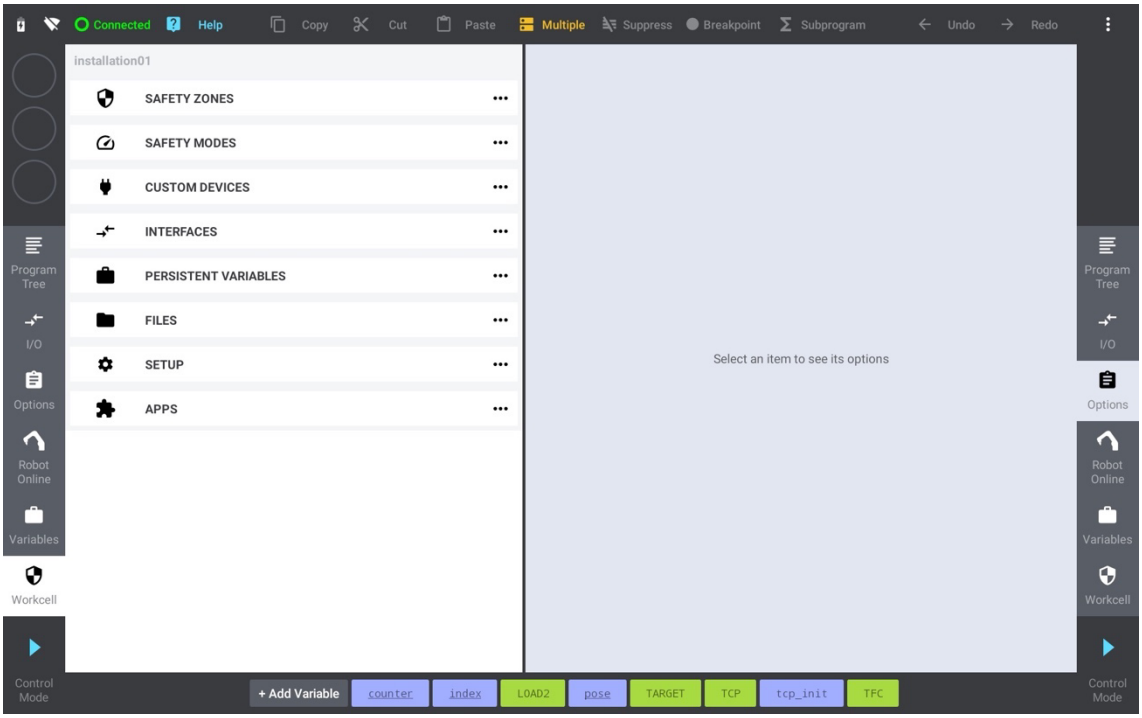
- SHP: shape
- ARR: array

You can add user defined variable by clicking **Add Custom Variable** button or map some system property (except I/Os) into a variable by clicking the **Map System Variable** button. To modify variable's value or properties, select the variable and open it's Options pane. To remove a variable, either click the **Cross** button or select the variable and click the **Cut** button.



3.7 Workcell

The Workcell group allows you to configure present robot installation and all elements related to the robot application setup. Whole chapter is dedicated to this part of the system [Ch. 8, page 96].



3.8 Expression Builder

The expression builder enables users to create a wide range of expressions for use with with various program commands, where access to objects or mathematical formulas is required. A typical expression consists of one or more operands, such as constants, variables or functions, combined with operators. These expressions are evaluated during program execution, returning a calculated value to be processed by the corresponding command.

To open the expression builder, select one of the following program commands, open its Options panel and click on the parameter that is designed to accept expression (see below).

Command	Parameters
IF	EXPRESSION
LOOP	EXPRESSION
MOVE	Target Pose, Knot Pose, Through Pose, Shape
SET	TARGET VARIABLE, EXPRESSION
WAIT	EXPRESSION
DIALOG	MESSAGE, DISMISS ON SIGNAL
SET	TARGET VARIABLE, EXPRESSION

The user can enter the expression via keyboard or construct the expression by selecting items from the hint bar. When the **Auto** hint mode is chosen, the hint bar's content is context-based, displaying only compatible operands and operators. In addition, the user has option to display all variables, operators, functions or special symbols. For more details see the Appendix - Expressions.

3.9 Teaching

It is of a common practice in the world of robot programming to jog or free-drive the robot manually and define poses and related motions based on the physical coordinates of the EOAT (end of arm tool) and context of the intended application. The basic way how to read and store the actual TCP or the robot configuration in the reusable Pose variable is provided by the TPUIO, e.g. when the new Pose variable is created/modified by the user.

To give the user more freedom and release his hands from the TP the recent software version implements a set of supportive *teaching functions*. Invoking the teaching function is simple, it requires to double click of the teaching button, either the one placed at the Tool IO of the robot arm, or the one at the back side of the TP.

Based on the context of the TPUI, the teaching function provides an automated action as stated in the following table.

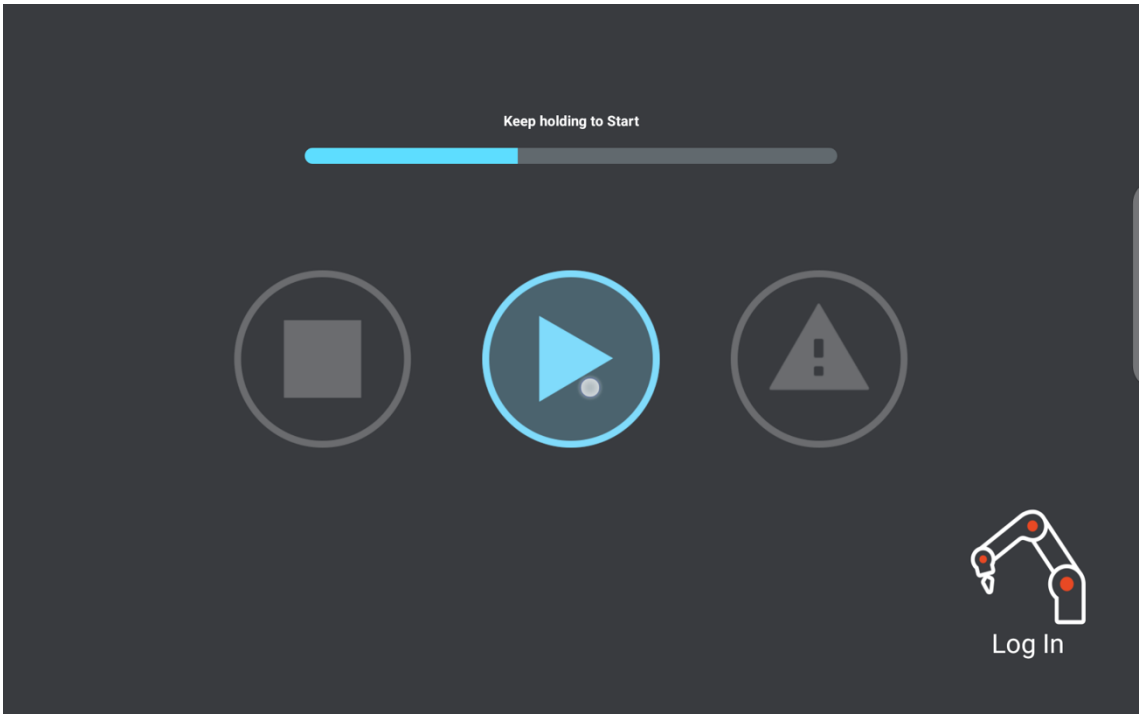
User Context	Teaching Function
Move command selected (with the empty target pose)	Creates the new Pose variable (labelled “P#number”) having the actual robot TCP/JointConfiguration (JCF) and assigns it to to the selected MOVE command target.
Move command selected (target pose already set)	Duplicates the selected MOVE command and place it after the actual selection. Creates the new Pose variable (labelled “P#number”) having the actual robot TCP/JCF and assigns it to the freshly created MOVE as target pose.
Non-array variable selected	Creates the new Pose variable (labelled “P#number”) having the actual robot TCP/JCF.
Array variable selected (array contains Pose typed column)	Takes the first column of the Pose type and define the actual robot TCP/JCF in the first empty row (or adds a new row at the end).
Array variable selected (array doesn’t contain Pose typed column)	The new Pose column is added to the array definition and fills-in its first row by the actual robot TCP/JCF.

Teaching functions are quite an efficient tool helpful in situations, where the row of move commands or poses are expected and defined from the physical robot configurations (teached). In this case the user can eliminate any extra interaction with the TPUI and focus on the robot arm positioning with the semi-automated program alteration provided by the teach button double clicks.

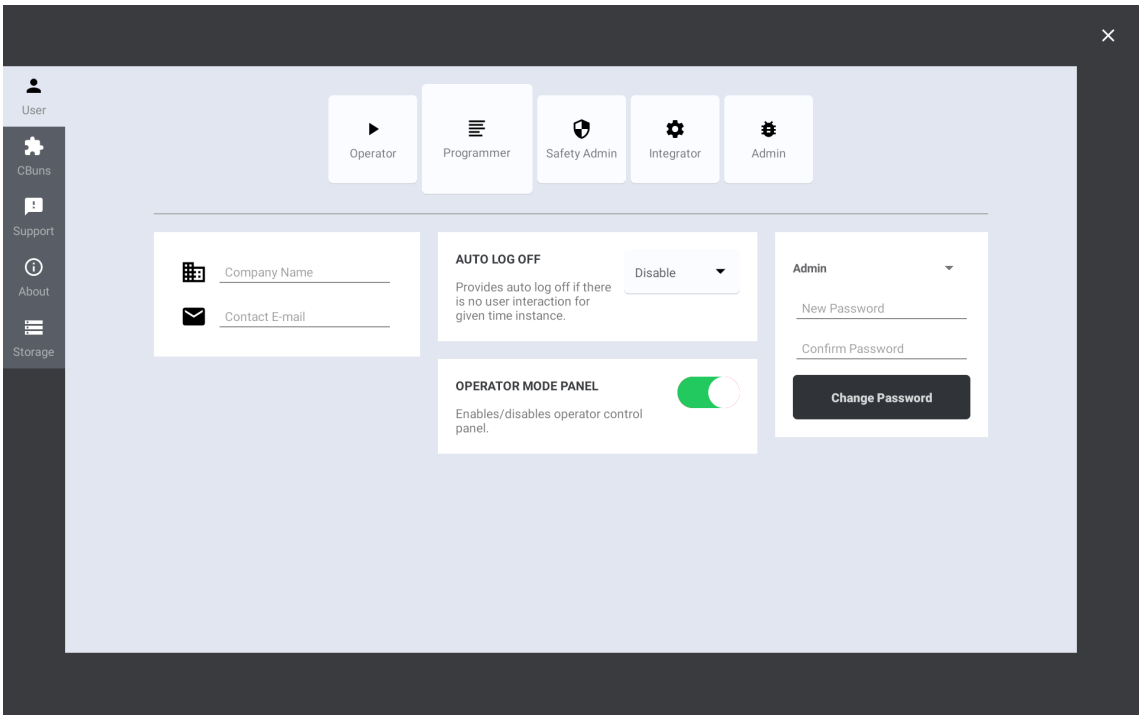
3.10 Operator Mode Panel

The Operator Mode Panel allows the robot programmer to restrict the TPUI to a simple control panel, providing the robot operator with a limited interface solely for launching or terminating the robot program. Please note that to launch or terminate the program, you need to click and hold the corresponding button.

Additionally, the robot operator has option to access robot warning and error messages.



To enable the Operator Mode Panel, navigate to *Settings -> User* and log in as any user (except Operator). Once logged in, you can activate the Operator Mode Panel feature. From now on, switching the user to Operator will display the Operator Mode Panel.



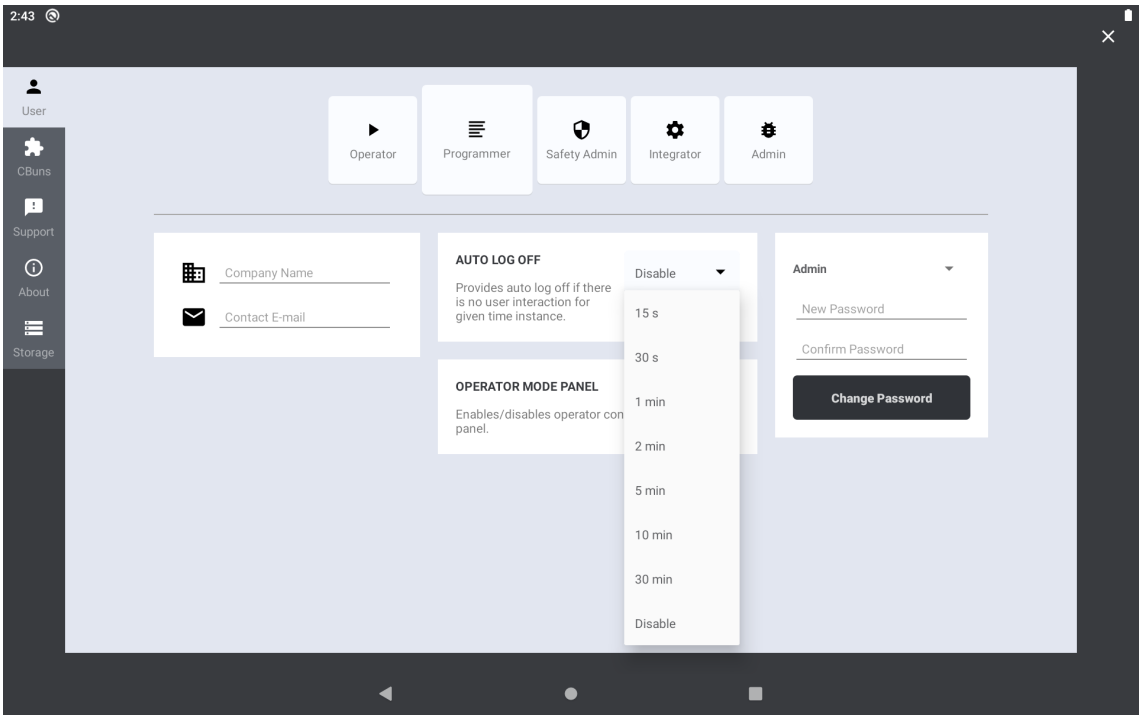
To close the Operator Mode Panel, click the **Log In** button, select the login user and enter the corresponding password.

To disable the Operator Mode Panel feature, navigate to **Settings -> User** and deactivate it.

3.10.1 Auto Log Off

The Auto Log Off feature automatically switches the user to Operator if there is no user interaction for the specified period. This is essential for maintaining security as it prevents unauthorized access to the robot programming interface, minimizing the risk of unintended changes to the robot program or settings. This feature is particularly valuable when combined with the Operator Mode Panel.

By default, the Auto Log Off feature is disabled. To activate it, navigate to *Settings -> User* and log in as any user (except Operator). Once logged in, you can select the maximum period of user inactivity.



4 Storage

4.1 Locations

Kassow Robots control software allows users to configure where files are stored (Programs, Workcells and Cbun Installers). Some of the storage locations are available by default (Tablet and Robot Storage), some require specific HW (USB Drive) and some have to be configured in advance (Google Drive).

Note: Available storage locations depend on specific storage operation.

4.1.1 Tablet Storage

 Built-in tablet (teach pendant) storage is designed for storing programs and workcells while the robot is offline (turned off). Ie. tablet storage allows users to manage their programs and workcells without the need to turn on the robot.

4.1.2 Robot Storage

 Built-in robot (cabinet) storage is the default location for storing programs and workcells while the robot is online (turned on). Robot storage also contains Cbun installers that are provided by Kassow Robots.

4.1.3 USB Drive

 USB Drive allows users to attach their own USB Flash Drive devices. Such storage can be used for storing programs and workcells and for installation of Cbuns provided by 3rd party.

Note: USB Flash Drive devices are only mounted if they contain one of the following filesystems: exFAT, VFAT, EXT2, EXT3, EXT4 or HFS+.

4.1.4 Google Drive

 Google Drive provides cloud storage for programs, workcells and 3rd party Cbuns Installers. Google Drive storage has to be configured in advance and requires internet access via WiFi.

4.2 Programs and Workcells

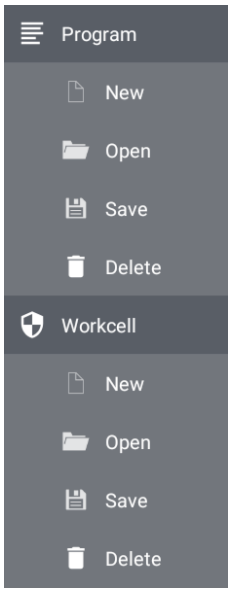
Both programs and workcells can be saved on Tablet Storage, Robot Storage, USB Drive or Google Drive. While files saved on Tablet and Robot Storage are associated with a particular robot, saving a file on USB Drive or Google Drive allows its access on any other robot.

Note: Regular saving of your program and workcell can help prevent data loss.

4.2.1 New File

To create new program or workcell:

1. Open Popup menu ⋮
2. Click **New**
3. Confirm by clicking **Yes**

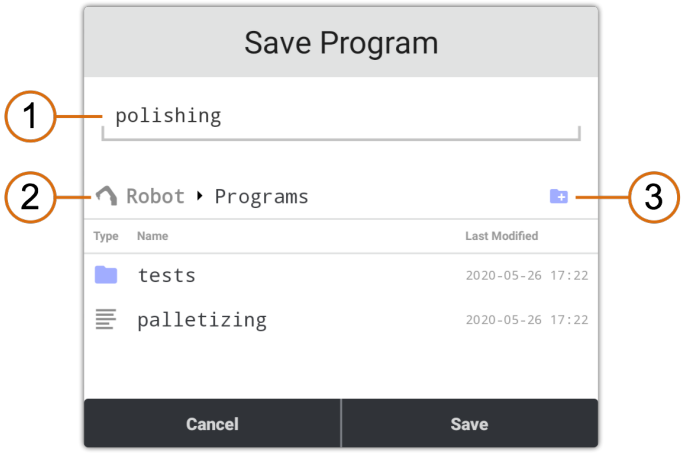


Warning: Any unsaved changes will be lost.

4.2.2 Save File

To save your program or workcell:

1. Open Popup menu ⋮
2. Click **Save**
3. Confirm by clicking **Yes**
4. Enter name, select location and click **Save**



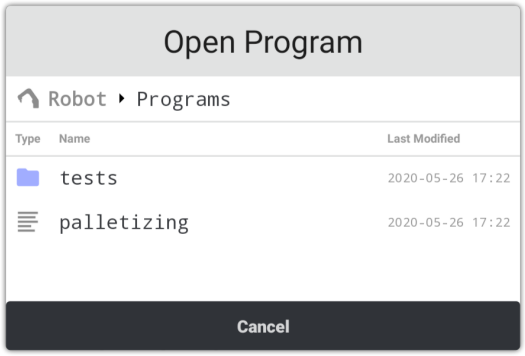
Item	Description
1	**File Name** Unique file identification in selected folder. File with same name will be overwritten.

2	**Storage Navigation** Displays selected storage location. Clicking the icon provides navigation through available storage locations.
3	**New Folder** Allows users to create new folder inside the selected folder.

4.2.3 Open File

To open your program or workcell:

1. Open Popup menu
2. Click **Open**
3. Select location and click the file to be opened



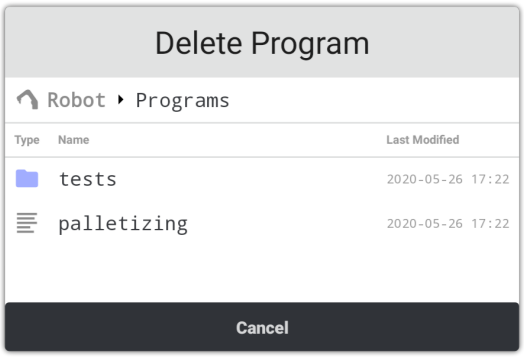
Warning: Any unsaved changes will be lost.

4.2.4 Delete File

To delete your program or workcell:

1. Open Popup menu
2. Click **Delete**

3. Select location and click the file to be deleted



5 Variables

Variables are one of the basic components of the robot program, enabling the storage and manipulation of various data, such as sensor readings or target robot poses. Variables facilitate calculations and decision-making process, empowering the robot to respond dynamically based on the data stored within them. In addition, variables can be used for accessing robot properties, such as I/Os, frames or loads.

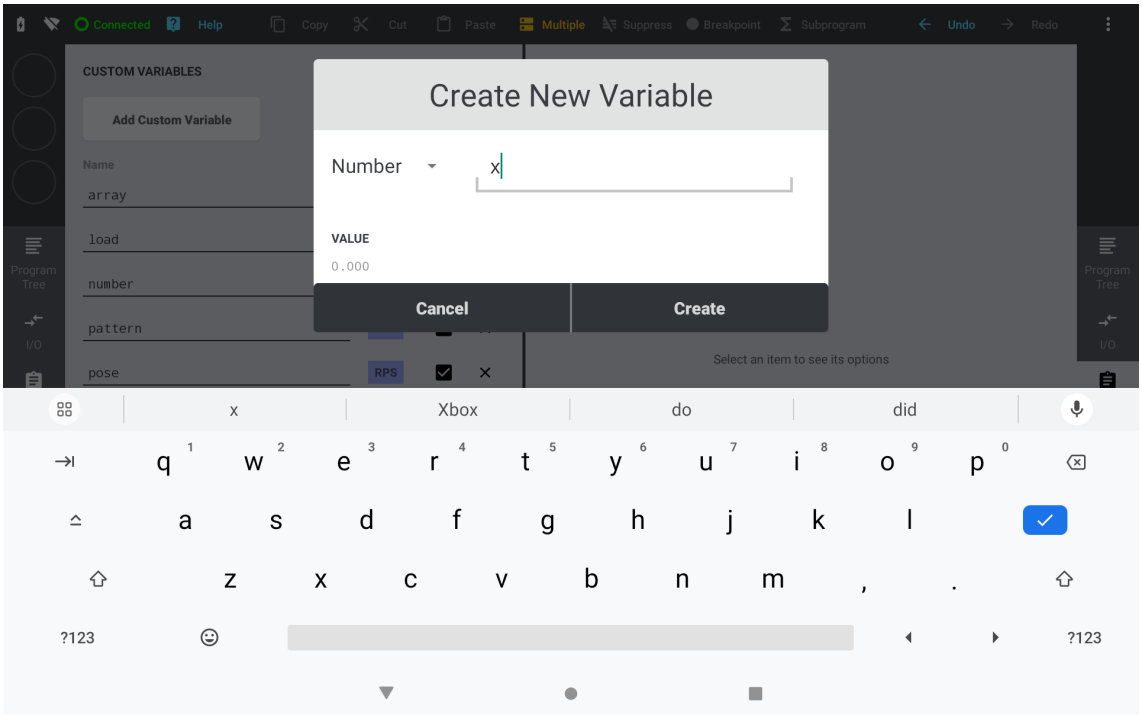
5.1 Custom Variables

Custom variables serve as placeholders for storing and manipulating data within program. The initial value of the custom variable is set by the user during the programming process. When the program is running, this value can be retrieved or updated by program or Cbun commands.

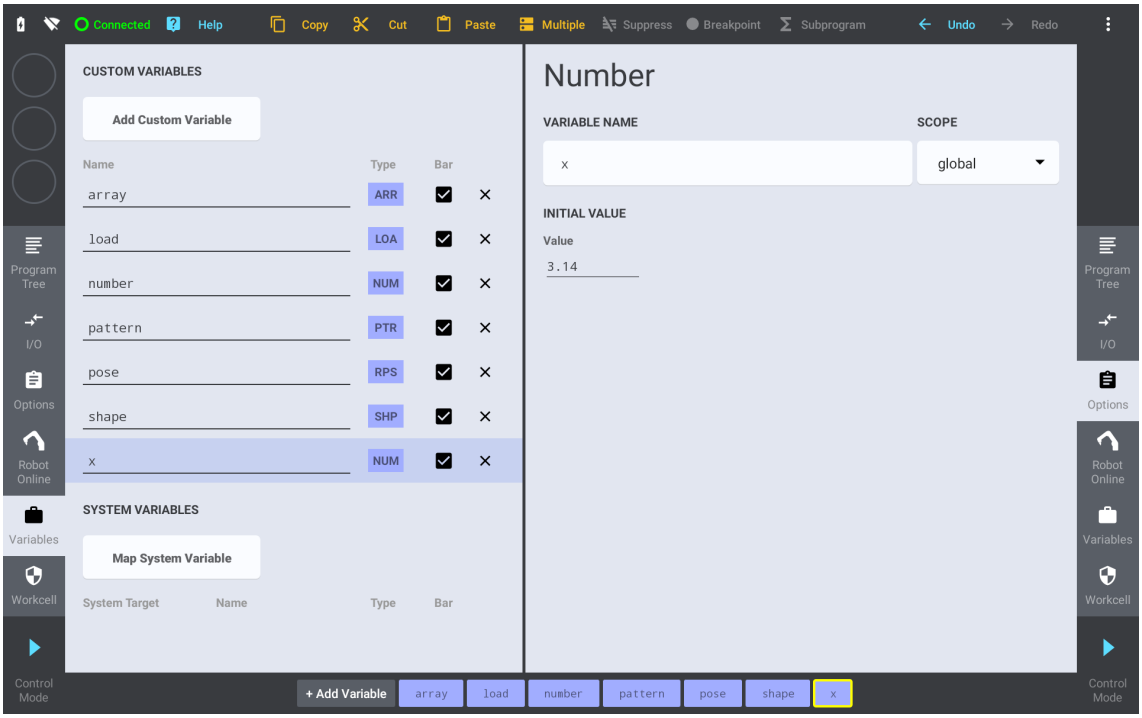
There are 2 ways to create a new custom variable:

- To create global variable, open the Variables tool and click the **Add Custom Variable** button. Optionally, you can switch the Bottom bar to Program Mode and click the **+ Add Variable** button.
- For persistent variable, navigate to *Workcell -> Persistent Variables* and click the **+ Add Variable**.

In both cases, the Create New Variable dialog appears. To create the variable, select the data type, enter a unique name and click the **Create** button.



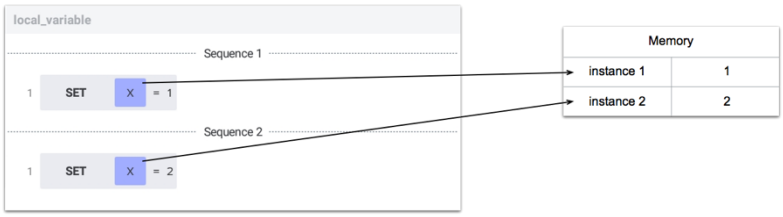
Once the variable is created, you can define the initial value via its options panel. Simply click on the value and modify. You can also rename the variable by clicking on its current name and making adjustments.



In addition, the custom variable's options panel enables selection of the scope. There are three possible options: Local, Global and Persistent. While variables with Local and Global scope are stored within the program tree, persistent variables are in the Workcell. Please refer to the detailed explanation of the custom variable scopes below.

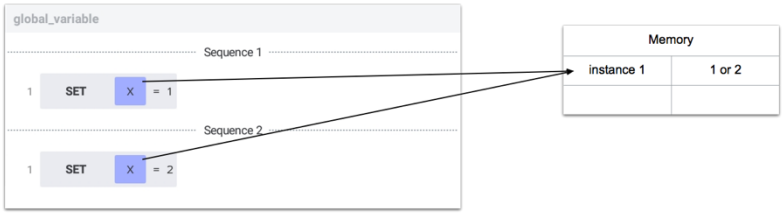
5.1.1 Local Scope

A variable with local scope is initialised to the user-defined initial value when the program is launched. Since the local variable exists within each program sequence independently, its value is not synchronised across the sequences. As a result, each sequence can store different value in the variable.



5.1.2 Global Scope

Similar to the local scope, a global variable is also initialised to the user-defined initial value when the program is launched. However, in this case, the value of the variable is shared across program sequences, allowing the value set by one sequence to be read by other sequences. This behaviour can be utilised for program synchronization, since one sequence can wait for value set by other sequence.



5.1.3 Persistent Scope

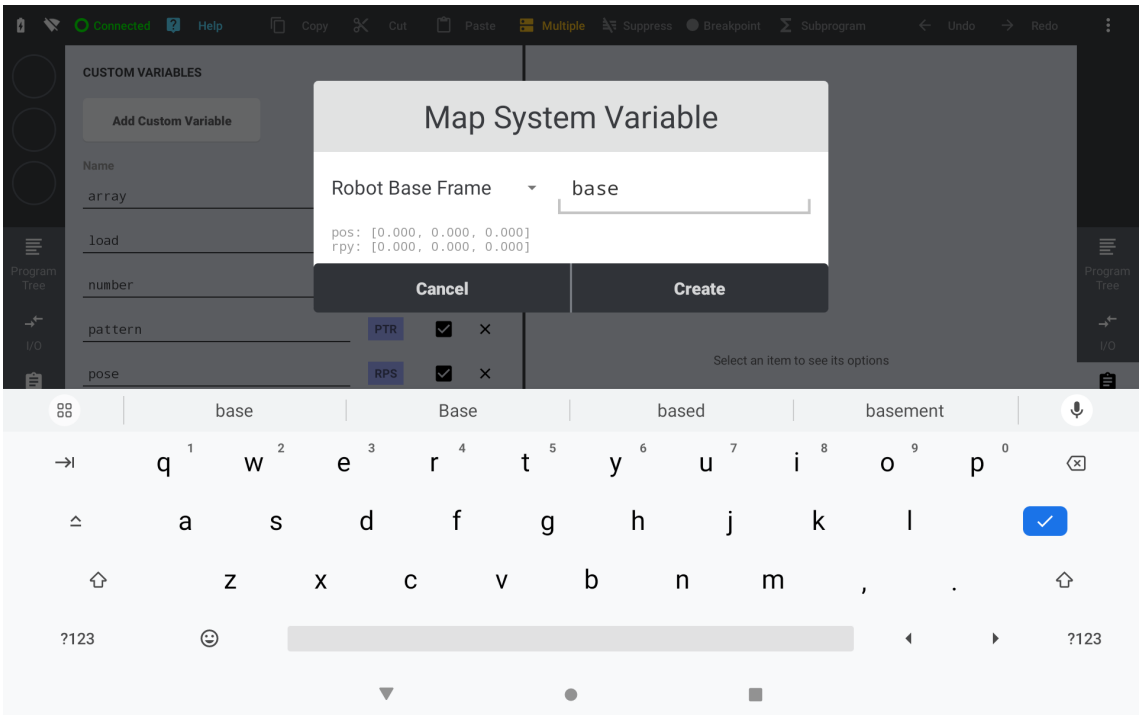
A persistent variable is like a global variable in that its value is shared across all sequences. However, unlike a global variable, a persistent variable exists outside the program execution and its value is recovered after system restart.

5.2 System Variables

System variables are created by mapping real robot properties, such as Loads, Frames, or I/Os, into variables. The value of a system variable always reflects the real state of the robot system. While some system variables are read-only, others can be written, allowing for updating of the corresponding system property. For instance, a system variable can be used to set the digital/analog output or configure the TCP frame during the program execution.

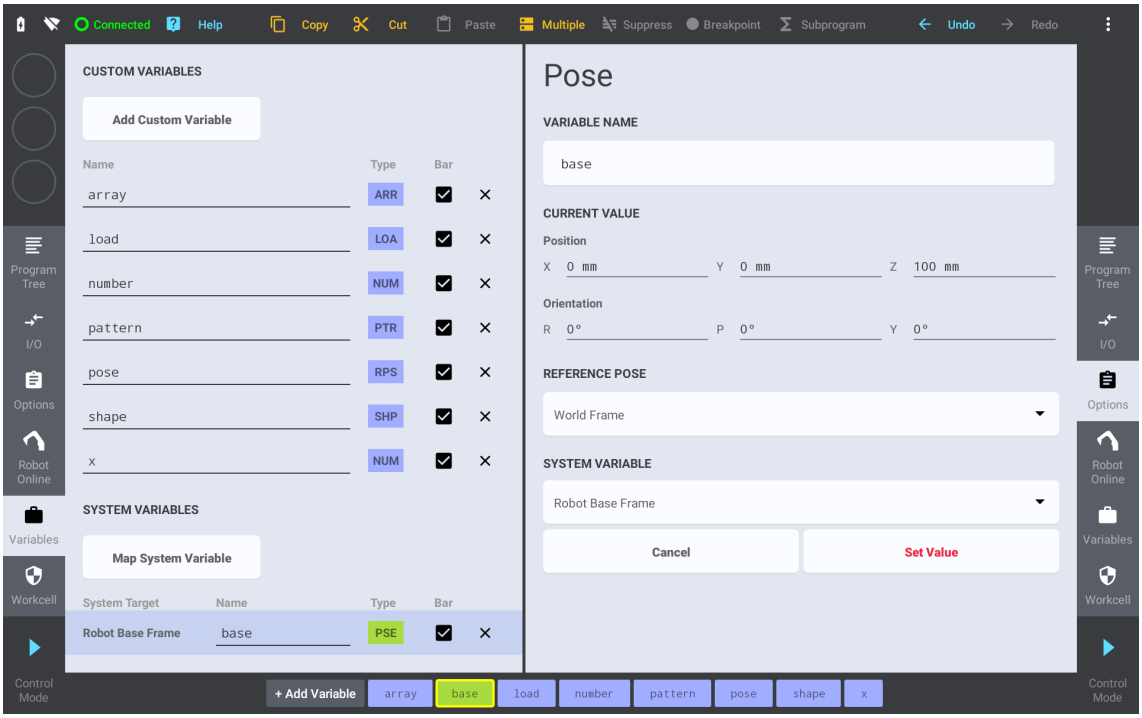
There are 2 ways to create a new system variable:

- To map I/O system variable, open the I/O tool and click the **Map...** button.
- For other system properties, navigate to Variables tool and click the **Map System Variable** button.



In both cases, the Map System Variable dialog appears. To create the variable, select the system property to be mapped, enter a unique name and click the **Create** button.

Once the variable is mapped, you can monitor and, if allowed, modify its value via the options panel. Simply click on the value and make the necessary modifications. To apply the changes, click the **Set Value** button. Additionally, you have the flexibility to change the mapped system property (if the new property shares the same data type) or rename the variable by clicking on its current name and making adjustments.



It is important to note, that system variables are always stored within the program. For the list of all accessible system properties, please see the System Variables [Appendix C, page 141].

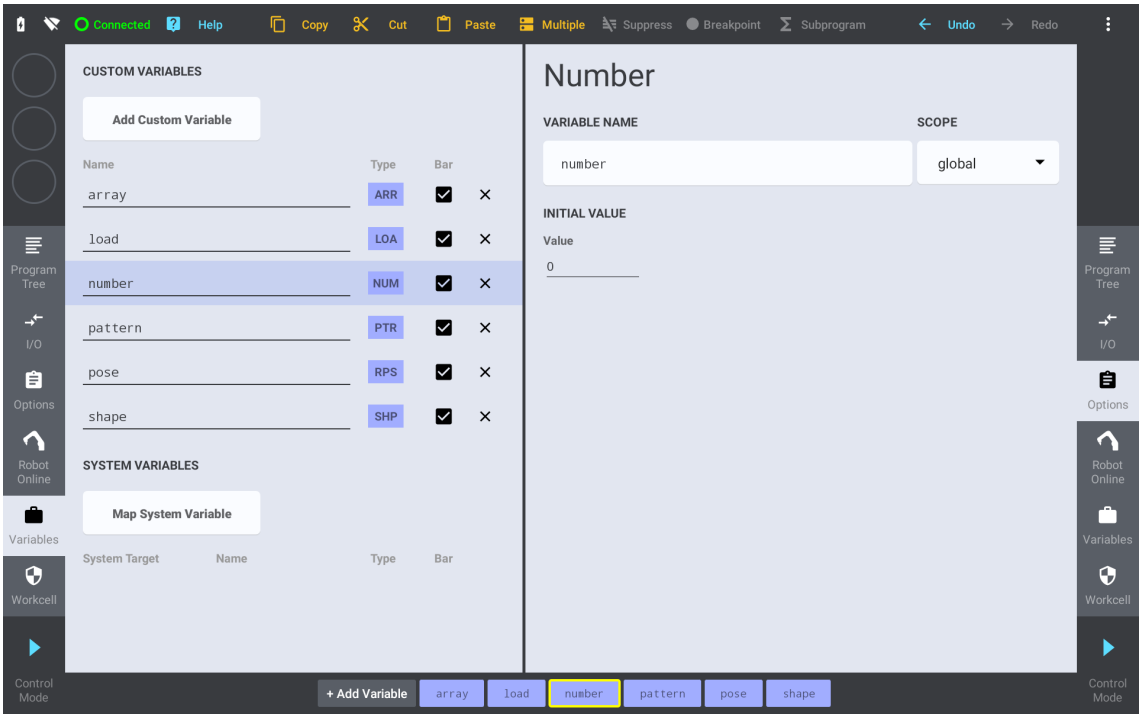
5.3 Data Types

The data type of a variable defines the type of value it can store and the operations that can be performed on those values. For example, arithmetic operations are applicable to the Number data type, while Pose data type supports dot notation for accessing its members, such as position and orientation.

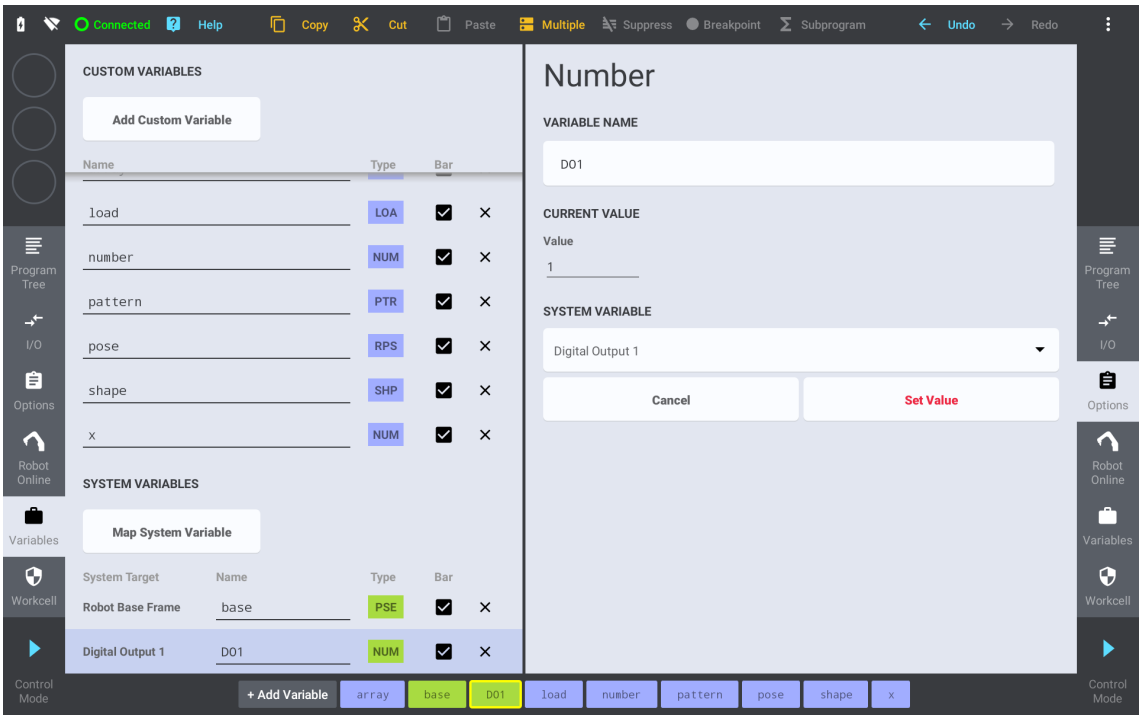
5.3.1 Number

The number data type is essential and versatile, capable of storing both integers (whole numbers without decimal points) and floats (real numbers with decimal points, providing greater precision than integers). This data type supports various arithmetic operations, such as addition or multiplication, and can be manipulated by the mathematical functions [Appendix B, page 136].

In addition, the number data type can be used in expressions for decision-making, where the zero value is interpreted as false, while any non-zero value is considered true.

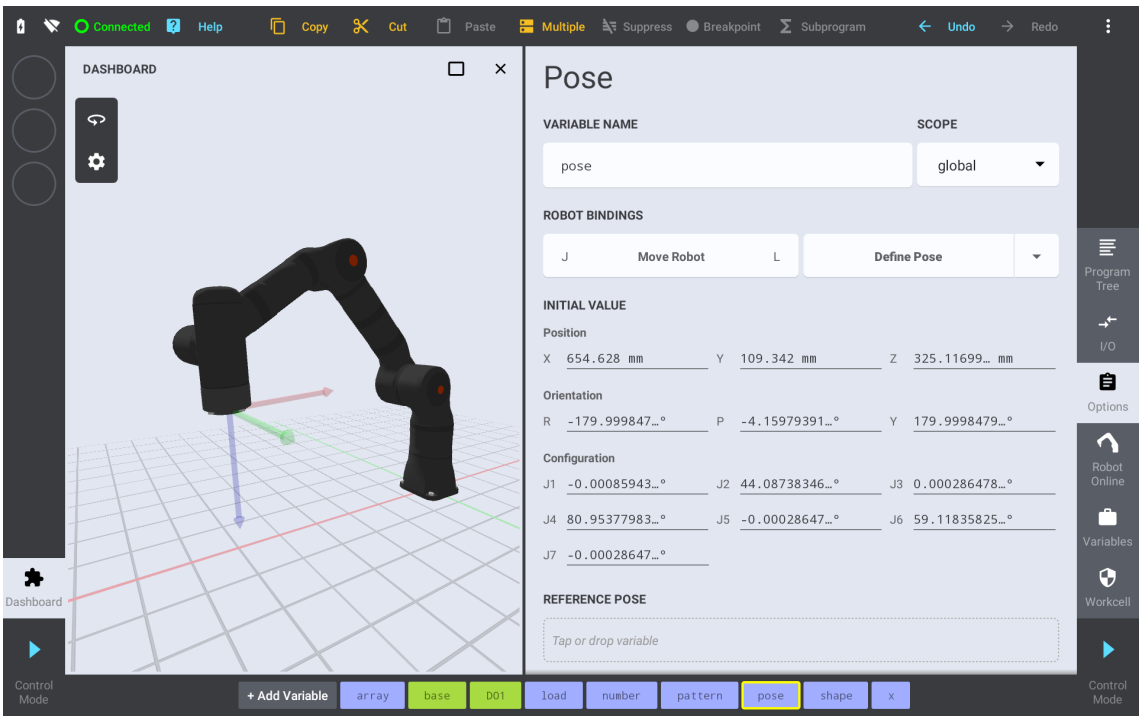


Furthermore, the number data type is used for mapping I/O ports into system variables. For instance, a zero value represents an inactive digital input/output, while a value of 1 indicates an active digital port. Both the user and the program can read the actual port value, and for output ports, set the requested state.



5.3.2 Pose

The pose data type typically represents the pose of the robot arm, fully defined by the joints configuration (J1 to J7) and TCP position (X, Y, Z) and orientation (roll, pitch, yaw). Additionally, the pose variable can be used for representation and manipulation of general frames, such as the frame of a pallet. For this purpose, the pose data type is equipped with an optional reference pose, allowing the creation of a chain of frames.



The pose variable options panel provides quick access to the pose coordinates. To verify the coordinates, click and hold the **Move Robot** button, jogging the robot to the specified joints configuration (**J**) or target position and orientation (**L**). The robot keeps moving as long as the button is held down and stops immediately as soon as the button is released.

The user can manually modify the pose coordinates by clicking on the respective coordinate and making adjustments. Additionally, it is also possible to redefine the pose variable using one of the built-in or Cbun Define Pose tools. To do this, simply click the **Define Pose** button to open the default Define Pose from TCP tool or select the desired tool from the drop-down menu (see Nitro Tool in [Appendix D – List of Extensions, page 144]).

To define a reference pose, either drop another pose variable into the Reference Pose drop box or click this box to create a new custom pose variable.

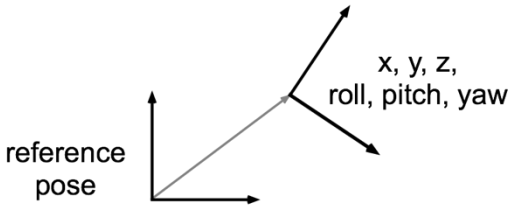
Pose Consistency

A robot pose represented by a single pose variable can be achieved through robot movement in both Jointspace and Workspace. It is important to note, that this applies only if the TCP and joint coordinates are aligned. The TPUI allows for the generation of consistent pose variables via the **Define Pose from TCP** tool or by using the teaching feature, ensuring that the coordinates are properly aligned. Any additional

modifications to the pose coordinates may lead to inconsistent robot pose.

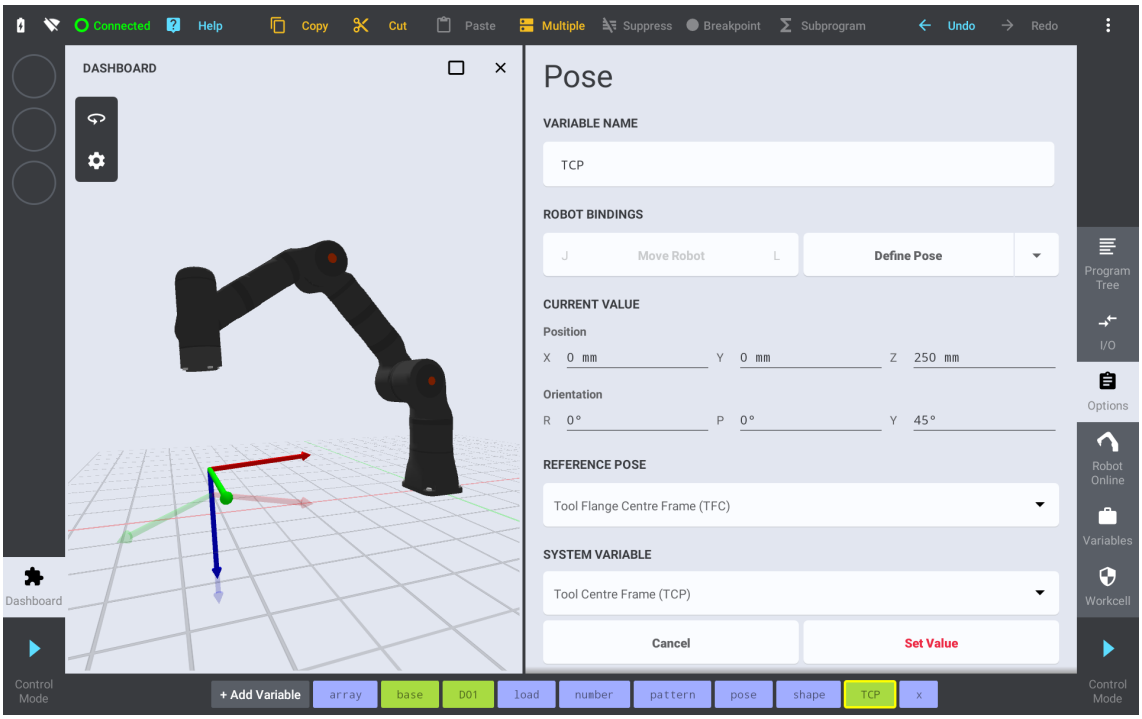
Reference Pose

By default, when the reference pose is empty, the position and orientation express the frame of the pose in the global world frame. Setting the reference pose to some other pose variable allows the expression of pose coordinates relative to this reference frame, creating a chain of frames.



System Pose

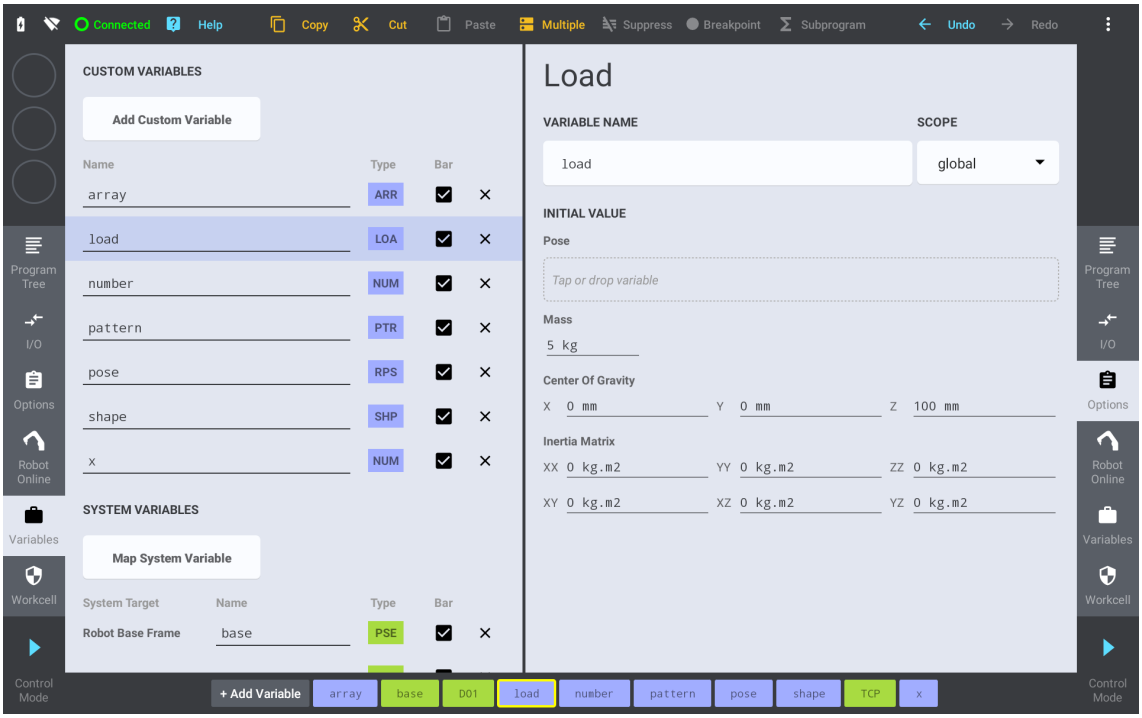
Furthermore, the pose data type is used for mapping robot-related frames into system variables. For instance, the current tool flange centre (TFC) frame can be mapped into a system pose variable. Both the user and the program can read the actual TFC coordinates. Additionally, for certain frames, such as Base or TCP frame, the system pose variable can be used for modifying the frame coordinates.



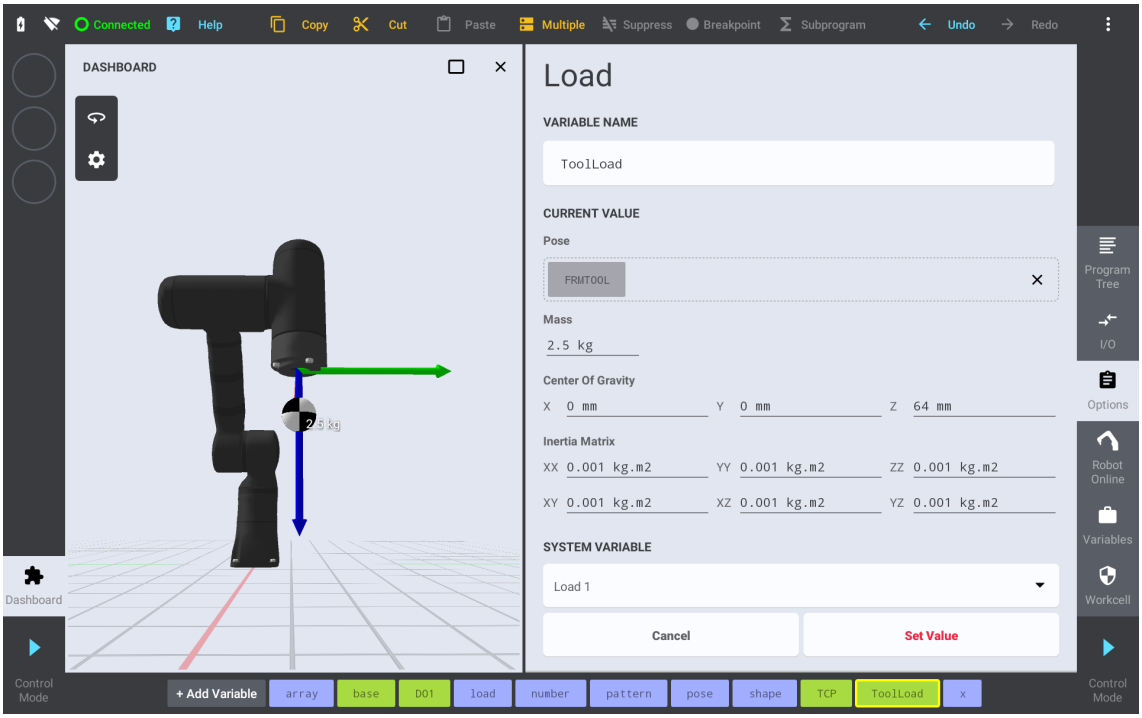
5.3.3 Load

The load data type represents an object that is physically attached to a robot, such as gripper (tool load), or manipulated by the robot (payload). This object is approximated as a rigid body, and therefore, the load data type is defined by its mass, centre of gravity (X, Y, Z) and inertia matrix (XX, YY, ZZ, XY, XZ, YZ).

The load variable options panel provides quick access to the load properties. To modify the mass, centre of gravity or inertia matrix, simply click on its property and make necessary adjustments.



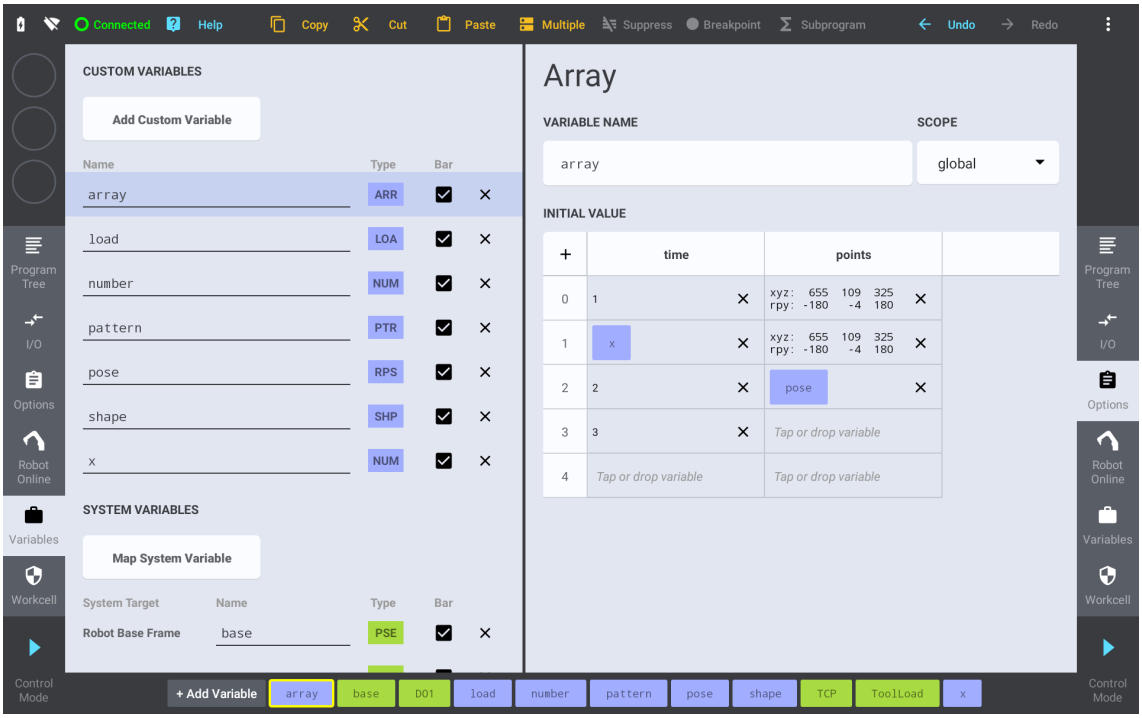
Furthermore, the number data type is used for mapping robot-related loads into system variables. For instance, the tool load and payload can be mapped into a system load variable. Both the user and the program can access and configure the actual load properties.



5.3.4 Array

The array data allows to store multiple values of various data types in the form of a 2D matrix. An array is organized into columns, where each column contains values of a specified column data type. Individual rows, columns or cells can be accessed from a robot program via subscript operator (array indexing).

An array improves the robot programming efficiency, as it allows to store multiple Poses, Loads, Numbers or Grid Patterns in a single array variable without the need to create many different variables of primitive data types.



To add a new column at the end of your array, open the array's options panel and click the **+** button. The Create New Colum dialog appears allowing you to select the column data type and enter the column name.

Click an empty cell or drop a corresponding variable into it to define a new cell value. To modify an existing cell value, either click the cell content and make necessary adjustments or drop a corresponding variable into the cell. Clicking the **X** button clears the content of the cell.

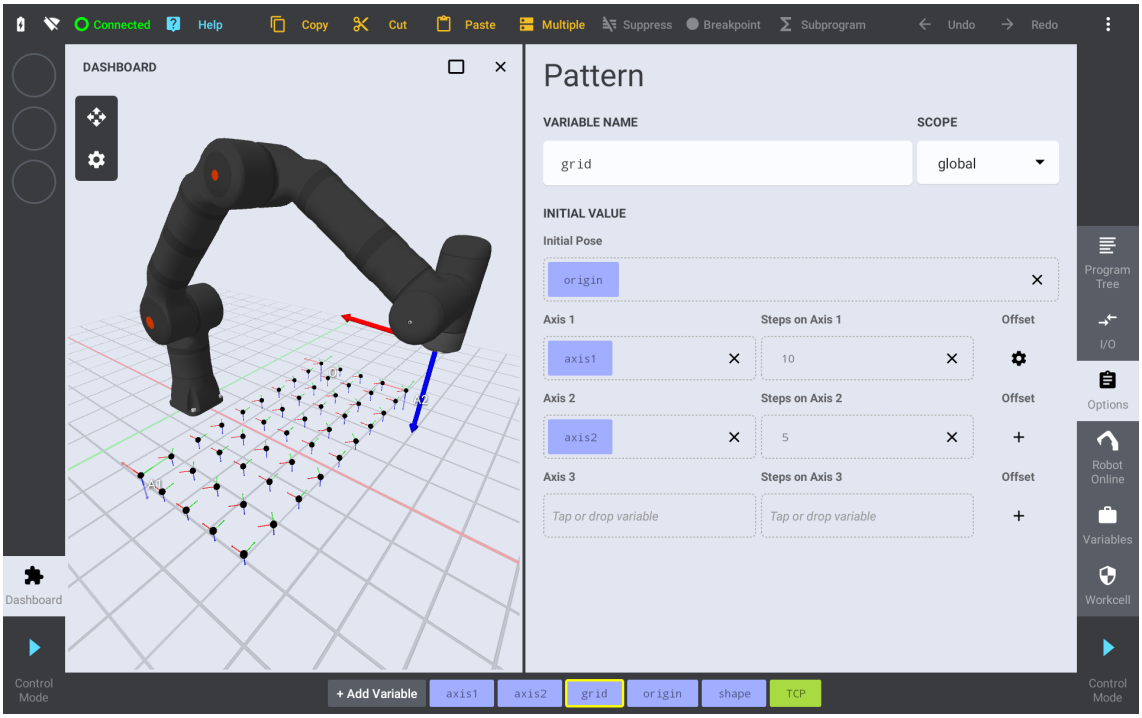
To remove a column, select the column by clicking its header and click the **Cut** button. The same procedure applies to the removal of a row, except for clicking the row header to select the row.

Array Capacity

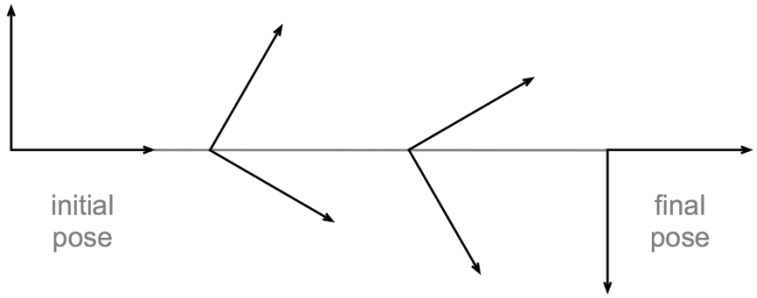
The maximum number of rows in a custom array depends on the scope of the array variable. While persistent and global arrays allow users to define up to 100 rows, the local array can consist of up to 1000 rows. The number of rows in system array is fixed and depends on the mapped system property. For instance, properties related to joints typically consist of 7 rows.

5.3.5 Grid Pattern

The grid pattern data type allows the construction of a 1D, 2D or 3D grid by using just few parameters. Each grid is defined by initial pose (origin) and up to 3 axes, consisting of a final pose and a number of steps on the axis. This plays a vital role in robot applications with regular robot pose distribution, such as palletizing.

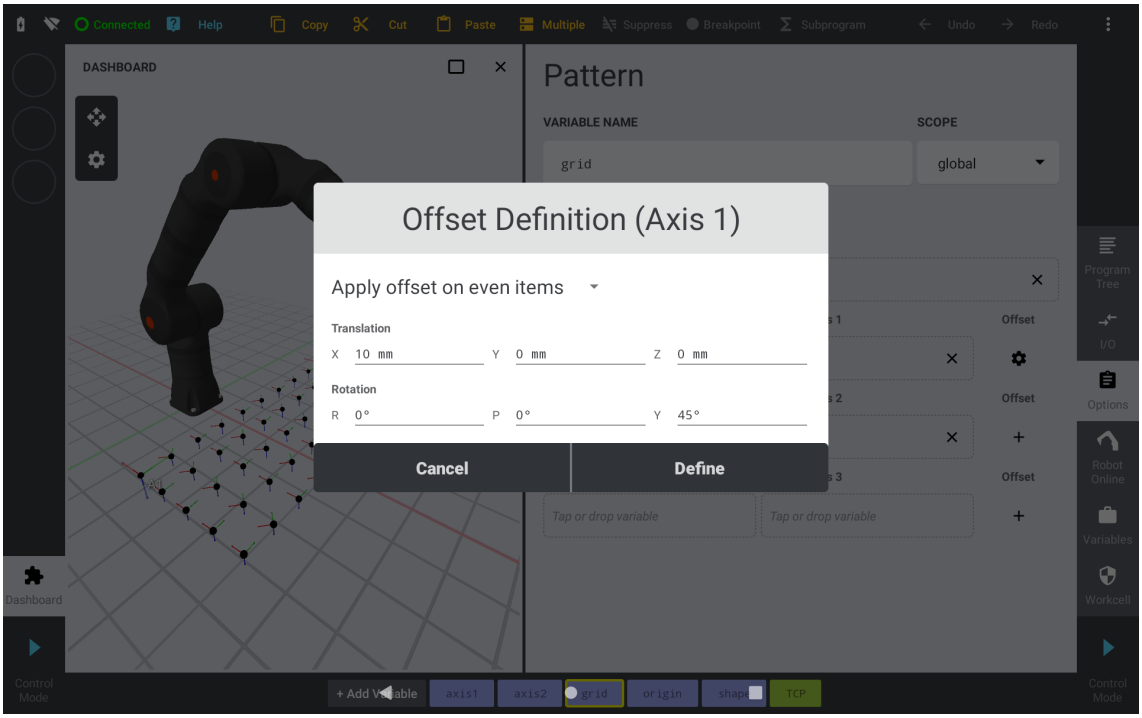


The grid pattern is used to generate evenly distributed poses between an initial pose (origin) and a final pose of the specified axes. This interpolation involves the position (x, y, z) and orientation (roll, pitch, yaw) of the grid pattern poses. It is important to note that the output poses from the grid pattern do not contain valid joint configuration and, therefore, should not be directly used for robot movement in Jointspace.



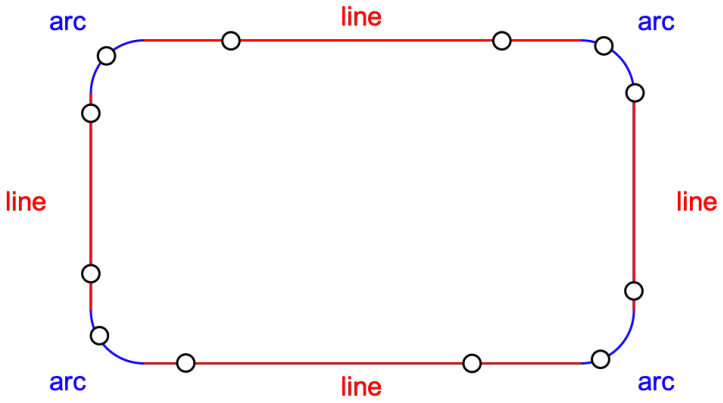
To create a new grid pattern, open its options panel and start by defining the origin. You can do this by either dropping a pose variable into the Initial Pose drop box or clicking the box to create a new pose variable using current TCP's coordinates. Proceed by adding up to 3 grid pattern axes. To define an axis, fill in the final pose and number of steps on the axis. Both parameters support drag-and-drop of corresponding variables, and you can also click on the box to create a new variable.

Furthermore, clicking the **+** button opens the Offset Definition dialog, allowing you to specify the translation and rotation of all, even or odd items on the axis. To remove an axis offset, open the Offset Definition dialog of the corresponding axis, and select **Do not apply offset**. To remove the whole axis definition, simply clear both the axis final pose and the number of steps.



5.3.6 Shape

The Shape data type empowers users to define complex robot trajectories by guiding the robot along a desired path and teaching pose markers. These markers are structured into a sequence that alternates between line and arc segments, contributing to the overall shape geometry. Beyond defining the geometry, each pose marker may play a crucial role in specifying the motion speed and tool orientation (fixed or tangent) along the entire trajectory.

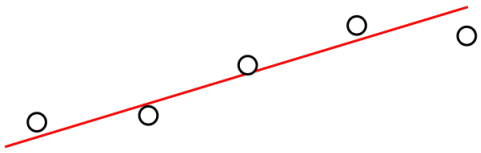


Please note, that due to its complexity, the shape data type can be used only within a custom variable with the local scope.

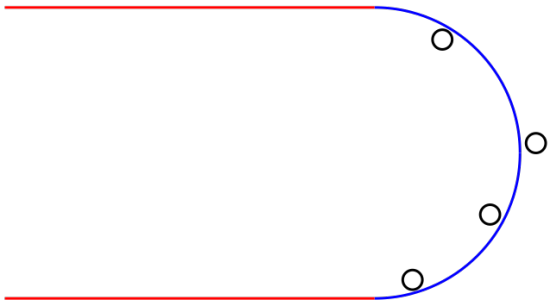
Introduction

The shape can take the form of either an open or closed trajectory. In a closed shape, the trajectory both begins and concludes at the same point and orientation, creating a seamless loop. On the other hand, an open shape allows for distinct start and end points or orientations. When utilizing an arc segment as the initial segment of a closed shape, the last segment must be a line, and vice versa. In closed shapes, the sequence of segments always starts and ends with a line segment.

Each line segment must comprise a minimum of two pose markers. If more than two markers are provided, an optimization process is initiated to identify the most suitable fit for the line trajectory. It is important to note, that markers must be defined in the correct order.



An arc segment must contain at least one marker, determining curvature of the corner. If more than one marker is provided, an optimization is used to find the best-fit arc.



The first marker of a shape trajectory has to establish the initial orientation (tangent vs fixed) and speed for the robot movement. Subsequent markers can either maintain the existing configuration or introduce changes in speed and/or orientation, ensuring precise motion control throughout the trajectory.

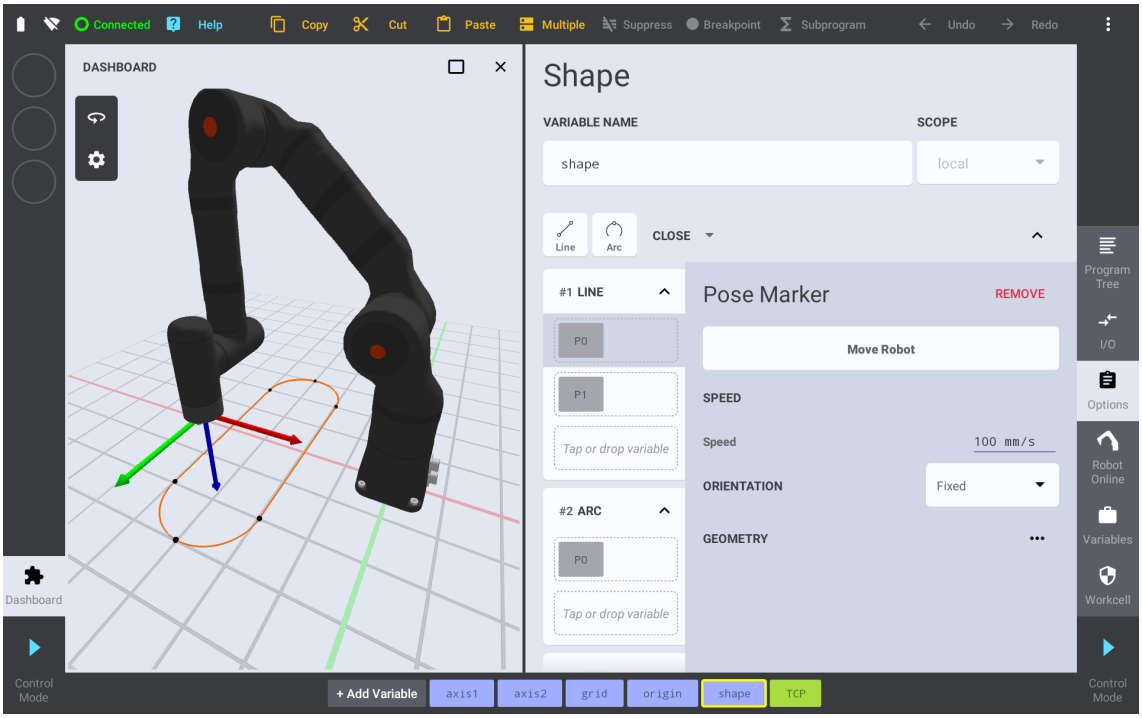
Options Panel

To define a shape, select it and access its options panel. Choose between an Open or Closed shape form and then add the first segment by clicking the **Line** or **Arc** button.

Once the segment is added, guide the robot tool (TCP) to the desired point on the path and use the teaching feature to add a new marker. Alternatively, you can drop a corresponding pose variable into the marker's drop box or click the box to define a new marker using the current TCP coordinates.

To modify the marker's properties, select the marker and make necessary adjustments. In addition to configuring the speed and orientation, the **Geometry** section allows you to redefine the marker's coordinates. You can either click the **Define Pose** button to use the current TCP coordinates or manually adjust individual coordinates. To verify the coordinates, click and hold the **Move Robot** button until the robot reaches the markers pose.

Furthermore, you can exclude the marker from geometry optimization by disabling the **Include in calculation** setting. This allows for finer control over the optimization process by selectively including or excluding specific markers (outliers) from the shape trajectory calculations.



To add another marker to a segment, guide the robot to a new pose along the segment geometry and use the teaching feature, or add the marker manually. Once the segment is fully defined, add another segment at the end of the segment list by clicking the corresponding segment button, and repeat the “Add markers” procedure. If required, the user has the option to add a new segment to the desired position by using drag-and-drop feature.

To remove a marker or segment, simply select it and click the **Remove** button.

Additionally, the shape options panel is equipped with extensive error reporting system. For details see Shape Errors [Appendix E, page 172].

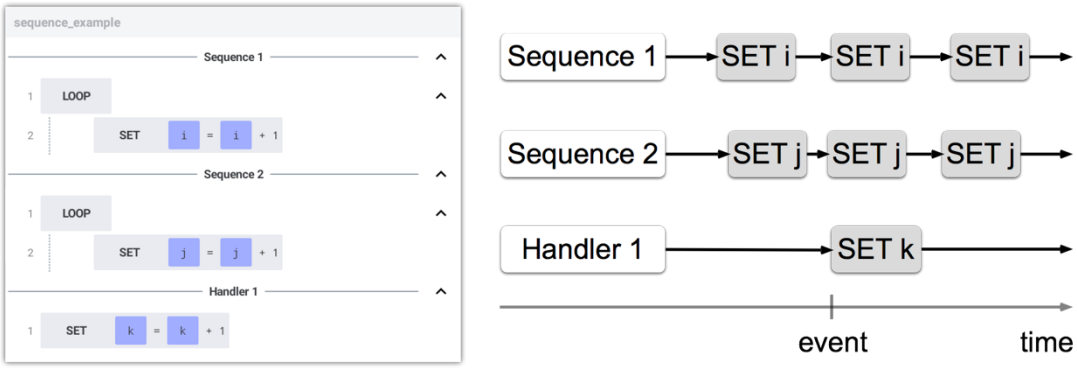
6 Commands

Each robot program consists of one or more sequences of commands that establish the general program structure or provide direct robot movement instructions to carry out specific manipulator tasks. Most commands can be parameterized, allowing for customization of their behavior. Furthermore, incorporating variables as command parameters enables dynamic control of command behavior, thereby enhancing flexibility and adaptability in robot programming.

The robot programmer can construct the program by dragging individual commands from the Command Box and dropping them into the Program Tree tool. Once a command is added to the program tree, its parameters can be configured by selecting the command and accessing the Options tool.

6.1 Sequence

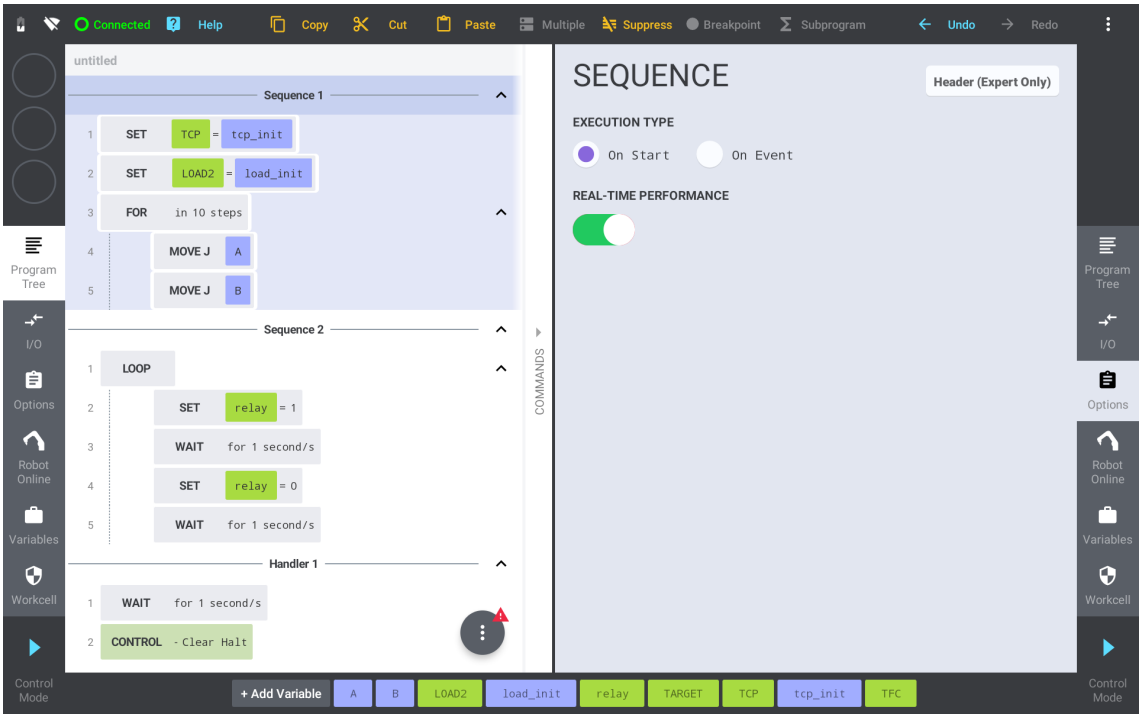
The Sequence functions as the root node command that encapsulates a block of instructions/commands. Each active robot program (cluster) can accommodate up to 10 concurrent sequences. While some sequences are executed at the beginning of the program, others may be activated by system events, such as alarms.



Commands within a sequence are executed sequentially, one after the other. However, individual sequences operate concurrently and asynchronously, functioning independently. Each sequence maintains its register of variables and Cbun device/command instances. To synchronize data across sequences, the robot programmer must explicitly use global or persistent variables.

Application programmers can choose to execute a sequence with real-time performance, crucial for tasks requiring precise timing, like high-frequency robot movement control. However, it incurs additional system resource costs that may not be warranted for low-priority tasks.

Moreover, programmers can specify additional C++ headers to be used by the C++ CALL command, enabling enhanced robot functionality in specific scenarios. This feature is aimed at advanced users and necessitates close collaboration with Kassow Robots' technical support.



6.1.1 Handler

Handler refers to a Sequence command with **On Event** execution type. The execution of such sequence is triggered by the occurrence of a selected system event, allowing the robot to respond to specific situations. This plays a crucial role for handling system alarms, since it allows the robot to notify its environment (equipment, PLC) and potentially resolve the issue.

kr2_signal::Alarm

The kr2_signal::Alarm event is generated when a program halts, typically due to a command exception, activation of E-Stop/P-Stop or other safety conditions. While standard sequences are halted in response to this event, the alarm handling sequence is initiated for execution, allowing the robot to provide necessary actions.

It is important to note, that if the program tree contains at least one alarm handling sequence, the default program halt handler is disabled. This implies that the handler sequence should either resolve the issue and clear the program halt or terminate the program execution entirely.

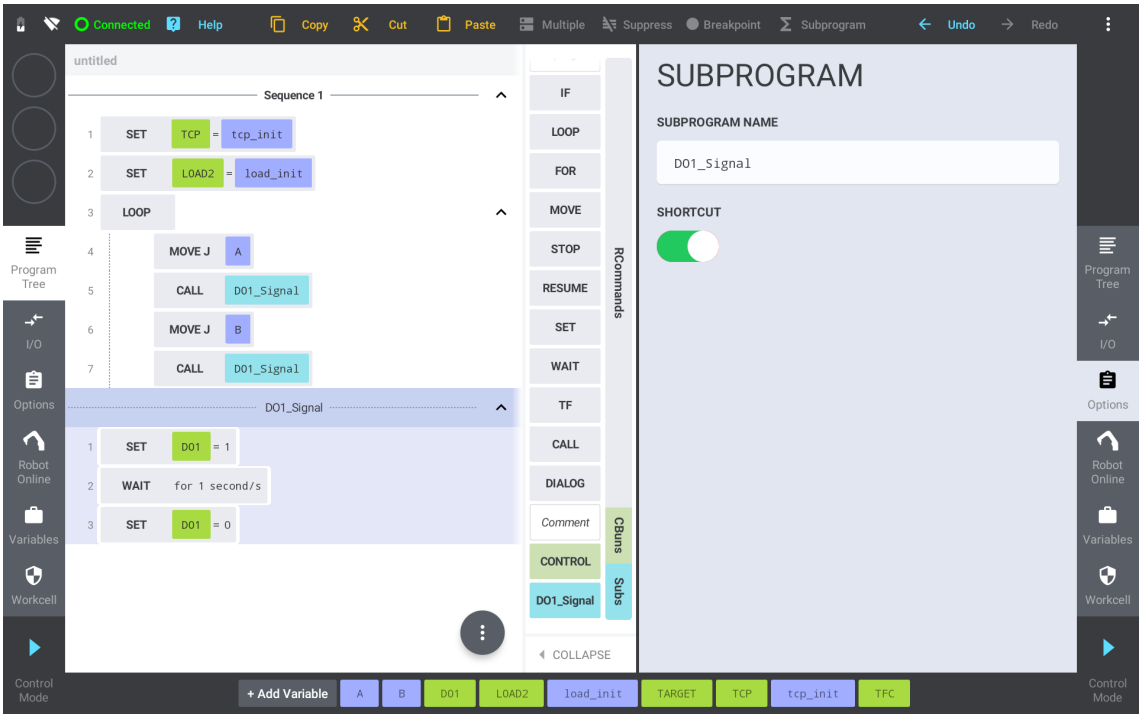
Please refer to the Alarm Handling [Appendix F, page 173] for more information.

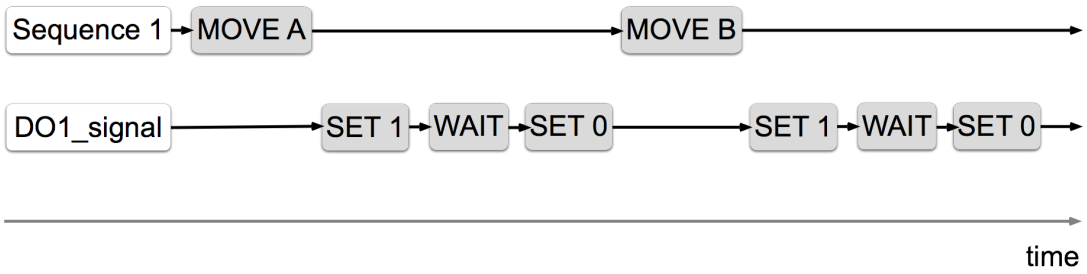
6.2 Subprogram

The Subprogram command is a root self-contained block of commands that performs a specific task or action. Subprograms are designed to be reusable and modular, enabling robot programmers to write code more efficiently and maintainable. Subprograms can be invoked by CALL command from different parts of a robot program, allowing programmers to reuse the same set of commands without duplicating it.

To create a subprogram, you can either drag-and-drop a new subprogram command into the open program tree or select a block of commands and use the **Subprogram** button to convert them into a subprogram.

The Subprogram options panel allows users to customize the subprogram label and activate the shortcut feature. When the shortcut is enabled, the subprogram invocation command becomes accessible through the Command Box.





6.3 IF

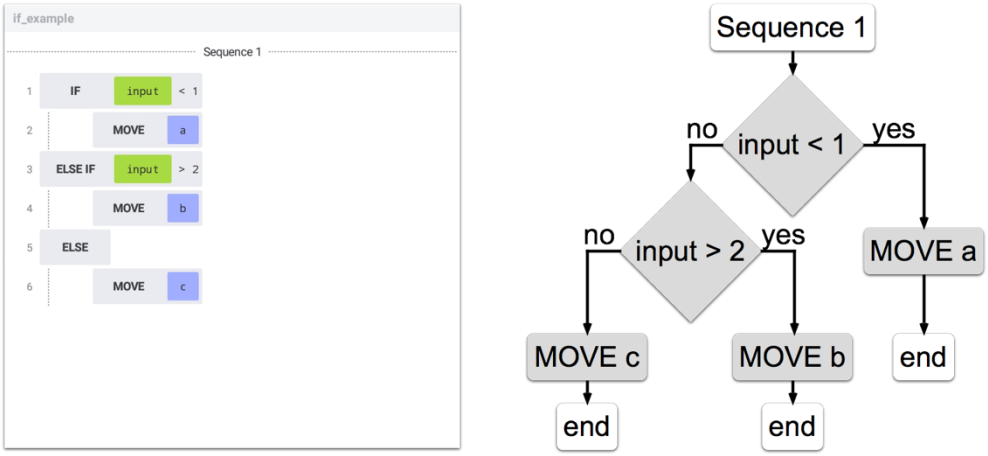
The IF command is a fundamental control flow structure that allows for conditional execution of a command block. It is used to make decision within a program, directing the flow of execution based on whether a given condition is true or false. There are three possible configurations of the IF command:

1. **IF:** The basic IF command is used to check a condition. If the condition is true, commands within the IF block are executed.

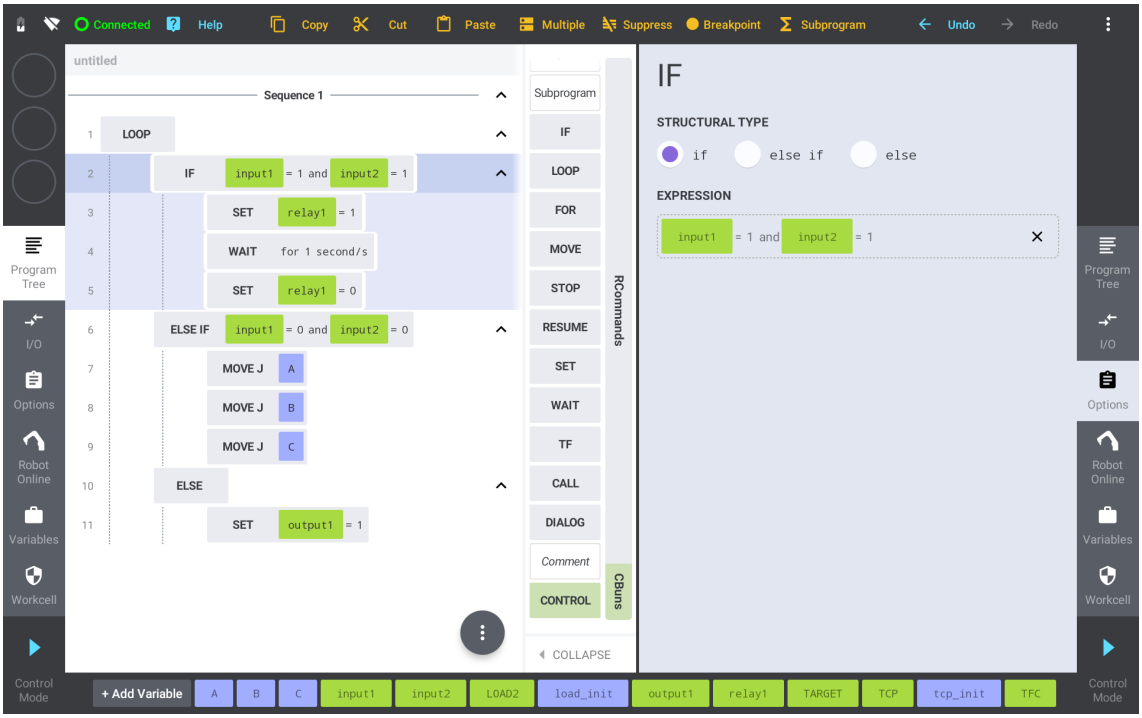
2. **ELSE IF:** The ELSE IF command is used to check an additional condition if the previous IF or ELSE

IF conditions are false. It allows for evaluation of multiple conditions in a sequential manner. If the condition associated with the ELSE IF is true, commands within that block are executed.

3. **ELSE:** The ELSE command is used to specify the code that should be executed if none of the previous conditions (in IF or ELSE IF) are true. It serves as a catch-all for cases where none of the preceding conditions are met.

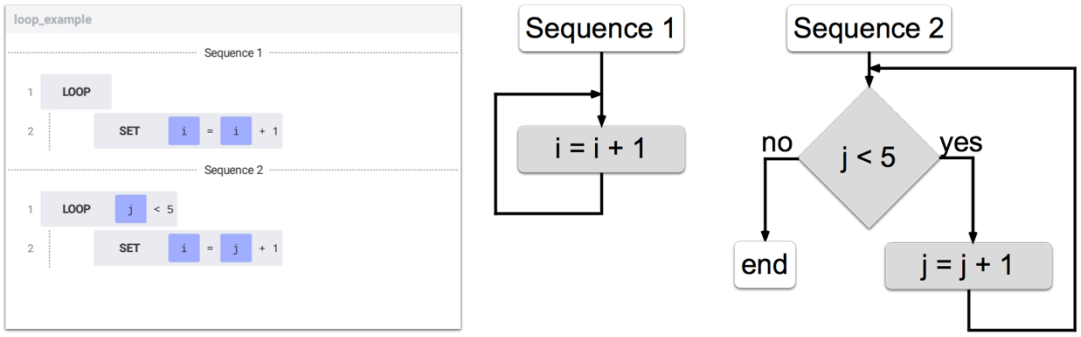


To define conditions for the IF and ELSE IF commands, simply click the expression box. This action opens the Expression Builder, allowing you to construct the condition using a wide range of operators, operands, and functions. Keep in mind, that the result data type of the expression must be a Number. The condition is met if the result of the expression is non-zero value.

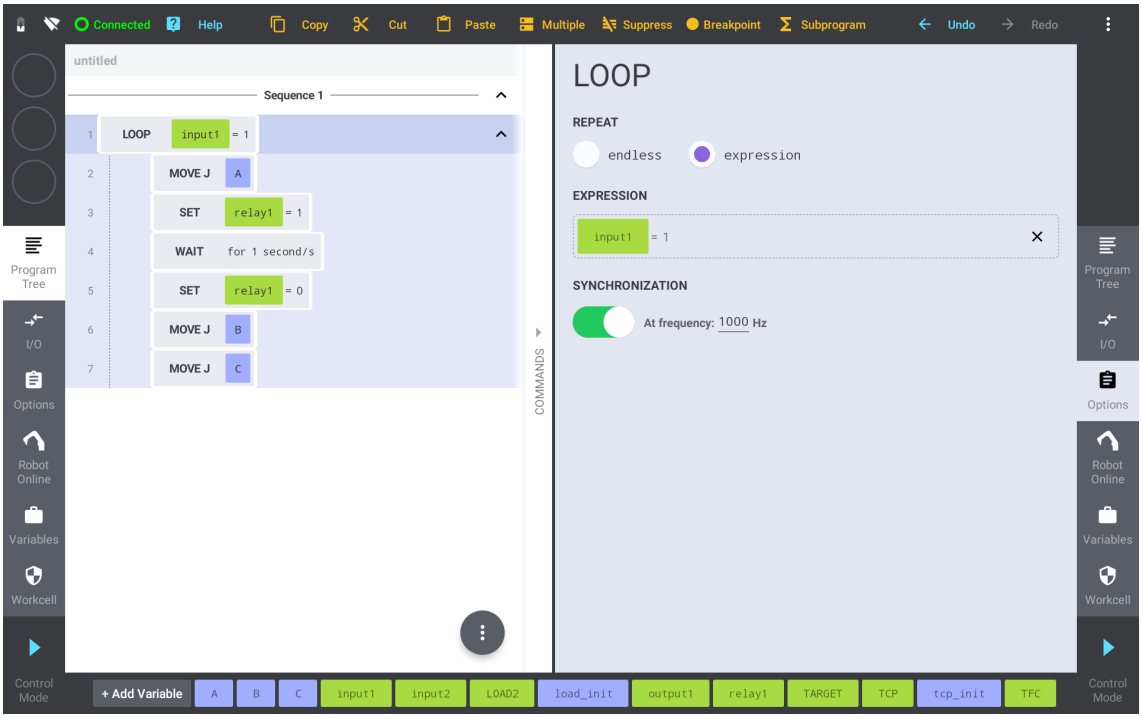


6.4 LOOP

The LOOP command is used to repeatedly execute a block of commands. The loop can operate either indefinitely or while a specific condition is satisfied. Additionally, the robot programmer can activate loop synchronization which limits the maximum frequency of loop execution.



The LOOP options panel allows to set endless or expression based (conditional) loop execution mode. The **Expression** box is enabled only for the conditional mode. To provide condition entry, simply click the box. This action opens the Expression Builder, allowing you to construct the condition using a wide range of operators, operands, and functions. Keep in mind, that the result data type of the expression must be a Number. The LOOP condition is met if the result of the expression is non-zero value.

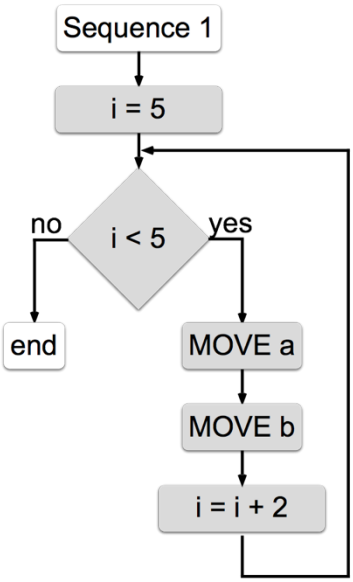


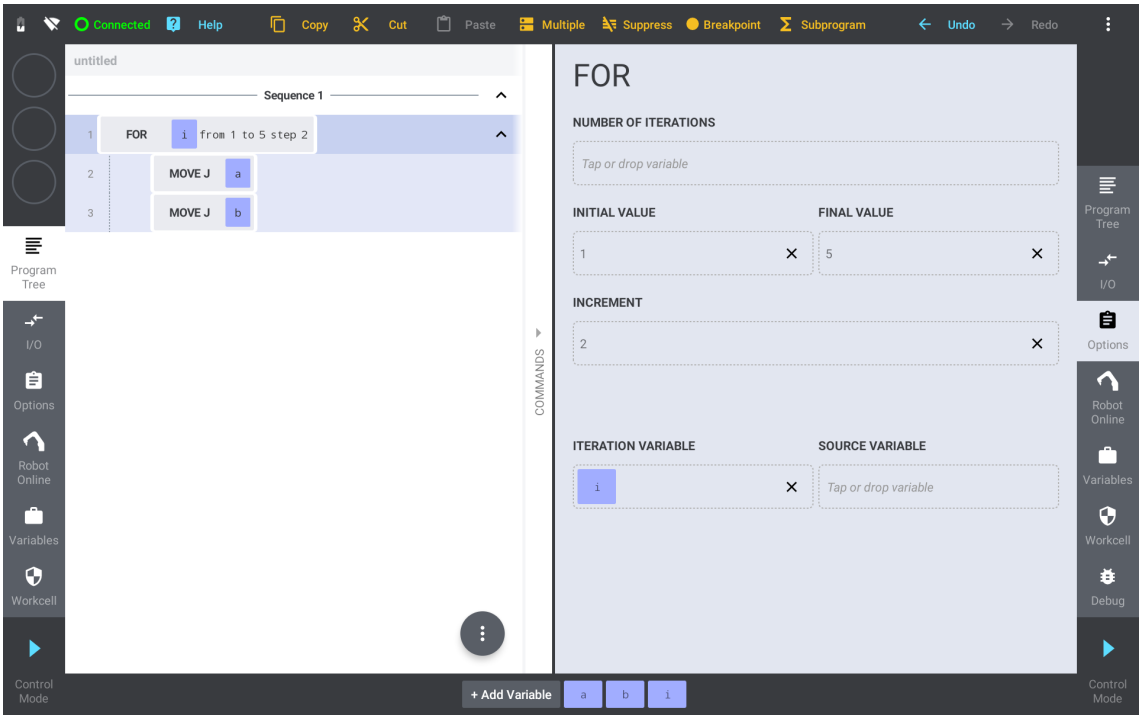
Furthermore, the user can enable and configure loop synchronization through the command's options panel. When the synchronization is disabled, the loop is executed at maximum possible frequency. This

can have a negative impact on performance and should be used with caution, by experienced users.

6.5 FOR

The FOR command is used to create a loop that iterates a specific number of times or over a range of values. The FOR loop body contains the set of commands to be executed during each iteration.





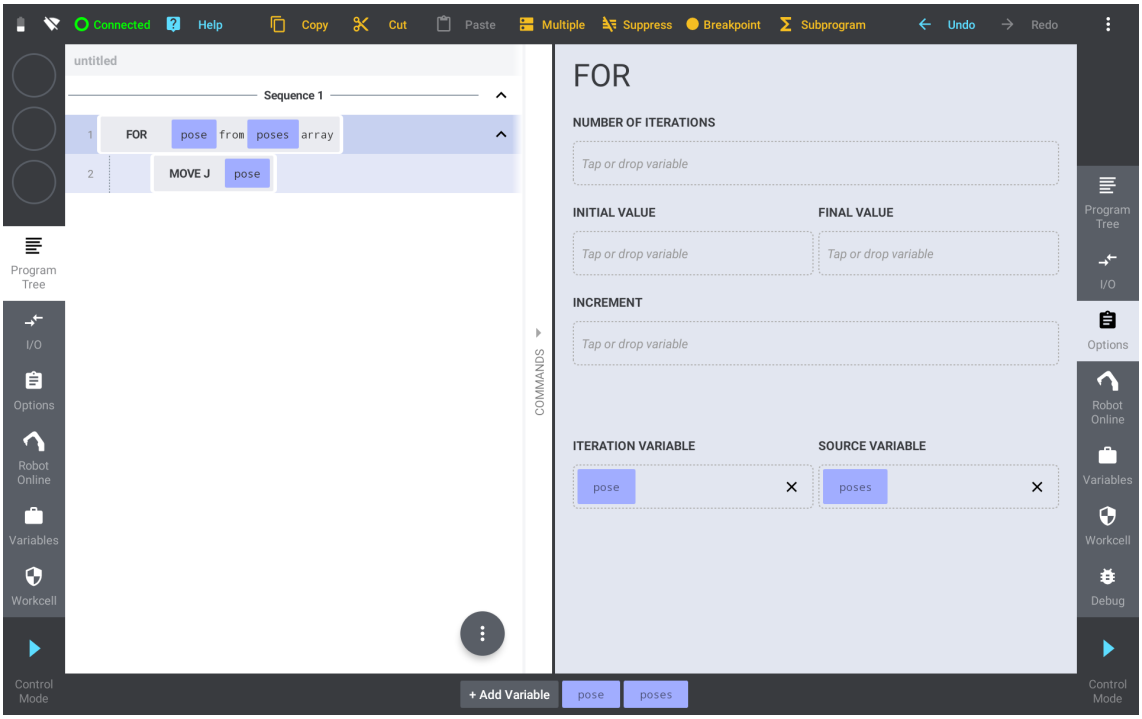
To execute a block of nested commands a specific number of times, define the **Number of Iterations** parameter. You can either click the box and enter the value or drop a Number variable into this box.

Sometimes, it is beneficial to keep track of the FOR loop iteration value. The FOR command includes an optional **Iteration Variable** parameter for this purpose. The iteration variable is updated at the beginning of each iteration and, by default, holds the current value of the iteration. This means, that the **Initial Value** is assigned to the variable prior the first iteration and the variable is automatically incremented/decremented by the **Increment** at the end of the iteration. If not specified, the default initial value is zero and the default increment is one. To define the **Iteration Variable**, drop corresponding custom local variable into the box or click the box to create a new variable.

Additionally, the FOR command allows to customize its behaviour by specifying a **Final Value** for the iteration index. For instance, to iterate from 3 to 5, you can set the **Initial Value** to 3 and the **Final Value** to 5. The same result can be achieved by setting the **Number of Iterations** to 3 and configuring either the **Initial Value** as 3 or the **Final Value** as 5. It is important to note, that setting all three parameters would make the FOR command overdetermined.

The FOR loop can automatically iterate through the **Source Variable** array or pattern. If the **Iteration Variable** is provided, it is automatically set to the next value within the array or pattern on each iteration. To define a **Source** Variable, drop a 1D array or pattern to the corresponding box.

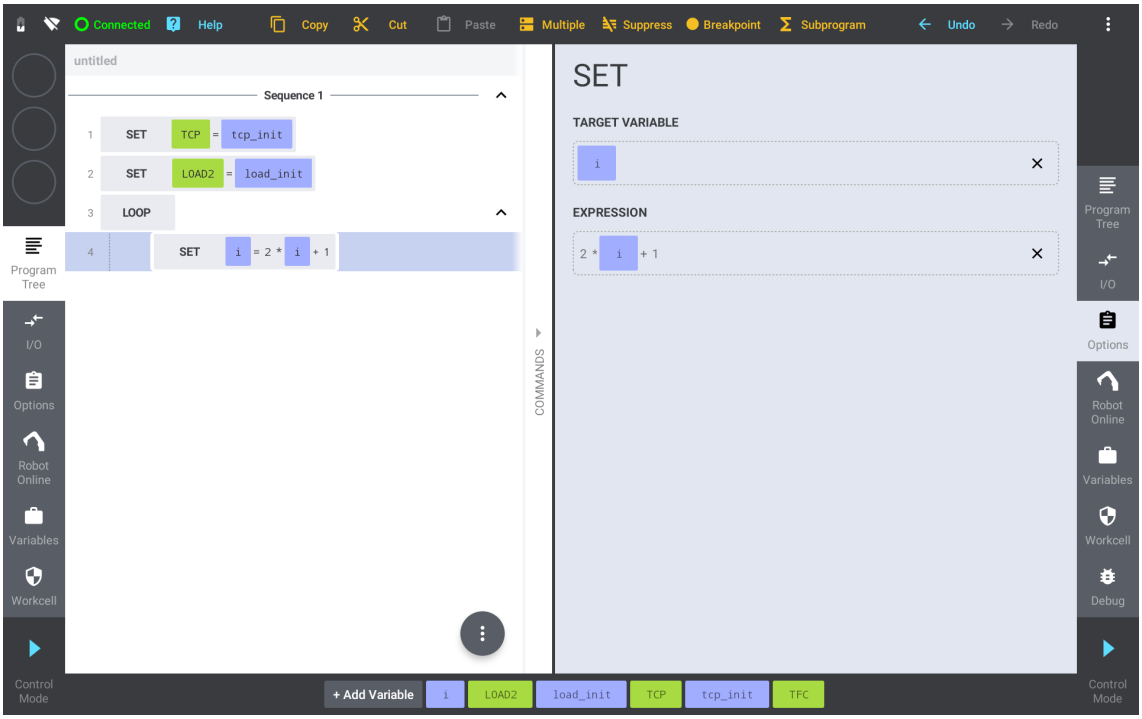
In addition, the robot programmer can combine the **Source Variable** with previous set of parameters, allowing to execute the iteration with some custom initial/final index or increment.



It is essential to align the data type of the iteration variable with the data type of the **Source Variable** items. For instance, if the **Source Variable** comprises array of poses, the data type of the iteration variable must have a data type of Pose. If the **Source Variable** is not specified, the iteration variable is used for loop indexing and therefore must have a data type of Number.

6.6 SET

The SET command enables users to assign the output of an expression evaluation to a target variable, allowing users to update the variable values and their members, as well as perform complex calculations. Furthermore, the SET command extends its functionality to the control of robot I/Os and configuration of other parameter's, such as frames and loads, by assigning new values into the corresponding system variables.



To define the **Target Variable**, simply drop a desired variable into the corresponding box. Clicking the box opens the Expression Builder, allowing you to define the variable in form of expression. Furthermore, the Expression builder enables users to utilise dot notation for accessing members of the variable.

The **Expression** box empowers users to combine one or more values, variables, operators, and functions that will be evaluated to produce an output value during the SET command evaluation. Drop a desired variable into the **Expression** box or click the box to enter the expression via Expression Builder. Please refer to the Appending – Expressions for further details.

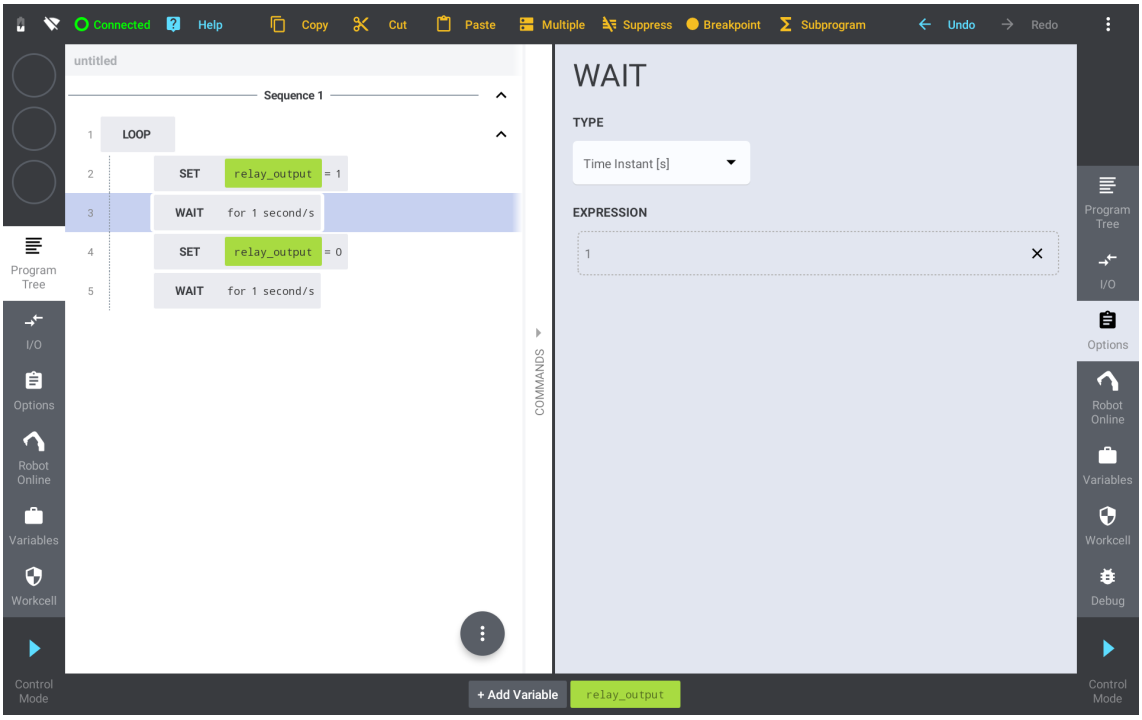
It is essential to align the data type of the target variable, or its member, with the data type of the evaluated output value. For instance, if a Number is used as the target variable, the output value of the expression must be Number as well.

6.7 WAIT

The WAIT command enables users to pause the program execution either for a specific duration or until/while a specified condition is met, allowing to synchronize the program execution with some internal or external signals.

6.7.1 Time Instant

The pause the program for a specific number of seconds, open the WAIT command's options panel and set the **Type** to **Time Instant**. Clicking the **Expression** box opens the Expression builder, allowing you to define the wait duration by using one or more constants, variables, operators, and functions. Additionally, you can also drop any Number variable into the **Expression** box.

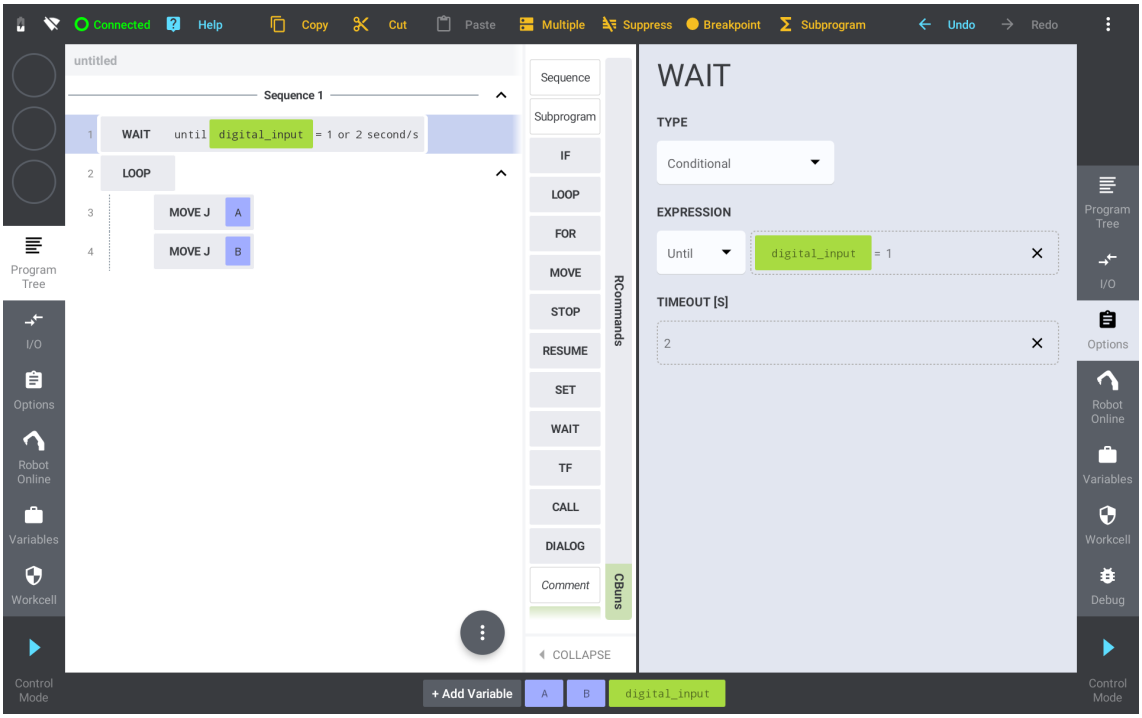


6.7.2 Conditional

The **Conditional Type** of WAIT command pauses the program execution **Until** or **While** the specified condition is met. The condition is met, if the output value of the expression evaluation is a non-zero value.

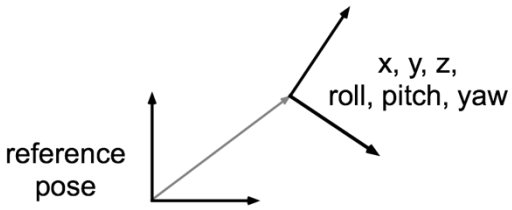
Clicking the **Expression** box opens the Expression builder, allowing you to define the wait condition by using one or more constants, variables, operators, and functions. Additionally, you can also drop any Number variable into the **Expression** box.

Additionally, the robot programmer is enabled to define an optional timeout duration. If the condition is not met within the specified period of time, the WAIT command is interrupted and the program execution continues.



6.8 TF (Transform)

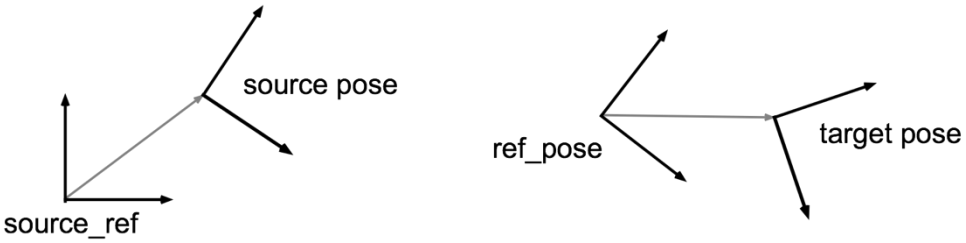
The TF (transform) command allows to perform various transformations of a Pose variable, affecting the position (X, Y, Z), orientation (roll, pitch, yaw) and reference of the Pose. This feature plays a vital role particularly in the context of frames and the intricate structures of frame chains.



The TF command offers 3 types of Pose transformations. It can either move or convert the pose into a new reference or translate and rotate the Pose.

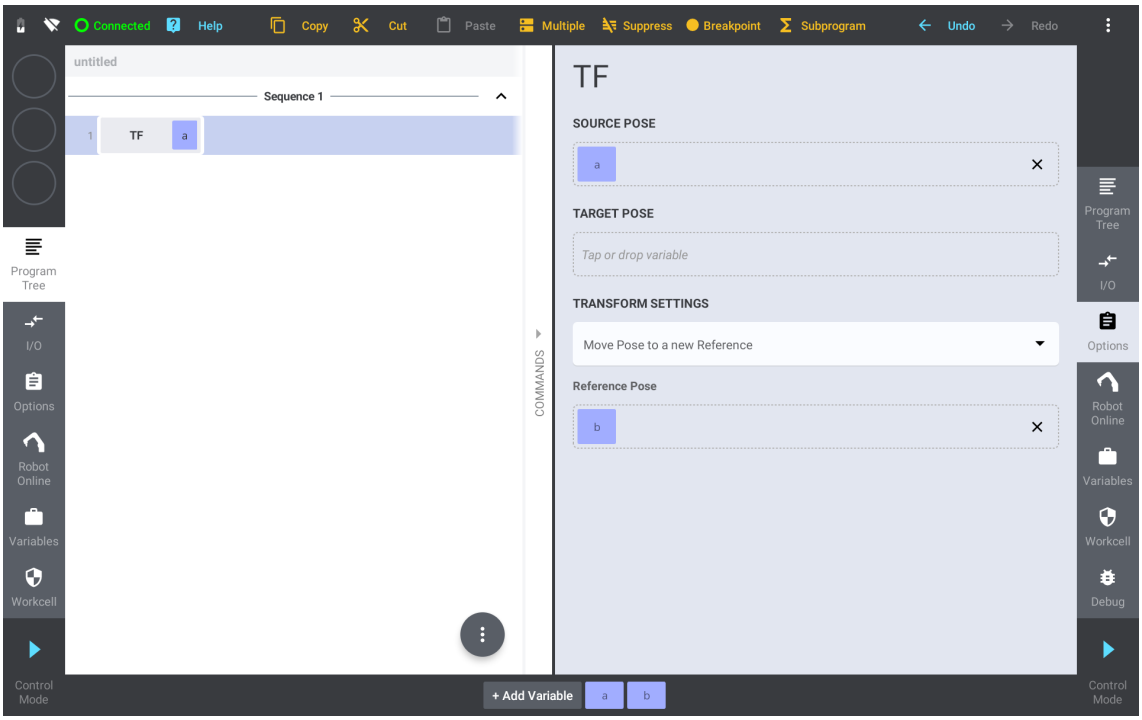
6.8.1 Move Pose to a new Reference

The **Move Pose to a new Reference** transformation empowers robot programmer to change the reference of the given pose while preserving the position (X, Y, Z) and orientation (roll, pitch, yaw) coordinates. This functionality is particularly useful in scenarios where a pose needs to be expressed relative to various frames, such as defining the same item position and orientation across multiple palettes.



To define the transformation, simply drop a pose variable into the **Source Pose** box and specify its new reference by dropping another pose variable into the **Reference** box. Clicking the boxes opens the Create New Variable dialog, allowing you to define new pose variables.

By default, the reference of the source pose is directly updated to the new reference pose, while keeping other source pose coordinates intact. If the **Target Pose** variable is defined, the output of the transformation is assigned into this variable, preserving the value of the source pose.

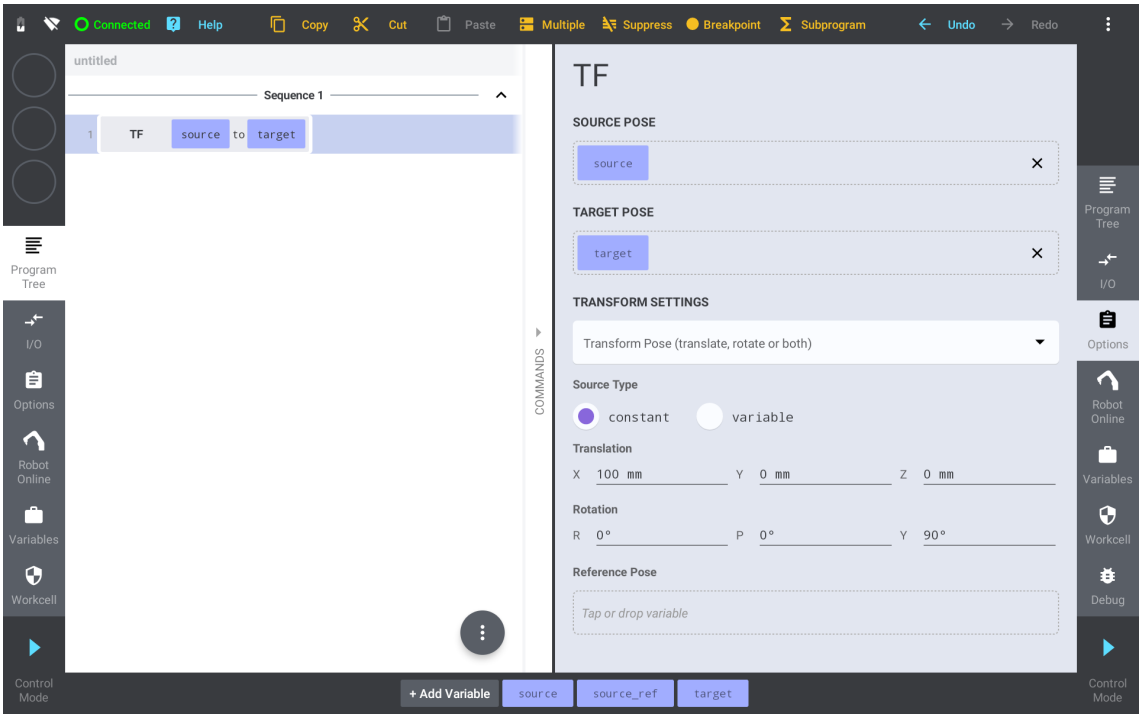


6.8.2 Transform Pose (translated, rotate or both)

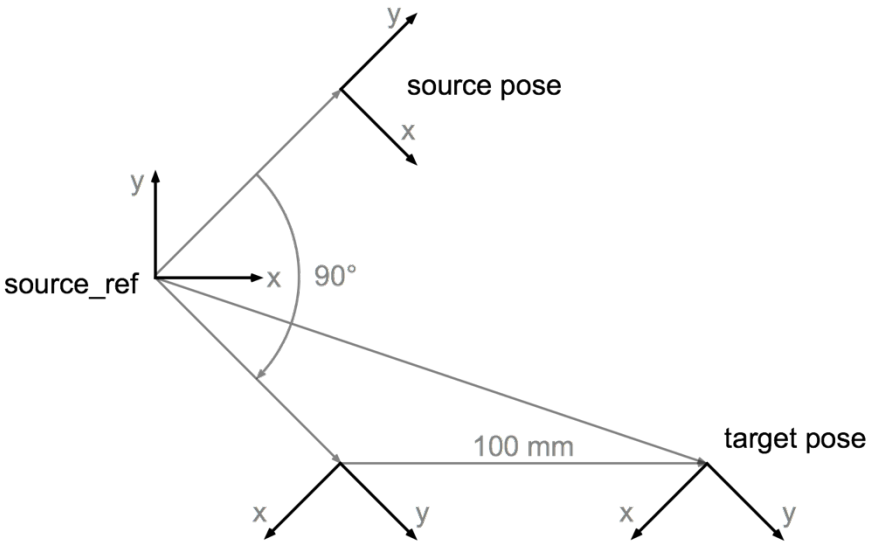
The **Transform Pose** feature enables robot programmers to translate and rotate the source pose within the specified reference frame. This plays a vital role when defining multiple poses relative to each other, since it allows to programmatically transform a single pose variable instead of defining multiple pose variables.

The **Source Pose** variable specifies the coordinates to be transformed. To define the source pose, either drop a pose variable into the **Source Pose** box or click the box to create a new pose variable via the Create New Variable dialog. By default, the TF command updates the value of the source pose.

If the **Target Pose** is provided, the result of the transformation is stored within this variable, keeping the source pose value intact. To define a target pose, simply drop a pose variable into the **Target Pose** box. Clicking the box opens the Create New Variable dialog allowing you to define a new pose variable.

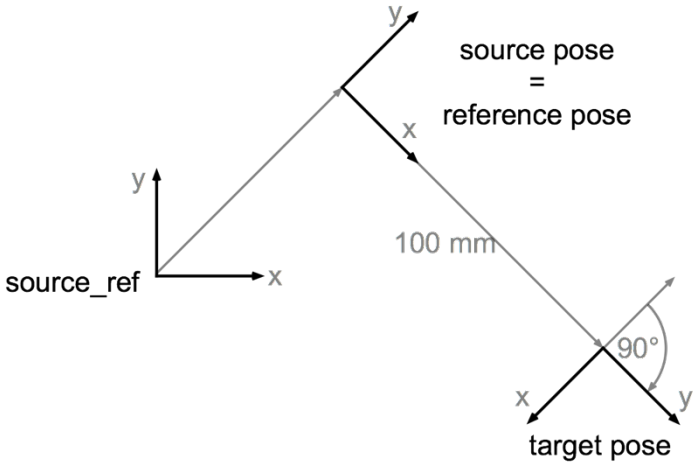


The transformation, involving translation and rotation, can be specified in two ways: either through constant X, Y, Z and roll, pitch, yaw coordinates, or by utilizing a pose variable. For a constant transformation, select the **constant** source type and adjust the **Translation** and **Rotation** coordinates as needed. Alternatively, the transformation can be defined by utilizing the **variable** source type. In this case, the translation and rotation are represented by the X, Y, Z, and roll, pitch, yaw coordinates of the variable.



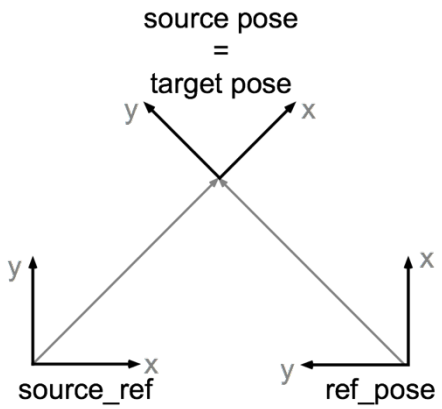
By default, the transformation is performed along the axes of the source pose reference frame. If the source pose reference is not defined, the transformation is applied in world frame.

Additionally, the robot programmer has the option to specify a custom **Reference Pose**. In this case, the translation and rotation are applied along the axes of the provided reference frame. A special case arises when the source pose is used as the reference pose as well. In this situation, the transformation is applied within the frame of the source pose.



6.8.3 Convert Pose to a new Reference

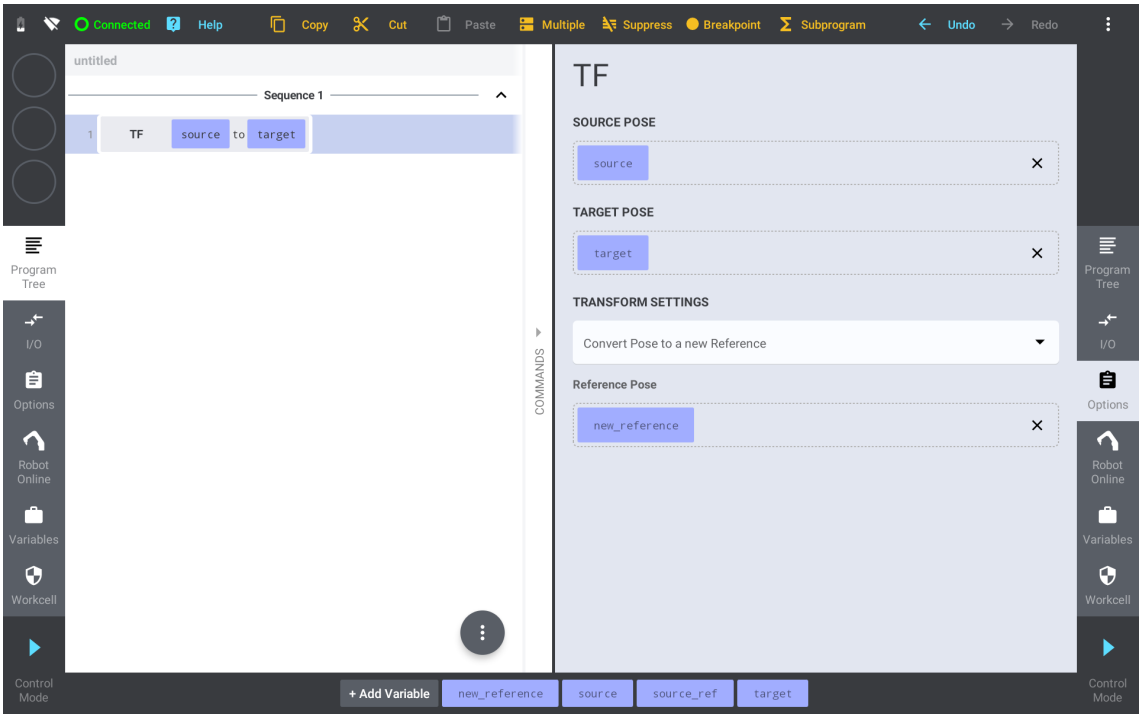
The **Convert to a new Reference** feature allows the robot programmer to exchange the reference frame of a pose while preserving its position and orientation within the world space. This is particularly useful when the same pose needs to be expressed in different frames.



The position (X, Y, Z) and orientation (roll, pitch, yaw) coordinates of the **Source Pose** are automatically recalculated. To define the source pose, either drop a pose variable into the **Source Pose** box or click the box to create a new pose variable via the Create New Variable dialog. By default, the TF command updates the coordinates and reference of the source pose.

If the **Target Pose** is provided, the result of the transformation is stored within this variable, keeping the

source pose value intact. To define a target pose, simply drop a pose variable into the **Target Pose** box. Clicking the box opens the Create New Variable dialog allowing you to define a new pose variable.



To define the **Reference Pose**, either drop a pose variable into the box or click the box to create a new pose variable.

6.9 CALL Command

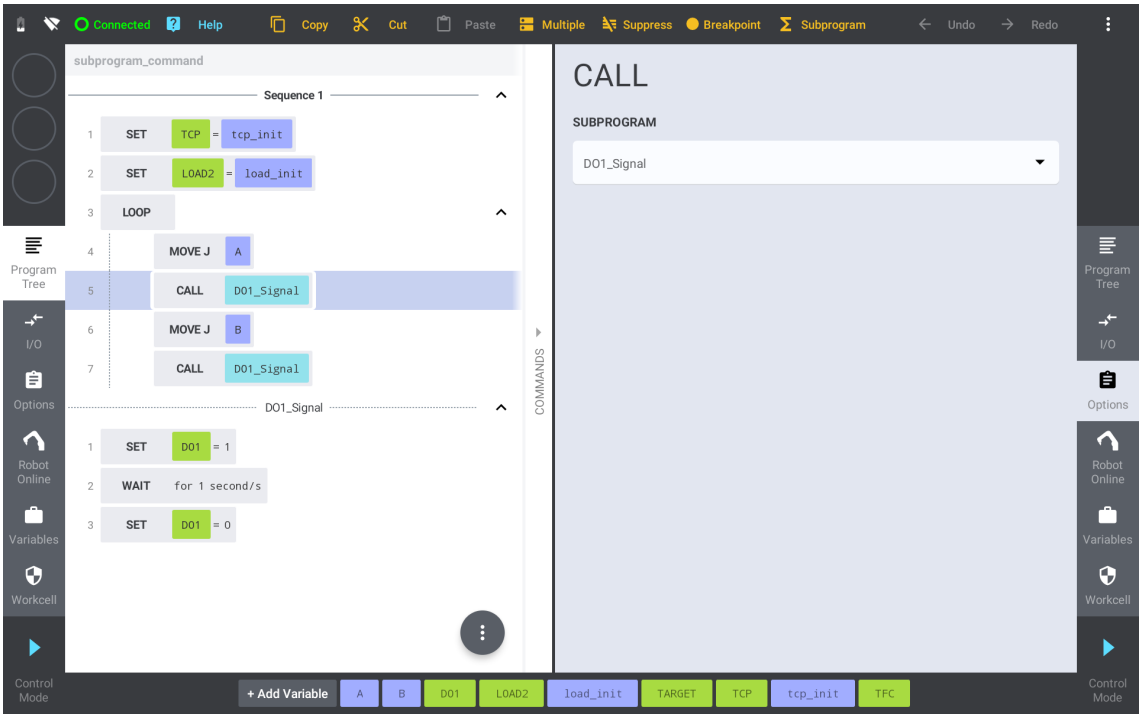
The CALL command enables the robot programmer to invoke the execution of a selected Subprogram. Furthermore, the CALL command serves as an interface for executing custom C/C++ code. This functionality proves valuable when standard program commands are not powerful enough for achieving the requested robot behaviour.

It is important to note that the CALL command operates synchronously, implying that the command remains blocked until the execution of the selected subprogram or custom C/C++ code is fully completed.

6.9.1 Subprogram Invocation

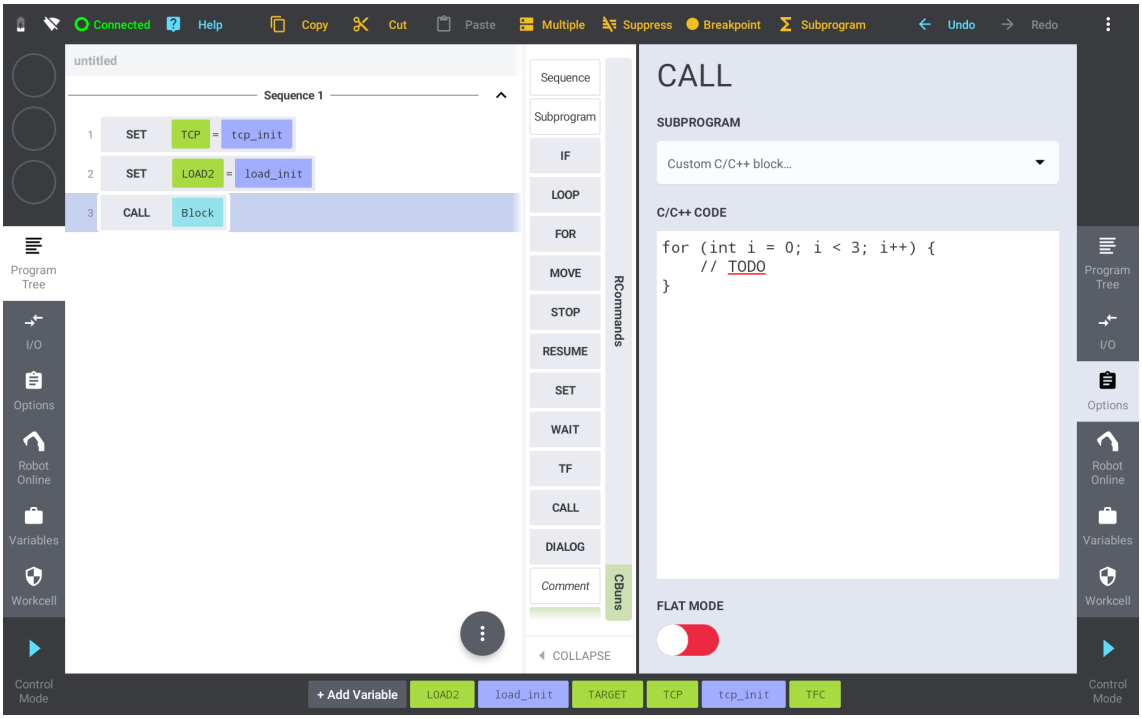
The basic functionality of the CALL command is represented by the Subprogram invocation. To call a Subprogram, drop the CALL command into the program tree, open its options panel and select a desired subprogram.

In addition, the CALL command with corresponding subprogram invocation is automatically added to the program when the user drops the subprogram shortcut into the program tree.



6.9.2 Custom C/C++ Block

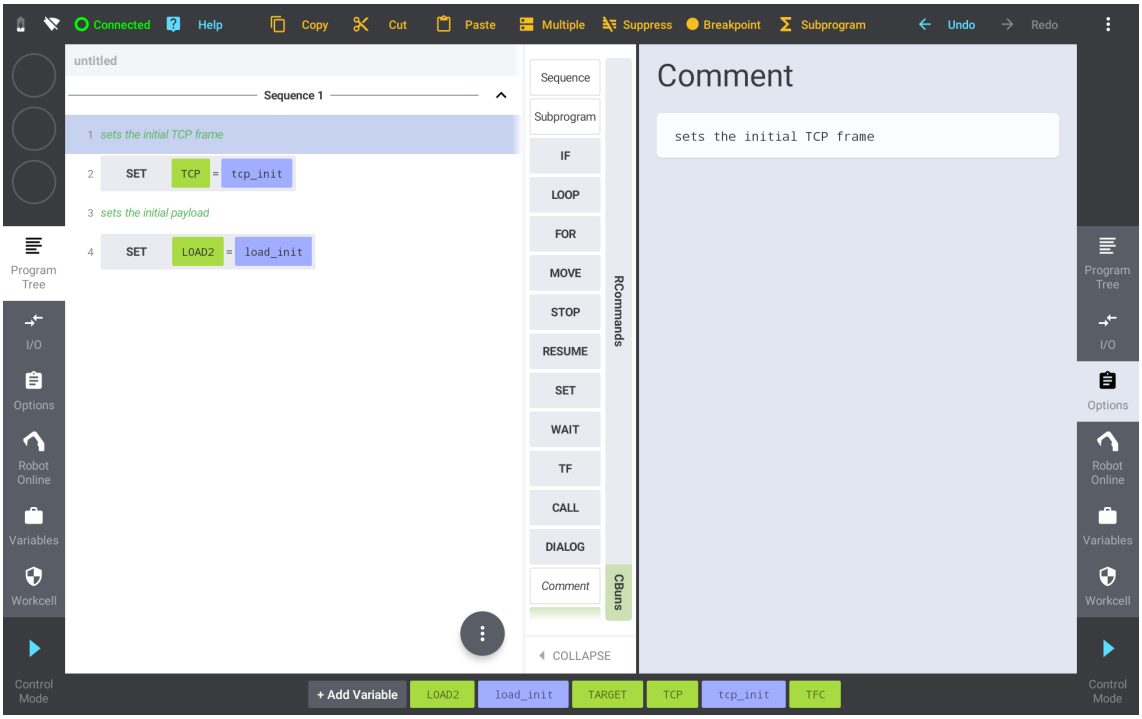
Expert robot programmers has the option to enhance robot program with custom C/C++ code. The code must be based on C++ Standard Library and Cbun Framework (RC, Program and IO API).



Please note that this feature is meant requires deep knowledge of Cbun Framework and its usage should be always discussed with Kassow Robots technical support.

6.10 Comment

The Comment command enables the robot programmer to enhance the robot program with comments, allowing easier orientation within the program tree. Comments do not affect program execution.



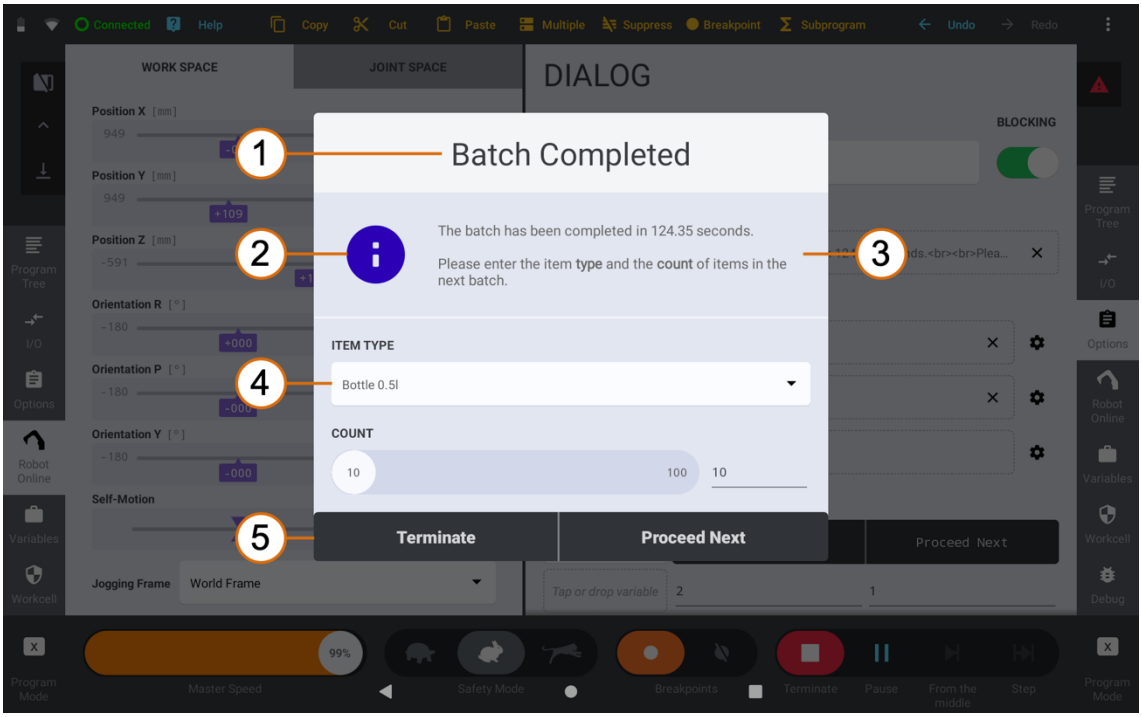
6.11 Dialog

The DIALOG command allows to invoke a custom dialog message that will be shown on the teach pendant screen each time the command is called. This custom dialog can serve both as output and input interface of the robot program, i.e. it can show (output) message to the user and collect (input) parameters entered by the user.

The DIALOG content is evaluated at each call of the command; therefore, it always shows the output based on the recent data. Note that the dialog content is not evaluated continuously once the dialog is visible.

The custom dialog can be always dismissed by clicking on any of its content buttons. In case the user clicks outside of the dialog, the dialog is just hidden temporarily. Clicking the warning icon at the top right corner of the application will show the dialog again (if it is active).

In addition, the dialog window can be dismissed automatically on signal (with no user interaction). This is suitable for situations when the dialog is shown while some defined conditions are met (such as human presence being detected by a laser scanner). Please note that for the entry variables, values for the used arguments are collected only if the dialog was dismissed by the user.



Item	Description
1	**Title** Dialog title should express the dialog purpose.
2	**Icon** (optional) The dialog can contain one of the following icons: Info, Warning or Error.
3	**Message** (optional) Dialog message should provide details of the dialog purpose. Dialog message can also display values of some variables (for example: The batch has been completed in 124.35 seconds).
4	**Input Parameter** (optional) The dialog can contain up to three input parameters. The input parameters are collected once the user clicks any of the control buttons and the collected values are assigned to the appropriate variables.
5	**Control Buttons** The dialog has one to three control buttons. All of the buttons will dismiss (close) the dialog. In addition, a value can be bound to each of the buttons and once the user clicks the button the value will be assigned to the target variable.

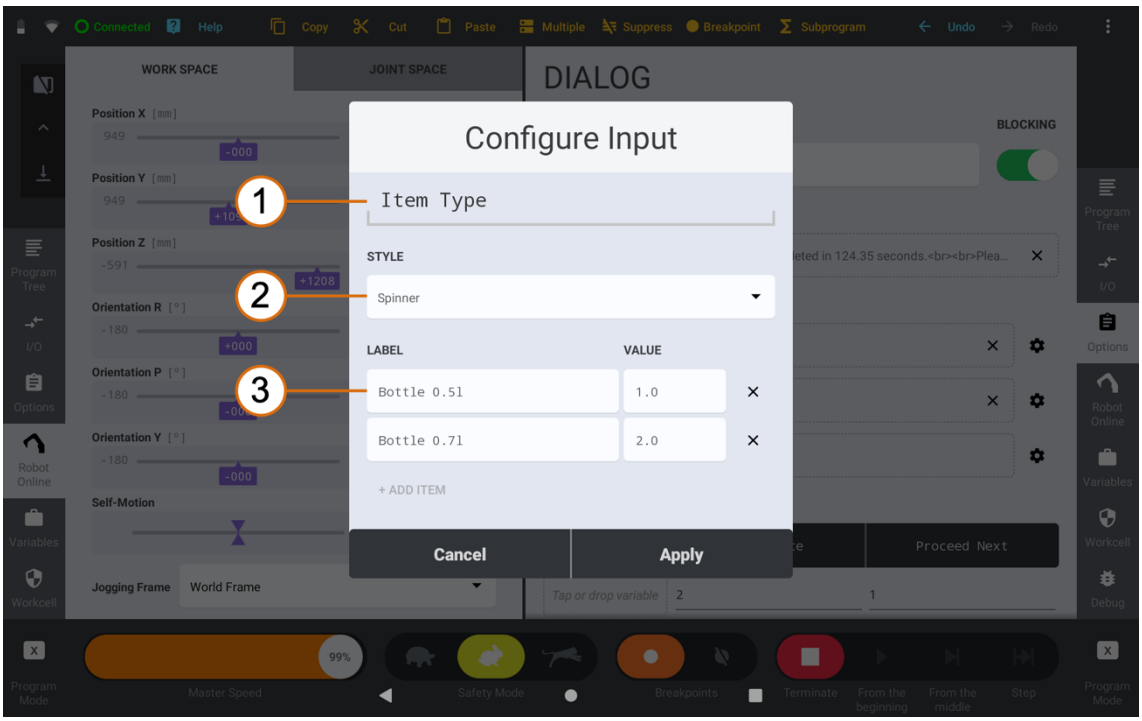
Various entry parameters can be available for the custom dialog message composition.

Item	Description
1	**Title** Allows the user to define the dialog title. The title should express the main purpose of the dialog.
2	**Blocking** Specifies whether the command is blocking. If blocking is enabled, the DIALOG command will be blocked unless the dialog is dismissed either by user or signal.
3	**Icon** Selects one of the following icons: None, Info, Warning and Error.
4	**Message** Clicking the message will allow the user to define the message via expression builder. The message allows concatenation of text and variables. In addition, the text also supports markdown language. Please see the example bellow: `"The batch has been completed in " + duration + " seconds.  Please enter the item **type** and the **count** of items in the next batch."`
5	**Input Parameter** Defines up to 3 input parameters. The input parameter is defined by dragging target variable into the input parameter box (Number, Pose and Load data types are supported). This variable will be updated with the user defined value once the dialog is dismissed. The target variable also defines the default value of the input parameter view. Clicking the settings button opens the Configure Input dialog.

6	**Buttons Layout** Selects one, two or three buttons dialog layout.
7	**Button Labels** Allows the user to define custom button labels.
8	**Button Target** Specifies the target to be edited on button click event, ie. once the user clicks one of the buttons, the button value will be assigned to the target variable.
9	**Button Values** A value to be assigned to the target variable once the button is pressed.
10	**Dismiss Signal Edge** Determines whether the dialog should be dismissed on signal rising edge, falling edge or toggle of the signal.
11	**Dismiss Signal Expression** Specifies the signal to be used for dismissing of the dialog. If the dismiss signal is not defined, the dialog can be closed only by the user.

Configure Input Dialog

The Configure Input Dialog allows to specify the input label and style.



Item	Description
1	**Label**

	Defines the input parameter label. The variable label is used by default.
2	**Style** Specifies the style of the input parameters. All supported data types provides the plain input method. In case of Number data type, also the switch, spinner and slider styles are supported.
3	**Style Configuration** Allows detailed configuration of the selected style. The style configuration options depend on the selected style. In case of spinner style, the configuration contains list of spinner items. The selected item value will be assigned to the target variable once the dialog was dismissed by the user.

6.12 MOVE

Move command represents the basic building block for handling various KR robot motions. Because of the complexity of its options and integral role of this structure for the robot programming it is described within this dedicated chapter.

This group of commands provides a primary interface for all kinds of elementary motions the user can grasp and utilise for his/her desired application to achieve various robot trajectories. The recent version of the software supports several types of motion commands (***Trajectory Type***), which can be chosen from the main roll down menu of the MOVE options [Figure 1/1].

Type	Label	Description
Joint	**MOVE J**	Move to joint position (configuration), the trajectory is linear within the Jointspace.
Linear	**MOVE L**	Move to cartesian coordinates (pose), the trajectory is linear within the Workspace.
Spline	**MOVE S**	Move to cartesian coordinates (pose), the trajectory is rendered by a set of splines.
Arc	**MOVE A**	Move to cartesian coordinates (pose) copying the arc segment with a derived radius.

Based on the selected trajectory type, the MOVE options panel provide layout for the settings and user parameters separated into two common groups: ***Trajectory*** and ***Advanced*** [Figure 1/2].

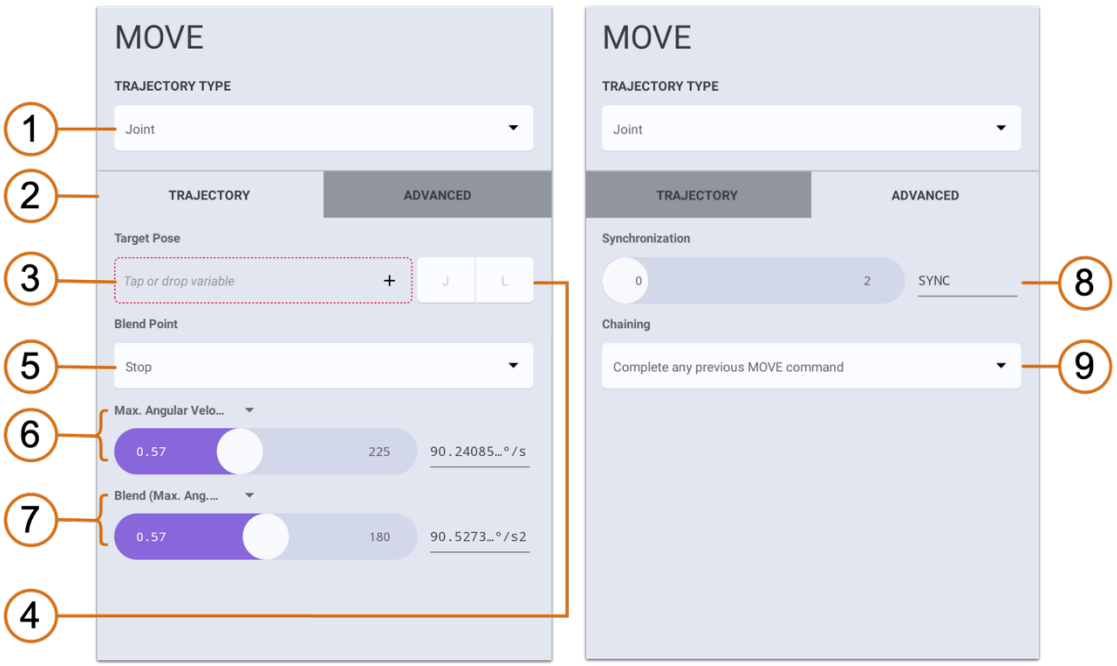


Figure 1. MOVE J Options layout

In general, the *Trajectory* tab encompasses a set of basic parameters necessary for the motion evaluation (target points or speed specification) while in *Advanced* the user can change the default behaviour of the command processing or specific geometrical features.

Target Pose

The common entry for all elementary motion commands is the **Target Pose** [Figure 1/3], which is also accompanied by the **Instant Jogging Tool** [Figure 1/4]. Using *J* or *L* buttons the user can immediately jog the robot to the pose entered within the *Target Pose* box. The *J* button uses the joint target position while the *L* uses the cartesian space and orientation for the jogging.

For a reminder please note, that each Pose variable contains both, the joint-position as well as the cartesian-space (or Workspace) coordinates. However, it is not verified by the system that those two sets are kinematically equivalent or even that those are valid when the values were modified/transformed programmatically or modified manually by the user.

6.12.1 MOVE J

The Jointspace move (MOVE J) is a basic, but very efficient kind of a motion command that simply drives the robot from the actual robot position into the target robot joint-configuration. The RC software provides the Jointspace reference trajectory independently for each of the 7 robot joint axes. The shape and properties of the motion speed/timing profile are determined by various user input parameters.

Blend Point

Each elementary motion command can be set with the *Stop* or *Via value for the* **Blend Point** option [Figure 1/5] This option determines how the robot approaches the given target, i.e., if the robot stops exactly at the target point before continuing pursuing the next target, or it is allowed to blend with the consecutive motion requests.

Geometry and Speed profile

The reference trajectory of the joint move is provided independently for each robot joint axis. Its geometry is trivial, based on the linear profile connecting the starting (actual robot joint configuration) with the target joint position, assuming valid joint boundaries.

The speed profile has rectangular composition (acceleration, constant target speed, deceleration), determined by the speed and blending parameters. The timing of the trajectory is shared for all joints, i.e. all joints start and finish their motion at the same instant.

By default, the target speed is specified by the **Max. Angular Velocity** option [Figure 1/6] and its value [degrees◦ per sec]. Alternatively, the timing of the segment can be set explicitly, use the spinner and choose the "*Duration Time*" instead the "*Max. Angular Velocity*".

- *Maximum Angular Velocity* – target joint speed [degrees◦ per sec] (each joint will
- *Duration Time* – explicit time interval [s] is required to reach the target joint configuration

The way how the MOVE J blends when used in the context of consequent move commands is based on

the same principle as in the case of MOVE L, except it is provided only in a single dimension. The resulting speed profile of such a blend is than given as a simple superposition of two consequent joint moves.

- *Blend (Time)* – the blending starts given time interval [s] before reaching the target robot configuration
- *Blend (Max. Angular Acceleration)* – a maximum joint acceleration (speed tilt) [deg/s^2] is allowed for all joints

(To get a better idea about the trajectory and blending composition, please check the next command description. MOVE L and MOVE J are quite similar at the level of their implementation, despite the fact they operate in different domains.)

Synchronization

Moves are processed sequentially and the consequent instruction or move command will be processed just at the moment, when the robot enters the blending phase of the actual move. This way, any decision about the next step or trajectory can be decided at the very last moment in the real-time.

Use the **Synchronization** [Figure 1/8] option located within the *Advanced* tab to change the interval to release the move instruction sooner, i.e., before reaching up the blending edge. The default value of this parameter is set to 0 meaning the program will continue to the next instruction right when the robot enters the respective blending area.

Immediate motion replacement

By default, each new executed move instruction is put at the end of the actual tracking plan. In some situations, it may be handy to cancel the present trajectory plan and replace it by the new one immediately. To allow that the **Chaining** option [Figure 1/9] must be set with the *"Interrupt any previous MOVE command"* option.

6.12.2 MOVE L

The linear move (MOVE L) represents the basic move command providing a coordinated robot motion to exercise the TCP along the line rendered in the cartesian-space coordinates, laid out by the initial (the actual TCP robot pose) and user defined *Target Pose*. The speed profile of this motion is determined primarily by the target speed, blend parameters and trajectory length.

Once the command is invoked, the RC draws up the linear reference trajectory representation which the robot follows by engaging all 7 joints. The orientation reference coordinates are evolving similarly, but within the context of rotational trajectories.

Blend Point

Analogously as in the case of the joint move, the MOVE L **Blend Point** *Stop* option [Figure 2/5] results in a trajectory that brings the robot TCP to the target pose before it continues to any further move operation.

When the MOVE L is set with the *Via* **Blend Point** option, the trajectory can be blended with the trajectory segment generated by the instruction which follows next in the program listing.

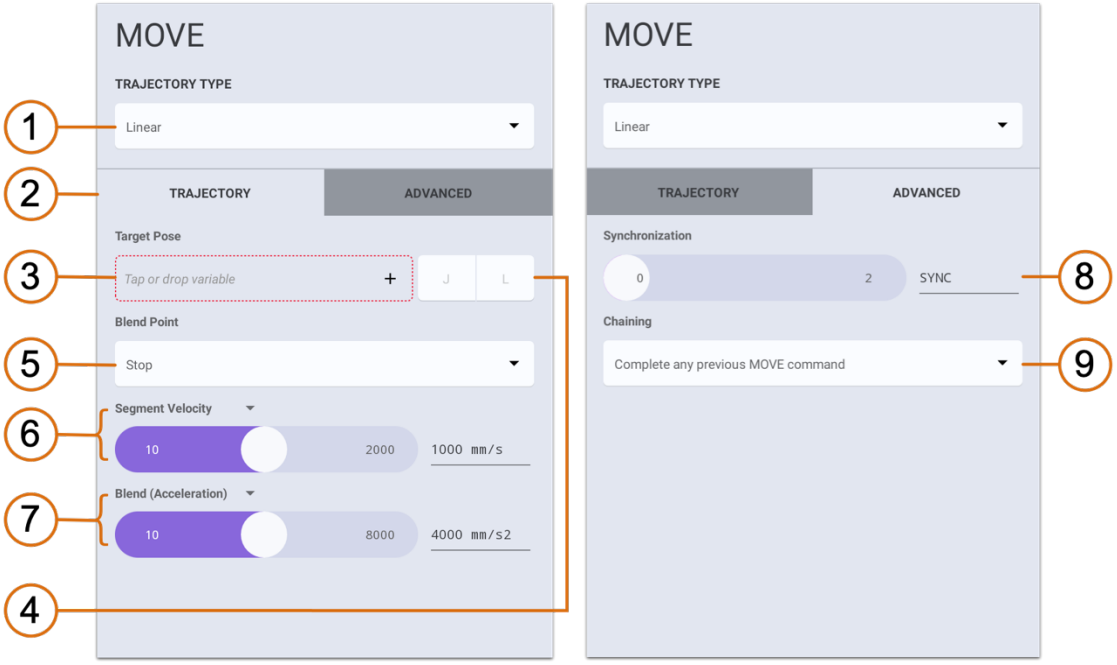


Figure 2. MOVE L Options layout

Geometry and Speed profile

The sequence of blending MOVE L commands is realized as alteration of blending and constant speed segments [Figure 3Figure 3]. The blending segment refers to the part of the trajectory, where the reference trajectory changes direction and speed to progress towards the next target pose. In other words, it is the part of the reference trajectory, where the TCP accelerates or decelerates.

The constant segment speed is specified by the ***Segment Velocity*** option [Figure 2/6] and its value. This is the target translational speed the TCP is requested to reach on its way towards the target pose. Alternatively, the timing of the segment can be set explicitly, use the spinner and choose the “Duration Time” instead the “Segment Velocity”.

- *Segment Velocity* – target speed [mm/s] to be reached on the linear segment
- *Duration Time* – explicit time interval [s] is required to reach the target pose

To determine the resulting geometry and the speed profile of a linear move, proportions of the blend segment are also required. There are 3 ways how to define the blend segment size (use the spinner to change the default “*Blend (Acceleration)*” option):

- *Blend (Distance)* – the blending starts at a specified distance [mm] from the target pose
- *Blend (Time)* – the blending starts given time interval [s] before reaching the target pose
- *Blend (Acceleration)* – only a maximum acceleration [mm per s^2] is allowed during the blending

When a sequence of linear moves is put together (by using the *Via* Blend Point option), the consequent moves provide the geometrical and speed blending around each target point.

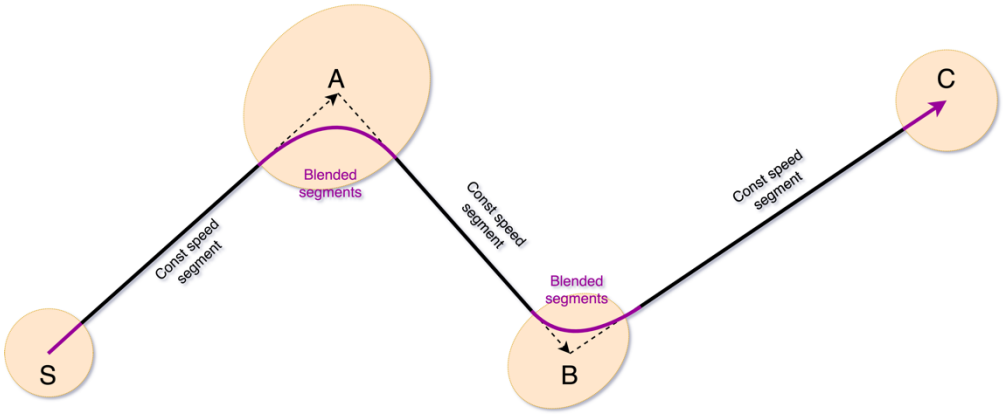


Figure 3. Blend of three consequent MOVE L commands (using via points)

Synchronization

Moves are processed sequentially and the consequent instruction or move command will be processed just at the moment when the robot enters the blending edge (sphere) of the actual move. This way the decision about the next step or trajectory can be decided at the very last moment in the real-time.

To adjust this synchronisation, you can use the **Synchronization** [Figure 2/8] option located within the Advanced tab to change the interval to unlock the present move sooner, before reaching up the blending edge. The default value of this parameter is 0, so it will continue at the next instruction right on entering the blend.

Immediate motion replacement

Same way as in the case of the joint move, each new executed move instruction is put at the end of the actual tracking plan. In some situations, it may be handy to cancel the present trajectory plan and replace it by the new one immediately. To allow that the **Chaining** option [Figure 2/9] must be set with the *"Interrupt any previous MOVE command"* option.

6.12.3 MOVE S

A different kind of the TCP trajectory tracking command, the Spline Move (MOVE S), is based on higher order curves representation. Unlike the MOVE L, the spline trajectory always passes through the defined target point (knot point) while keeping the input translational speed. The motion can be configured to use *tangential orientation* option, which provides the orientation exactly within the direction of the TCP motion.

This set of features is handy for various advanced applications like welding, dispensing or cutting. The combination of those features makes the MOVE S a better choice also for the imported curves processing (e.g., by using the Step File Cbun).

Knot Pose

It is more common to refer to *knot points* when a geometry is described by a set of polynomial curves, i.e.,

when the trajectory is being processed as a spline. In our spline moves we do that to emphasize a bit more specialised role of the entry points used for this kind of trajectory parametrisation. By a ***Knot Pose*** we label the respective target knot point also accompanied by the orientation coordinates. This pose is set by the user within the MOVE S interface [Figure 4/3].

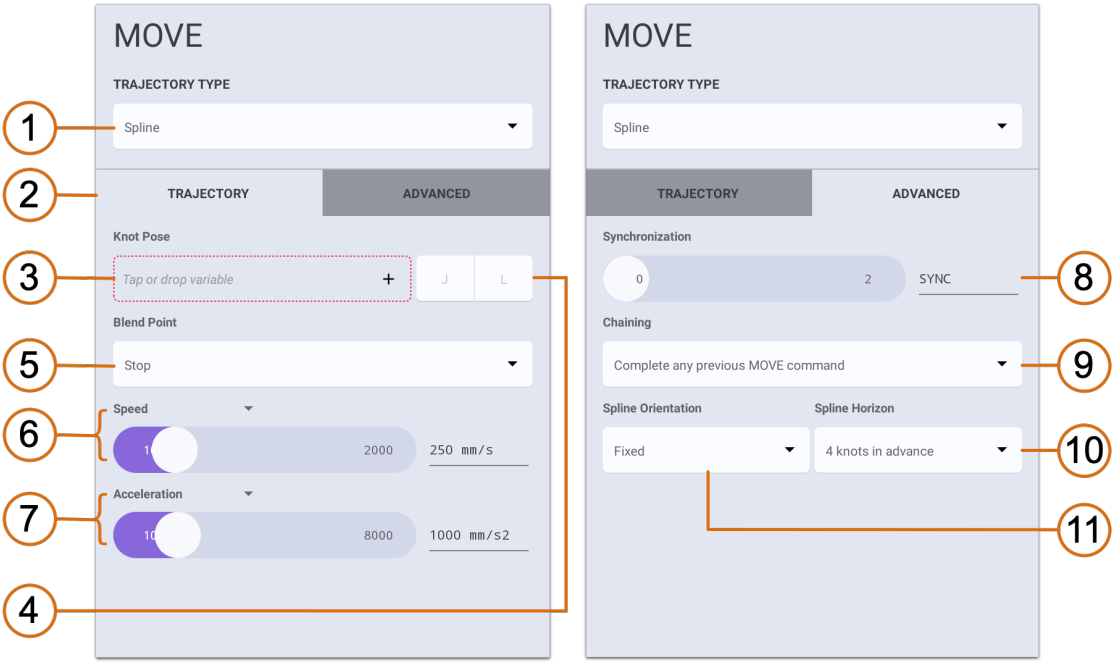


Figure 4. MOVE S Options layout

Geometry

The trajectory composed as a sequence of MOVE S commands is determined by their knot poses with the result curve optimised for low variations and geometrical tensions. In other words, the result shape should be kink, noose or slings free if that was possible for a given combination of arguments.

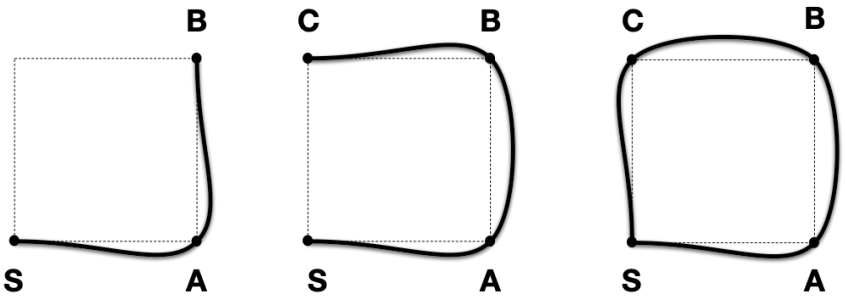


Figure 5. Composition of successive spline segments

The basic concept of the spline composition is illustrated at the picture above [Figure 5]. Consequent spline segments are glued together in a way to guarantee the smoothness and knot points path inclusion

while the first and last spline of this composition are treated specifically. Because the direction of the spline can't be determined by a consequent or previous segment at terminal knot points, it is inferred from the first/last couple. Taking the sample illustration from above, the [S, A] and [C, S] segments are determining the direction for their splines.

Please note the spline move is set to **STOP** blend point [Figure 4/5] by default. To allow the blending (gluing) between the successive commands the **VIA** option needs to be set.

Segment Speed

Aside of geometrical differences, the speed control of the spline move can be beneficial in various applications (as welding, dispensing, gluing) for which the linear motion won't guarantee absolute constant speeds (blended segments).

There are two main parameters determining the speed profile of the sequence of spline moves: the **Speed** [Figure 4/6] and the **Acceleration** [Figure 4/7]. The first one defines the level of the target absolute translational speed, while the latter specify the max. slope that can be used to achieve that within the corresponding segment.

For example, a succession of MOVE S with the different speeds and accelerations per segment,

```
/* assume the robot is located at the S pose in the beginning) */
MOVE_S A  [speed: 200mm/s, acceleration: 500mm/s^2]
MOVE_S B  [speed: 300mm/s, acceleration: 200mm/s^2]
MOVE_S C  [speed: 100mm/s, acceleration: 600mm/s^2]
MOVE_S S  [speed: --, acceleration: 150mm/s^2]
```

will result in the following kind of speed profile, where the speed is adjusted for each segment respectively [Figure 6].

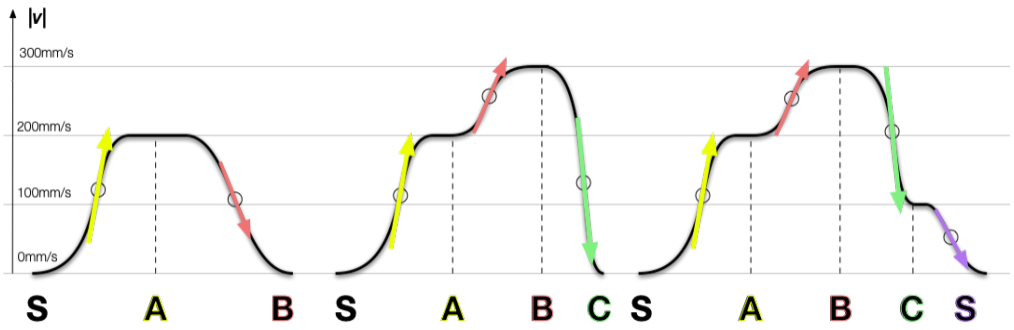


Figure 6. Speed profile of successive spline segments

Tangential orientation

The **Spline Orientation** [Figure 4/11] is set as *Fixed* for the MOVE S command by default. However, with the present implementation of the spline moves provide also support for the *tangential orientation*, useful

especially for applications where it is required the TCP keeps its direction along the spline trajectory (e.g. welding).

If the spline move command is set as tangential, the main spline is accompanied by a secondary curve, which is used as the orientation handle for TCP orientation. Since the resulting TCP orientation is treated in a way to follow the tangential direction of the main trajectory, the entry pose orientation is used only to rule out the single remaining degree of freedom (free or perpendicular axis), which is located around the tangential axis.

The user can choose from two options, which axis of the entry knot pose to use to determine the perpendicular axis of the tangential orientation, the X – *Tangent X* or Z – *Tangent Z*. This choice is relevant mainly regarding the way the entry poses are being created or teached.

A rule of thumb is that for manually teached entry poses, it is recommended to choose *Tangent Z* option, while for the sampled or generated poses (e.g. from the Step File Cbun) it is more adequate to use the other, *Tangent X*.

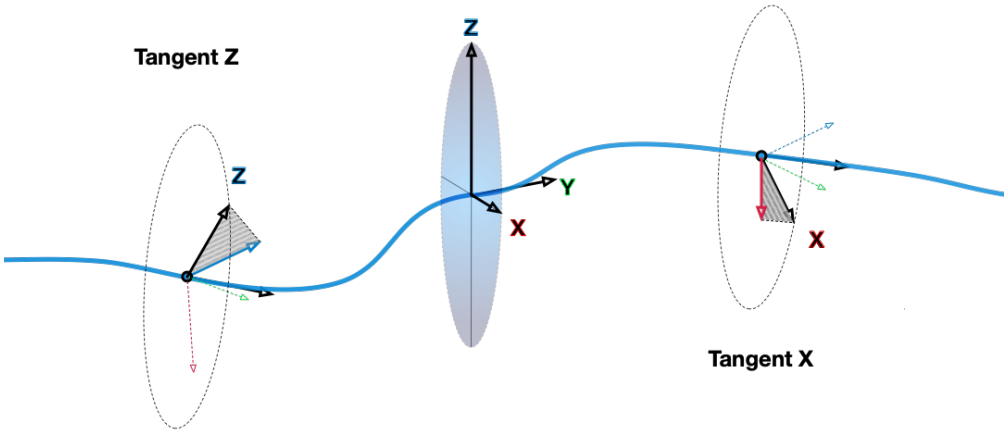


Figure 7. Tangential motion axis handling schema

Real-time processing

As in case of other elementary move commands, the spline move is also implemented in a way which allows to provide the instruction at the very last moment of the execution. This allows to build the trajectory on the fly i.e., while the robot is moving along it, just a segment ahead. Based on this feature the robot program can do decisions about the next target and amend the plan in the real-time.

Moreover, for the optimal curve shape it is crucial to keep several knot-points ahead. The size of this **Spline Horizon** can be modified in **Advanced** tab of the MOVE S command options [Figure 4/10]. The default value is using 4 knot-points. The size of this buffer affects the smoothness and shape of the resulting geometry. Basically, less points in the horizon results in more ragged curve shape and accelerations, but it requires less calculation power to update the trajectory in one step of the RC.

From the program perspective, the *Spline Horizon* also affects the way how the MOVE S instructions are executed and how the IP (instruction pointer) is updated while the spline move is being processed by the RC. For a particular MOVE S the command doesn't block the execution and continues right to the next instruction if the spline handler doesn't have the required number in its internal buffer yet.

The **Synchronization** option [Figure 4/8] is the other way how to modify the MOVE S instruction processing. The default value used for this slider field is set to 0, which means the command will wait up to a sync event, after which the instruction is “released” and the IP will move and execute the next instruction in the program. In majority of cases, the spline move sync event is simply a moment when the TCP passes through the knot-pose defined by the MOVE S command. The **Synchronization** value can be than used to adjust the time of the sync event and schedule it before the default timing using the nonzero time interval.

To eliminate any MOVE S command synchronisation, it is also possible to set the **Synchronization** *as ASYNC* (simply by dragging the slider to the rightmost position) and let the RC feed the spline move requests into its internal buffer. Although this option is not recommended because of possible negative effects on the RC performance, it can be handy in some scenarios where the set volume of entry points is well known or under control by the user.

Similarly, as in other kinds of move commands, also spline move allows completely replace/reset the actual trajectory plan by a fresh new MOVE S command. By using the **Chaining** option “*Interrupt any previous MOVE command*” [Figure 4/9] the RC will be commanded to cancel any previous move commands and blend the actual dynamical state to move towards a new target point immediately.

6.12.4 MOVE A

The arc move (MOVE A) is a specialised TCP tracking command providing the circular motion which can be helpful in a set of applications requiring regular and precise shapes and rounded edges.

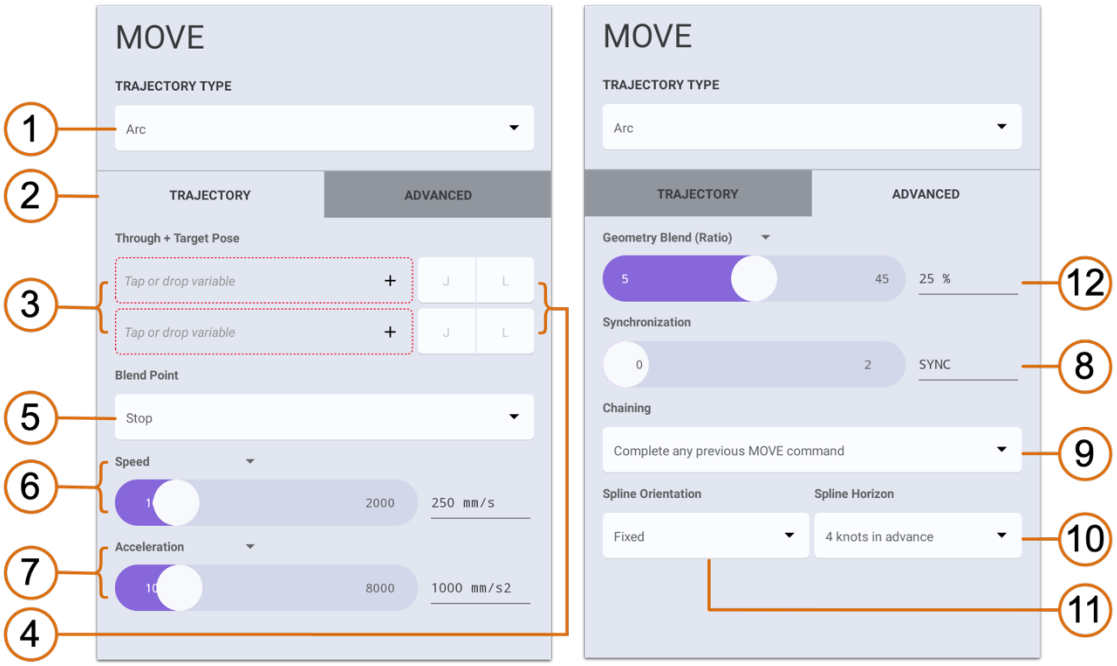


Figure 8. MOVE A Options layout

The recent MOVE A command interface allows to define the arc (portion of circle) by two entry poses **Through + Target** [Figure 8/3]. The first one is used to determine the arc radius while the latter provides the

information about the end of arc (it is supposed to be the target point as in case of any other elementary moves).

When the MOVE A is executed in the context of the program, the shape and the placement of the arc trajectory is determined from the actual robot pose (TCP) and two MOVE A entry poses. The rest of the command parameters help to adjust the arc trajectory blending, speed or instruction synchronisation to fit users' needs.

The s*peed profile* of the arc move is composed in analogous way as it is provided for the spline moves [Page 80], assuming the **Speed** and **Acceleration** [Figure 8/6,7] is specified by the user.

It also applies for the *real-time* and *synchronisation* features, which are based on the same principles.

Blending

With the **Blend Point** [Figure 8/4] it is possible to set the arc move segment as *Via.* Similarly, as for other move commands, this option basically allows to blend with any segment processed following the actual MOVE A.

The arc segment is blended into a continuing trajectory by the use of spline curves at both ends (if needed). The length of this blending sub-segment can be changed by the user, and it is expressed as a portion of the arc (in %), **Geometry Blend (Ratio)** [Figure 8/12]. The following picture briefly illustrates the basic concepts and arc move composition on 3 consequent arcs interconnected by line moves.

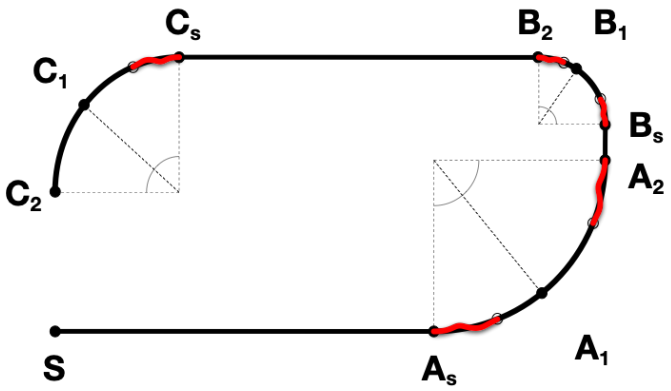


Figure 9. Arc move composition and their blend sub-segments (red)

Please note the default *Stop* value of the **Blend Point** option doesn't provide blending with a subsequent motion command and the robot stops tracking the TCP at the target pose before continuing with any further motion.

Tangential orientation

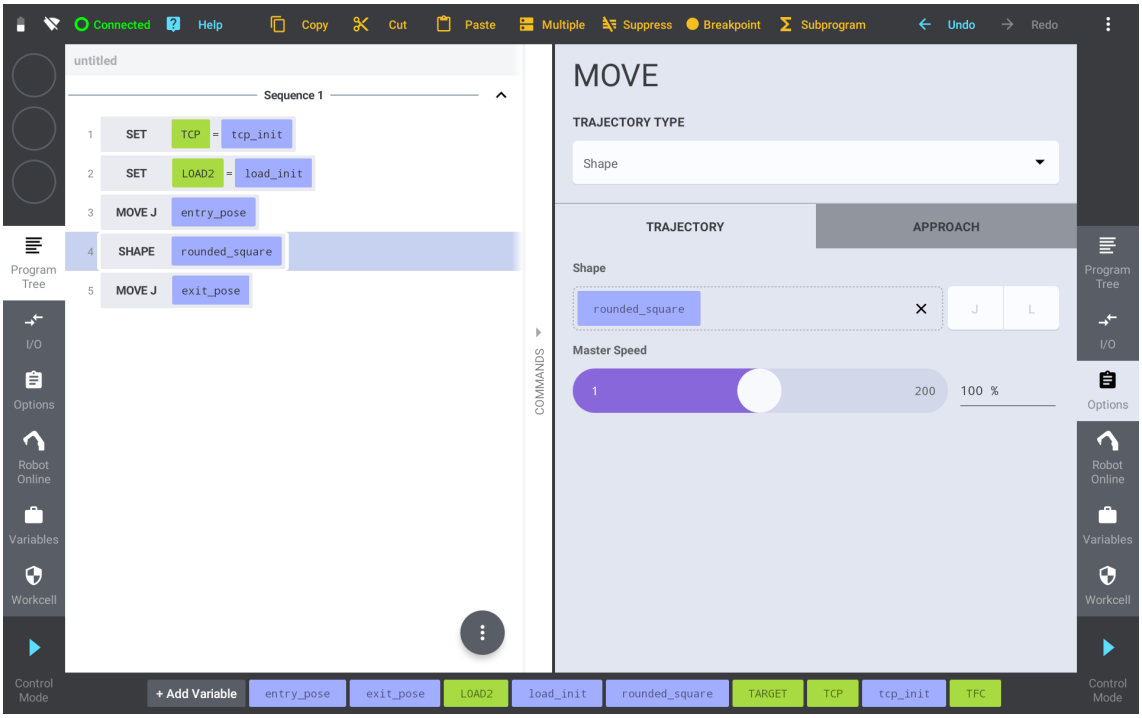
The arc move orientation tracking is providing tools based on same principles as in case of the spline move that was described few pages above [Page 80]. Also here, the entry target pose orientation is used to determine the perpendicular axis while the tangential direction is based on the trajectory shape.

6.12.5 SHAPE

The SHAPE move is a TCP tracking command that provides robot movement along a complex trajectory defined by the shape variable. The shape variable not only outlines the geometry of the trajectory but also incorporates speed and orientation profiles. The SHAPE move command is designed to execute the specified movement in accordance with the Shape variable, offering a comprehensive solution for intricate robot motions.

To add a SHAPE move, open the MOVE command's options panel and set the **Trajectory Type** to **Shape**. Specify the **Shape** by dropping a corresponding variable into the box or click the box to create a new shape variable. Please note that the jogging buttons are disabled for the SHAPE move.

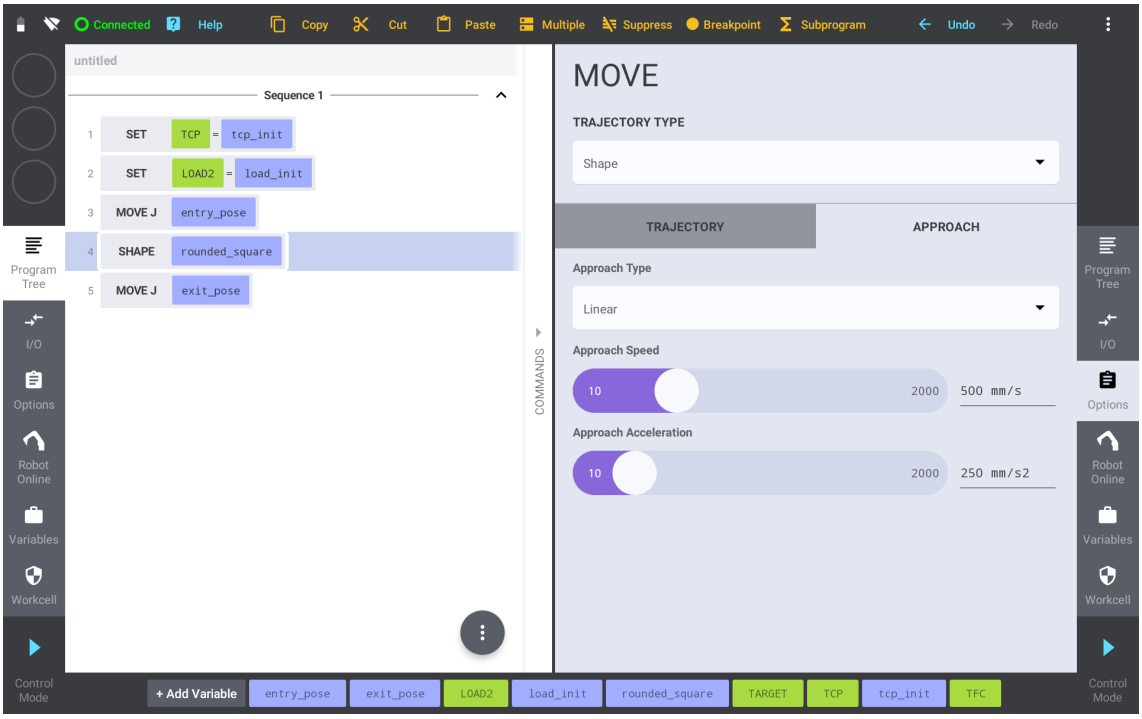
The **TRAJECTORY** options tab provides the robot programmer with the capability to configure the SHAPE move **Master Speed**, allowing for scaling of the speed profile specified within the shape variable. This parameter influences not only the target motion speed but also impacts the acceleration and deceleration of the robot movement.



The **APPROACH** options tab empowers the robot programmer to customize the robot's movement during the approach phase towards the specified shape trajectory. This phase involves the robot transitioning from its current (entry) pose to the initial pose of the shape.

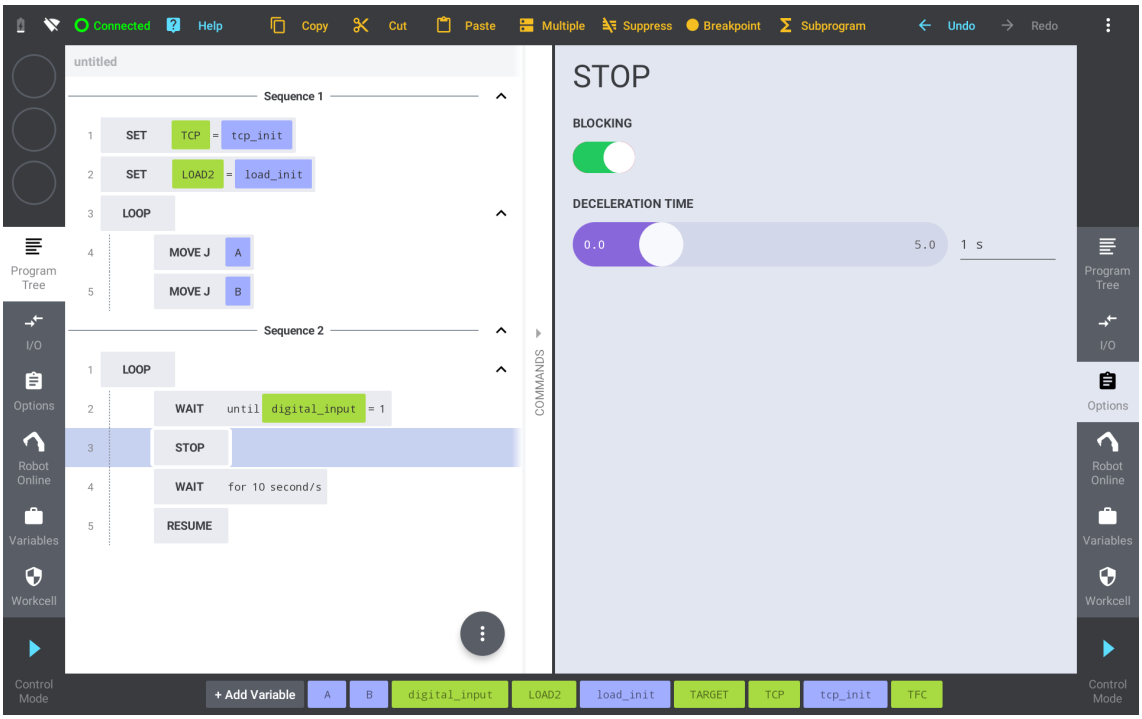
The user has the option to choose between the **Linear** and **Joint** approach type. The linear approach is provided along the line in robot's Workspace from the entry pose to the initial pose of the shape. The joint approach simply drives the robot to the shape's initial joint configuration.

Furthermore, the robot programmer can define the **Approach Speed** and **Approach Acceleration** parameters to regulate the dynamics of the movement during the approach phase.



6.13 STOP

The STOP command suspends the robot motion until the RESUME command is used to cancel this state. The instruction can be called asynchronously from any running sequence.

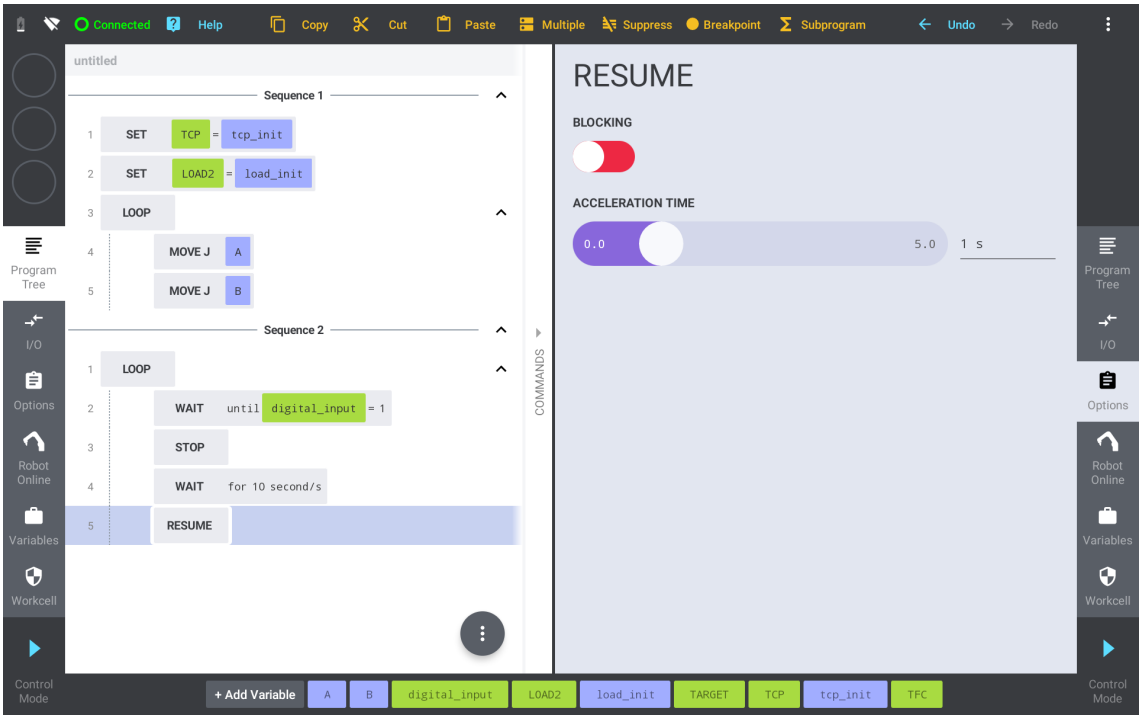


By default, the command is set as **Blocking**. In this case the instruction processing is blocked until the robot motion is stopped. To use the STOP asynchronously, turn of the blocking feature. In addition, the rate of the deceleration can be adjusted via the **Deceleration Time** value.

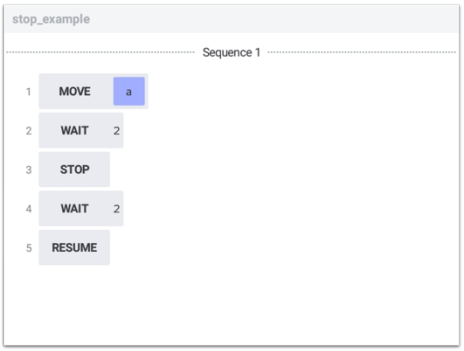
6.14 RESUME

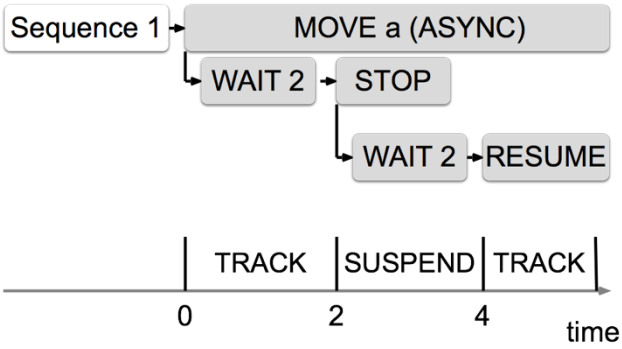
The RESUME command cancels the motion suspended state, which was previously invoked by the STOP command. The instruction can be called asynchronously from any running sequence.

The robot programmer has the option to configure the RESUME command via its options panel. If the **Blocking** is enabled the instruction processing is blocked until the robot motion is fully resumed. In addition, the rate of the acceleration can be adjusted via the **Acceleration Time** parameter.



The following example demonstrates the possible use of RESUME command after async MOVE command.





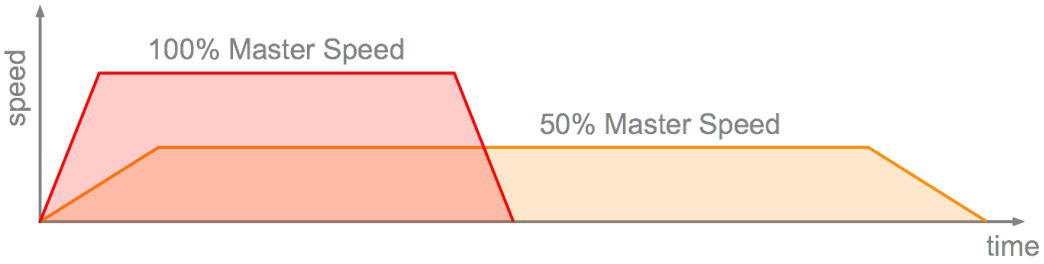
7 Program Control

The Program Control chapter describes how users can run and debug program. At the beginning the execution params are introduced such as master speed and safety mode. Later the program launch is explained as well as program termination and program debugging. All Program Control items can be found in the Bottom Bar (Program Mode) (see 3.1.5).

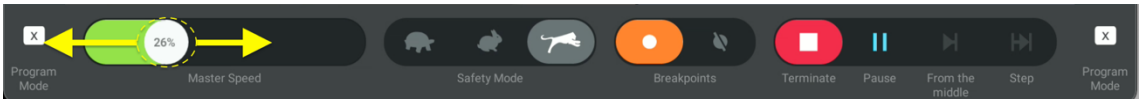
7.1 Execution Params

7.1.1 Master Speed

The *Master Speed slider* allows the user to scale down the motion speed and acceleration during the program execution. The Master speed can be set between 1% and 100%, where 100% means standard motion speed and 1% is the lowest, but the safest speed. Note that the decreasing of Master Speed leads to longer program execution time.



With the recent versions of the RC software, it is possible to change the master speed “on the fly” i.e., also while to robot is in automatic mode providing the user program or in the middle of the motion. For the safety reasons the user needs to **double-tap** the *Master Speed slider* to unlock it to be able to move it.



7.1.2 Safety Mode

The Safety Mode switch determines whether the robot should move fast or provide higher safety. There are 3 states available: Safe, Reduced, Normal. While the Normal state does not limit the speed, Safe and Reduced state limits the maximum speed of any part of the robot. That means, that any part of the robot will move faster than the applied speed limit.

Default speed limits are:

- Safe (Turtle) – 0.25 m/s
- Reduced (Rabbit) – 1 m/s
- Normal (Cheetah) – Unlimited

7.2 Program Launch

Kassow Robots control software allows users to launch their programs in just few clicks.

- Please stand clear (all persons incl. the programmer/operator) of the robot before launching a program.
- Run a new program at low speed to avoid hard collisions with surrounding equipment due to programming errors.

To launch your program:

1. Check program validity (all components have to be valid)
2. Open Program Mode
3. Adjust execution params (see 6.1)
4. Click **From the begging**

7.2.1 Move to Start Pose

To avoid any unexpected movement at the beginning of program execution, the first MOVE command pauses program and user interaction is requested. First MOVE command is always provided in Jointspace and with limited parameters.

To move robot to start pose:

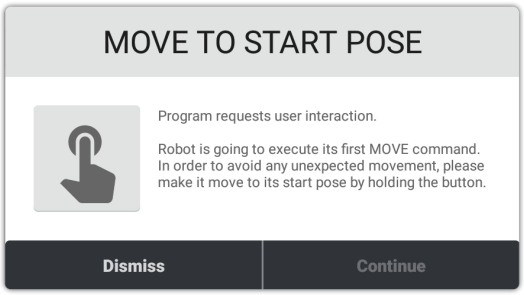
1. Press and hold
2. Wait until the start pose is reached (teach pendant vibrates)
3. Click **Continue** to resume the program execution

7.3 Program Termination

Users can invoke immediate program termination in any moment. Program termination interrupts also any ongoing robot movement.

To terminate program execution:

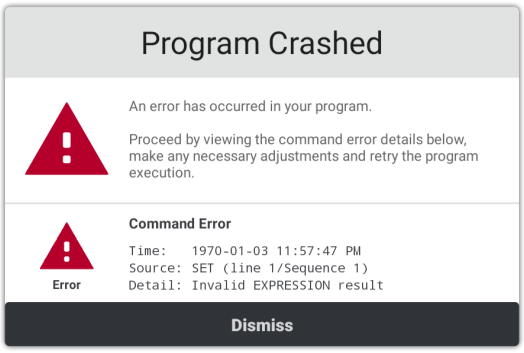
1. Open Program Mode
2. Click **Terminate**



7.4 Program Failure

Program failure occurs if the program stops functioning properly and terminates itself. Most program failures are result of command or Cbun error. Typical causes include invalid program data, unexpected command operation, robot failure or HW issues.

Each program failure is reported to user with all details relating to it.



7.5 Alarm Handling

Alarm is response of the system to the occurrence of exceptions (anomalous conditions requiring special handling) during the program execution. Alarm is typically triggered on command exception, Cbun exception, P-STOP, E-STOP or violation of safety functions.

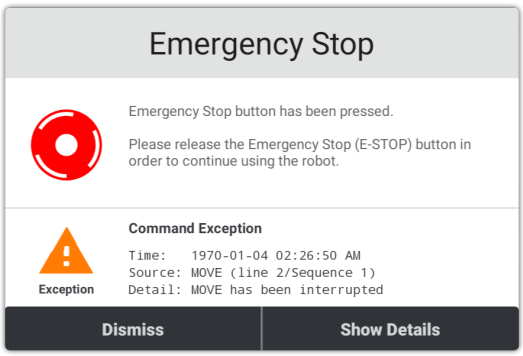
Alarm always causes the program to halt until the exceptional state is properly resolved either by user (default handling) or by one or more dedicated program sequences (custom handling).

7.5.1 Default Handling

Default alarm handling lets the user to resolve the exceptional state manually and decide whether the program execution can continue or should be terminated. Default alarm handling is available only if there is not any custom handling used.

Exception Resolution

Most of the exceptional states must be resolved by user, otherwise the program cannot be recovered. For example, if E-STOP button was pressed during the program execution, the button must be released and the robot has to be re-initialized in order to resolve the exceptional state.



Execution Control

Once the exception is resolved, the execution control can be provided. The user can proceed by:

1. **Program Recovery and Continue**

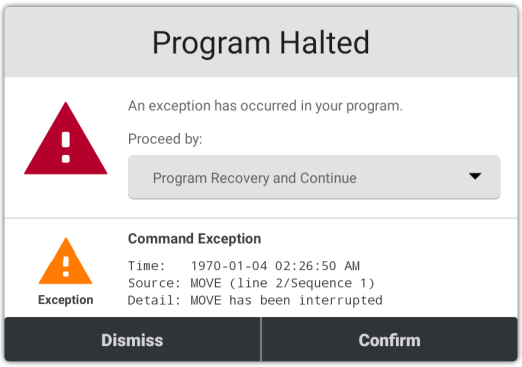
Repeats command that caused the alarm (i.e.. thrown an exception) and continues program execution.

2. **Ignore and Continue**

Ignores command that caused the alarm (ie. thrown an exception) and continues program execution.

3. **Program Termination**

Terminates program execution immediately.



7.5.2 Custom Handling

Kassow Robots control software allows users to define one or more sequences that are executed on alarm event (see 5.2.1). If program contains at least one such sequence, the default alarm handling is not used, and we are talking about custom alarm handling.

All alarm handling sequences start simultaneously when the alarm event occurs. At the same time, all other sequences are halted until the program halt state is cleared. Beware that if any exception is thrown in the alarm handling sequence, this sequence becomes halted too and the only way to resolve this situation is the program termination.

If another alarm event occurs during the ongoing custom handling, each alarm handling sequence runs again as soon as it completes the previous execution.

7.6 Debugging

Debugging is the process of finding and resolving unexpected behaviour, exceptions and errors within the program. Kassow Robots control software provides several tools for proper program debugging.

7.6.1 Pause

To pause program execution either:

1. Open Program Mode
2. Click **Pause**

or:

1. Click the Toggle button

Note: Robot motion is also suspended on program pause.

7.6.2 Continue

To continue program execution either:

1. Open Program Mode
2. Click **Continue**

or:

1. Click the Toggle button

Note: Robot motion is also unsuspended on program continue.

7.6.3 Step

Once the program is paused (either via pause button, toggle button or breakpoint), the user can control step-by-step the program execution. During this process, all MOVE commands become interactive, ie. the robot moves as long as the user holds the Step button.

In case of multiple sequences, single command in one of the sequences is called on each step button click. The execution order is not guaranteed.

To execute single command:

1. Open Program Mode
2. Press and hold **Step**
3. Wait until the command is completed (teach pendant vibrates)

7.6.4 Breakpoints

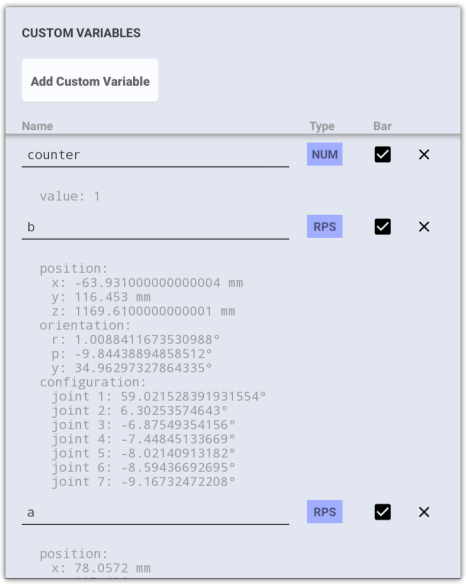
Breakpoint is an execution stop point in a program. When breakpoint is reached, it pauses the program execution and lets the user to debug the program. Note that a breakpoint can be set on any program command and even on multiple commands.

To set a breakpoint:

4. Select a program command line (with no breakpoint)
5. Click **Breakpoint**

To remove a breakpoint:

1. Select a program command line (with breakpoint)



2. Click **Breakpoint**

To enable breakpoints:

1. Open Program Mode
2. Enable **Breakpoints**

To disable breakpoints:

1. Open Program Mode
2. Disable **Breakpoints**

7.6.5 Runtime Values

Once the program is paused (either via pause button, toggle button or breakpoint), the user can inspect runtime value of custom variables (Local and Global Scope). These runtime values can be accessed via Variables Tool (see 3.6).

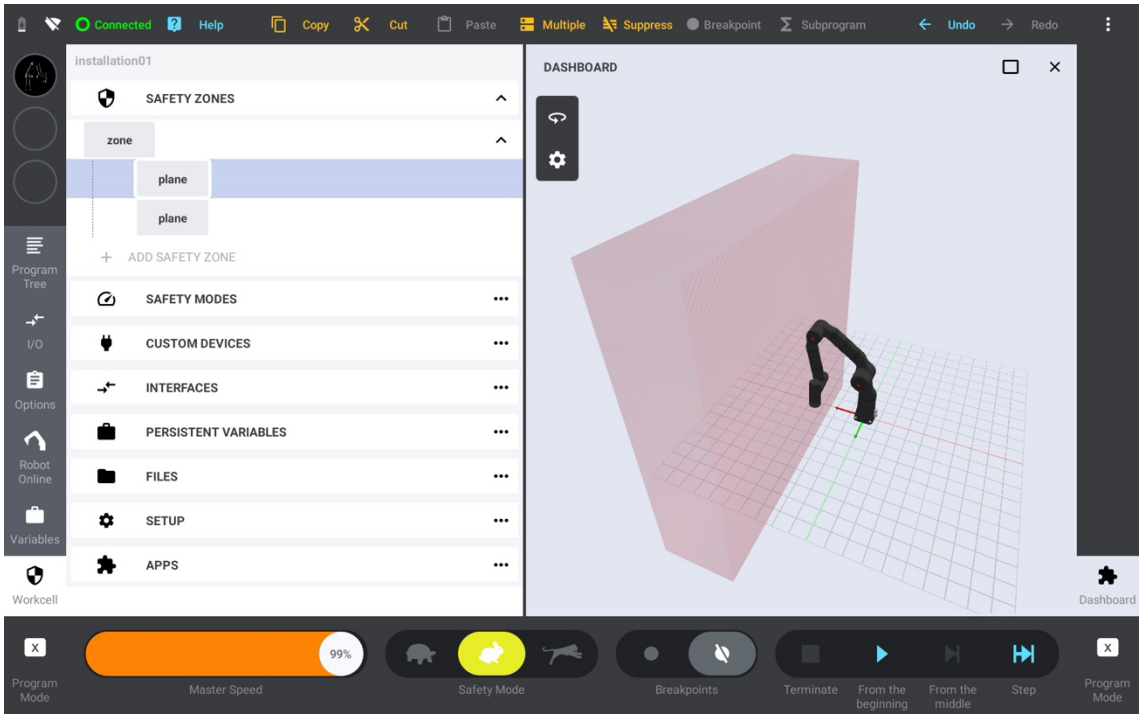
In case of debugging multiple sequences, runtime values are available only for those variables that are involved in sequence that is going to be executed next, ie. contains highlighted command.

8 Workcell

Anytime the robot is used for some specific application, the robot installation setup should be adjusted. This may involve configuration of safety zones, activation and mounting of 3rd party accessories (grippers, cameras, etc.), interfaces setup or persistent variables declaration. All these settings are grouped in so-called Workcell.

8.1 Safety Zones

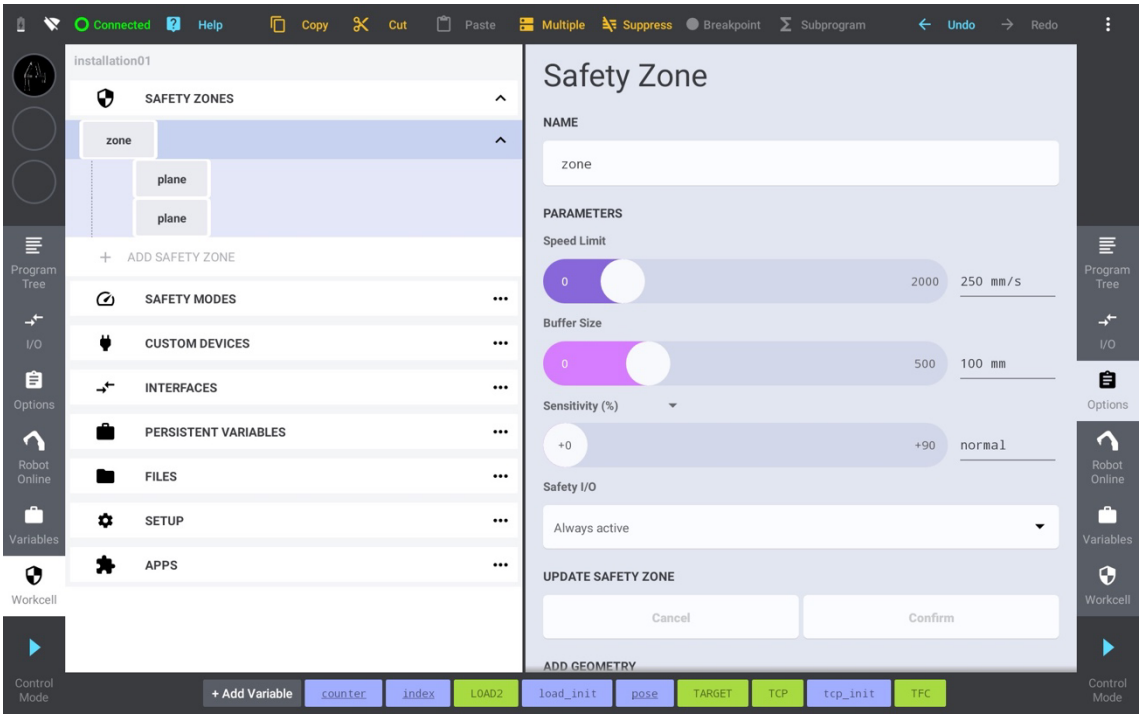
Safety Zones represent virtual boundaries in the robot Workspace. The robot will reduce its speed or stop completely if any part of the robot enters the safety zone. This can be used for protection of sensitive equipment or areas with human presence.



8.1.1 Safety Zone

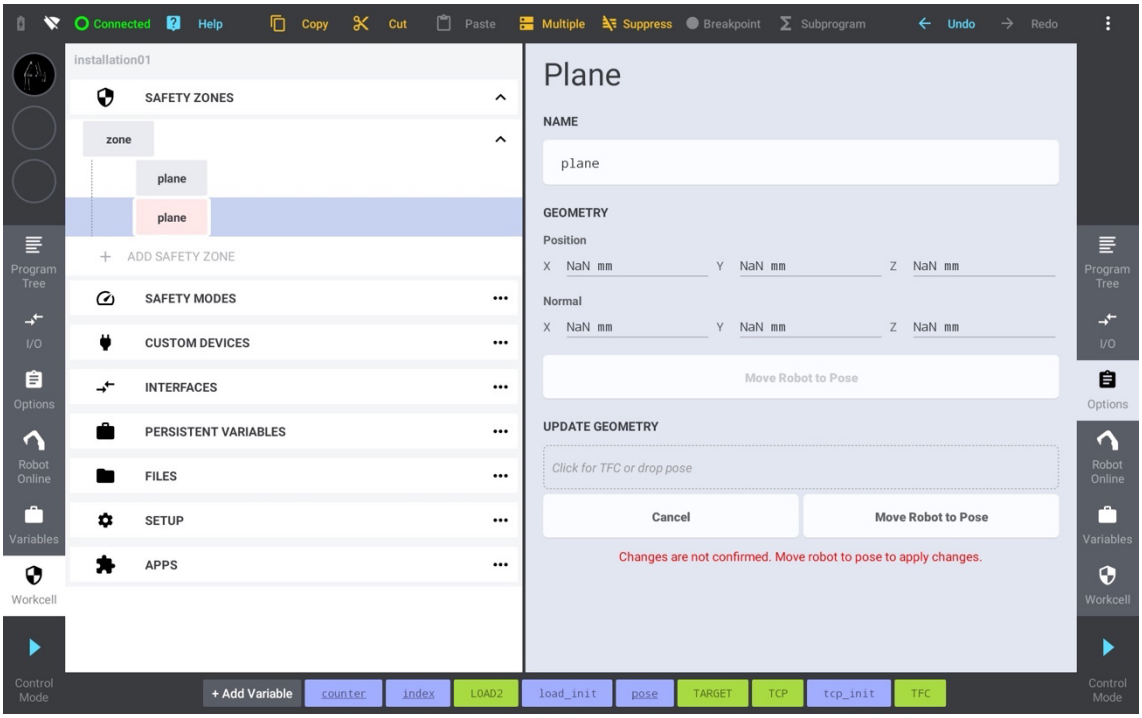
The safety zone represents a single virtual boundary. Its geometry is defined as an intersection of planes (linear subspaces). If the safety zone does not contain any geometry, it applies in the whole Workspace of the robot.

Click **+ ADD SAFETY ZONE** button to add a new safety zone to the Workcell. Do not forget to confirm the newly added safety zone by clicking the **Confirm** button in its options panel.



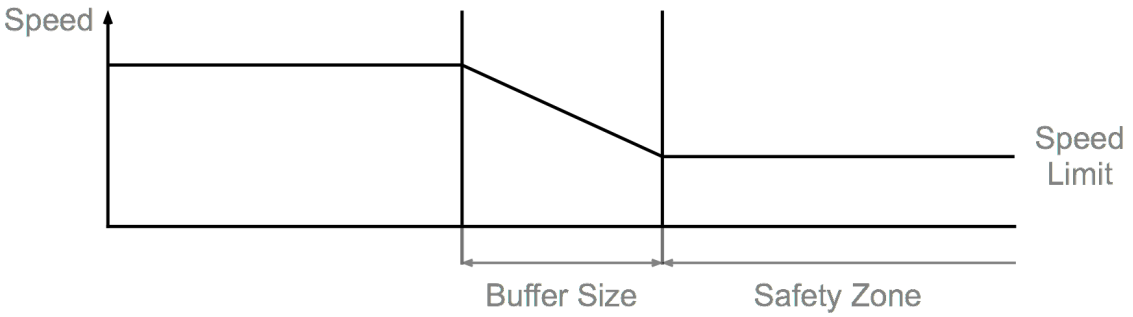
Select a safety zone and click **Cut** button to permanently remove the safety zone from the Workcell.

The safety zone is invalid in case its parameters were adjusted, but not confirmed. If any of the safety zones is invalid, the program cannot be launched. To fix it, either apply the new configuration by clicking the **Confirm** button or discard it and revert the previous configuration by clicking the **Cancel** button.



By default, the safety zone is always active, but it can also be activated/deactivated dynamically using Safety I/O.

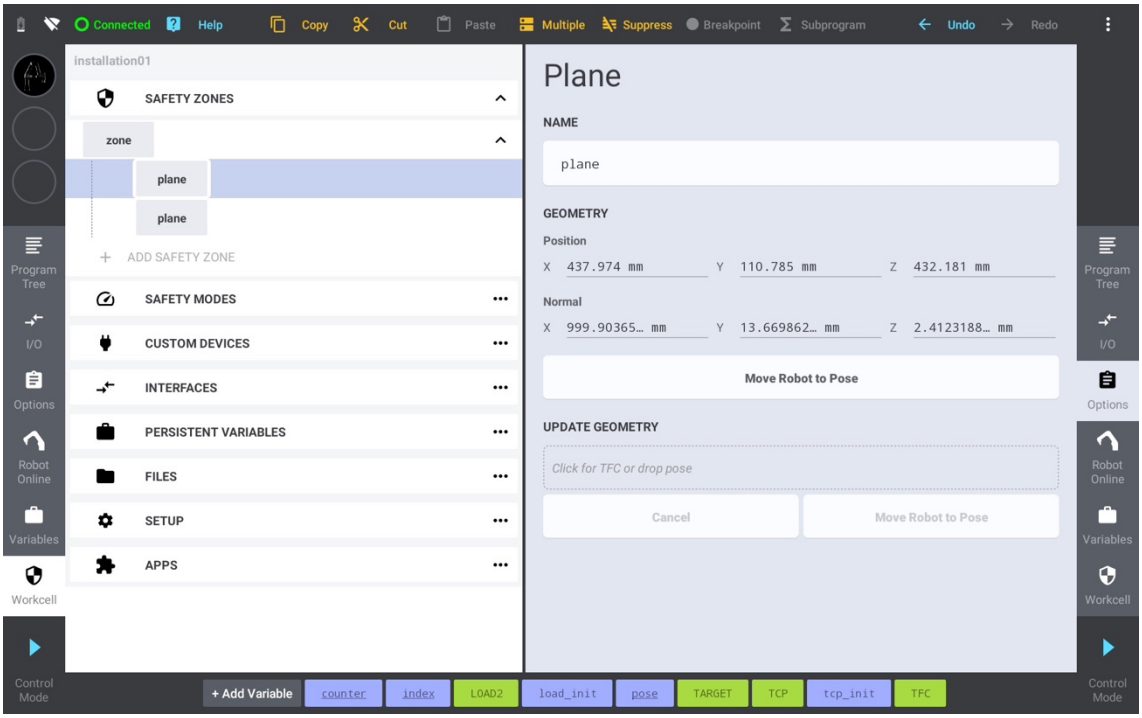
The robot will reduce its speed while it gets closer to the zone and it will keep the movement bellow the Speed Limit or stop completely inside the zone. The deceleration rate depends on the Buffer Size parameter.



The sensitivity can be configured either by ratio for all joints (%) or independently for each joint (Torques).

8.1.2 Plane Geometry

The plane geometry is a basic building block of a safety zone geometry. It represents a flat surface that divides robot Workspace into two subspaces. The plane geometry is defined in a point-normal form, i.e. using a point in the plane and a vector orthogonal to it (the normal vector).



To add a plane geometry to a safety zone:

1) Select the target safety zone and open its options panel.

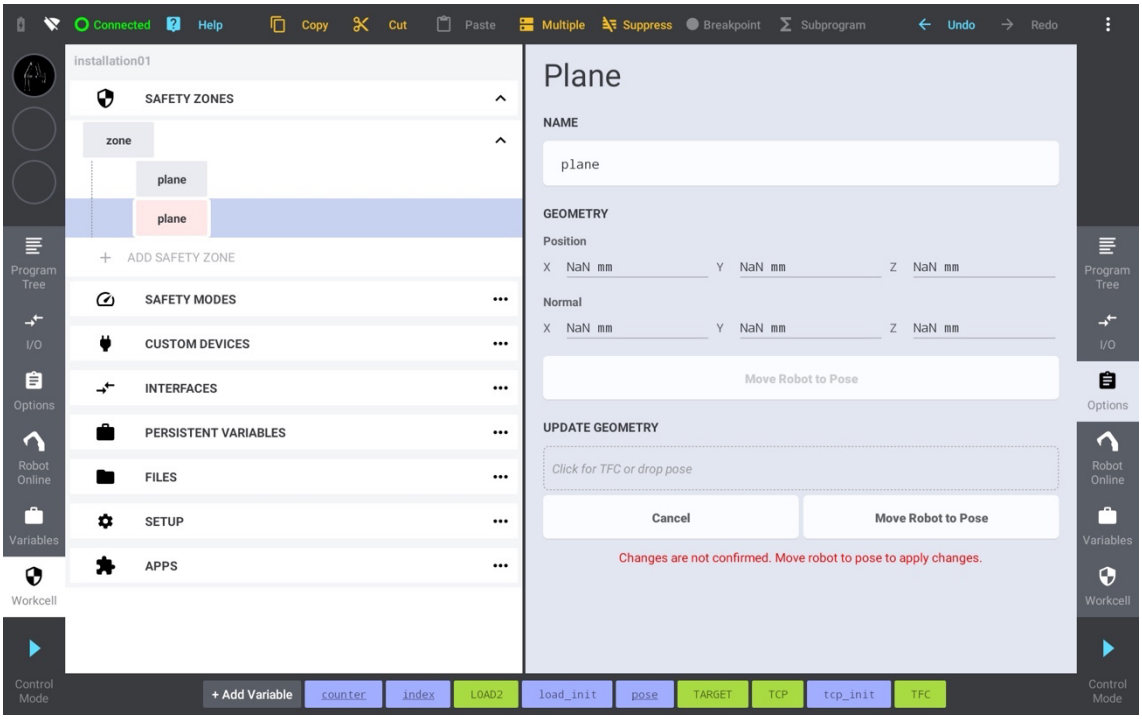
2) Drag&drop predefined pose variable to the **ADD GEOMETRY** drop box or click the box to define safety zone from the current TFC position/orientation. The plane will be aligned with the X and Y axes of the input pose and the positive direction of the input pose's Z axis defines the protected linear subspace.

3) Click and hold **Move Robot to Pose** button until the robot reaches the target pose.

4) Click **Confirm** to apply the geometry.

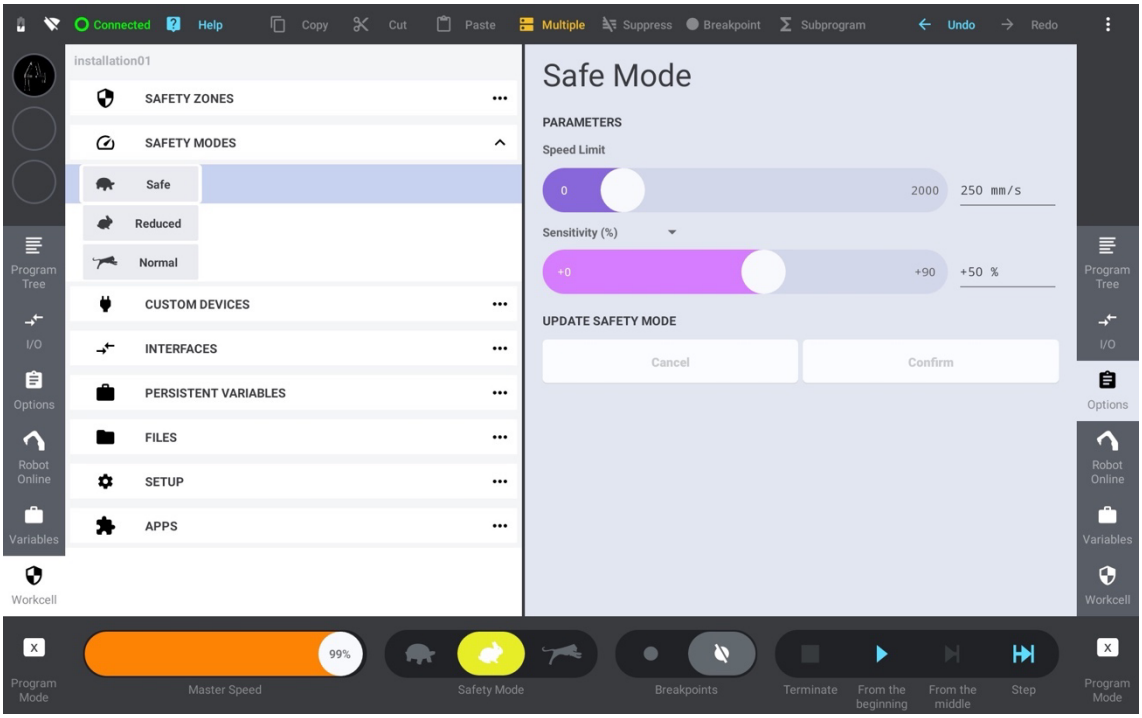
Select a plane geometry and click **Cut** button to permanently remove the geometry from the safety zone.

The plane geometry is invalid in case its parameters were adjusted, but the new configuration was not confirmed. Note that if any of the geometries is invalid, the program cannot be launched. To fix it, either apply the new configuration by moving the robot to pose and clicking the **Confirm** button or discard it and revert the previous configuration by clicking the **Cancel** button.

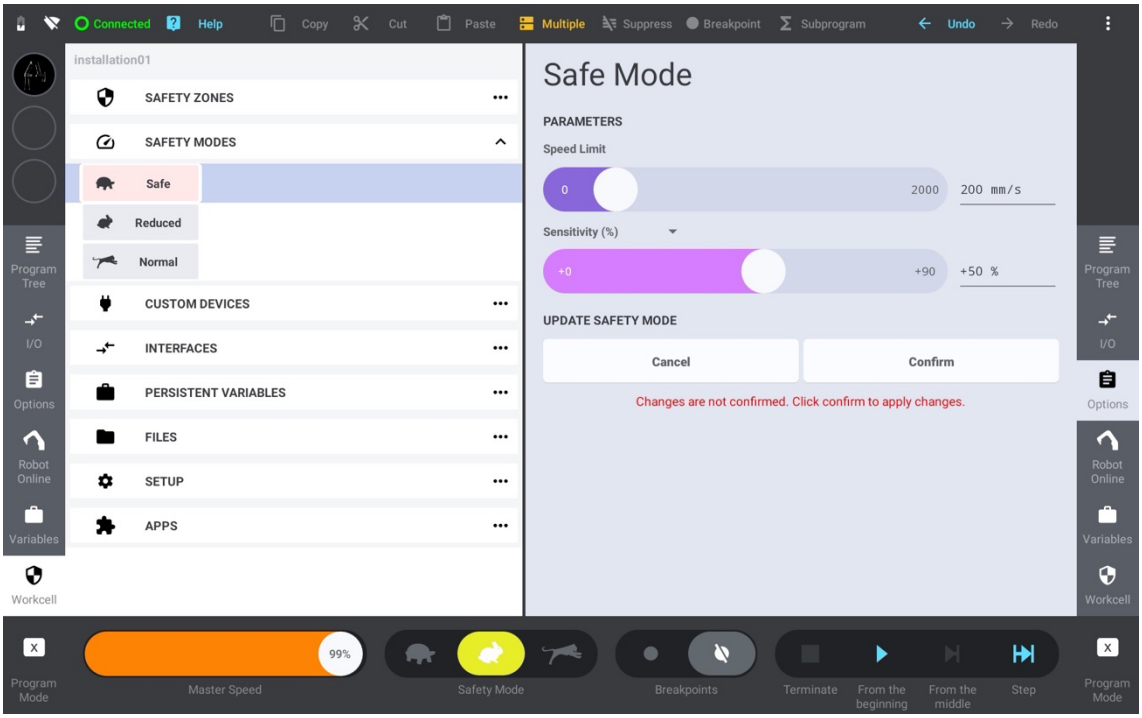


8.2 Safety Modes

Safety Modes Workcell section allows you to configure parameters of the available safety modes. Select a safety mode to open its options panel. The speed limit defines the maximum speed that the robot can move within the Workspace. The sensitivity can be configured either by ratio for all joints (%) or independently for each joint (Torques). Adjust the parameters and click the **Confirm** button to apply new safety mode configuration.

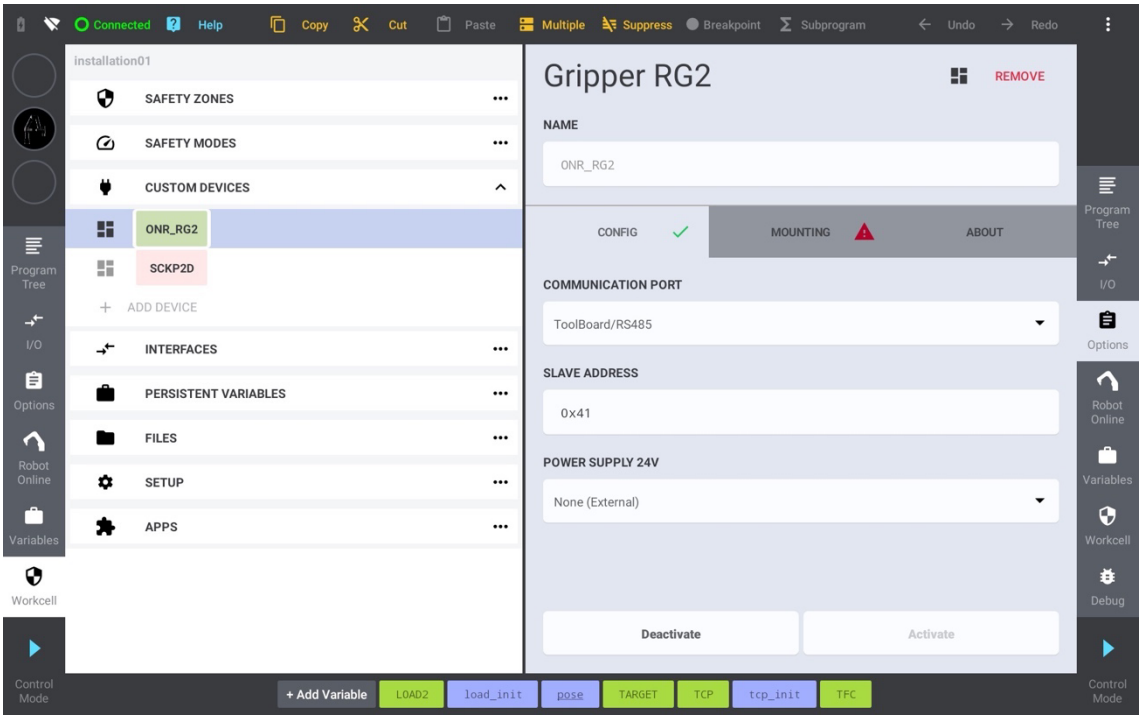


The safety mode is invalid in case its parameters were adjusted, but the new configuration was not confirmed. Note that if any of the safety modes is invalid, the program cannot be launched. To fix it, either apply the new configuration by clicking the **Confirm** button or discard it and revert the previous configuration by clicking the **Cancel** button.

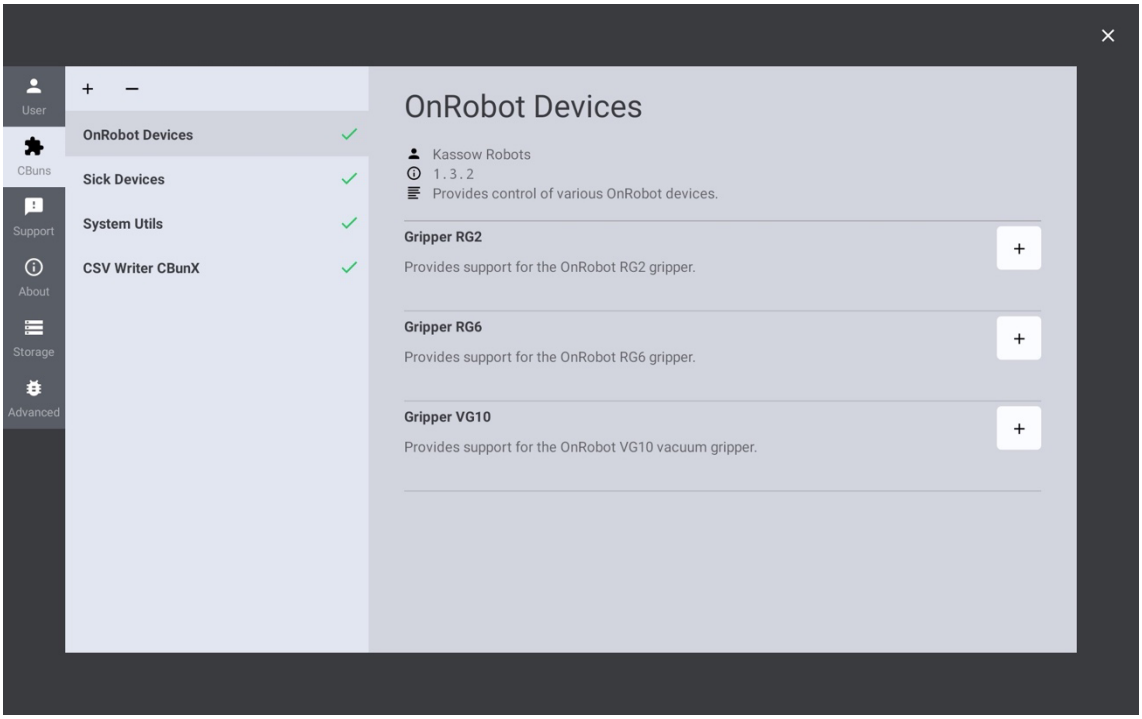


8.3 Custom Devices

Custom device typically provides a support to robotic peripherals, like grippers or sensors.



Click **+ ADD DEVICE** to open *Settings->Cbuns*, select a Cbun and click the **+** button to add a new device.

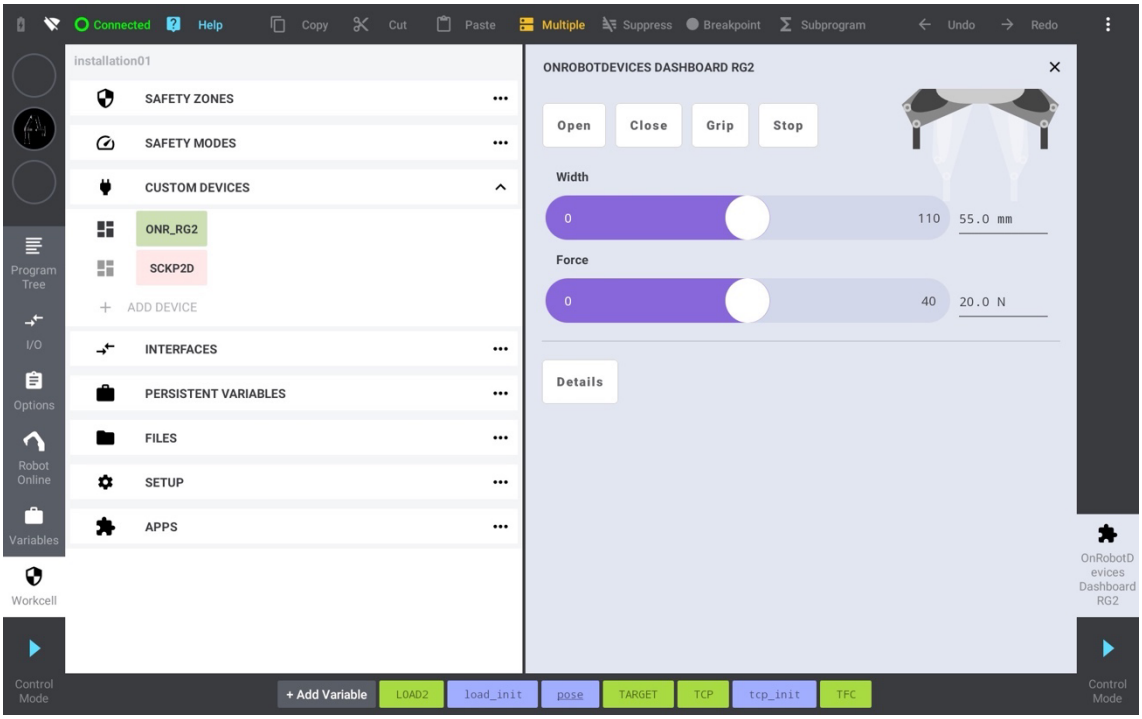


Select a device and click **Cut** button to permanently remove the device from the Workcell.

The device is invalid in case it is not activated. Note that if such device is involved in Program Tree, the program cannot be launched. To fix it, either activate the device by clicking the **Activate** button or cut/suppress the device commands in the Program Tree.

8.3.1 Device Dashboard

Some custom devices may provide optional dashboard for online device control and monitoring. Clicking the dashboard icon (if enabled) launches the device dashboard on the opposite side of the screen. Once the dashboard is open, it can be closed by clicking the close button.

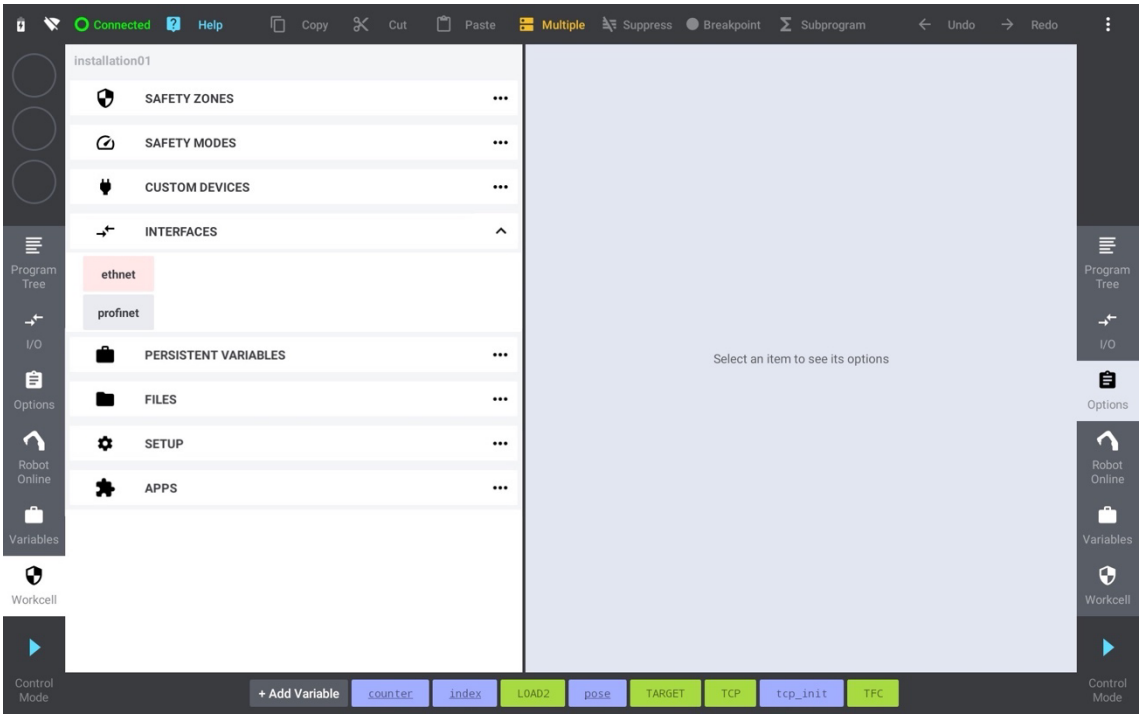


Note that you can also create a dashboard shortcut by dragging the dashboard icon to one of the shortcut placeholders.

8.4 Interfaces

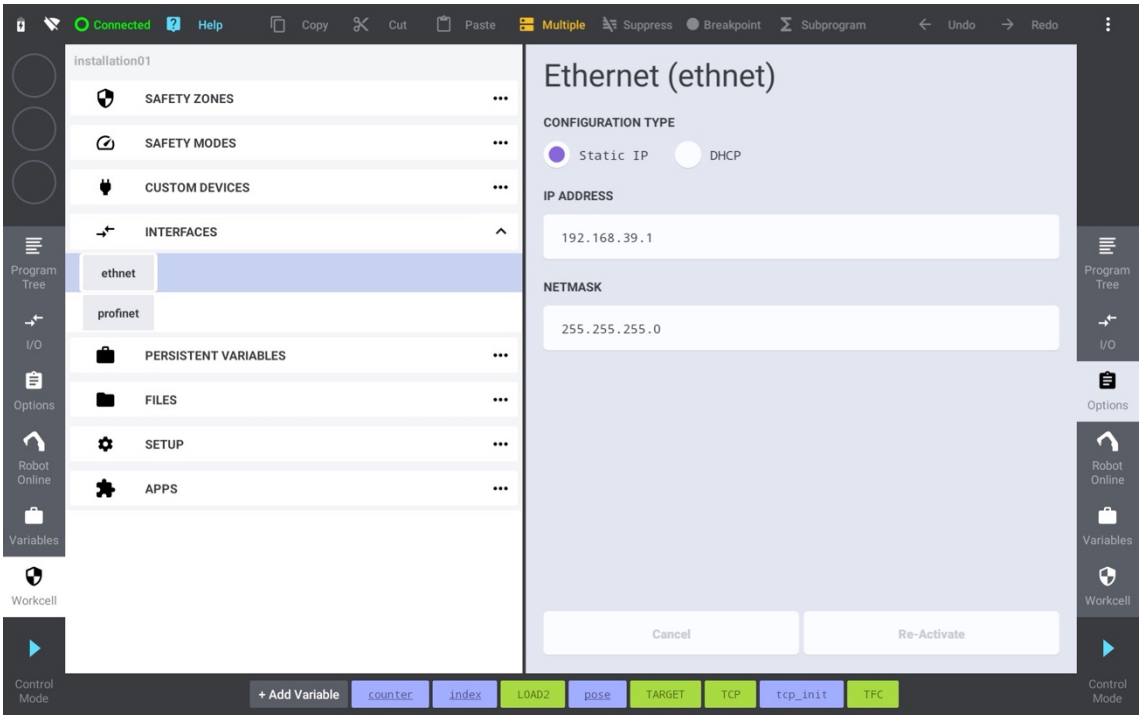
The Interface sections allows you to configure robot's network interfaces.

Interface	Description
ethnet	This standard ethernet interface can be used for internet access (necessary for remote support) or for communication with other devices/equipment on your local network.
profinet	Allows to integrate the robot into a Profinet network. The Profinet interface is available only if the robot is equipped with optional Profinet HW.



8.4.1 Ethernet Interface

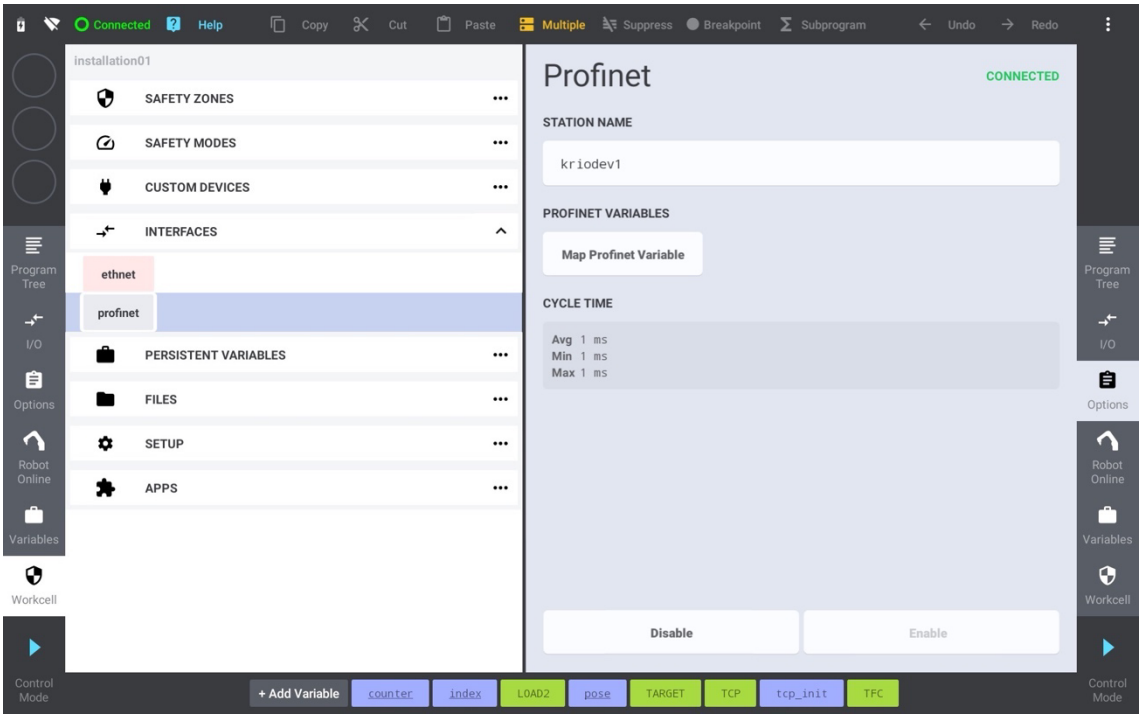
Each robot is equipped with a single standard ethernet interface (ethnet). The options panel of ethernet interface involves assigning IP address (DHCP or Static IP) and setting netmask. Adjust the parameters and click the **Re-Activate** button to apply new configuration.



8.4.2 Profinet Interface

Allows you to integrate robot into a Profinet network as s Profinet IO Device. The Profinet interface is available only if your robot is equipped with optional Profinet HW.

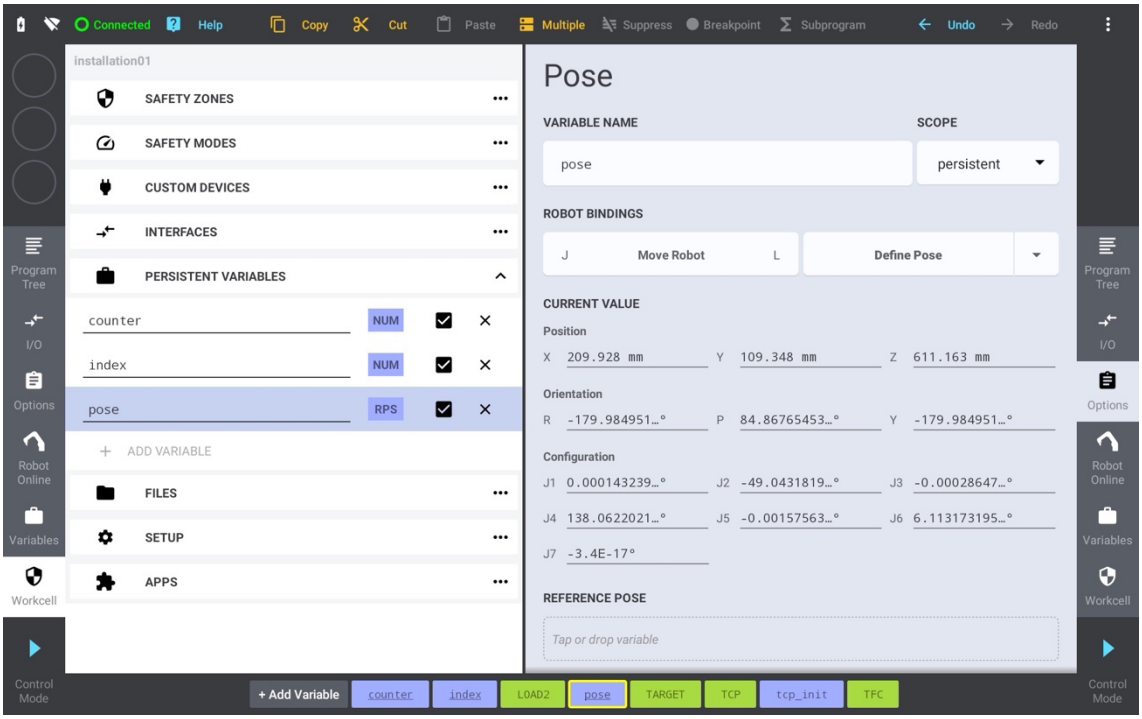
The Profient options panel allows you to enable and disable Profinet interface, monitor its state and performance. There is no need to configure anything on the robot's side, since this is provided remotely via Profinet IO Controller (for example PLC).



8.5 Persistent Variables

Persistent Variables Workcell sections allows you to manage the list of persistent variables. A persistent variable exists outside the program execution, therefore it keeps its value on program restart as well as on the robot system reboot. The persistent variables are stored and loaded together with the Workcell.

If the persistent variable scope is changed to local or global, the variable is removed from the *workcell* variables register and is placed into program variables register.



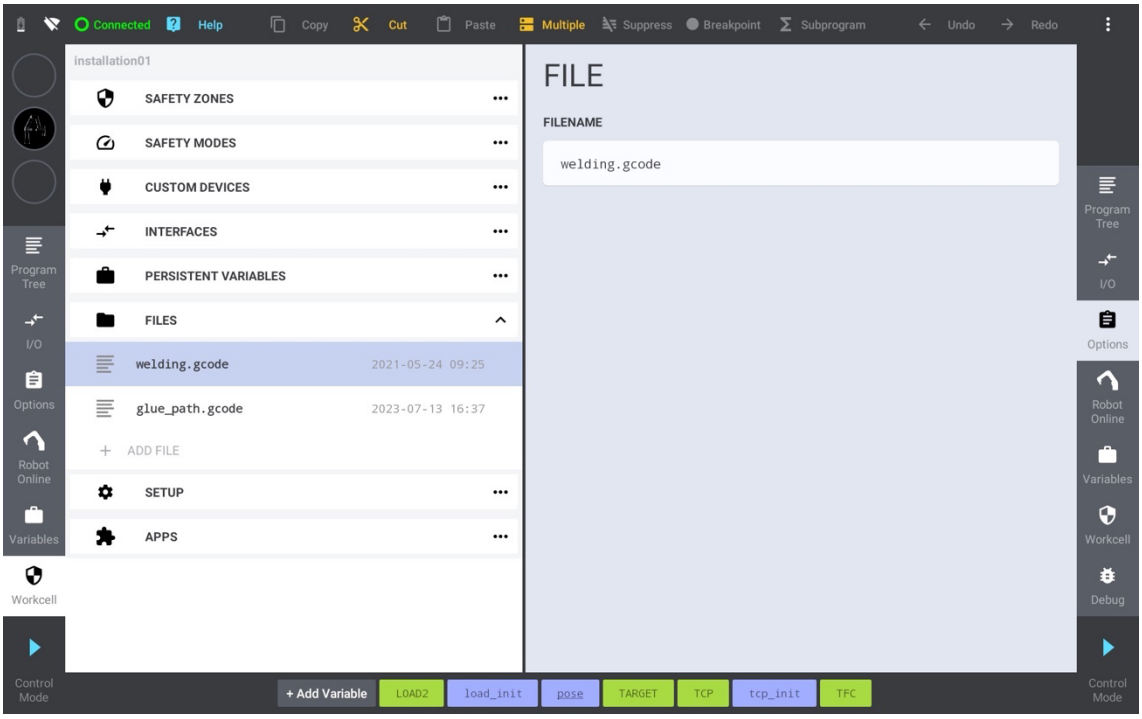
Each line represents a single persistent variable. You can see the variable label and data type (see below). The shortcut checkbox determines whether the variable will be listed in the bottom bar.

- NUM: number
- PSE: pose
- RPS: robot pose
- LOA: load
- PTR: grid pattern

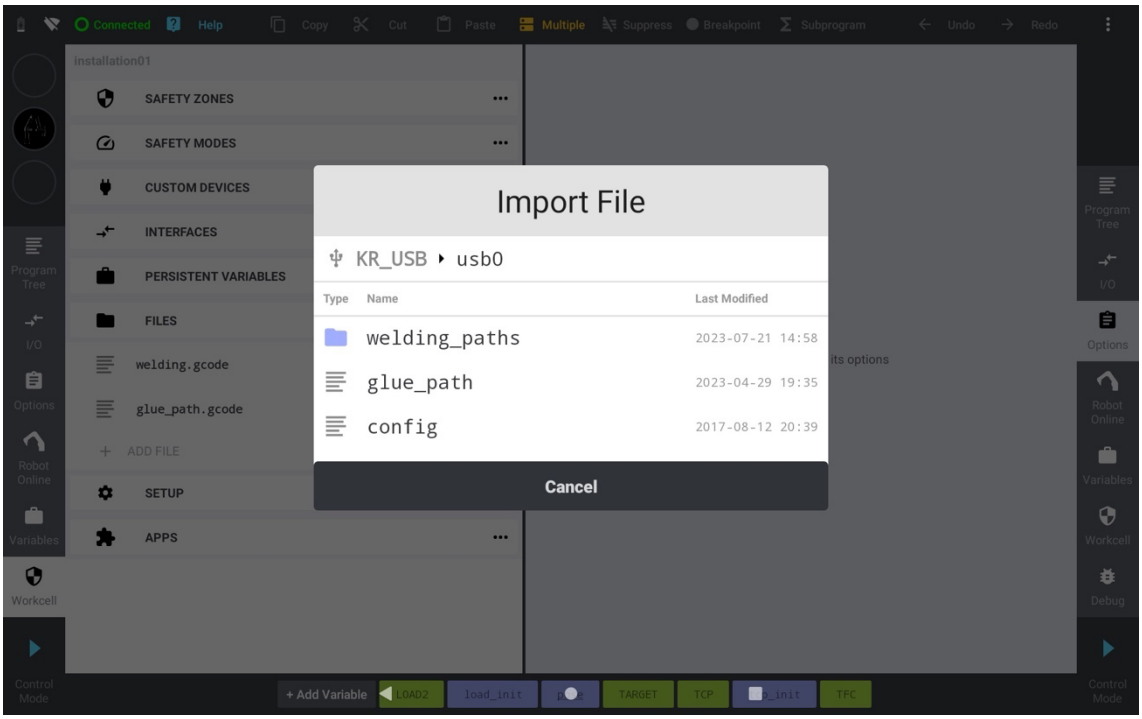
A persistent variable can be added to the Workcell by clicking the **+ ADD VARIABLE** button. Click the **X** button or select a variable and click the **Cut** button to permanently remove the variable from the Workcell.

8.6 Files

The Files Workcell section allows you to import and manage files. These files can be listed and used by Cbuns. Once the files are imported in the Workcell, they are stored together with the Workcell.



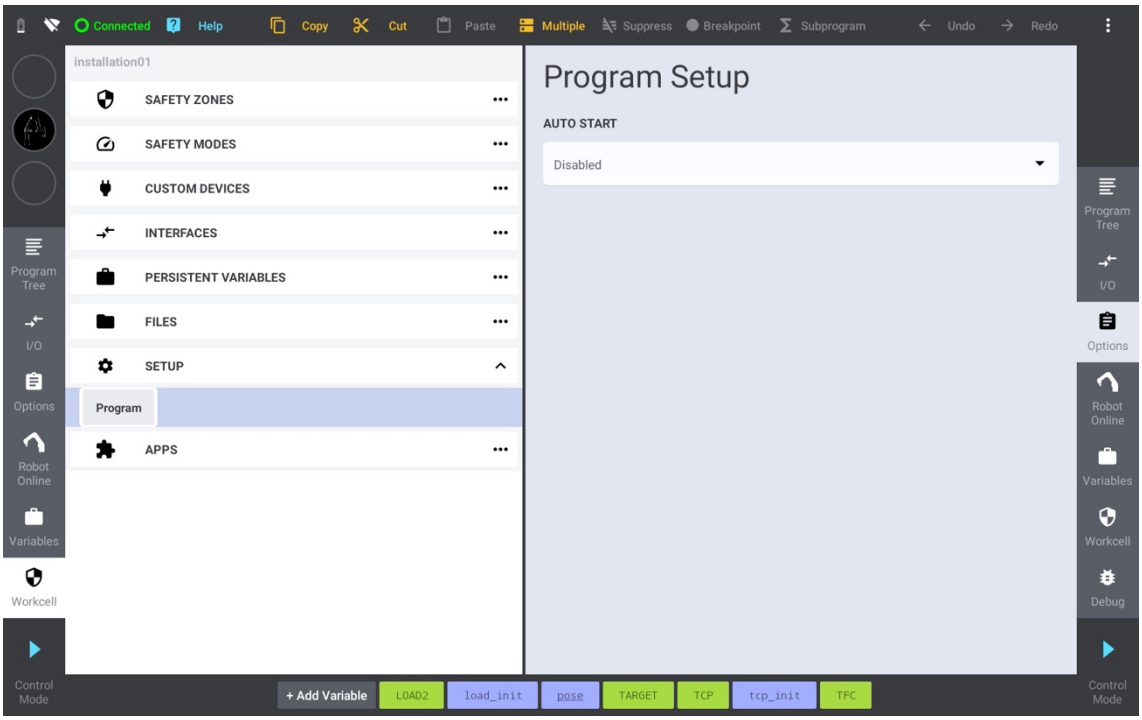
A file can be added to the Workcell either from a USB stick or the Google Drive. Use cabinet's USB port to attach you USB stick or log in to your Google Drive via *Settings->Storage*. Once the source storage is setup, click the **+ ADD FILE** to open the import file wizard.



Select a file and click **Cut** button to permanently remove the file from the Workcell.

8.7 Setup

The Setup Workcell section contains a single element called Program, that allow you to enable/disable program auto-start. If auto start is enabled, the program execution starts automatically as soon as the robot is turned on.



To activate auto-start:

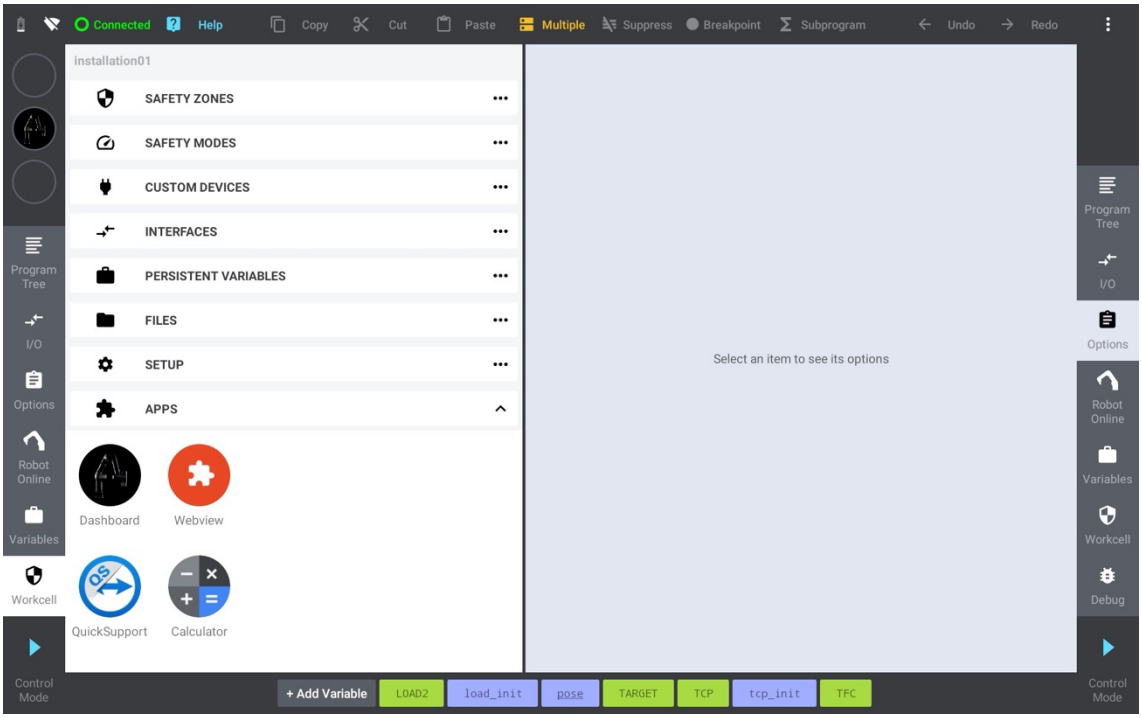
1. Terminate any running program cluster.
2. Open program to be launched on robot start.
3. Select On Power Up option from Auto Start spinner.

Note that the auto-start activation will fail in case the program cluster or workcell is not valid.

To deactivate auto-start, select Disabled options from Auto Start spinner.

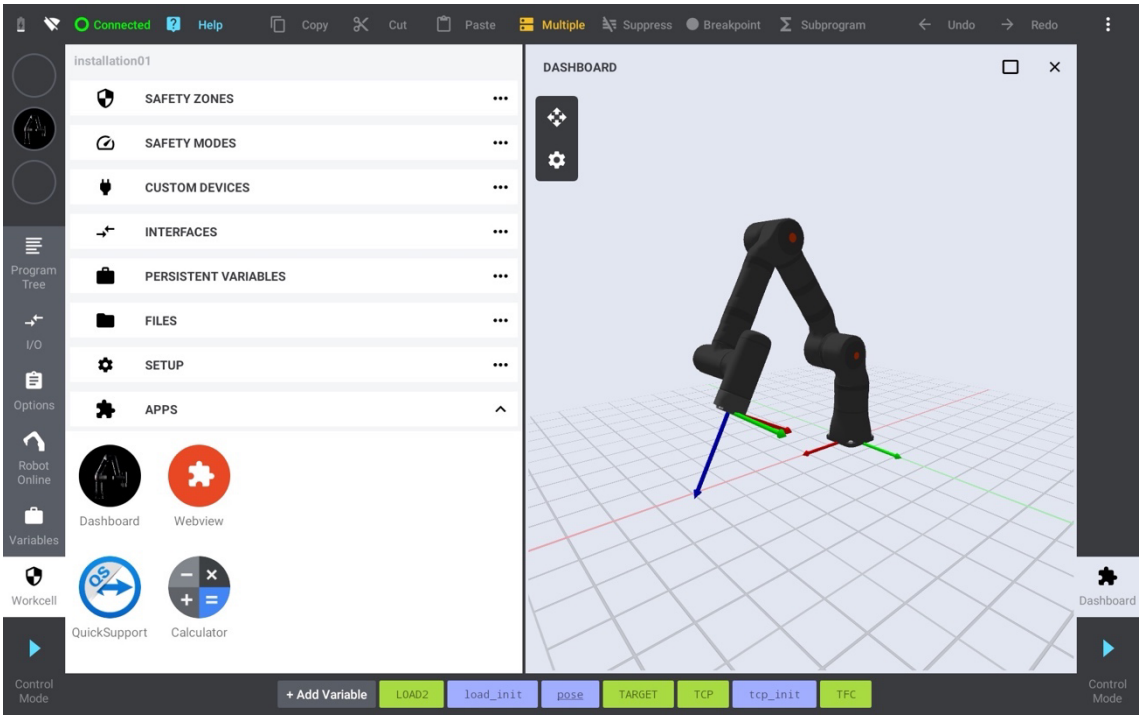
8.8 Apps

The Apps Workcell section allows you to launch installed CbunX applications as well as whitelisted Android applications. All CbunX applications are listed in the first row, whereas the Android applications are listed in the second row.



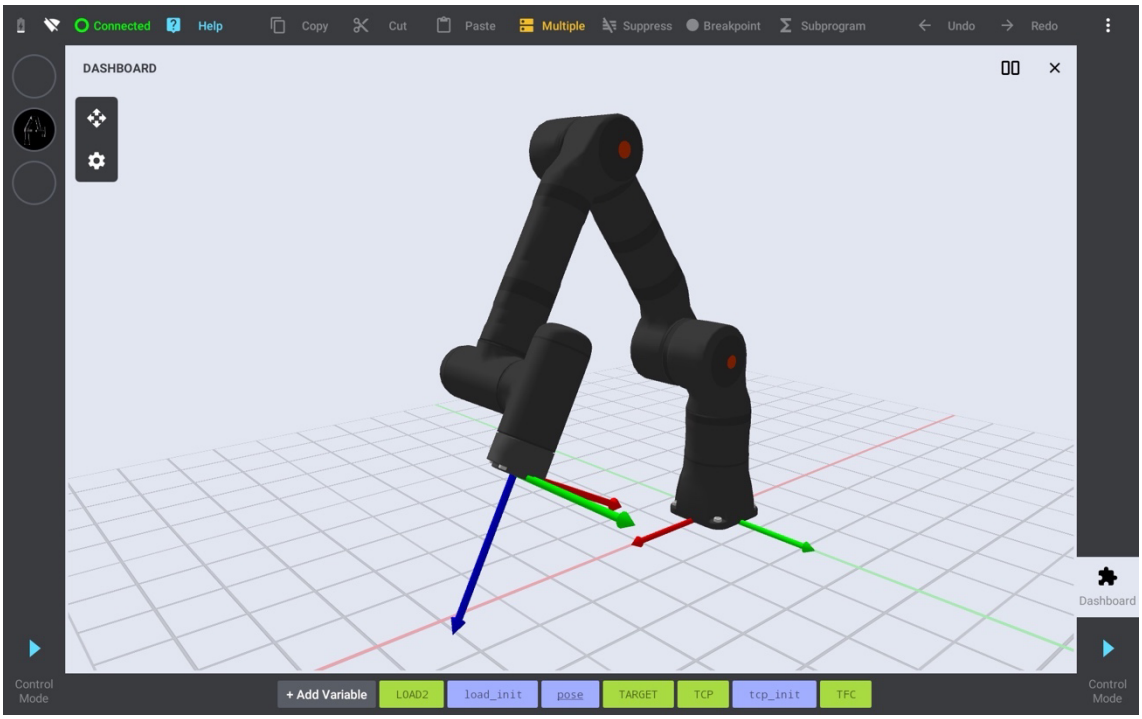
8.8.1 CbunX Application

A CbunX application is a Kassow Robots specific application, that is fully integrated into the Teach Pendant's control app. A CbunX application is automatically added to the list as soon as the relevant Cbun is installed. Once this Cbun is uninstalled, the CbunX application is removed too.



Clicking the icon launches the CbunX application in the opposite side of the screen. Once the CbunX application is open, it can be closed by clicking the close button.

There are 2 CbunX application launch modes: split-screen and fullscreen. Some CbunX applications may allow dynamic switching between these 2 modes.



8.8.2 Android Application

An Android application is any standard Android application, that was whitelisted by the Teach Pendant's control app. If Android application is installed on the Teach Pendant, it can be added to the whitelist from *Settings->Advanced->Single App*.

Clicking the icon launches the Android application in the new independent screen. Once the Android application is open, it can be closed by clicking the back navigation button (or using the back navigation gesture).

9 Software Extensions (Cbuns)

The Kassow Software provide its own extensions technology, allowing the high degree of variability based on C++ interfaces and dynamic libraries. Check the list of RC pre-installed modules in Appendix D [– List of extensions (Cbuns)].

The Cbun (Capability Bundles) module represents the KR framework item, encapsulating the application, system or peripheral extensions. The set of basic extensions are now maintained by KR, but integrators and device manufacturers are invited to implement and provide support for their own devices and applications.

The basic Cbuns control interface is located at the *Cbun Manager* [Settings/Cbuns], where the user review and adjust the set of loaded Cbun modules. The Cbun manager allows the basic package functionalities:

- Load Cbun module [1]
- Unload Cbun module [2]
- Add the Cbun instance into *Workcell* [3]

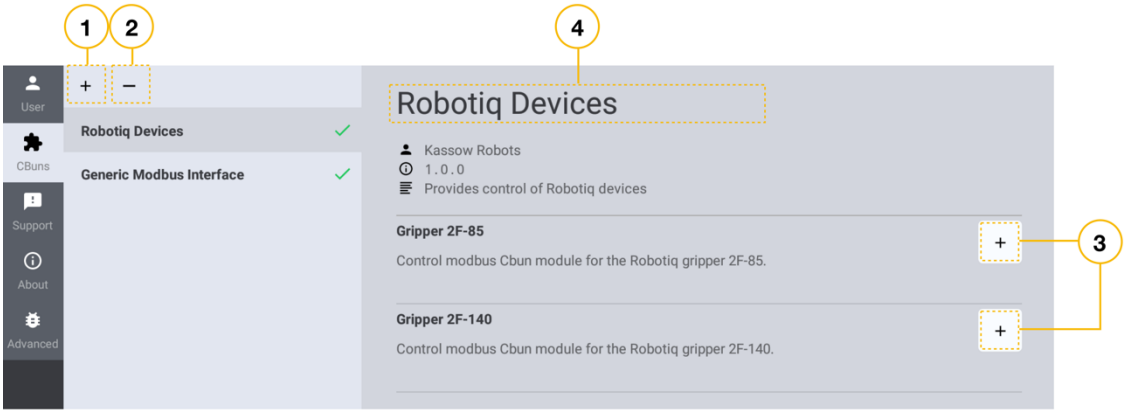


Figure 10. Cbuns Manager view

9.1 Cbun Installation

The present KSW supports the installation of the Cbun from three kinds of sources.

- Cbuns coming with the core software package (Cbuns maintained by Kassow Robots).
- Installation of the package located at the USB

- Compilation and installation from the custom Cbun source code.

In this manual we describe only the first 2 options. The custom Cbun development and compilation is beyond the scope of this handbook.

9.1.1 Installation from the KSW packages

When the installation dialog is opened, the default source is pointing to **Robot** folder [5]. That's the part of the KSW core software package containing the set of Cbuns maintained by Kassow Robots.

Choose desired Cbun, verify the version and click “Install” to proceed with the installation.

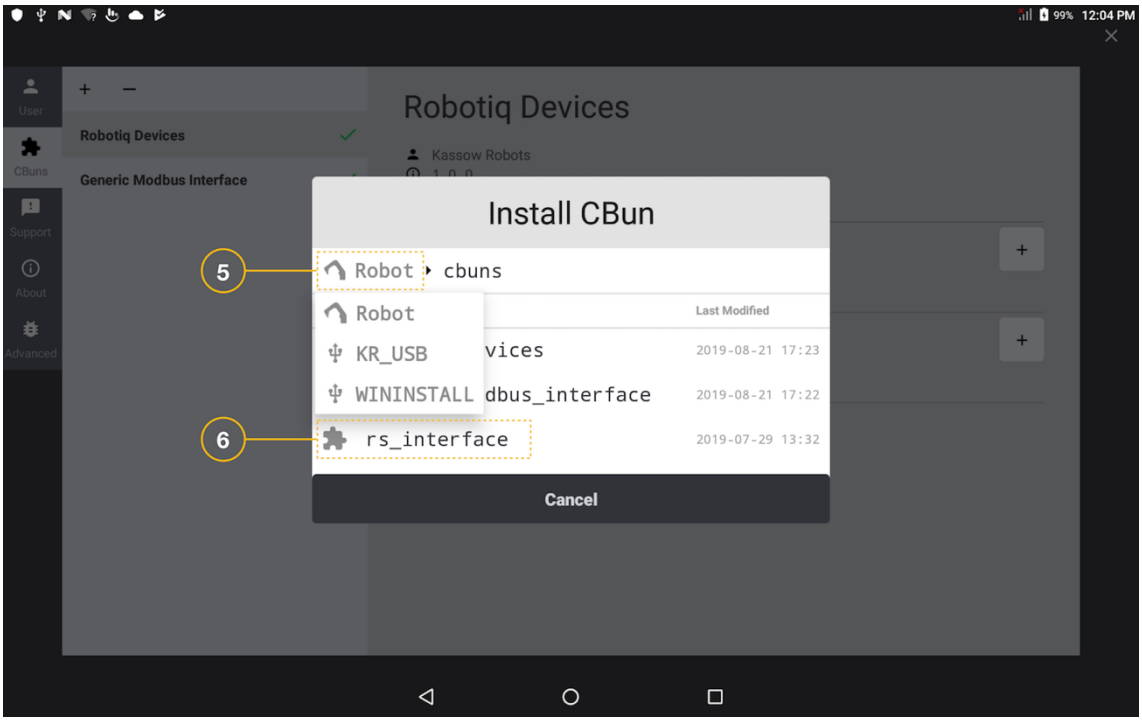


Figure 11. Cbun installation dialog

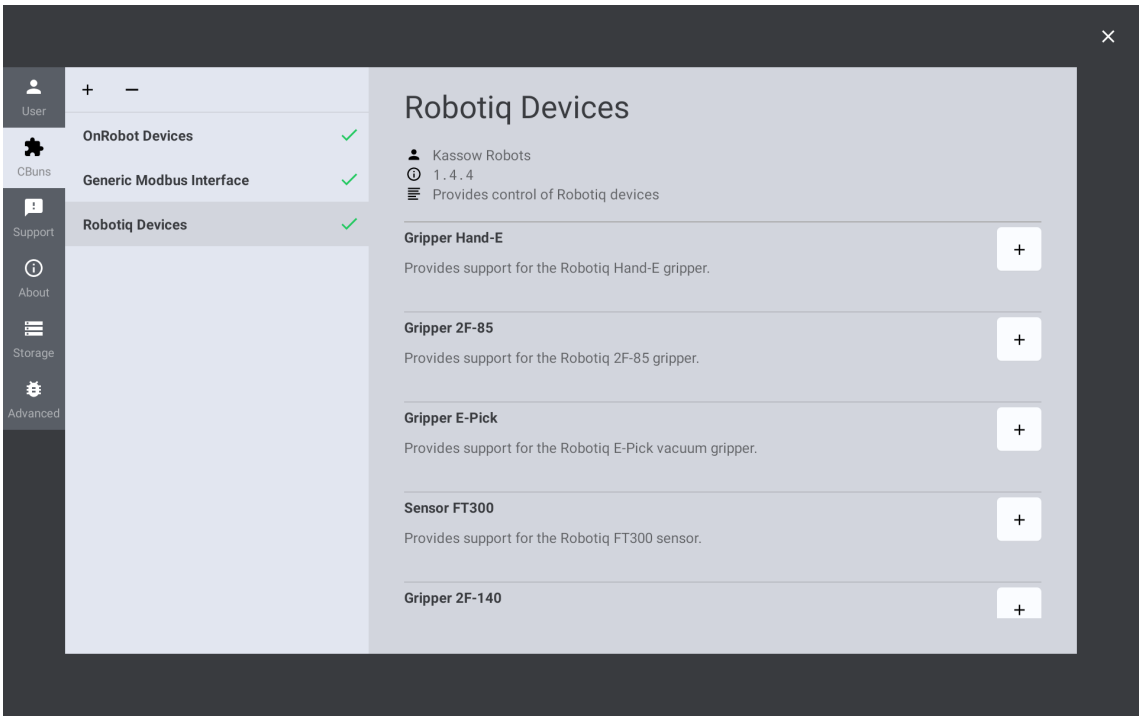
9.1.2 Installation from USB

1) Put the Cbun file (*.Cbun) on to the USB flash disk.
2) Plug the USB flash with the Cbun into RC.
3) Open Cbuns Manager and click the "+" button in the top left corner [1].
4) At the install dialog chose the USB as install source.
5) Select the Cbun for installation [5].
6) Confirm the installation by clicking the "Install" button.

9.2 Device

The Cbun device serves as a bridge between the robot and its peripheral devices such as grippers, welding machines, cameras and others. The device element encapsulates communication with the equipment and manages the associated physical device within the Cbun module.

Although many devices can be connected without a proprietary Cbun, this abstraction empowers the robot programmer to efficiently control and monitor the robot peripherals without the necessity to implement intricate communication logic.



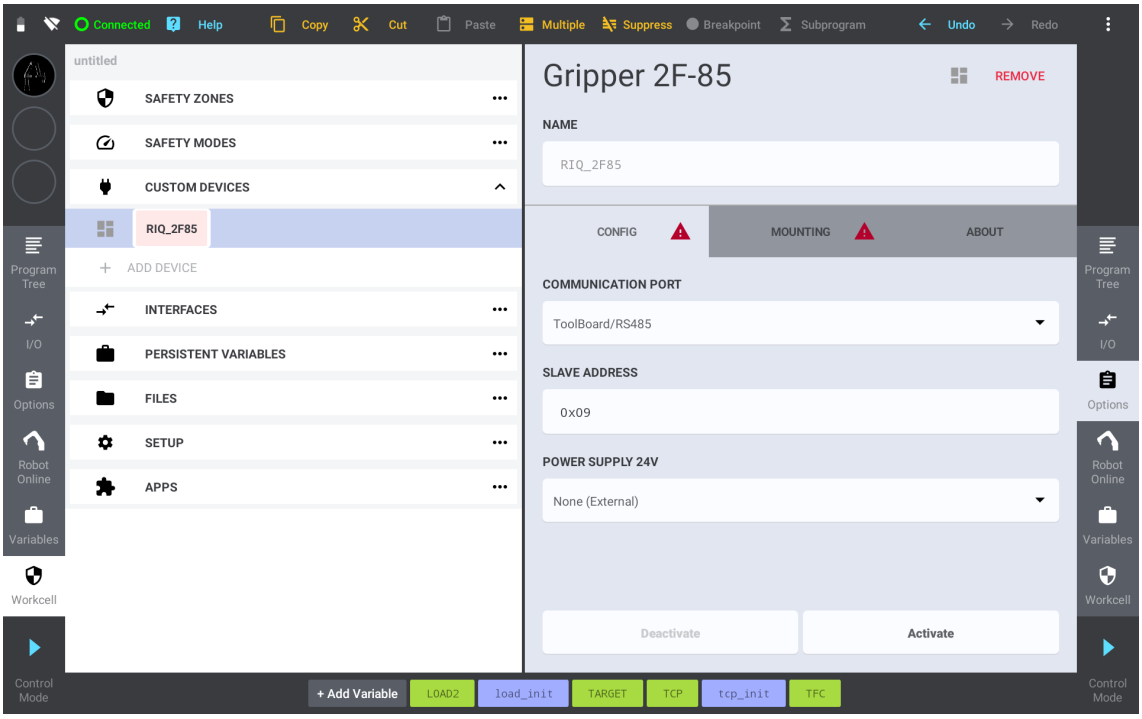
To add a device, install the corresponding Cbun and click on the **+** button of the selected device to create its instance. It is important to note that a single device instance refers to a specific individual device. For instance, if two identical grippers are attached to the robot, the user must create two instances of the gripper device. Each instance is responsible for controlling and monitoring the associated gripper.

All device instances are listed in Custom Devices section of the Workcell tool. To access the configuration of a device, select its instance and navigate to the Options tool.

9.2.1 Activation

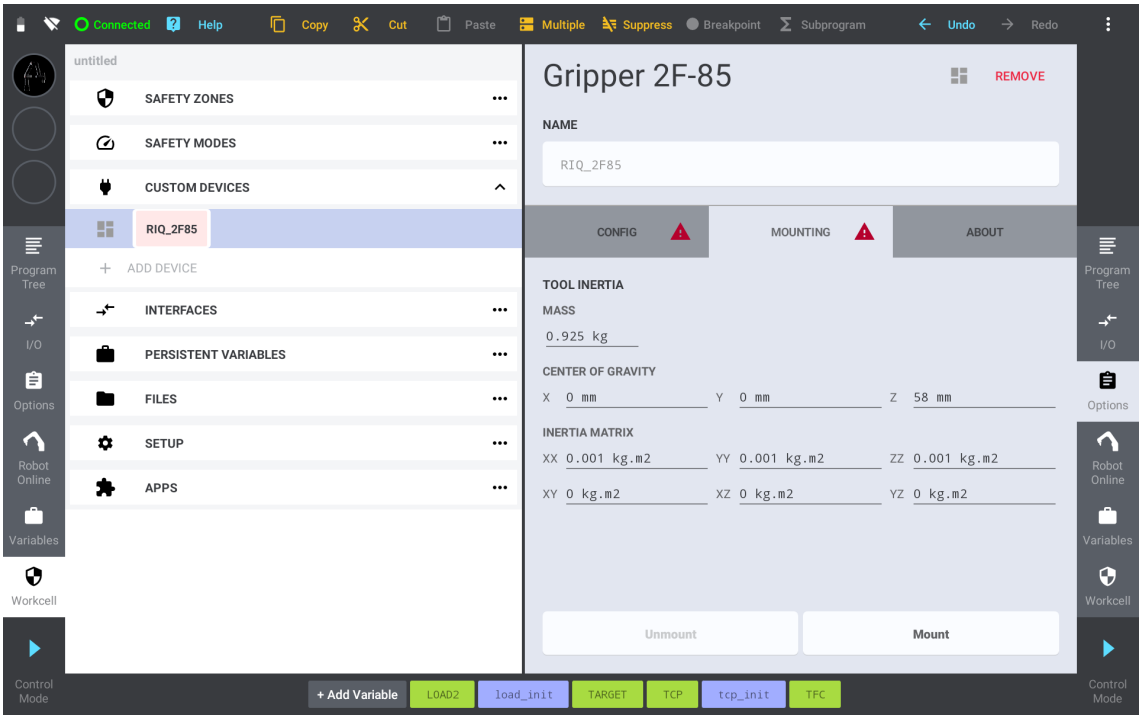
The **Config** tab enables the robot programmer to activate the device instance. The device activation is typically responsible for providing power supply and establishing communication. The specific set of activation parameters may vary for different devices.

If the device is not activated, it's methods and functions are invalid. Please note, that the device instance needs to be activated prior the execution of a robot program that involves the instance of the device.



9.2.2 Mounting

The **Mounting** tab offers and interface for automatic configuration of robot's TCP (Tool Center Point) and tool load. This feature is particularly valuable for devices mounted to the robot's tool flange. If needed, the robot programmer has the option to adjust the default mounting parameters.



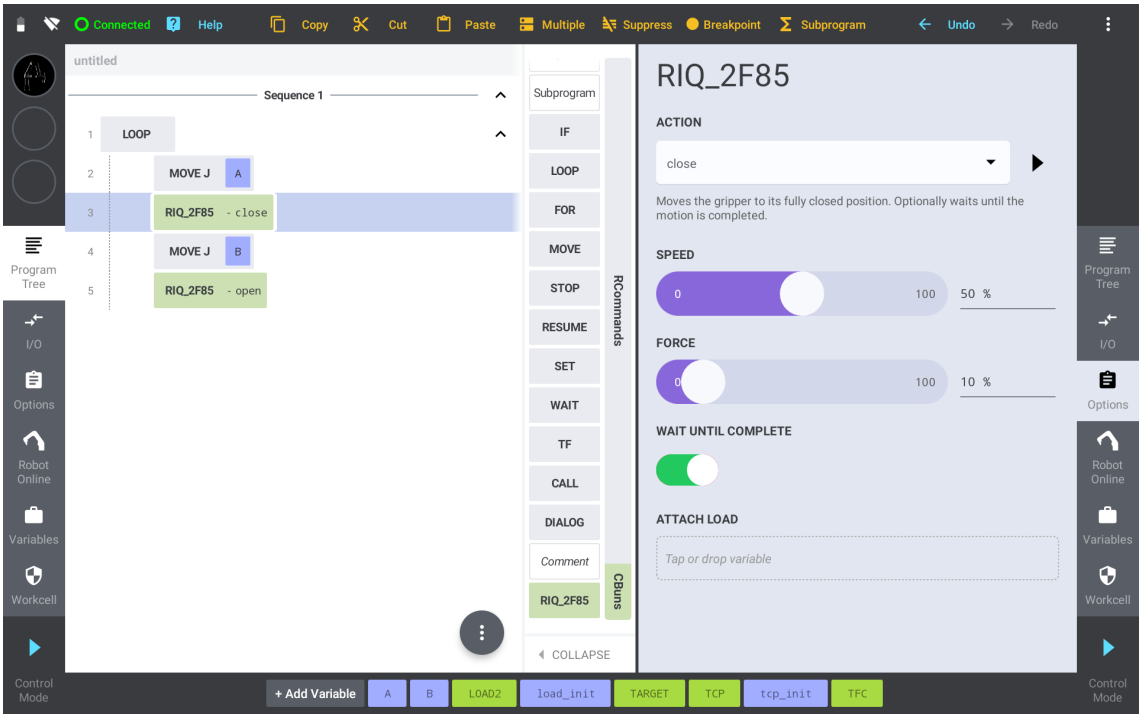
The device mounting feature is optional and some Cbun developer may choose not to implement it., especially if the device is not physically attached to the robot. In such cases, the Mounting tab will not be accessible.

9.2.3 Methods

Each Cbun device provides one or more methods (denoted as Actions) that allow the control of the robot's peripheral from a robot program. For instance, a gripper device may offer one methods for gripping and releasing an object.

To add a device method to the program, drag the device instance (*Workcell -> Custom Devices*) or its shortcut (Command Box) and drop it to the desired position within the Program Tree. Navigate to command's options panel and select the requested action.

The specific list of the method parameters may vary for each action. It is crucial for the robot programmer to define all non-optional parameters, otherwise, the device command will be invalid and prevent the launch the robot program.

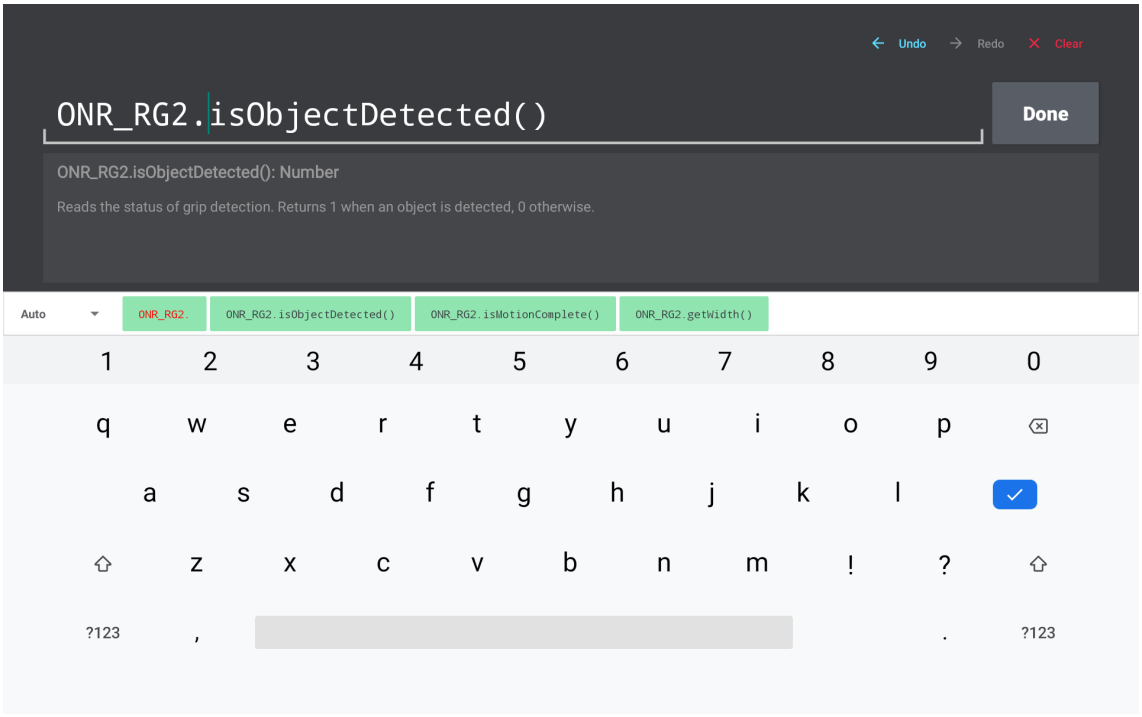


If supported, the robot programmer can test the method functionality by clicking the **play** button.

9.2.4 Functions

The Cbun device may offer functions that allow obtaining data from the device within a command expression. Unlike device methods, functions always return a value.

To use a device function within the Expression Builder, enter the device instance name and utilize dot notation. All available functions are listed in the hint bar. To use the function, simply click the corresponding hint item. If needed, ensure to define the function's parameters.

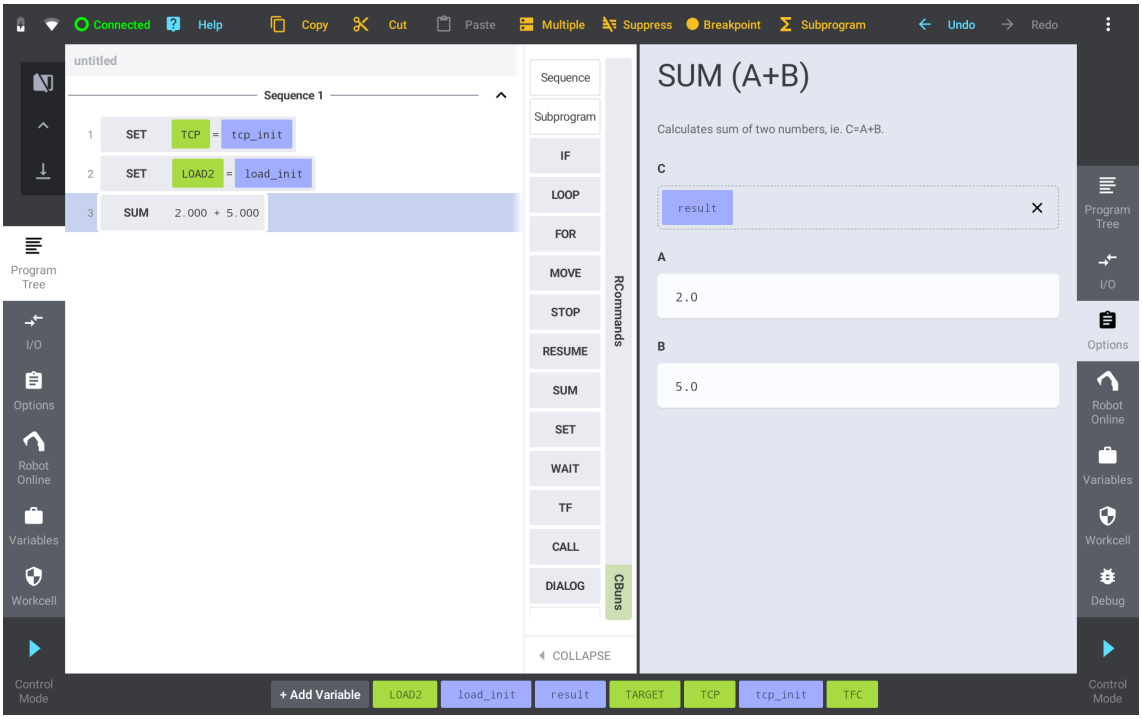


9.2.5 Dashboard

Some custom devices may provide optional dashboard for online device configuration, control and monitoring. Clicking the dashboard icon (if enabled) launches the device dashboard on the opposite side of the screen. Once the dashboard is open, it can be closed by clicking the close button.

9.3 Command

The Cbun module may offer additional Program Tree commands, extending the basic functionality of the robot programming interface. These Cbun commands are automatically listed within the Command Box when the Cbun is installed. The robot programmer can drag a Cbun command and drop it into a desired position within the Program Tree.



To configure command's parameters, select the command and navigate to the Options tool. The specific list of the command parameters may vary for each command. It is crucial for the robot programmer to define all non-optional parameters, otherwise, the command will be invalid and prevent the launch the robot program.

9.4 Application

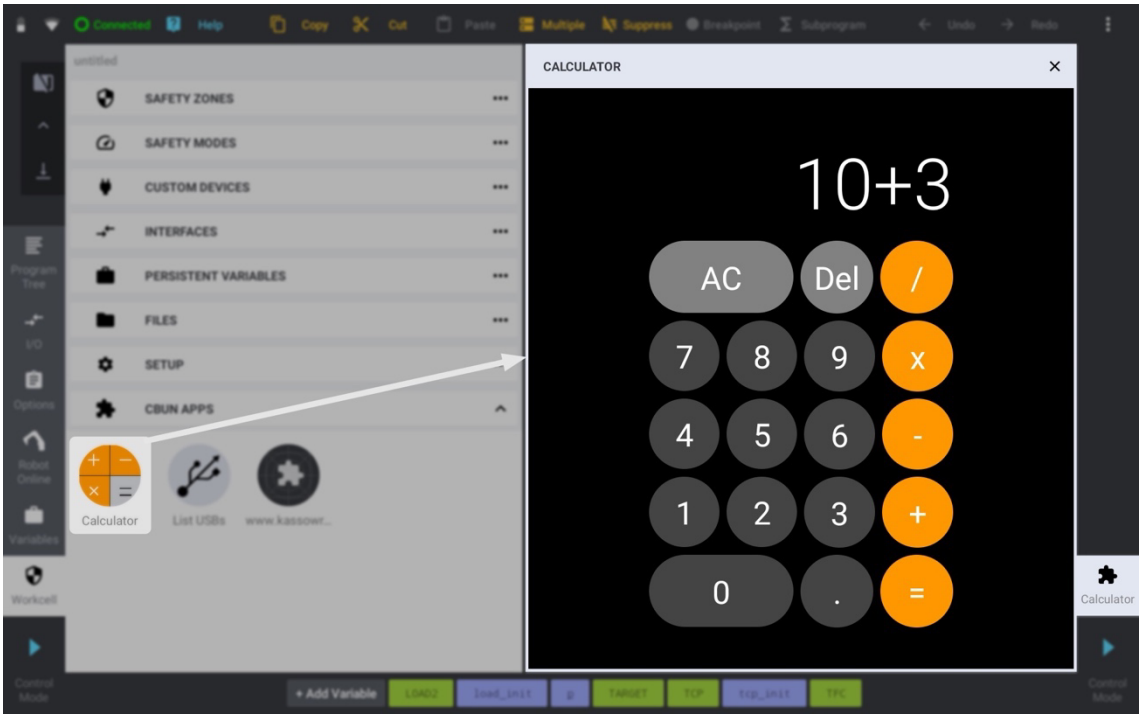
The Cbun application is a compact software program to run within a Teach Pendant's user interface. It encompasses various functionalities, offering users with a range of features to enhance the efficiency of robot programming, configuration and overall usage.

If an installed Cbun module includes applications, they become automatically accessible within the Teach Pendant's control app.

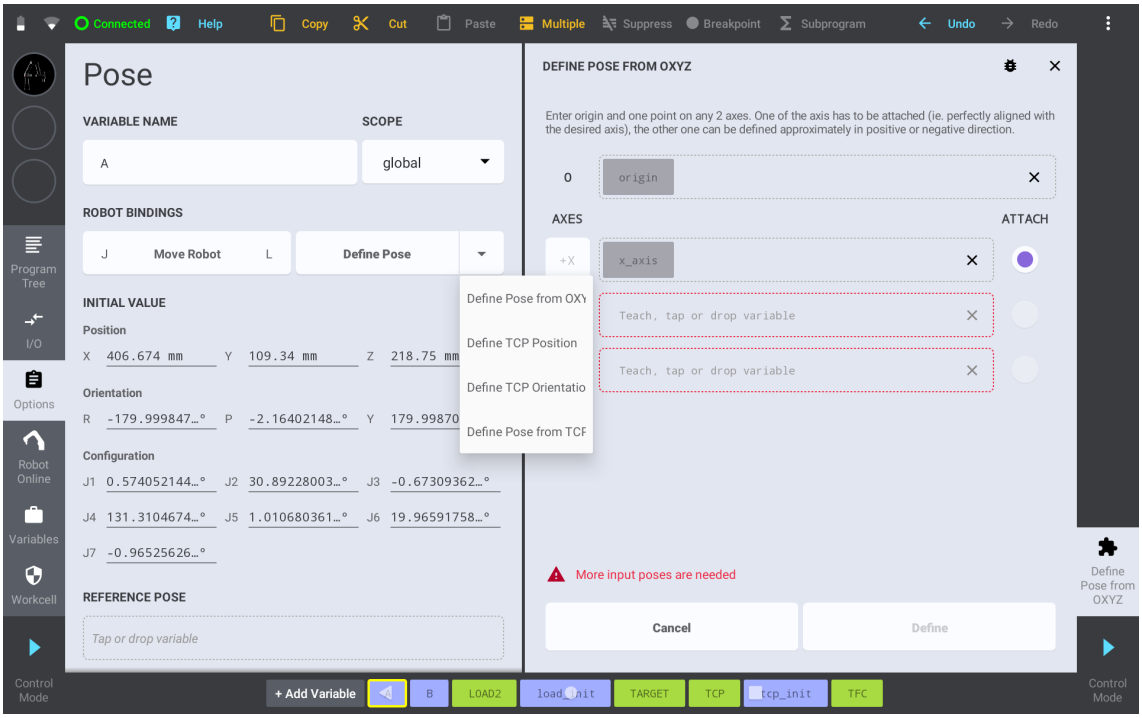
There are two types of Cbun applications:

- Standalone application
- Tool application

All standalone applications, such as calculator, web browser or robot 3D visualization, are listed in the Apps section of the Workcell tool. The user can open the application by clicking its icon.



The tool applications are designed to fulfil specific user intents. For instance, a Cbun module may provide additional tools for defining the coordinates of a pose variable.

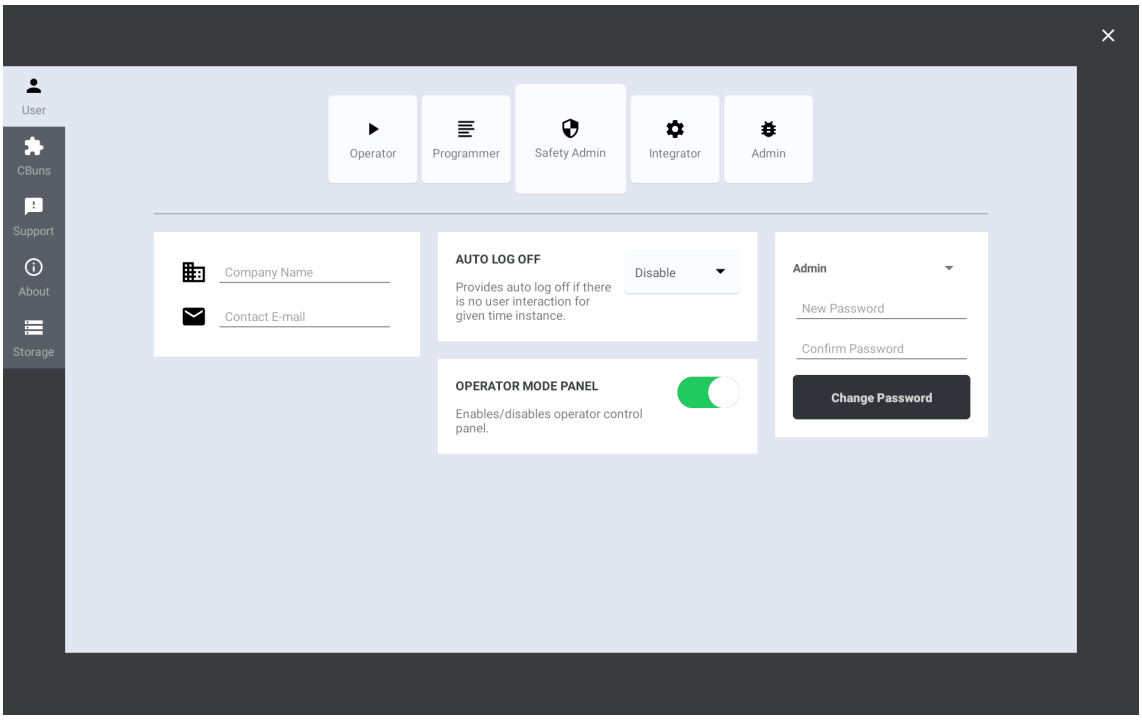


10 Settings

Kassow Robots settings provide users with the flexibility to fine-tune the robot system to meet their specific needs and preferences. The settings screen, accessible through the Teach Pendant's hamburger menu, allows to switch user profiles, manage Cbuns, send robot logs, update a robot software and much more.

10.1 User

The User settings section serves as an interface for switching among user profiles. With five available user profiles, it enables efficient management of access to diverse robot features and settings. The user profiles are inclusive, indicating that a higher privileged profile encompasses access to the features of any lower privileged profile.



The User section enables the definition of a company name and contact e-mail. This information proves valuable when sending data logs, facilitating Kassow Robots technical support in accurately assigning received logs to a specific company and e-mail account. In addition, the User section provides interface for configuration of Operator Mode Panel [3.10, page 35] with it's Auto Log Off feature.

10.1.1 Password

All users, except Operator, are password-protected. The default password for **Programmer**, **Safety Admin** and **Integrator** is “kassow”. For the **Admin** password, users can obtain it from Kassow Robots technical support upon request. Furthermore, each user can change their own password and the password of users with lower privileges.

10.1.2 Privileges

In the following table you can find the overview of supported profiles and list of features accessible to each of them.

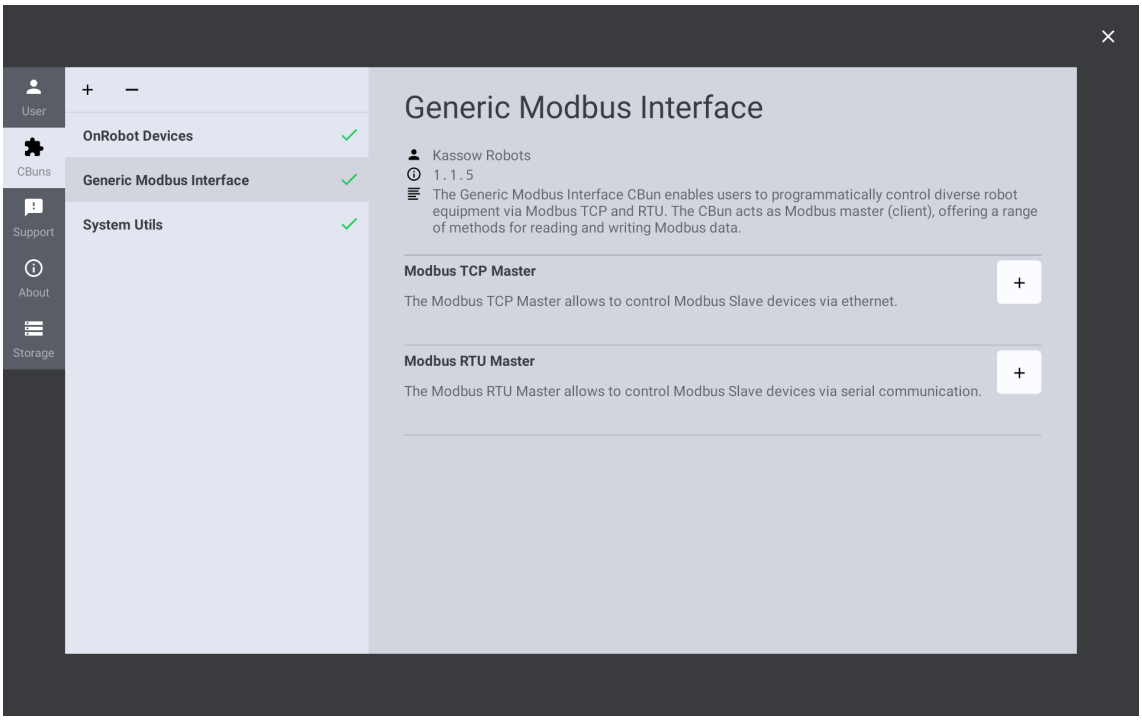
User Profile	Allowed features
OPERATOR Can only start and stop programs.	• robot start • program start/stop
PROGRAMMER The low access profile. Allowed to load or modify the program. No safety or the workcell can be provided by this user.	• copy/cut/paste on commands/variables • command suppress option • undo/redo • modify program (add, remove, move commands, change params) • modify variables (add, remove variables, change params) • set breakpoint • convert to subprogram • program management (new/open/save/delete) • pause program • enable/disable breakpoints • update system • send logs • control I/O • install/uninstall Cbuns • add/remove Cbun devices • configure interfaces (ethnet, profinet)
SAFETY ADMIN Safety functions and the workcell settings can be modified by this user.	• copy/cut/paste on safety zones, (custom devices, persistency variables) • suppress on safety zones, (custom devices)?!? • import workcell • workcell management (new/open/save/delete) • set safety mode, master speed • modify safety zones (add, remove, change params) • modify custom devices, persistent variables (add, remove, change params) • undo/redo
INTEGRATOR All documented functions of the system	• remove workcell files • set initial robot configuration

are available for this profile.

ADMIN Can access all the system features with no limits.	• advanced settings access • enable maintenance mode • firmware update

10.2 Cbuns

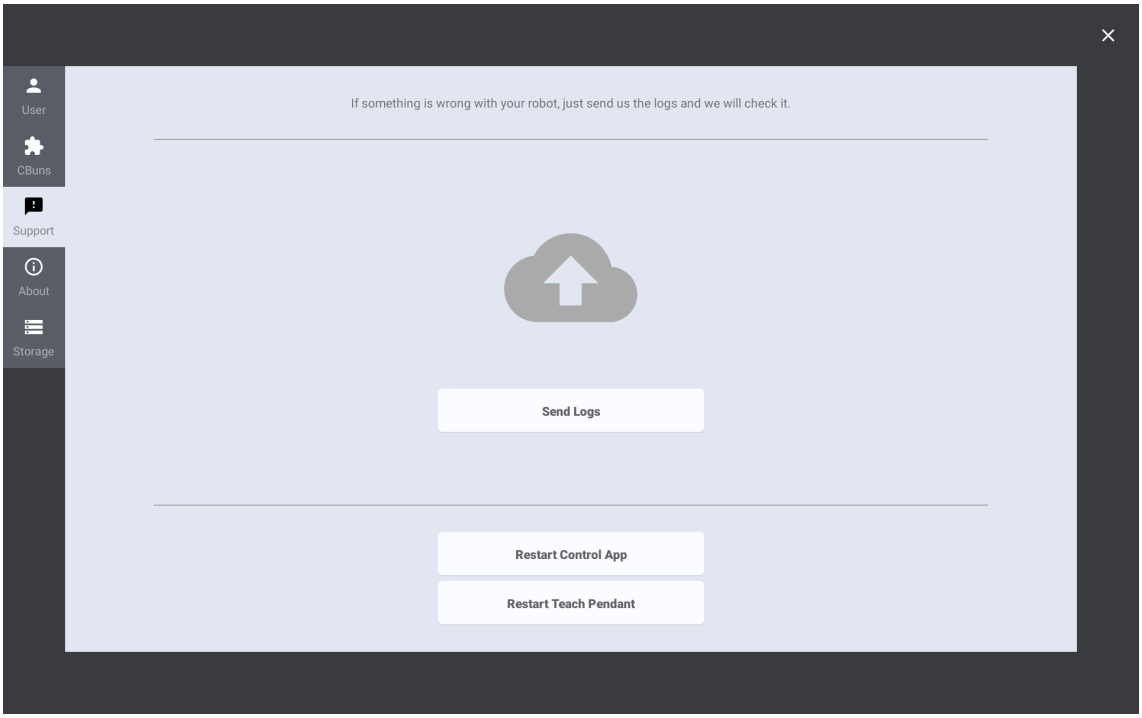
The Cbuns section serves as the interface for Cbun management, granting robot programmers the ability to install and uninstall Cbuns from the robot's built-in Cbun repository, a USB drive, or Google Drive. Furthemore, the Cbuns section allows users to create instances of Cbun devices. For more details refer to Software Extensions (Cbuns) chapter [Ch. 9, page 110].



10.3 Support

The Support tab primarily allows robot programmers to collect and send robot logs to Kassow Robots technical support via internet. This plays a crucial role for investigation of any unexpected behaviour and robot system debugging.

For the sake of efficient robot trouble shooting and problem identification, the RC keeps a massive record of various kind of data (not longer than recent 2 minutes of the recent robot operations). Those data (datalogs) can be easily delivered to the safe and dedicated cloud storage.



10.3.1 Send Logs

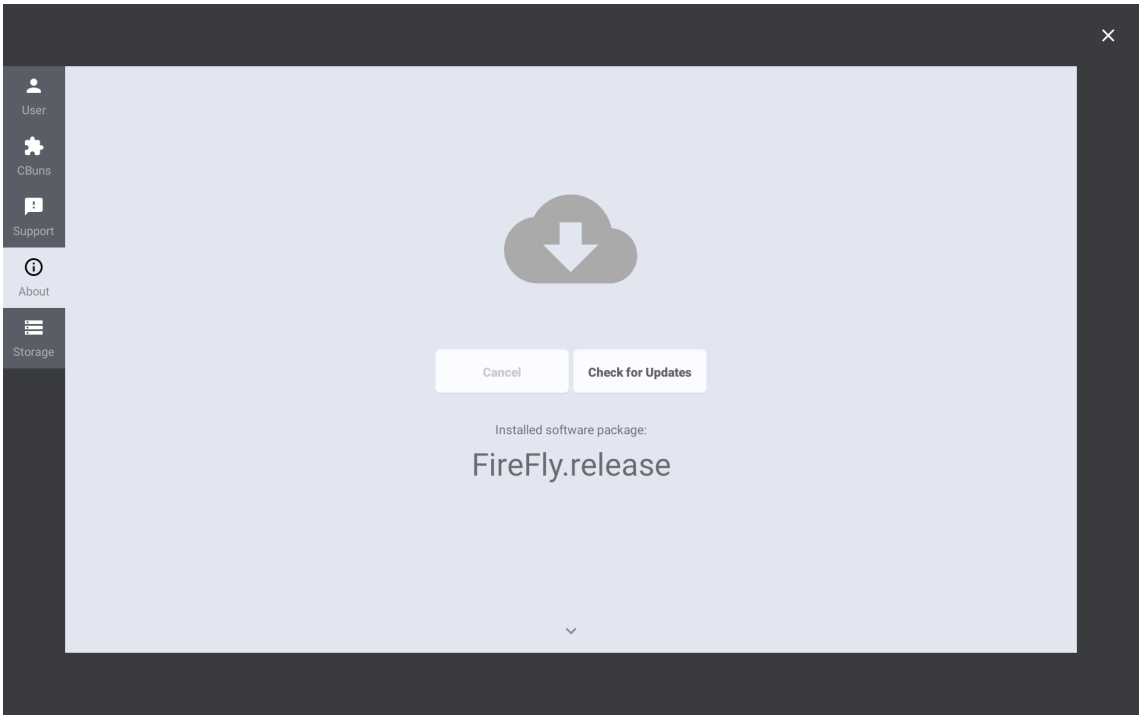
While the data extraction procedure is not complicated, it is crucial to follow these steps accurately to shared meaningful data with Kassow Robots technical support team.

1. Check the robot is not moving (pause or terminate the running program).
2. Press the E-Stop button.
3. Wait until the progress bar ceases (the data are being extracted in this phase).
4. Navigate to *Settings -> Support* and click on the **Send Logs** button.
5. Select a Wi-Fi to get an internet access and click on the **Continue** button.
6. Choose data logs to be send, typically the latest ones, and click on the **Next** button.
7. Roughly describe the issue and click on the **Collect and Send** button.
8. Wait until the data are successfully uploaded.

Please note that our team is not automatically notified when the new datalogs are uploaded. Always contact the Kassow Robots support to get assured your issue was reported and is scheduled for examination.

10.4 About

The About section provides detailed info about installed robot software package, including its label, versions of individual software components and changelog. Furthermore, the About section enables users to update the software package via internet from Kassow Robots cloud.



10.4.1 Check for Updates

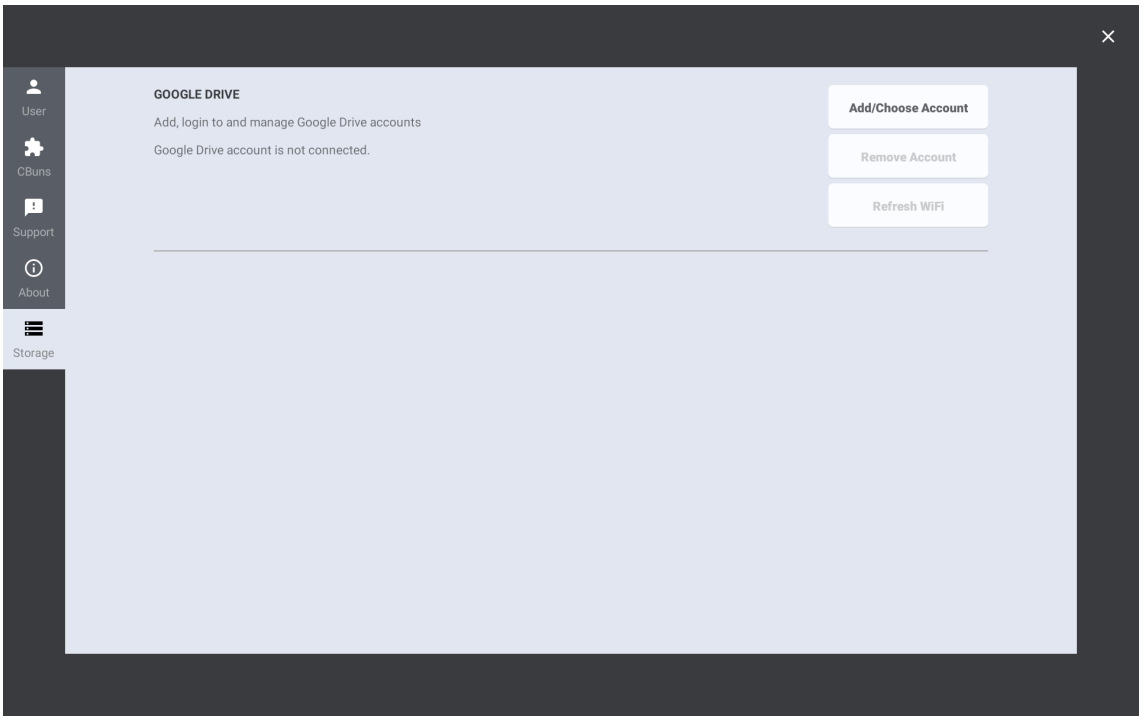
To update the robot RC software, proceed as follows:

1. Click on the **Check for Updates** button.
2. Select a Wi-Fi to get an internet access and click on the **Continue** button.
3. Check the software package details and click on the **Download and Update** button.
4. Wait until robot is updates. This could take approx. 5-10 minutes to complete.
5. Restart the robot by turning it off and on.

The RC software update doesn't involve the firmware update automatically. If the robotic arm hardware or the cabinet IO requires the firmware upgrade, please follow the instructions from the maintenance section later in this chapter.

10.5 Storage

The Storage tab serves as an interface for Google Drive configuration, allowing robot programmers to add or remove their Google account. The Google Drive account allows to save Programs and Workcells to your Google Drive for later access. Additionally, it can be used for for installing Cbuns.



10.5.1 Google Drive

To add your Google Drive account, proceed as follows:

1. Click on the **Add/Choose Account** button.
2. Select a Wi-Fi to get an internet access and click on the **Continue** button.
3. Sign in with your Google account.
4. Set a lock screen (without Secure start-up), if required by your account's policy.
5. Choose the newly added account to continue to Boson.

To remove a Google Drive account, follow these steps:

1. Click on the **Remove Account** button.
2. Choose an account to be removed and follow the remove account wizard.
3. Remove screen lock (if enabled) and return to the control app.

10.6 Advanced

The Advanced section serves as an interface for expert configuration of your robot system. This advanced setting is accessible only by the **Admin** user. Modifications to these advanced settings should be made upo specific request from Kassow Robots technical support. It is crucial to emphasize that altering the advanced configuration without proper knowledge can lead to permanent damage to your robot.

Section	Description
ADVANCED CONFIGURATION	Allows to export and import the RC and robot configuration set (cfg file). This is useful when some uncommon system configurations are required. The USB stick is only medium supporting import/export of the cfg file. It must be plugged into the RC USB outlet.
JOGGING MASTER SPEED	Enables the option to disable the master speed effect on robot jogging.
USER INTERFACE	Allows to specify the command size and to disable program line numbers visibility.
SINGLE APP	The single app mode is the default setting which doesn't allow the TP app exit back into the android system or provide the switch into any other application. Additionally, the Single App section allows to select whitelisted Android apps. All Whitelisted Android apps are accessible via *Workcell -> Apps* section.
AUTOMATIC WIFI CONNECTION	By default, the Wi-Fi connection is enabled only if it is needed, for instance during the robot update or send logs process, otherwise it is automatically disabled. Disabling this feature allows the user to manually control the Wi-Fi connectivity.
DEBUG MODE	This option allows to access several tools which are handful for some system and software analysis.
DEVICE OWNER	The system default setting. Don't disable the option until instructed by the Kassow Robots support team.
TABLET SCREEN LOCK	The system default setting. Don't disable the option until instructed by the Kassow Robots support team.
ANDROID DEBUG BRIDGE	The system default setting. Don't disable the option until instructed by the Kassow Robots support team.
AUTO SHUTDOWN	The system default setting. Don't disable the option until instructed by the Kassow Robots support team.
TABLET LOGS	Explicit request to send the application crash data into the cloud (useful in cases of failure or application crashes).
ROBOT CONFIGURATION	The robot model configuration. Please consult with the Kassow Robots support team before changing the predefined settings.
ROBOT CONTROLLER	The maintenance and diagnostic tools.

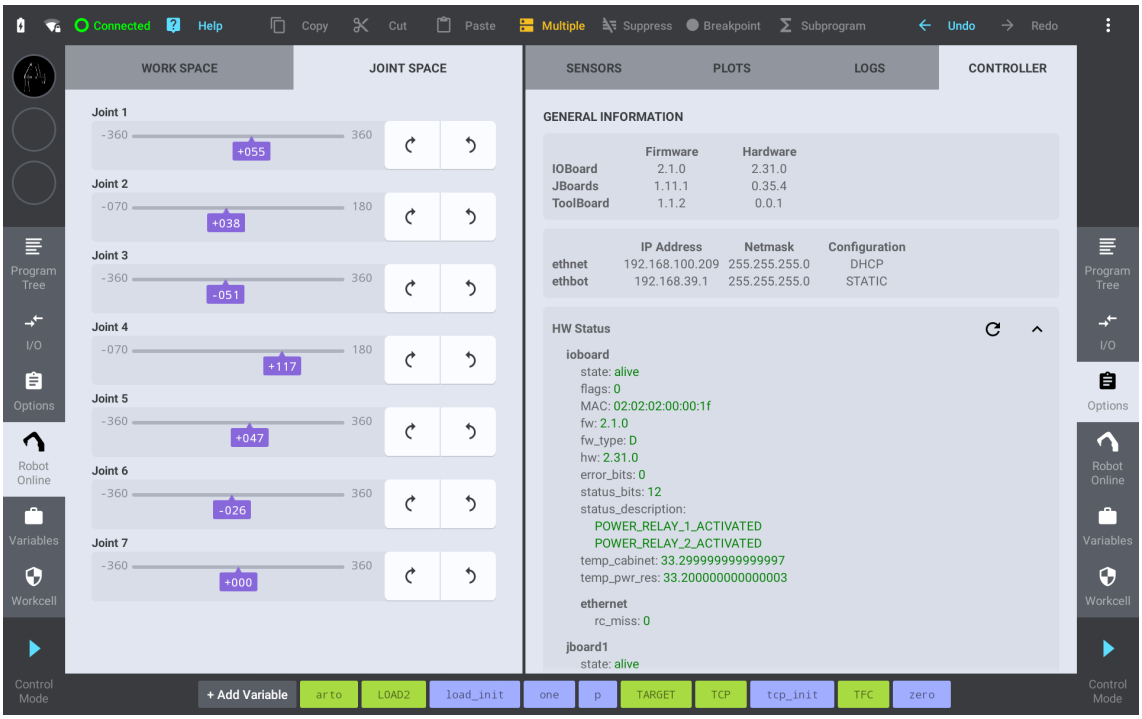
LANGUAGE	Allows the robot user to specify language localisation: • English • German • Danish • Spanish • Czech • Chinese • Japanese • Hungarian *(Please note, localisation applies only to UI elements, not to system messages or embedded documentation.)*

11 Maintenance

The Maintenance pan allows robot users to conduct comprehensive robot diagnostics and perform maintenance task, including firmware updates, robot zeroing or partial system update. Accessing the robot maintenance tool can be achieve by long-clicking on the **Robot Online** tool tab or navigating to *Settings -> Advanced* and clicking on the **Controller...** button.

11.1 General Information

The General Information sections serve as an interface for robot diagnostics, providing detailed info about robot hardware and firmware versions. Additionally, it displays the latest know configuration of robot's ethernet interfaces.



The **HW Status** section can be populated with the actual data extracted from the RC. Use the refresh button to update the data. In some case this tool can be useful on efficient inspection of the hardware issues, related to IO, Joint Boards or Tool IO.

The **HW Datalogs** section serves as a shortcut to *Maintenance -> Logs* tab, providing users with direct access to download and process critical robot logs on the Teach Pendant screen. This proves crucial in investigating and resolving robot arm-related issues.

11.2 Factory Poses

The Factory Poses sections enable the user to jog the robot arm to one of the predefined factory poses, offering instant access to essential robot joint configurations.

11.2.1 Move to Zero Pose

Holding the **Move to Zero Pose** button jogs the robot to the zero pose, where all joint angles are zero. This functionality is crucial for various maintenance tasks, including firmware update or robot zeroing.

11.2.2 Move to Transport Pose

The **Move to Transport Pose** button enables users to fold the robot to a transport pose, ensuring that the robot arm fits into its transport package box. To jog the robot to the transport pose, simply click and hold the button and wait until the robot reaches the desired pose.

11.3 Maintenance Mode

The maintenance mode refers to a special mode of your robot system, allowing to perform various maintenance tasks such as firmware update and calibration data download. The user must enable the maintenance mode prior running these tasks.

To enter maintenance mode, proceed with the following steps:

1. Terminate both program execution and robot movement.
2. Press and release E-Stop button, but do not initialize the robot arm.
3. Navigate to Maintenance Mode section and click on the **Enable** button.
4. Log in as **Admin**.
5. Wait until the toggle starts blinking (cyan).
6. Click the Toggle button to enter maintenance mode.
7. Wait until the robot switches into maintenance mode (toggle button is purple).

To exit maintenance mode, follow these steps:

1. Navigate to Maintenance Mode section and click on the **Disable** button.
2. Click the toggle button to initialize the robot.

11.4 Firmware Update

The firmware update allows to improve and extend functionalities of the embedded/electronic parts of the robot (namely the IO Board, Joint Boards and Tool IO).

Here we us the firmware term to refer to any software running by all the robot hardware parts (meaning IO Board, Joints and Tool IO). This bit of software keeps all electronic parts alive, providing the proper control and communication on each active part of the robot.

As mentioned earlier, the firmware update is not a part of the standard RC software update routine. Typically, it might be recommended by the Kassow team after some significant improvements on hardware control or the communication level and in some accordance with the RC software version.

The firmware packages are stored using the **.fwu* extension and can deliver the simple joint or even the whole set of updates for each part of the robot.

Procedure

The regular procedure for the firmware update consists of following steps:

1. **Choose the media source**
   You can install the *fwu* package from the Google drive or the USB storage. For the first option make sure your Google account is set up properly [38]. To go with the USB put the package onto the USB stick and plug it into the RC. But don't initialize the robot (don't use the blue blinking toggle button).

2. **Restart the RC**
   But don't initialize the robot (don't use the blue blinking toggle button).

3. **Enter the Maintenance Mode**
   Engages the mode used for firmware updates and readings [page 127]

4. **Start the update**
   Press the **Update** button located within the **Firmware Update** section and enter the admin password.

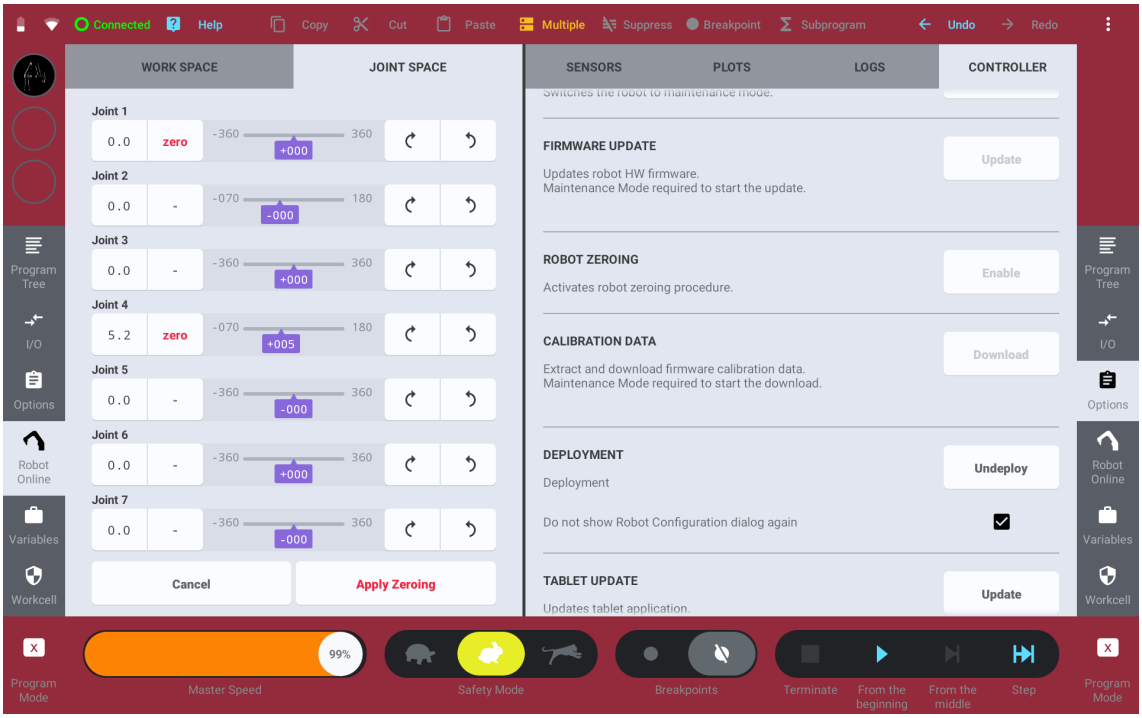
5. **Locate and choose the package**
   Choose the media source [38]. Then navigate to the intended fwu package, click and confirm with *"Proceed"*.

6. **The firmware update progress**
   Wait until the update process was successfully finished. If there was a failure, please contact the Kassow team and share with us any necessary information (RC software version, fwu package). Eventually extract and upload the datalogs.

7. **Disable the Maintenance Mode**
   Return back to regular operations [page 127].

11.5 Robot Zeroing

The robot zeroing procedure allows rough robot zeroing after exchange of one or more joints, allowing to specify zero angle of new joint/s. The robot zeroing procedure should be used only upon a specific request from Kassow Robots technical support.



To perform robot zeroing, proceed with following steps:

1. Navigate to Robot Zeroing section and click on the **Enable** button.
2. Log in as **Admin**.
3. Move the desired joint to zero angle and click on the **dash (-)** button. Alternatively, if the joint angle is not reachable, measure the current joint angle, modify the corresponding value and click on the **dash (-)** button.
4. When all exchanged joints are zeroed, click on the **Apply Zeroing** button.
5. Verify the zeroing data and click on the **Upload** button to apply the zeroing.

Consult with Kassow Robots support team before you decide to do any changes in factory delivered zeroin changes!

Be aware that zeroing will (very probably) negatively affects robot accuracy, kinematic calibration and makes your previous poses and programs unaligned to intended robot positions.

11.6 Calibration Data

The Calibration Data section provides an interface for download of robot calibration data, allowing backup of robot's kinematic calibration and detailed diagnostics of robot arm kinematics.

To download calibration data, proceed with the following steps:

1. Plug in a USB drive or add a Google Drive account.
2. Enter the maintenance mode.
3. Select the devices to download calibration data from and click on the **Next** button.
4. Choose the target storage location and click on the **Download** button.
5. Wait until the calibration data are downloaded, this may take few minutes.
6. Exit the maintenance.

11.7 Deployment

The Deployment section allows robot administrators to deploy and un-deploy the Teach Pendant. Deployment is a prerequisite before delivering the Teach Pendant to the end customer. However, in certain situations, such as on-site diagnosis by Kassow Robots technician, temporary un-deploy may prove useful. This feature should be used only upon request from Kassow Robots technical support team.

11.8 Tablet Update

The Tablet Update section serves as an interface for updating and downgrading Teach Pendant software only. This is crucial when replacing the Teach Pendant or robot controller, enabling the robot programmer to align the Teach Pendant's software version with the robot controller's SW.

To change tablet SW version, proceed with the following steps:

1. Navigate to Tablet Update section and click on the **Update** button.
2. Log in as **Admin**.
3. Select a Wi-Fi to get an internet access and click on the **Continue** button.
4. Choose the corresponding software package (based on technical support recommendation).
5. Click on the **Download and Update** button.
6. Wait until the tablet is updated, this may take few minutes.

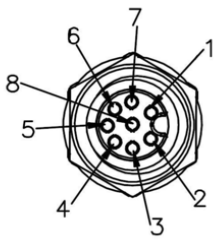
Appendix A - Tool IO

The Tool IO supports several hardware communication interfaces, in particular Analog, Digital/IO and RS-485. It also provides the necessary power supply and the back-drive button access for convenient teach functions. Digital inputs and outputs can be accessed directly from the UI or by variable mapping from within robot programs.

I/O (G2 robots with flat caps - from mid 2023)

Our robots produced since 2023 contain extended multipurpose digital IOs with the option to configure several as extra groundings in case of need to deliver higher power. Description of the hardware and how it routes to the UI is stated in the following charts.

M8 8-pole female connector

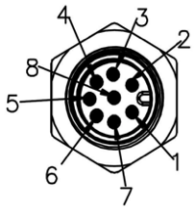


Connector 1, Tool IO M8 female, 8 poles

Digital IO		Pin no.	Configurations	Specification
TAIC05	IN	1	-	Current input 4-20mA in +
	IN	2	-	Current input 4-20mA in -
TAO01	OUT	3	4-20mA or 0-10V	Analog output 1 configurable current or voltage
TAO02	OUT	4	4-20mA or 0-10V	Analog output 2 configurable current or voltage
TAI01M	OUT	5	Digital Output Disconnected or 24V	Multi-Purpose In/Out 1
TDO01M	IN		Analog Input	

			Nominal 0-10V, max 30V	
TAI02M	OUT		Digital Output Disconnected or 24V	Multi-Purpose In/Out 2
TDO02M	IN	6	Analog Input Nominal 0-10V, max 30V	
	G		GND	or Ground = Chassis = Earth = 0V
TAI03M	OUT		Digital Output Disconnected or 24V	
TDO03M	IN	7	Analog Input Nominal 0-10V, max 30V	Multi-Purpose In/Out 3
-	-	8	GND	Ground = Chassis = Earth = 0V

M8 8-pole male connector



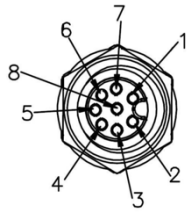
Connector 2, Tool IO male M8, 8 poles

Digital IO		Pin no.	Configurations	Specification
IO/API	IN	1	-	RS 485 + baud rate programmable up to 1Mbps
	IN	2	-	RS 485 – baud rate programmable up to 1Mbps
TAI04M	OUT		Digital Output Disconnected or 24V	
TDO04M	IN	3	Analog Input Nominal 0-10V, max 30V	Multi-Purpose In/Out 4

TAI05M	OUT	4	Digital Output Disconnected or 24V	Multi-Purpose In/Out 5
TDO05M	IN		Analog Input Nominal 0-10V, max 30V	
TAI06M	OUT	5	Digital Output Disconnected or 24V	Multi-Purpose In/Out 6
TDO06M	IN		Analog Input Nominal 0-10V, max 30V	
TAI07M	OUT	6	Digital Output Disconnected or 24V	Multi-Purpose In/Out 7 or Ground = Chassis = Earth = 0V
TDO07M	IN		Analog Input Nominal 0-10V, max 30V	
	G		GND	
TAI08M	OUT	7	Digital Output Disconnected or 24V	Multi-Purpose In/Out 8
TDO08M	IN		Analog Input Nominal 0-10V, max 30V	
-	-	8	-	Ground = Chassis = Earth = 0V

I/O (G1 with round caps - produced before 2023)

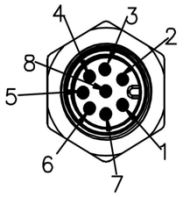
M8 8-pole female connector



Connector 1, Tool IO M8 female, 8 poles

Digital IO		Pin no.	Configurations	Specification
-	-	8	-	GND
TPSU01	OUT	5	12V or 24V	Power output 1, 12V or 24V +5% / -15% Max 700mA *
TDO01	OUT	7	12V or 24V	Digital output 1, 12V or 24V +5% / -15% Max 100mA *
TDO02	OUT	6	12V or 24V	Digital output 2, 12V or 24V +5% / -15% Max 100mA *
TAO01	OUT	4	4-20mA or 0-10V	Analog output 1, configurable current or voltage
TAO02	OUT	4	4-20mA or 0-10V	Analog output 2, configurable current or voltage
TAI05	IN	1	-	Analog input 1, 4-20mA in +, Max 12V relative to GND
	IN	2	-	Analog input 2, 4-20mA in -, Max 12V relative to GND

M8 8-pole male connector



Connector 2, Tool IO male M8, 8 poles

Digital IO		Pin no.	Configurations	Specification
-	-	8	-	GND
TPSU02	OUT	5	12V or 24V	Power output 2, 12V or 24V +5% / -15% Max 700mA *
TDO03	OUT	7	12V or 24V	Digital output 3, 12V or 24V +5% / -15% Max 100mA *
TDO04	OUT	6	12V or 24V	Digital output 4, 12V or 24V +5% / -15% Max 100mA *

TAI03	IN	4	-	Input 3, Digital or Analog 1-10V Max 30V
TAI04	IN	3	-	Input 4, Digital or Analog 1-10V Max 30V
IO/API	IN	1	-	RS 485 + baud rate programmable up to 1Mbps
	IN	2	-	RS 485 - baud rate programmable up to 1Mbps

**Note. The total combined current that can be drawn from the Tool IO is below 800mA @24V or 1.6A @12V. Exceeding the specification may damage the electronics or cause robot to stop.*

Appendix B - Expressions

The *expression* is a combination of one or more operands (literals, variables, functions) and operators. This compound is evaluated during the program execution and returns a calculated result. It is an important part of various program command inputs.

Operators

The following tablet lists all available operators and their precedence. Operators are listed top to bottom, in descending precedence.

Precedence	Operator	Description
1	a[i, j]	Array indexing
2	a.b	Member access
3	-a !a (not a)	Unary minus Logical not
4	a * b a / b	Multiplication Division
5	a + b a - b	Addition Subtraction
6	a < b a <= b a > b a >= b	Less Less or equal Greater Greater or equal
7	a = b a != b	Equality Non-equality
8	a & b (a and b) a | b (a or b)	Logical AND Logical OR

Members

The following table lists available members of different data types. Properties are accessible via dot notation.

Data Type	Operator	Description	Member Type
Number	-	-	-
Position	a.x	X	Number
	a.y	Y	Number
	a.z	Z	Number
Orientation	a.rpy	Roll, Pitch, Yaw	Number[3]
	a.quat	Quaternion	Number[4]
Configuration	a.j1	Joint 1	Number
	a.j2	Joint 2	Number
	a.j3	Joint 3	Number
	a.j4	Joint 4	Number
	a.j5	Joint 5	Number
	a.j6	Joint 6	Number
	a.j7	Joint 7	Number
Pose	a.pos	Position	Position
	a.rot	Orientation	Orientation
	a.conf	Configuration	Configuration
	a.ref_pose	Reference Pose	Pose
Inertia Matrix	a.xx	Moment of inertia	Number
	a.yy	Moment of inertia	Number
	a.zz	Moment of inertia	Number
	a.xy	Moment of inertia	Number
	a.xz	Moment of inertia	Number
	a.yz	Moment of inertia	Number
Load	a.pose	Pose	Pose
	a.mass	Mass	Number
	a.cog	Center of gravity	Position
	a.imx	Inertia matrix	Inertia Matrix
Grid Axis	a.pose	Final pose	Pose
	a.steps	Number of steps	Number
GridPattern	a.pose	Initial pose	Pose
	a.axis1	Axis 1	Grid Axis
	a.axis2	Axis 2	Grid Axis
	a.axis3	Axis 3	Grid Axis
	a.next	Next pose	Pose
	a.count	total count	Number
Array	a.size	Number of rows	Number

Functions

Current version of Kassow Robots control software provides built-in functions divided into modules.

Math Module

Math module provides mathematical functions for manipulation with Numbers.

Constants

Function	Description
math.pi()	Returns the mathematical constant π = 3.141592...
math.e()	Returns the mathematical constant e = 2.718281...

Theoretic and Representation

Function	Description
math.abs(x)	Returns the absolute value of x.
math.ceil(x)	Returns the ceiling of x, i.e., the closest integer value greater than or equal to x.
math.floor(x)	Returns the floor of x, i.e., the closest integer value less than or equal to x.
math.round(x)	Returns the closest integer value to x.
math.trunc(x)	Returns the integer part of x.

Power and Logarithmic

Function	Description
math.exp(x)	Returns e^x.
math.log(x)	Returns the natural logarithm of x (to base e).
math.log10(x)	Returns the base-10 logarithm of x.
math.pow(x, y)	Returns x raised to the power x, ie. x^y.
math.sqrt(x)	Returns the square root of x.

Trigonometric

Function	Description
math.acos(x)	Returns the arc cosine of x (in radians).
math.asin(x)	Returns the arc sine of x (in radians).
math.atan(x)	Returns the arc tangent of x (in radians).
math.atan2(x, y)	Returns the arc tangent of y/x. Takes in account signs of both inputs, so it can compute the correct quadrant and return the result between - π and π.
math.cos(x)	Returns the cosine of x in radians.

math.hypot(x)	Returns the Euclidian norm, ie. sqrt(x*x + y*y). This is the length of the vector form from the origin to point (x, y).
math.sin(x)	Returns the sine of x in radians.
math.tan(x)	Returns the tangent of x in radians.

Angular Conversion

Function	Description
math.degrees(x)	Converts angle x from radians to degrees.
math.radians(x)	Converts angle x from degrees to radians.

Min & Max

Function	Description
math.min(x, y,...)	Returns the smallest number of two or more arguments.
math.max(x, y,...)	Returns the biggest number of two or more arguments.

Program Module

Program module provides function related to program execution.

Function	Description
program.time()	Returns time in seconds since the program execution start as a floating-point number. The precision is 1 microsecond.
program.debug()	Returns 1 if the program is in debug mode, otherwise returns 0.

Geometry Module

Geometry module provides functions for manipulation with geometry objects (such as poses).

Function	Description
geometry.dist(A, B)	Returns the distance between pose A and pose B in millimetres.

Constructors

Constructor functions enable users to create values of KR data types that are not accessible directly through variables.

Function	Description
Position(x, y, z)	Constructs position from X, Y and Z coordinates.
RPY(r, p, y)	Constructs rotation from roll, pitch and yaw (RPY).

Quaternion(x, y, z, w)	Constructs rotation from quaternion coordinates (X, Y, Z and W).
Configuration(j1, j2, j3, j4, j5, j6, j7)	Constructs joint configuration vector.
InertiaMatrix(xx, yy, zz, xy, xz, yz)	Constructs inertia matrix.

Type Casting

Typa casting functions provide conversion of one data type to another.

Function	Description
Int(x)	Converts (casts) number x into integer. Since the Number data type can store both integer or floating point value, this functions allows to explicitly tell the Number to store the value as an integer.
Double(x)	Converts (casts) number x into double. Since the Number data type can store both integer or floating point value, this functions allows to explicitly tell the Number to store the value as a double.

Appendix C - System Variables

I/O (G2 – flat caps from mid 2023)

Name	Type	Description
Digital Input (1- 16)	Input	16x analog digital inputs (IO Board, 30 V)
Current Input (1, 2)	Input	2x analog inputs (IO Board, 20 mA)
Voltage Input (1, 2)	Input	2x voltage input (IO Board, 10 V)
Relay Output (1- 4)	Output	4x relay output (IO Board)
Digital Output (1- 8)	Input	8x digital output (IO Board, 24 V)
Current Output (1- 2)	Output	2x current output (IO Board, 20 mA)
Voltage Output (1- 2)	Output	2x voltage output (IO Board, 10 V)
Tool IO Digital Output (1- 4)	Output	4x digital output (Tool Board, 12 or 24 V)
Tool IO Power Supply Output (1- 2)	Output	2x power supply output (Tool Board, 12 or 24 V)
Tool IO Analog Output (1- 2)	Output	2x analog output (Tool Board, 20 mA or 10 V)
Tool IO Analog Input (1- 4)	Input	4x analog input (Tool Board, 20 mA, 10 V)

I/O (G1 - round caps)

Name	Type	Description
Digital Input (1- 16) DI01 … DI16	Input	16x analog digital inputs (IO Board, 30 V)
Current Input (1, 2) AI01, AI02	Input	2x analog inputs (IO Board, 20 mA)
Voltage Input (1, 2) AI03, AI04	Input	2x voltage input (IO Board, 10 V)
Relay Output (1- 4) RO01, RO02, RO03, RO04	Output	4x relay output (IO Board)
Digital Output (1- 8) DO1 … DO8	Output	8x digital output (IO Board, 24 V)
Current Output (1- 2) AO01, AO02	Output	2x current output (IO Board, 20 mA)
Voltage Output (1- 2) AO03, AO04	Output	2x voltage output (IO Board, 10 V)
Tool IO Digital Output (1- 4)	Configurable	4x digital output (Tool Board, 12 or 24 V)

TDO01 … TDO04	Output (12V / 24V)	
Tool IO Power Supply Output (1-2) TPSU01, TPSU02	Configurable Output (12V / 24V)	2x power supply output (Tool Board, 12 or 24 V)
Tool IO Analog Output (1- 2) TAO01, TAO2	Configurable Output (Voltage / Current)	2x analog output (Tool Board, 20 mA or 10 V)
Tool IO Analog Input (3 - 4) TAI03, TAI04	Input	2x analog Voltage inputs (Tool Board, 20 mA, 10 V)
Tool IO Analog Input (1) TAI05	Input	1x input current (Tool Board, 20 mA, 10 V)

Frames

Name	Type	Description
Robot Base Frame	Output	Attached to the base link of the robot.
Robot Link 1 Frame	Input	Attached to the link 1 of the robot.
Robot Link 2 Frame	Input	Attached to the link 2 of the robot.
Robot Link 3 Frame	Input	Attached to the link 3 of the robot.
Robot Link 4 Frame	Input	Attached to the link 4 of the robot.
Robot Link 5 Frame	Input	Attached to the link 5 of the robot.
Robot Link 6 Frame	Input	Attached to the link 6 of the robot.
Tool Flange Centre Frame (TFC)	Input	Attached to the tool flange of the robot.
Load 1 Frame	Output	Attached to the load 1 (tool load).
Load 2 Frame	Output	Attached to the load 2 (payload).
Tool Centre Frame (TCP)	Output	Exact working point of the robot tool.
Last Target Pose	Input	Last target in robot trajectory.

Loads

Name	Type	Description
Load 1	Output	Describes the rigid body of the attached tool.
Load 2	Output	Describes the rigid body of the attached payload.

Arrays

Name	Size	Type	Description

Robot Joint Angles	7	Input	Angles [rad] of individual robot joints.
Robot Joint Velocities	7	Input	Velocities [rad/s] of individual robot joints.
Robot Joint Accelerations	7	Input	Accelerations [rad/s^2] of individual robot joints.
Robot Joint Torques	7	Input	Torques [Nm] of individual robot joints.
Safety Joint Torque Deviation [raw]	7	Input	Internal torque deviation values (No filtering applied)
Safety Joint Torque Deviation [filtered]	7	Input	Internal torque deviation values (Smoothed by a filter)

Appendix D – List of extensions (Cbuns)

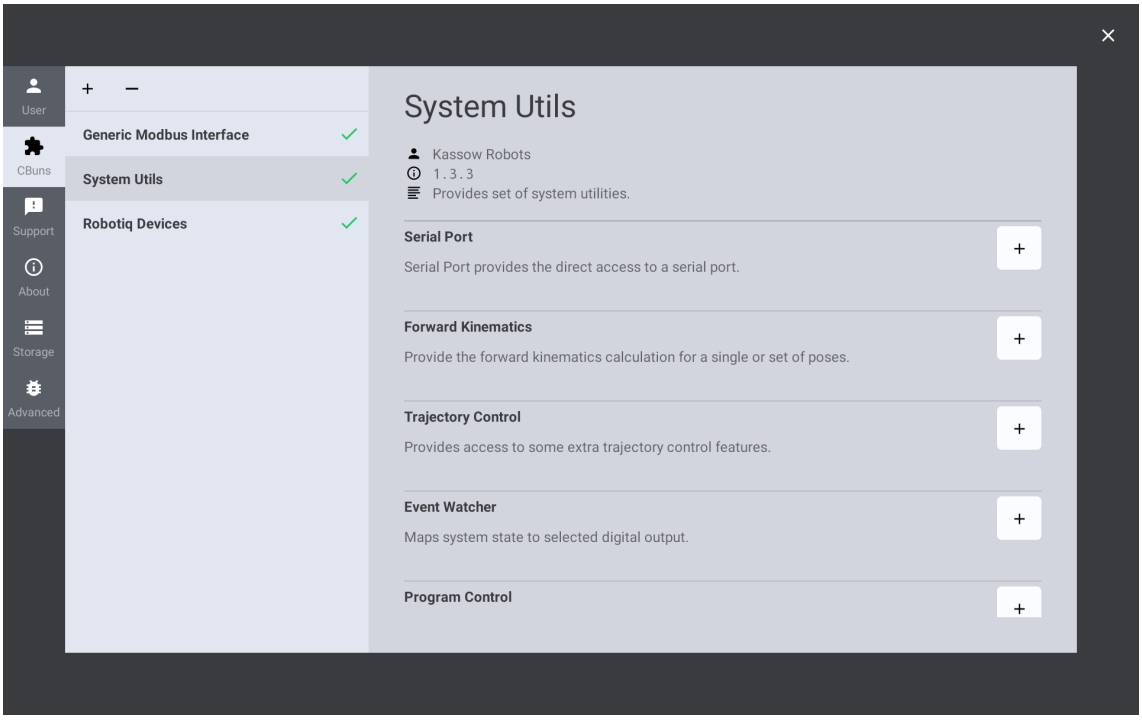
Here we share the list of the official extension modules maintained by the KR. These modules allow to integrate several external devices (e.g., grippers) or to access more specialised system functions.

Cbun Module	Description
Generic Modbus Interface	This extension module provides numerous published methods to access Modbus RTU slave device connected to the RC through the RS-485, RS-232 or the USB (RS adapter) interface.
System Utils	Provides access to various system extensions and functionalities that are not included in the standard user interface (yet). Contains the safety events mapping to I/O (Event Watcher) or interfaces the control program execution and handle alarms (Program Control).
STEP file processor	CAD based curves can be imported and processed by the software into KR robots programming environment. By using this Cbun it is possible to let the robot follow the defined trajectory which makes the process of teaching robot complex movements much easier.
Robotiq grippers	Robtiq grippers (2F-85, 2F-140, Hand-E and the vacuum EPick) are easy to integrate by using this Cbun. The Robotiq force-torque sensor support (FT300/FTS300) are also included in this package.
OnRobot grippers	KR software installation also includes the OnRobot extension module, but it provides support only for the limited set of OnRobot devices. Check the OnRobot software support website for the full range support of their products integration with the KR: https://onrobot.com/en/robot-compatibility/onrobot-solutions-for-kassow

The documentation and more detailed information about the recent versions of those modules are available at https://docs.kassowrobots.com/. To get the access to this resource please contact the Kassow Robots technical support team.

System Utils

The System Utils Cbun enhances the user's capabilities within the standard robot programming interface by providing additional system tools and features. The System Utils Cbun is available within the robot's built-in Cbun repository and it's updates are delivered together with the robot SW update. To access the following features, install the System Utils Cbun first.



Serial Port

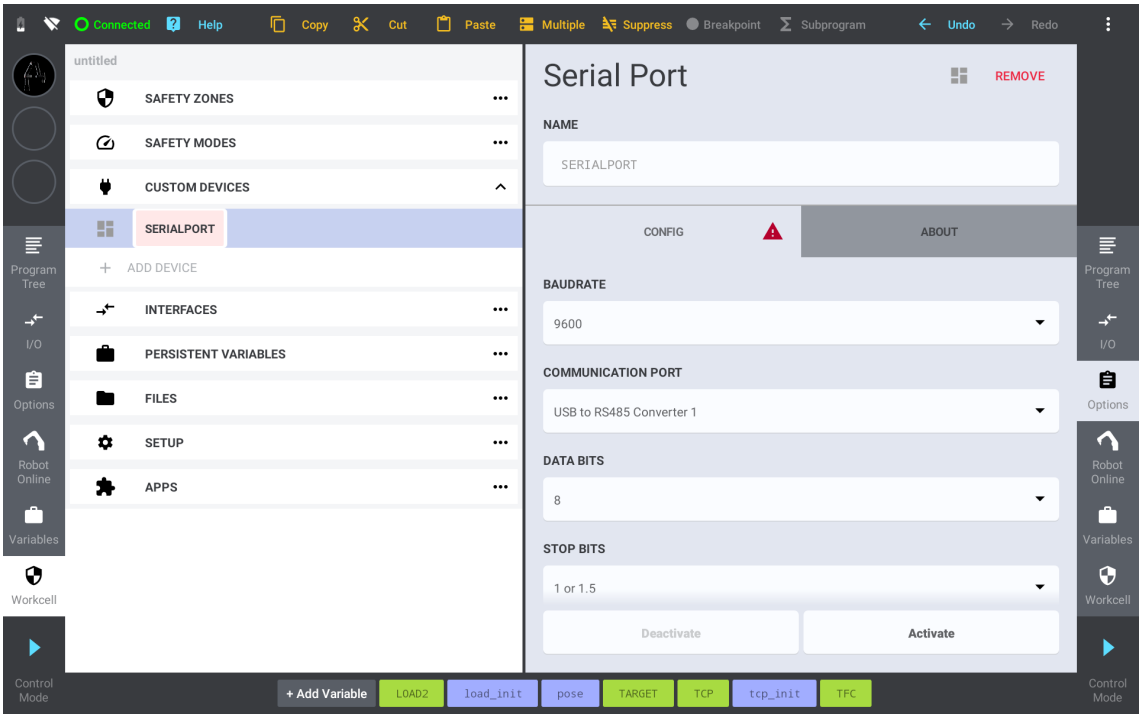
The Serial Port device offers robot programmers and interface to access a selected serial port from the robot program, allowing to programmatically exchange data with robot peripherals by using serial communication.

To add a new serial port interface into the *Workcell -> Custom Devices* section, click on the Serial Port **+** button. Each physical serial port must be controlled with a single independent Serial Port device instance.

Subsequently, navigate to the Workcell tool, select the desired **SERIALPORT** instance and open its options panel. The options panel provides access to a list of parameters, allowing you to configure the key properties of a serial communication, such as baud rate, data bits, stop bits and parity. Please note, that the parameters must be set according to the robot equipment documentation, otherwise the robot will not be able to communicate with the connected peripheral.

Additionally, the user has the option to specify a physical serial port to be controlled by the selected Serial Port interface. The user can access not only the built-in serial ports on IO Board and Tool IO, but also the additional serial port provided via USB-Serial converters.

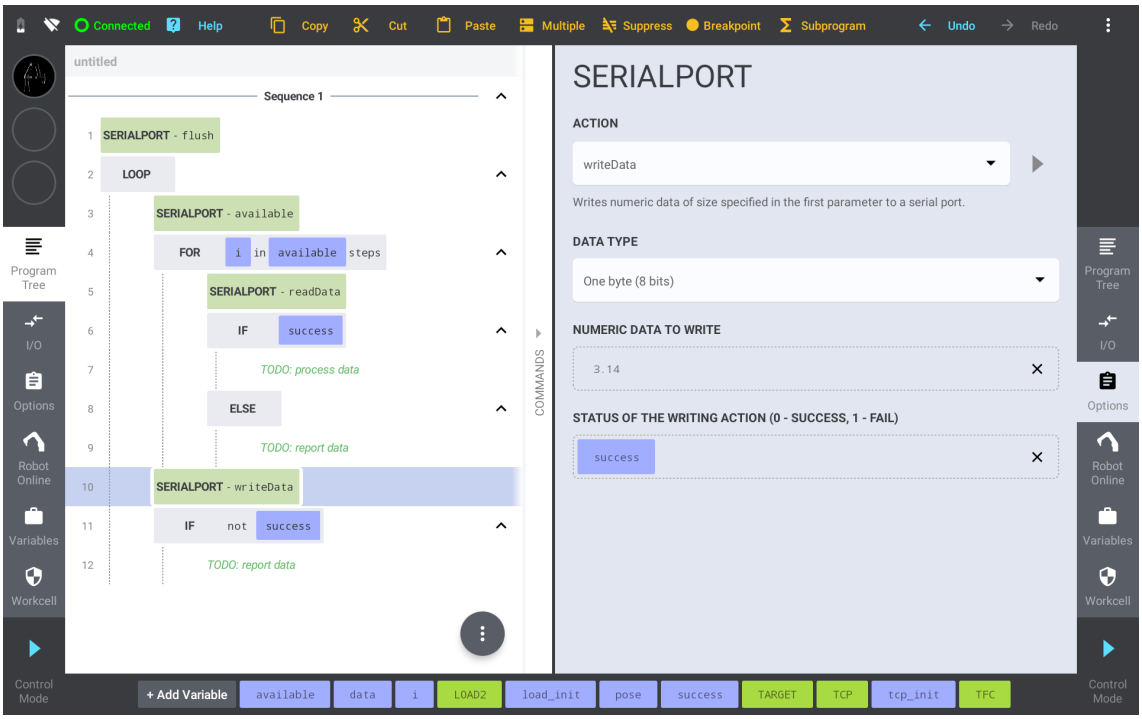
Finally, click on the **Activate** button to apply the configuration and enable the device for programming.



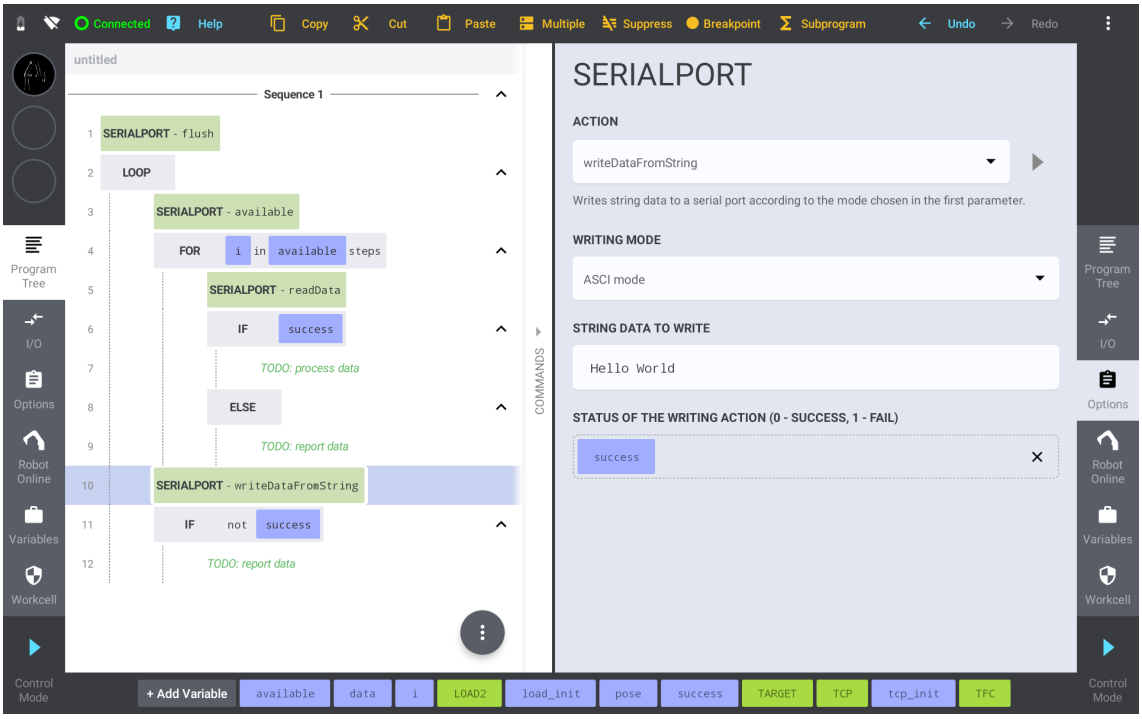
To access serial port interface within a robot program, drag the **SERIALPORT** command from the Commands Box, drop it into a desired position in the Program Tree and select one of the actions.

The **writeData** command performs transmission of specified numeric data. The user can enter the data in form of constant value or drop a number variable into the corresponding box. Additionally, the user has the

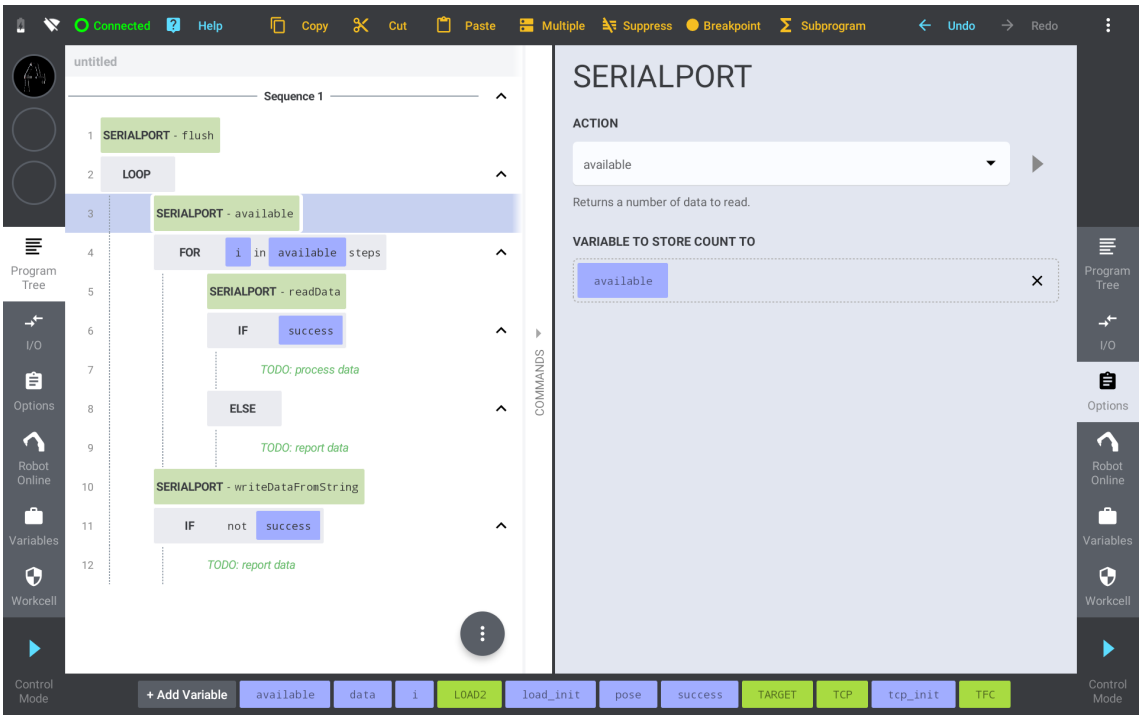
option to specify the width of the data. The result of the action is stored to a given number variable.



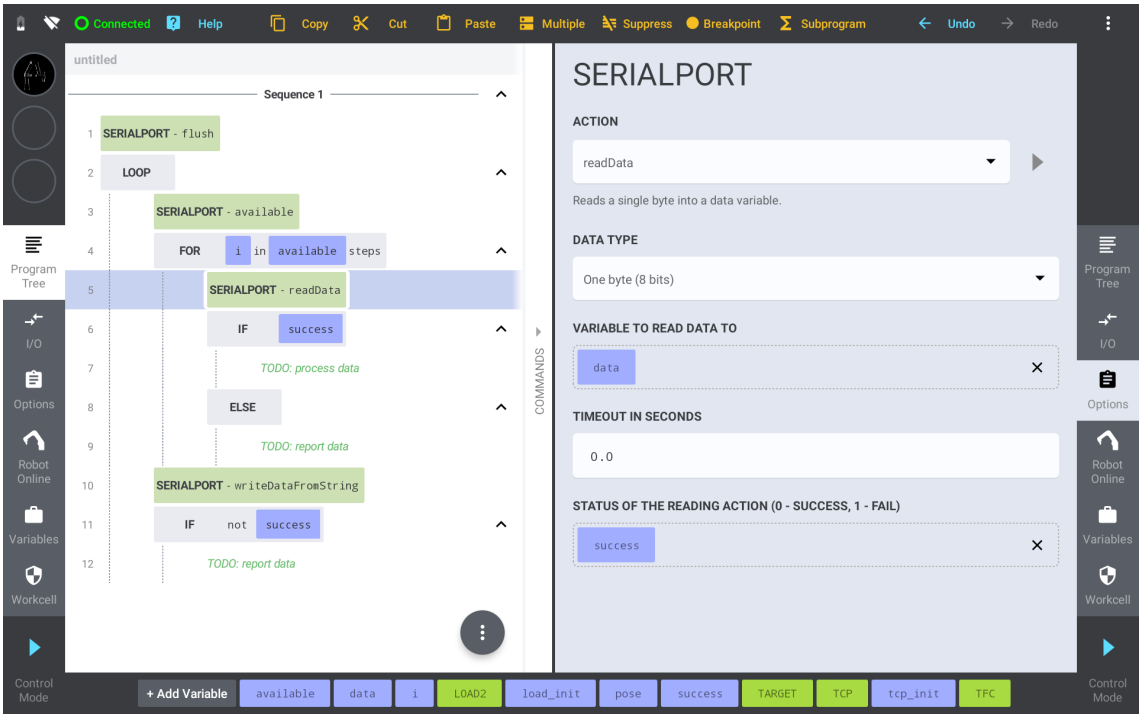
The **writeDataFromString** command sends specified text data. The data can be entered in form of an ASCII string or as a HEX string, using two 0_9A_F characters for 1 byte of data. The result of the action is stored to a given number variable.



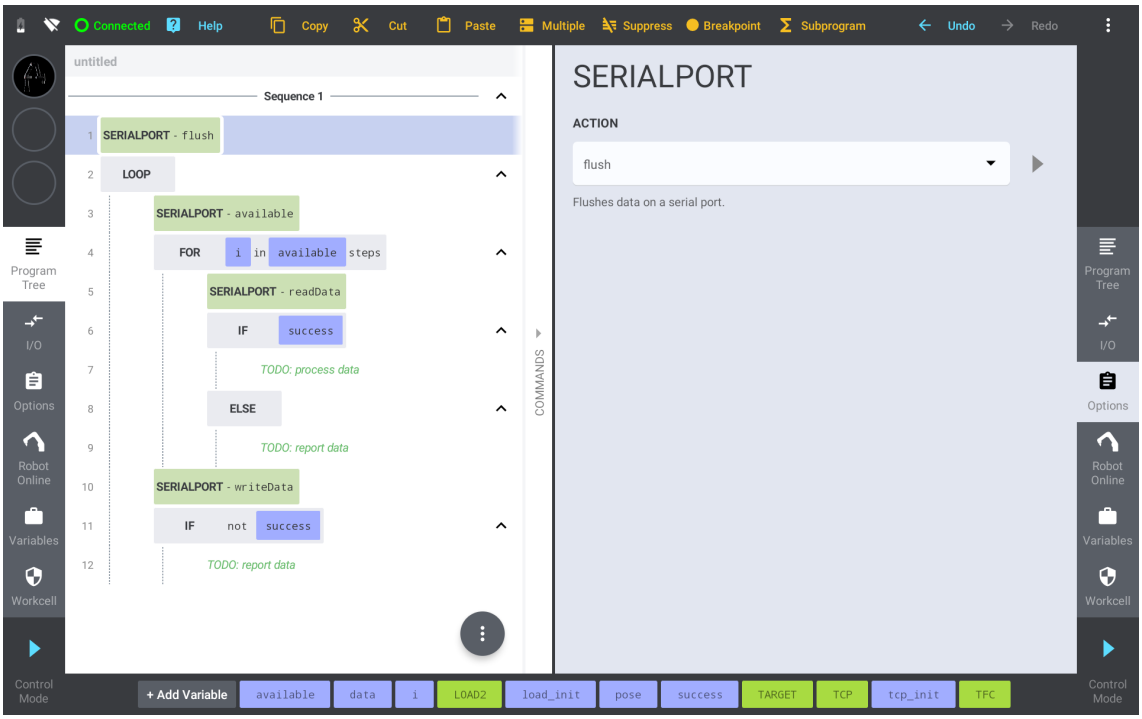
The **available** action assigns the number of available data bytes into a given number variable.



The **readData** command reads the first 1, 2 or 4 bytes, converts them into an integer and assigns this numeric value into a given number variable. Additionally, the success or failure is indicated via a given status number variable.

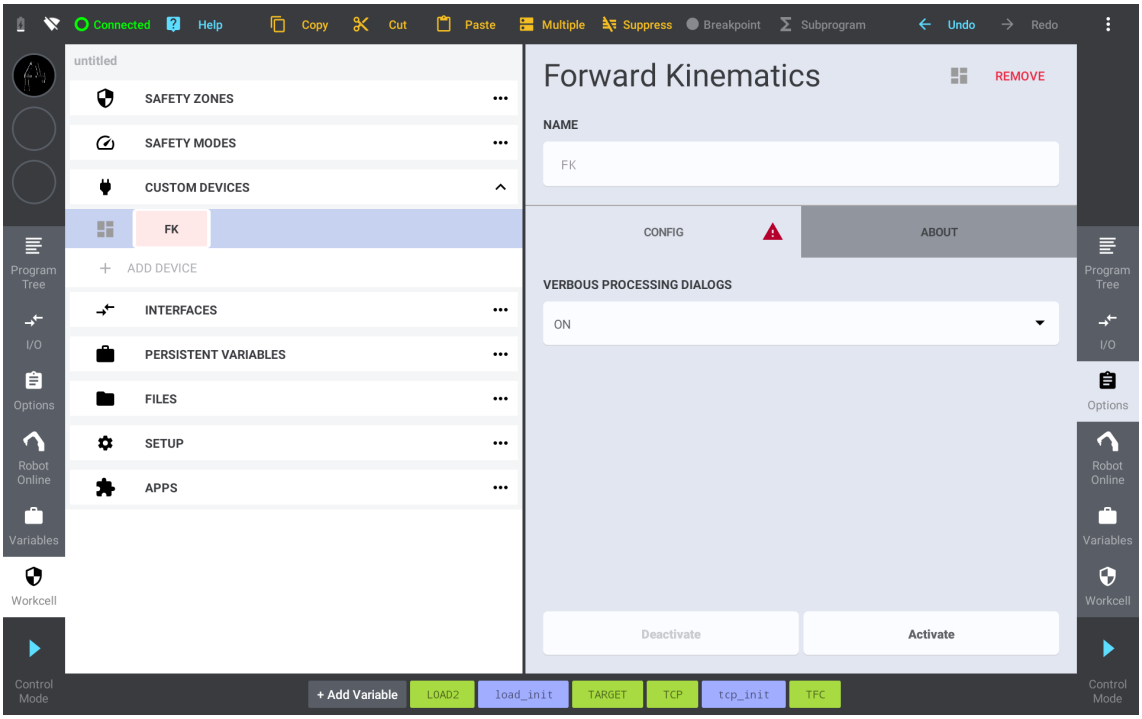


The **flush** command reads and flushes all available data on the serial communication line.



Forward Kinematics

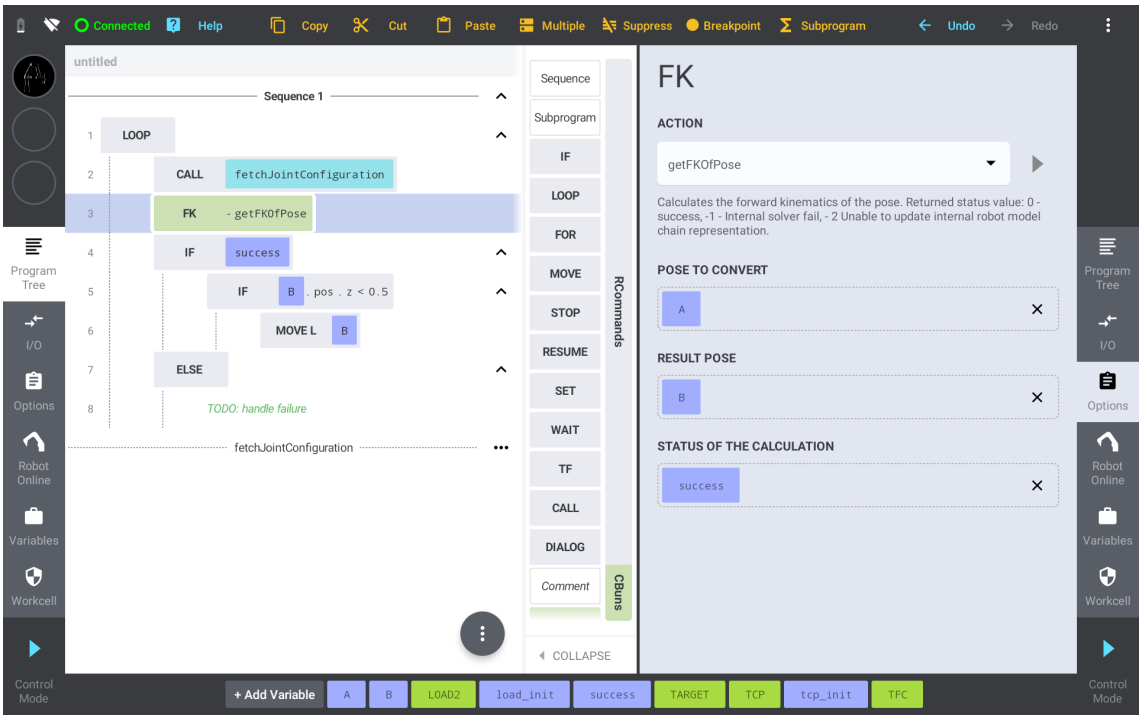
The FK instance allows users to calculate end effector's position and orientation from joint configuration.



The option to toggle **Verbose Processing Dialogs ON** and **OFF** is essential during the robot programming phase, as it serves a crucial role in communicating internal errors to the user through error dialogs. Click on the **Activate** button to apply the configuration and enable the forward kinematics instance for programming.

To calculate forward kinematics within a robot program, drag the **FK** command from the Commands Box, drop it into a desired position in the Program Tree and select one of the actions.

The **getFKOfPose** command reads the joint configuration from the **Pose to Convert** variable, calculates the corresponding position and orientation of the robot's end effector, and assigns the calculated coordinates into the **Result Pose** variable. The joint configuration and reference of the result pose variable is not affected. In addition, the **Status of the Calculation** variable is set to -2 in case of solver configuration failure and to -1 on a calculation failure.

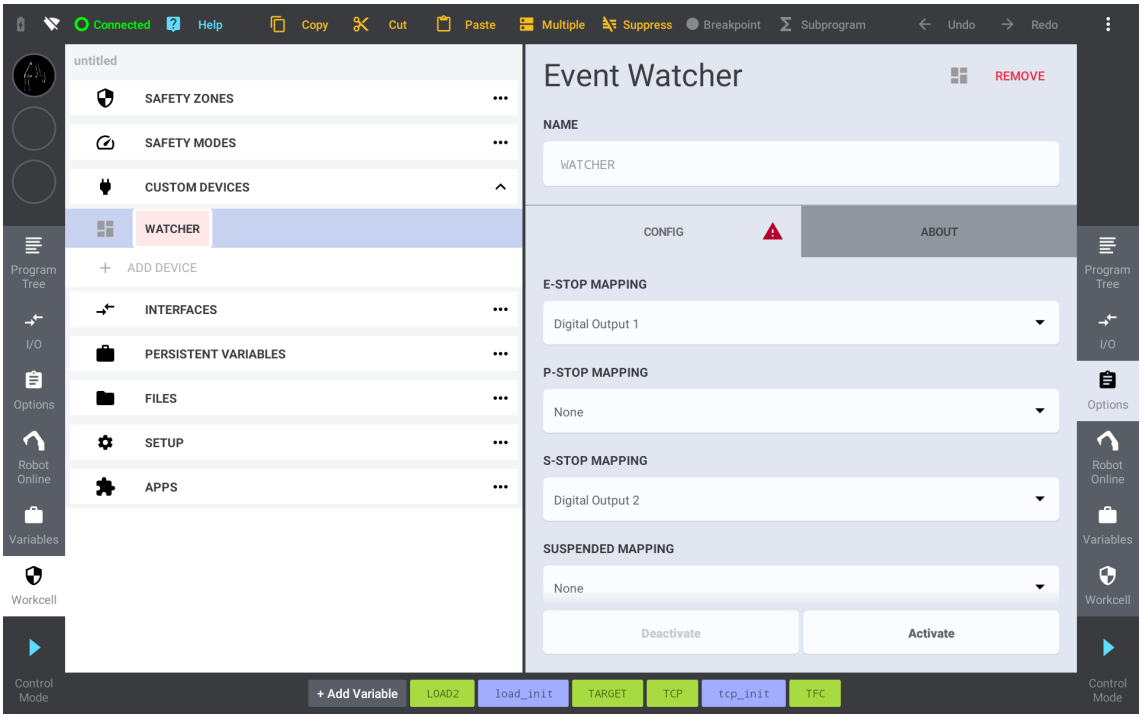


The **getFKOfPosesArray** command offers functionality similar to the **getFKOfPose** command but extends its capability by accepting an array of poses as both input and output variables. To use this function, the user needs to specify the number of poses to be processes. This enhancement allows for the efficient calculation of forward kinematics for multiple poses in a streamlined manner.

Event Watcher

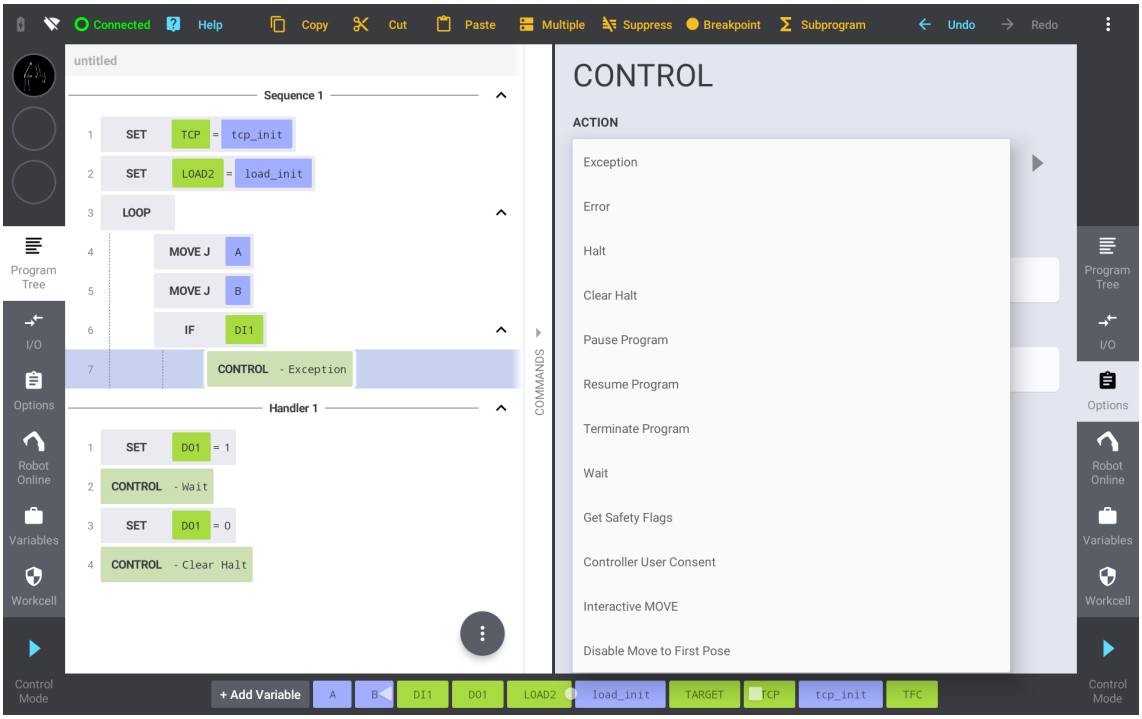
The Event Watcher allows robot programmer to map an occurrences of robot's safety events to activation of specific digital outputs and relays. Upon the detection of a specified safety violation, the corresponding output port is enabled, and when the safety issue is resolved, the output is promptly disabled.

To enable event watching feature, add a single instance of the **Event Watcher** into the Workcell, specify an associated output port for each safety event and click on the **Activate** button.

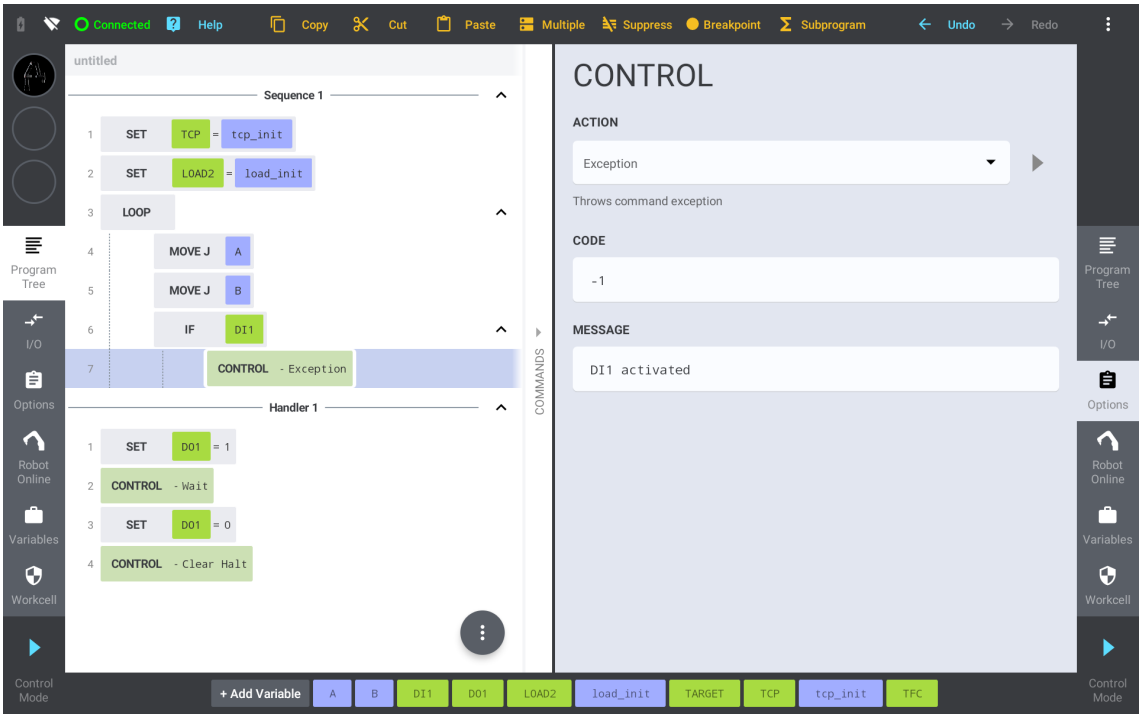


Program Control

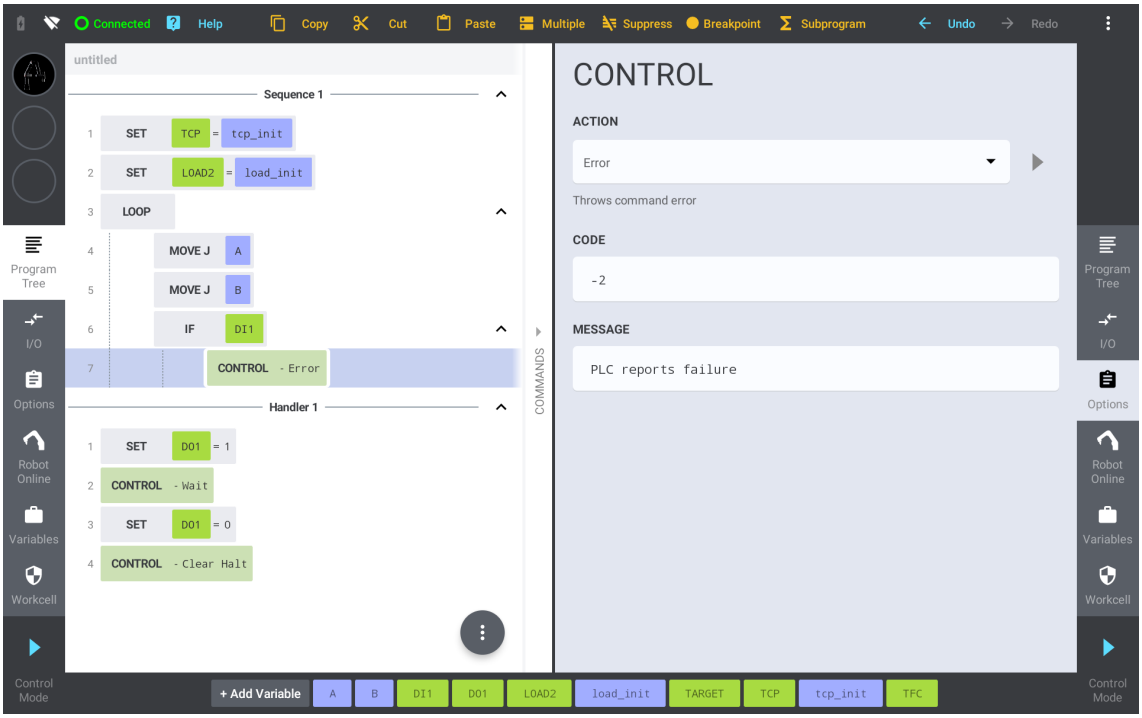
The **Program Control** instance is designed for advanced control of the program execution, allowing the user to throw exceptions and errors, pause, resume and terminate program execution or handle alarm events.



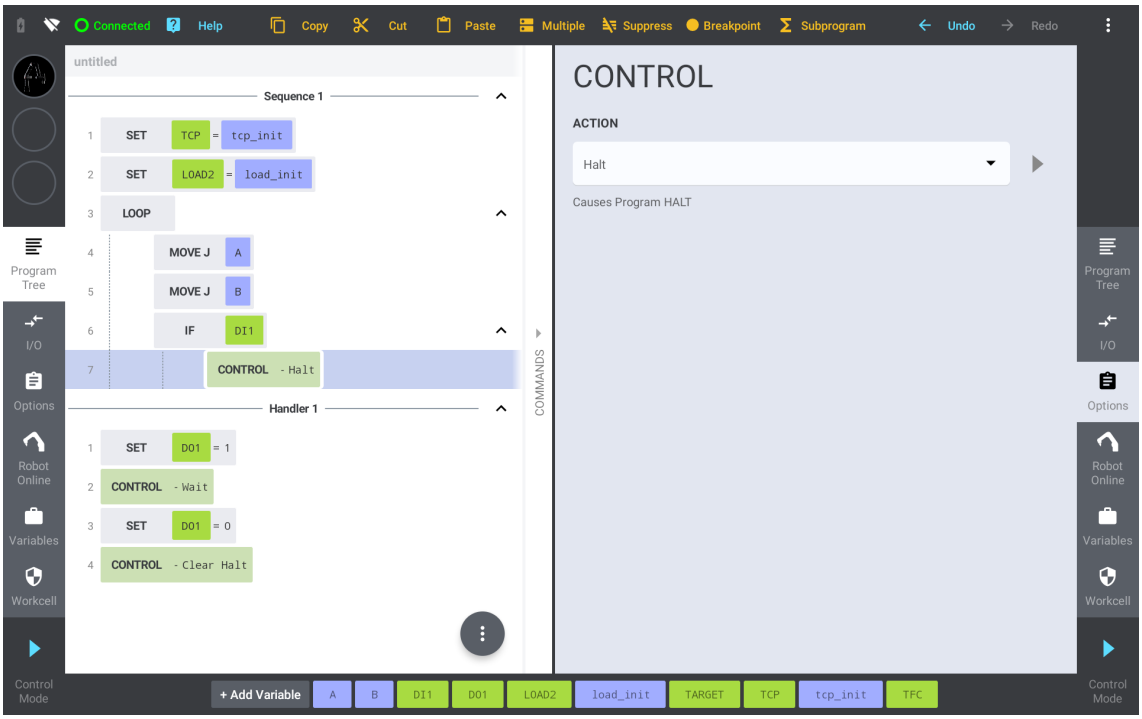
The **Exception** commands throws a non-fatal exception, triggering an alarm and halting the program execution. The specified **Code** and **Message** is displayed on Teach Pendant within an exception dialog.



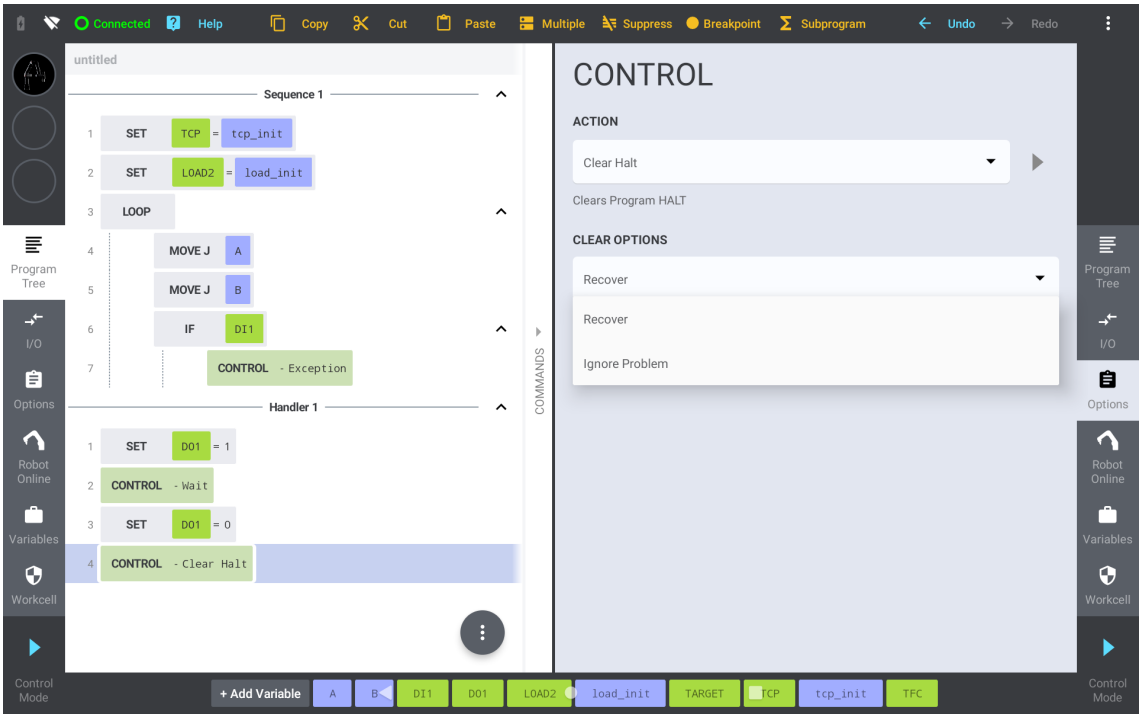
The **Error** command throws an error, causing immediate termination of the program execution. The **Code** and **Message** is displayed on Teach Pendant within a program failure dialog.



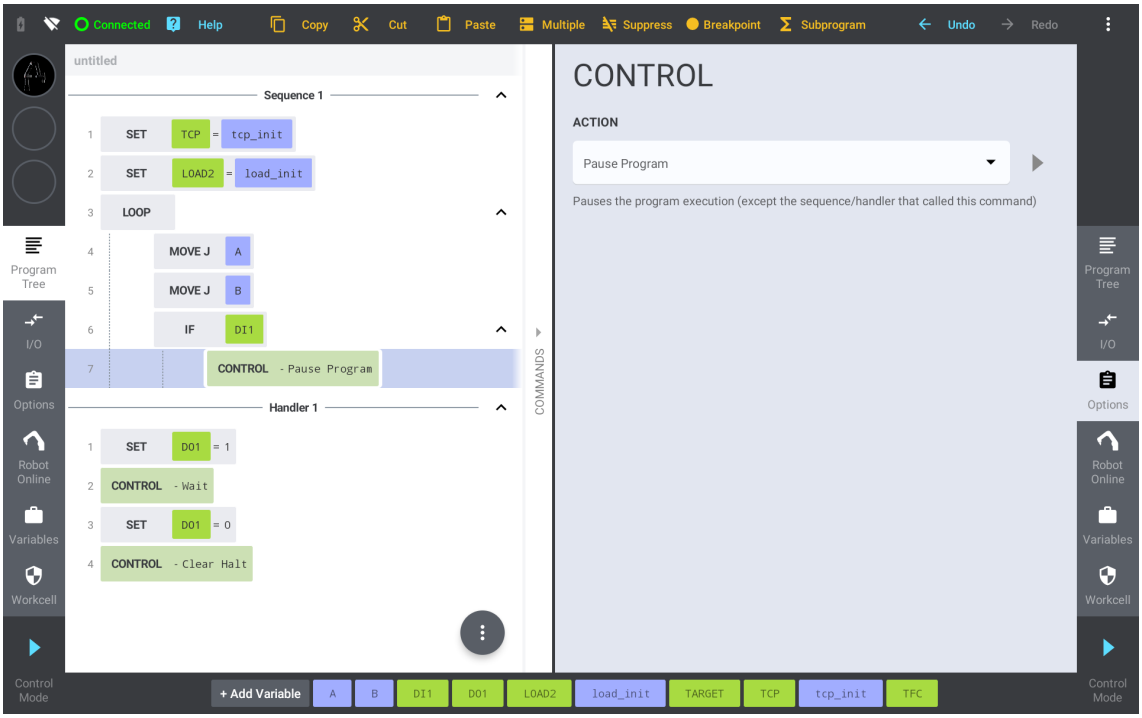
The **Halt** command halts the program execution, pausing all sequences and invoking alarm handlers.



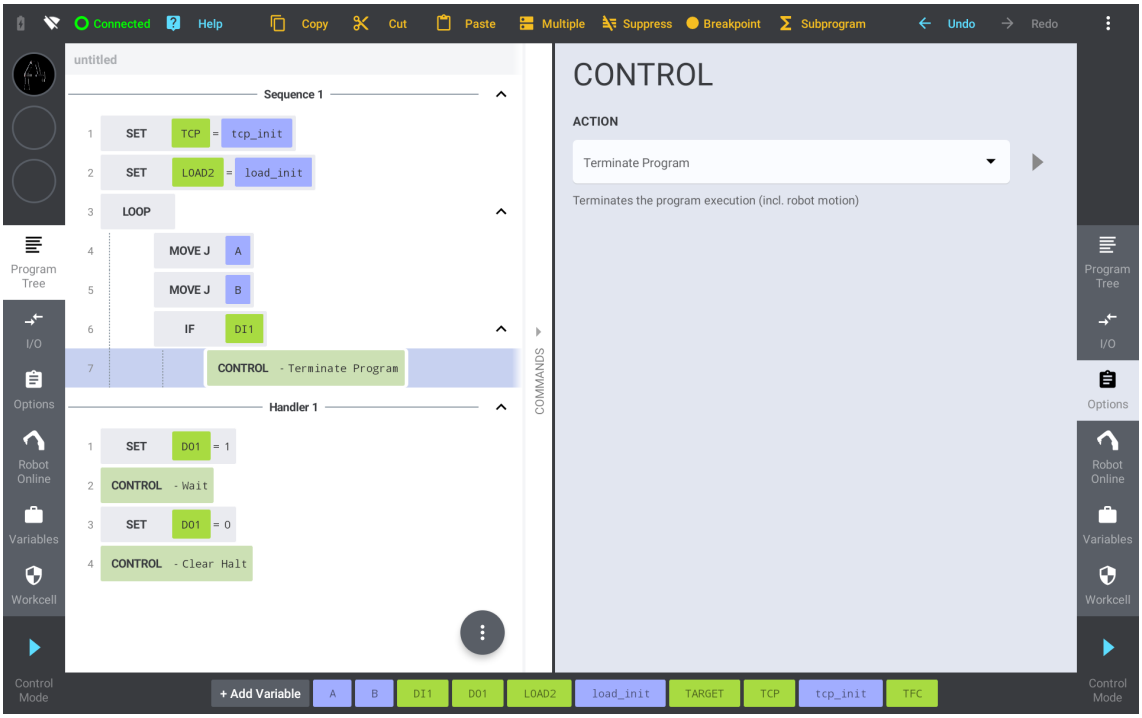
The **Clear Halt** command allows the user to resolve the programs halt within an alarm handling sequence. If cases where the halt is caused by a command exception, the robot programmer has the option to specify whether the command should recovered (executed again) or ignored (skipped).



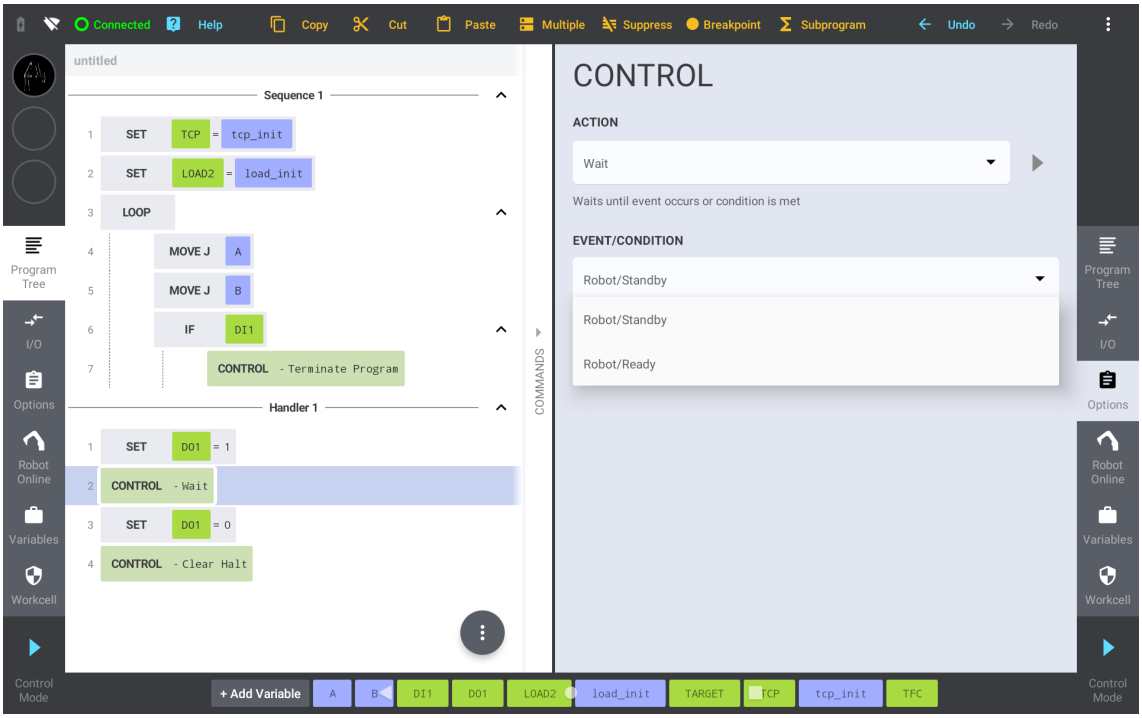
The **Pause Program** command pauses the program execution, making all sequence and handlers stop. Only the user or a Cbun can resume the program execution.



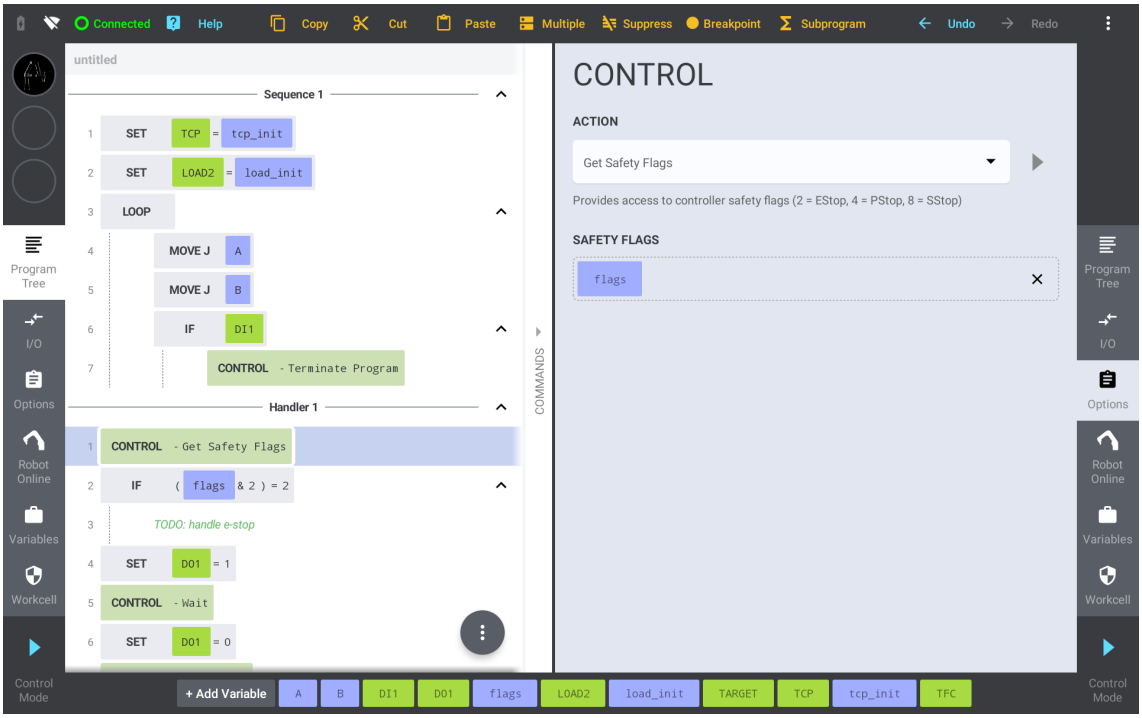
The **Terminate Program** command permanently stops the program execution. Furthermore, it terminates any ongoing robot movement and clears the planned trajectory.



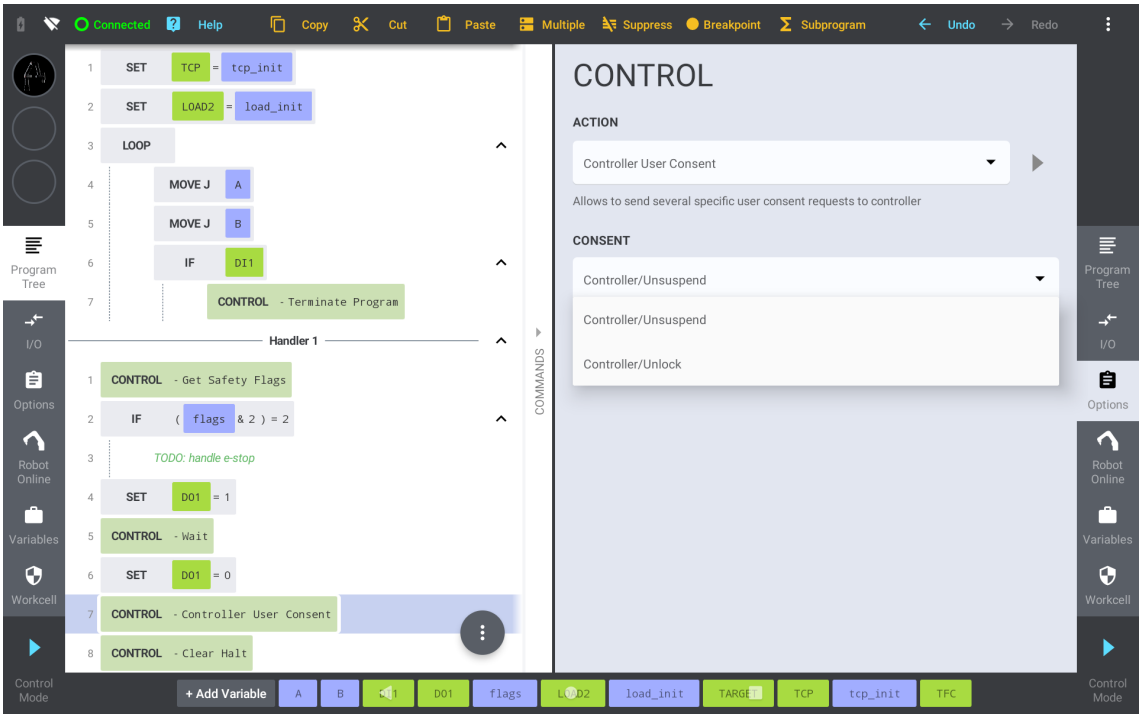
The **Wait** command is blocked until the specified condition is met. The command can wait for robot standby flag, indicating that a robot move can be performed or for ready to clear halt event.



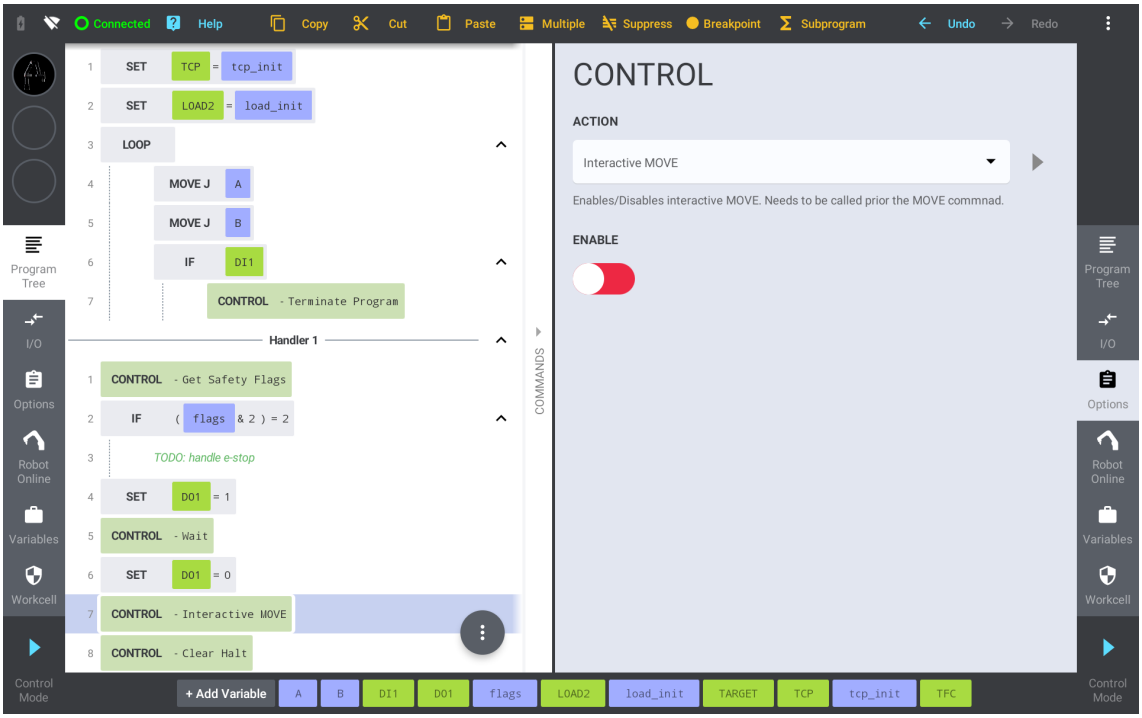
The **Get Safety Flags** command returns active safety flags. If multiple safety flags appear at a same time, the output value represents an OR combination of these flags.



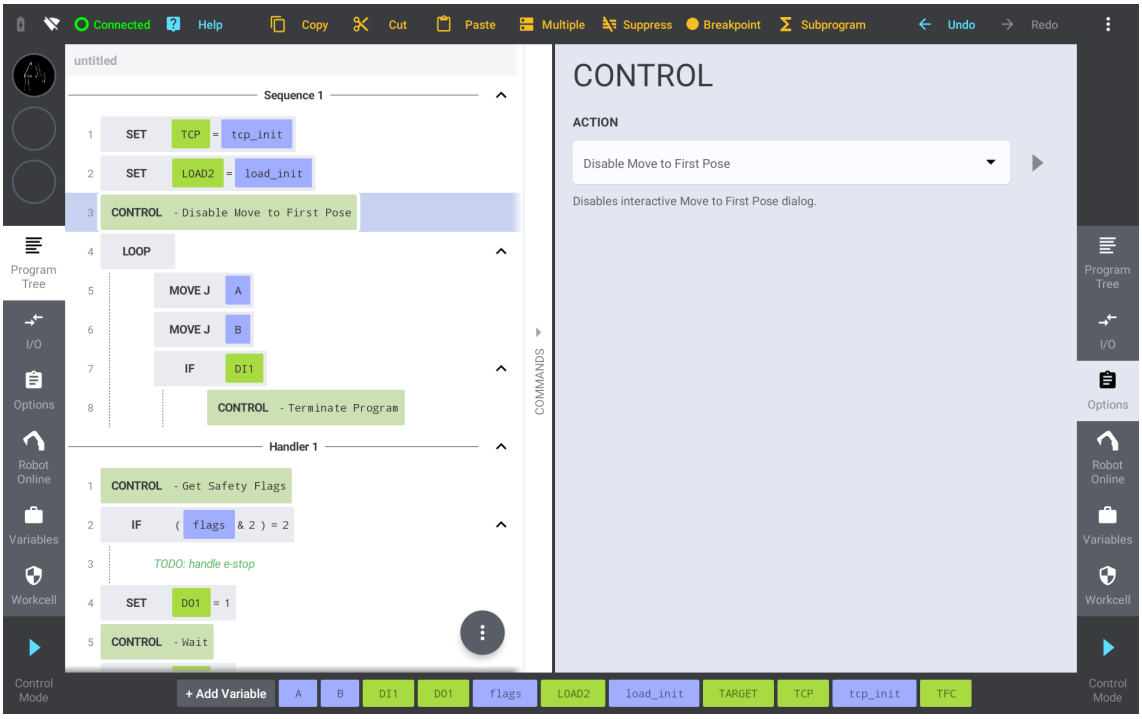
The **Controller User Consent** command allows the user to unsuspend a motion, for instance if it was previously suspended by the program pause event. Additionally, it can clear the robot controller's halt.



The **Interactive Move** command toggles the interactive motion mode prior calling a MOVE command. Each program halt automatically activates the interactive mode, requiring the user interaction on the next MOVE.



The **Disable Move to First Pose** command allows suppresses the activation of interactive motion mode for the first MOVE command within a robot program. As a result, the **Move to First Pose** dialog is not shown. Please note, that this command needs to be placed before the first MOVE command.



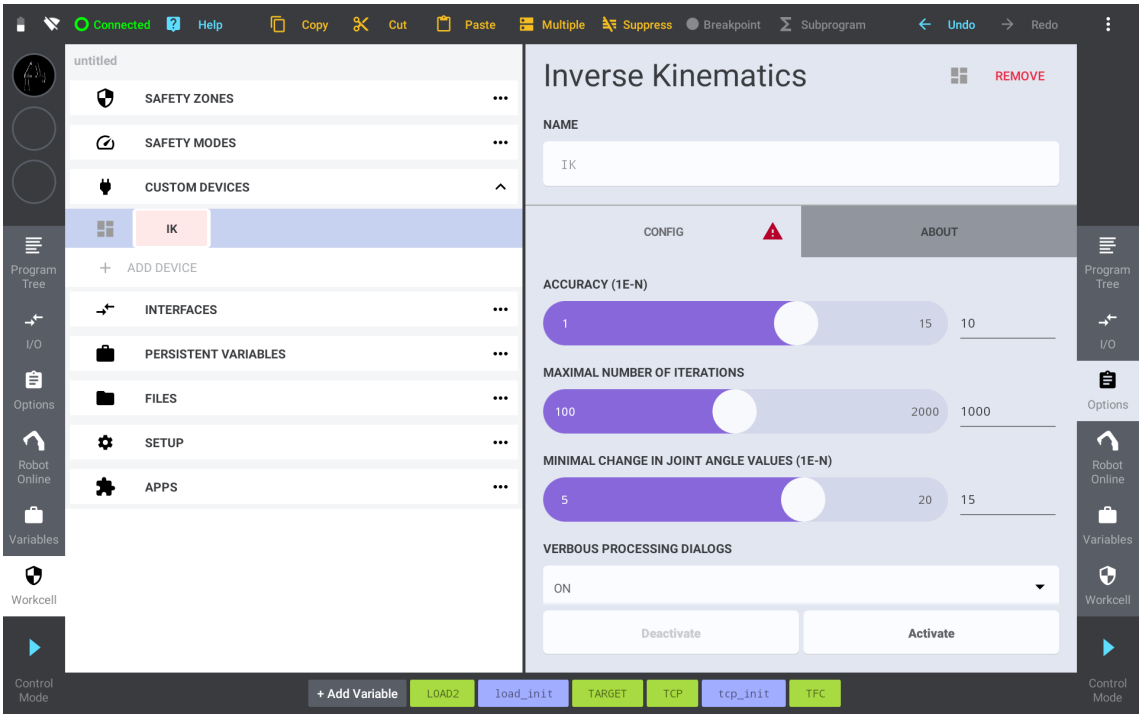
Inverse Kinematics

The **IK** instance allows the user to programmatically calculate a joint configuration from the given position and orientation of the robot's end effector. This feature is crucial for programming robot movements when, for instance, target end-effector coordinates are obtained from a grid pattern or a camera system, but the robot programmer intends to control the robot's motion in Jointspace.

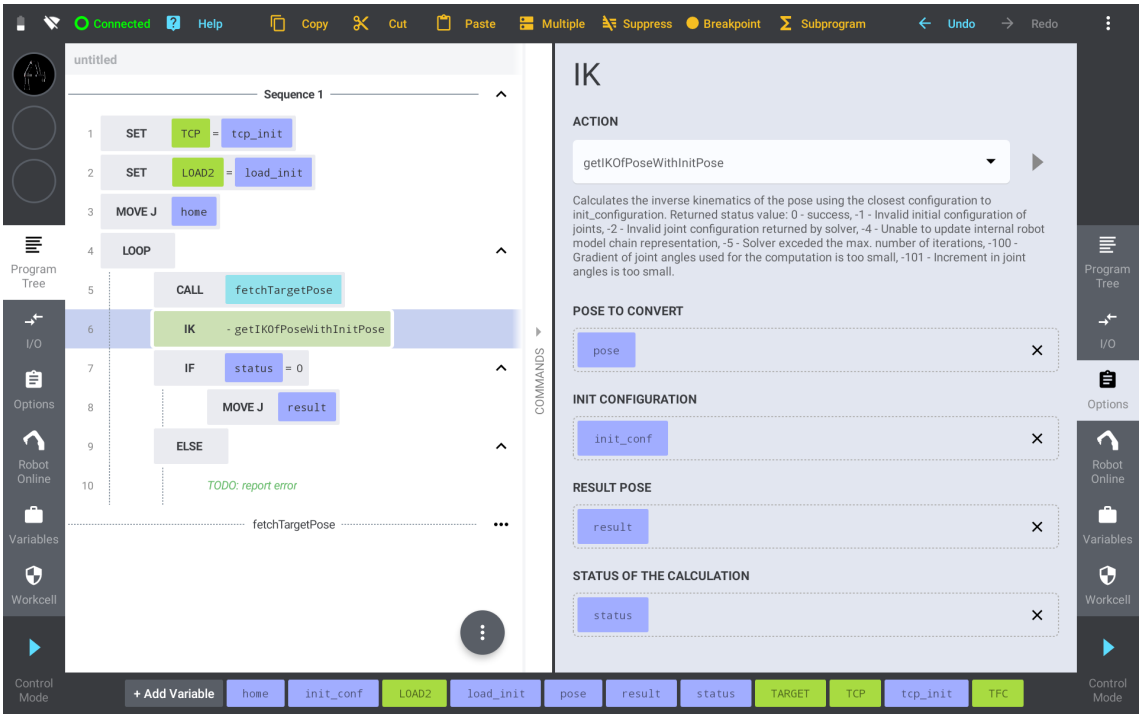
Solving inverse kinematics for 7-axes robots is a complex mathematical task, usually offering multiple solutions, as there are different joint configurations that achieve the same end-effector pose. The calculation of the resulting pose is done iteratively, where more iterations lead to improved accuracy at the cost of increased processing time.

Begin by adding a single **IK** instance and navigating to its options panel. The **Accuracy** parameter specifies the desired accuracy in robot's Jointspace. The **Maximal Number of Iterations** limits the number of iterations to be used for reaching the specified accuracy. The higher accuracy usually requires more iterations and takes longer processing time. The inverse kinematics task will stop when the calculated joint angle increments are smaller than the **Minimal Change in Joint Angle Values** parameter, avoiding unnecessary computations if the joint angle increments do not effectively change anymore.

The option to toggle **Verbose Processing Dialogs ON** and **OFF** is essential during the robot programming phase, as it serves a crucial role in communicating internal errors to the user through error dialogs. Click on the **Activate** button to apply the configuration and enable the inverse kinematics instance for programming.

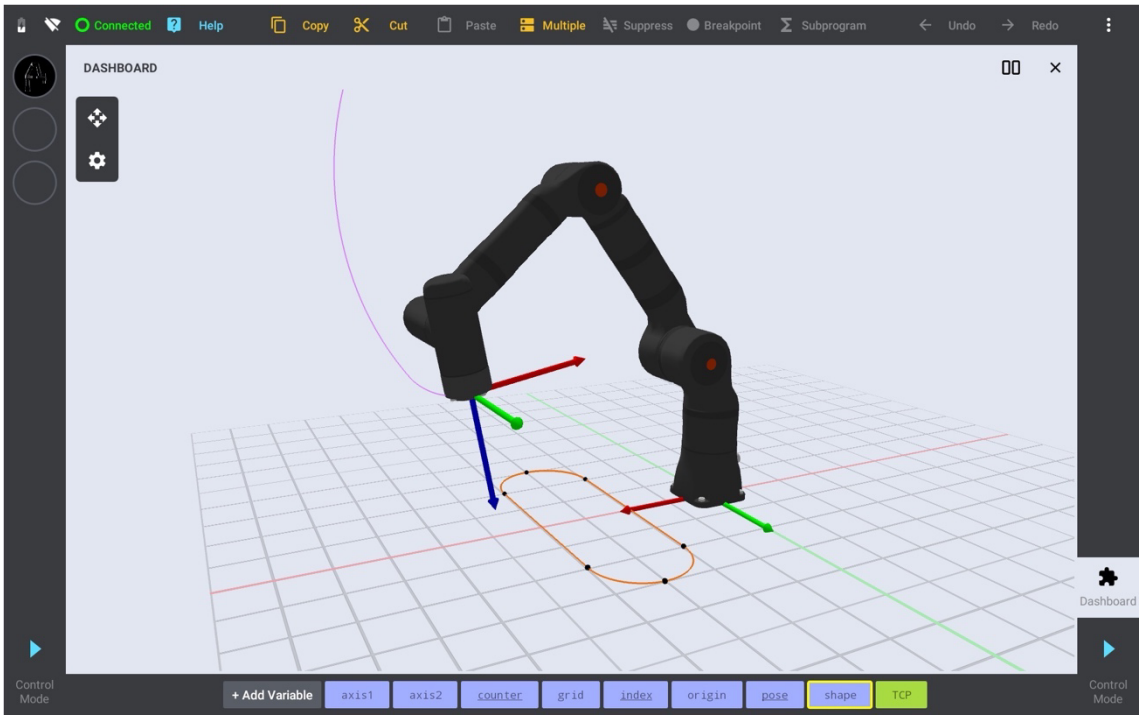


The **getIKOfPoseWithInitPose** command computes the inverse kinematics of the **Pose to Convert**, identifying the solution closest to the specified joint configuration within the **Init Configuration** variable. The resulting joint configuration is assigned to the **Result Pose** variable, without affecting its other coordinates. The **Status of the Calculation** variable specifies the cause of a possible failure. Additionally, the user can calculate inverse kinematics of multiple poses by using **getIKOfPosesArrayWithInitPose.**



Nitro Dashboard

The Nitro Dashboard Cbun introduces a Dashboard application, offering a 3D visualization of the robot arm and its virtual environment, including frames, loads and safety zones. Users can manipulate the camera by selecting the appropriate mode and using drag gesture for rotation or movement, while the pinch gesture controls zooming in and out. In addition, the Dashboard application supports both split-screen and fullscreen modes.



Furthermore, the Dashboard application offers the following set of features.

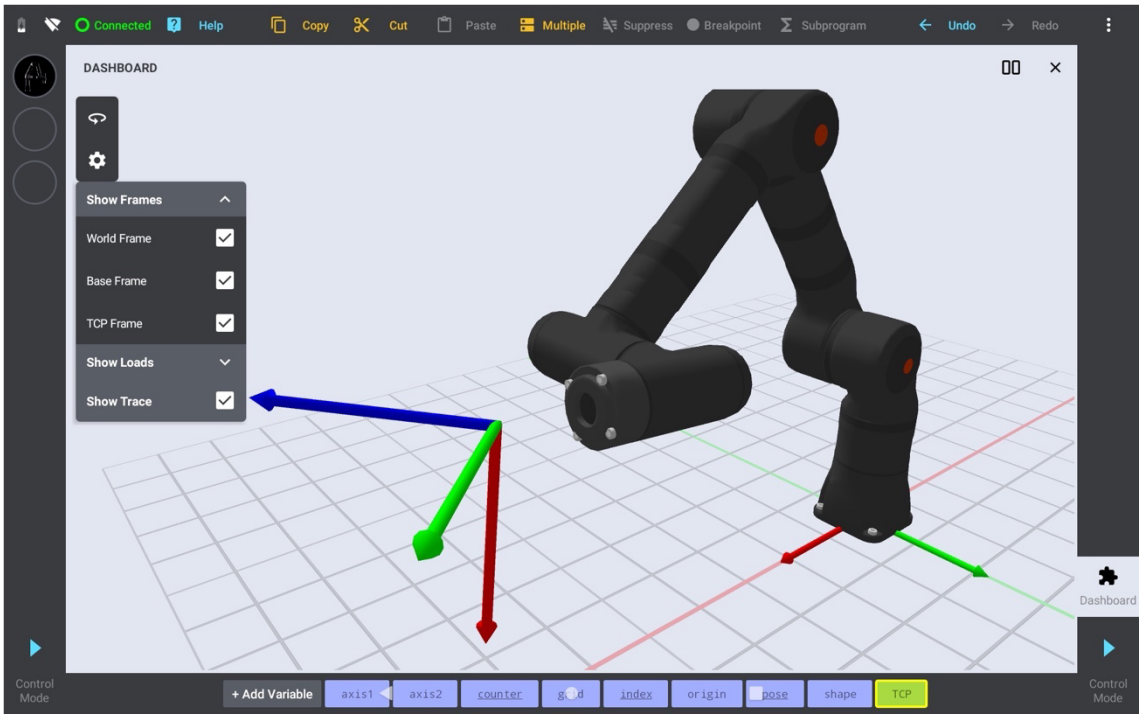
Robot Arm Position

The 3D robot arm continuously mirrors the actual pose and joint configuration of the real robot arm. Additionally, the 3D robot arm accurately represents the configuration of the system's BASE frame. This feature ensures real-time and accurate visual representation, aligning the virtual robot model with the physical robot arm state.

Robot Frames

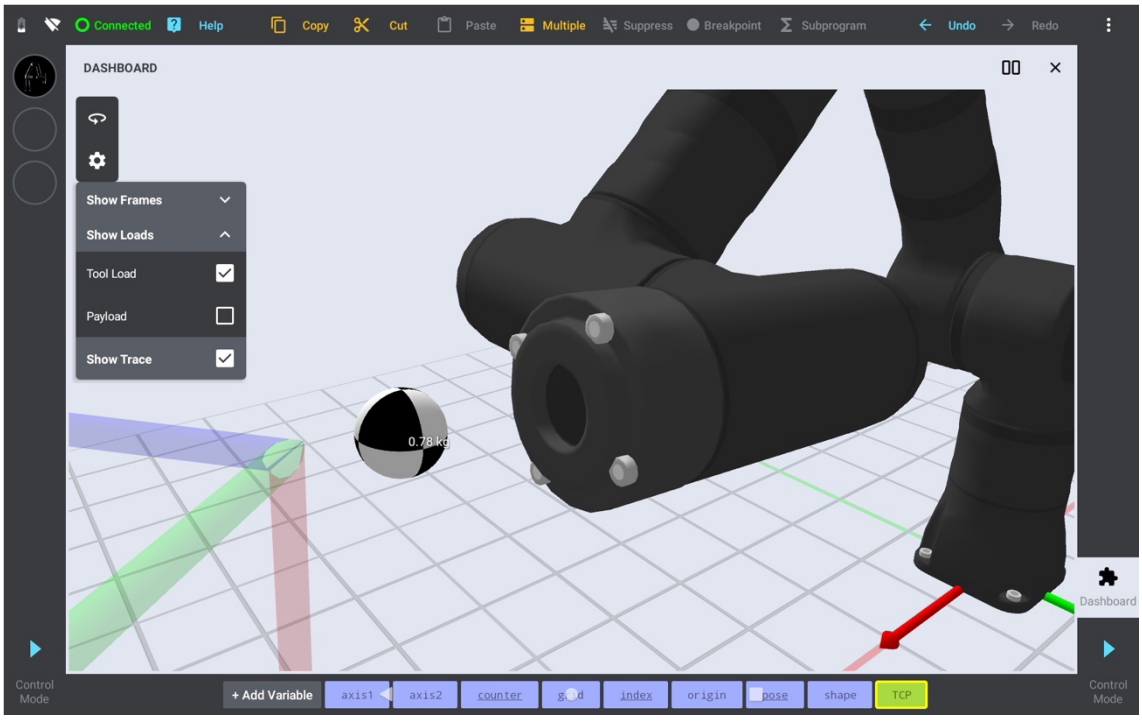
The Dashboard Application excels in visualizing crucial robot frames, including the world frame, base frame and TCP frame. Axes of each frame are represented by three perpendicular arrows (x = red, y = green, z = blue) originating from the frame's origin. This visualization provides a clear and intuitive visual reference for understanding the orientation of these key robot frames.

Users can customize the visibility of robot frames by accessing the Dashboard **Settings** and expanding the **Show Frames** section. Simply toggle the frame visibility by clicking the corresponding checkbox.



Robot Loads

The Dashboard application enables visualization of both tool load and payload. Each load is represented by a black and white sphere accompanied by a text label, offering real-time information on load position and mass.



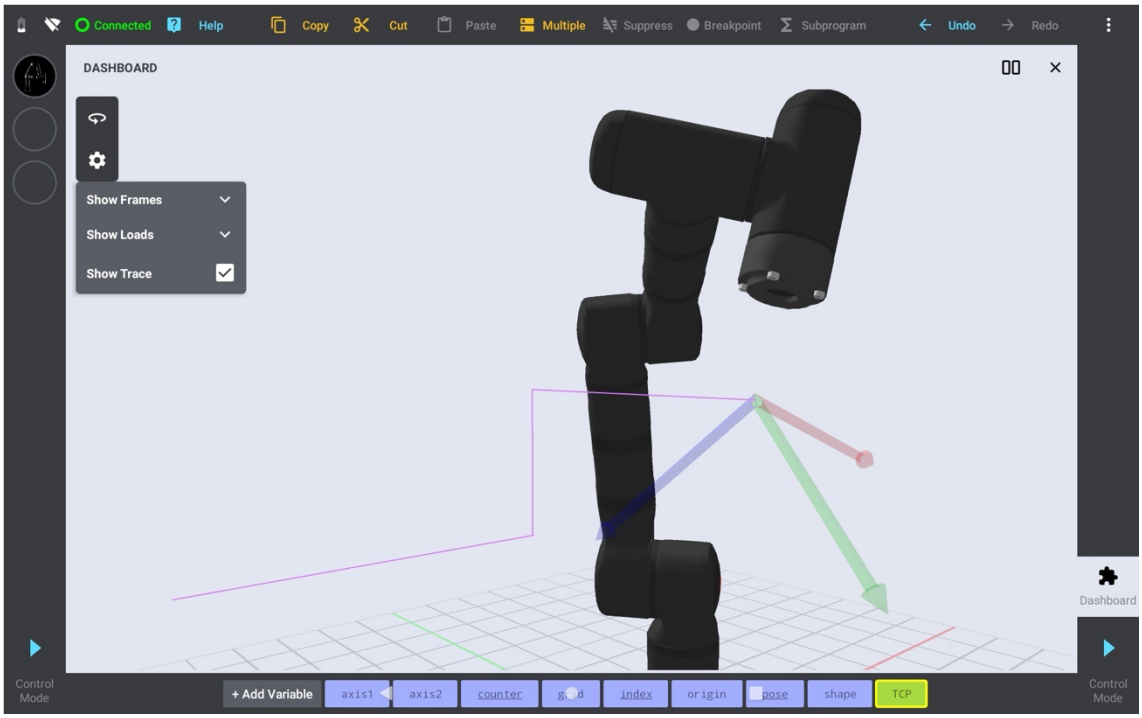
The visualized sphere's centre is precisely aligned with the load's configured centre of gravity. Moreover, the size of the sphere corresponding to the load's mass. A larger sphere indicates a greater mass.

Users can customize loads visibility by navigating to the **Show Loads** section within the Dashboard **Settings** and clicking on the checkboxes of the corresponding loads.

Trace Path

The trace path feature in the Dashboard application visually represents the trajectory of a robot's performed movement. Continuous generation of new trace path segments occurs at the robot's TCP. The oldest segments are automatically removed to maintain a clear and concise visualization.

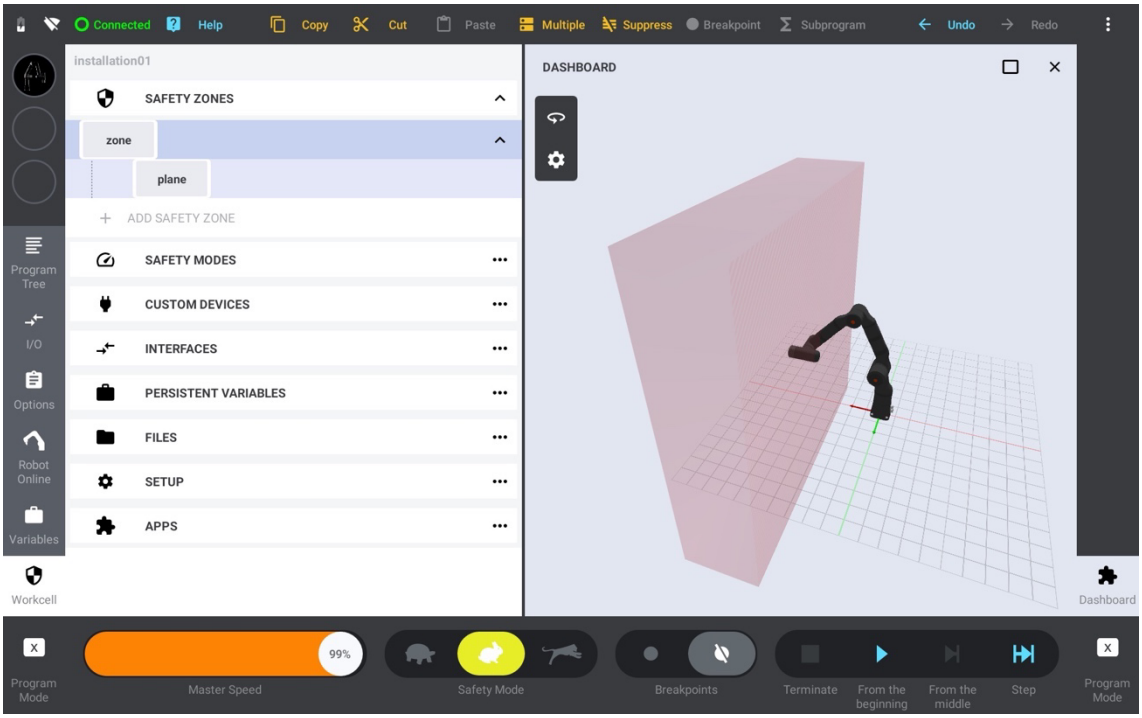
To toggle the trace path visibility, access the Dashboard **Settings** and click on the **Show Trace** checkbox.



Safety Zones

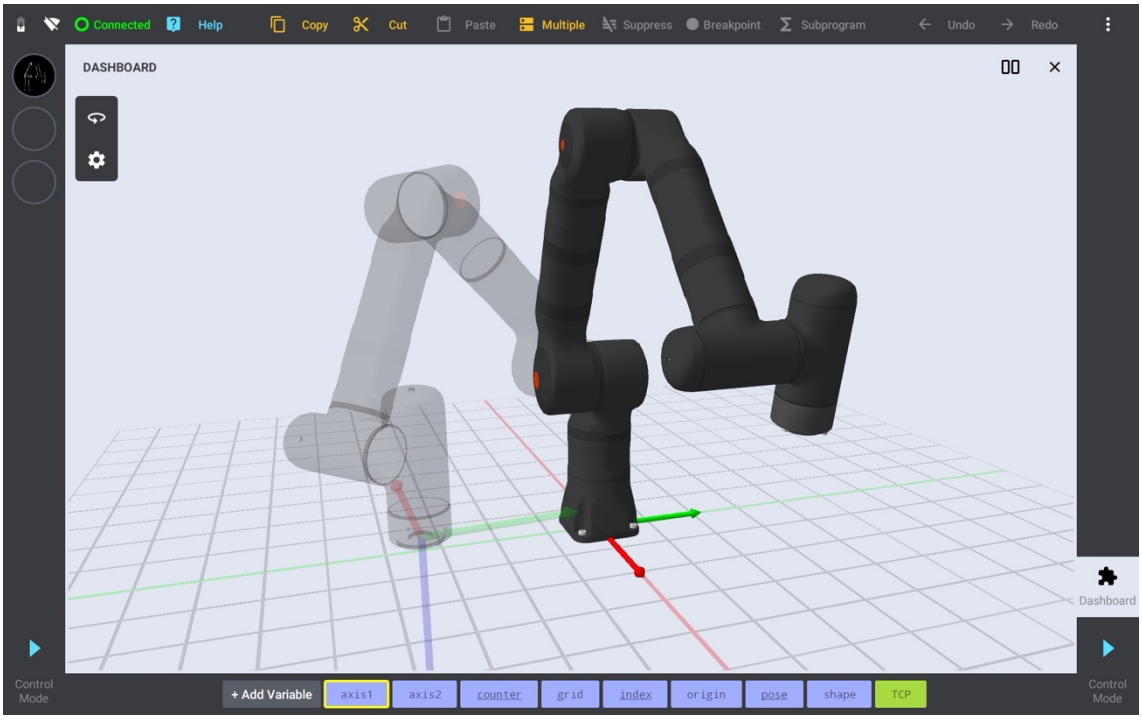
The Dashboard application serves a crucial role in defining and verifying robot safety zones, ensuring accurate positioning and orientation of safety zone geometry. Each safety zone's plane geometry is visually represented by a semi-transparent red box, with the front hatched side aligned to the plane geometry. The complete volume of the box is placed within the specified subspace.

To display a single plane geometry, access the Safety Zones section in the Workcell tool and select the desired plane geometry. In addition, the Dashboard application allows the user to visualize all safety zone geometries simultaneously by selecting a safety zone. In this instance, the resulting safety zone geometry is represented as an intersection of individual geometries.



Pose Variable

The Dashboard application automatically visualizes any selected pose variable, allowing the user to verify pose coordinates and consistency. Whereas the position of the semi-transparent robot arm is based on the joint configuration, the semi-transparent frame represents the position and orientation of the pose.

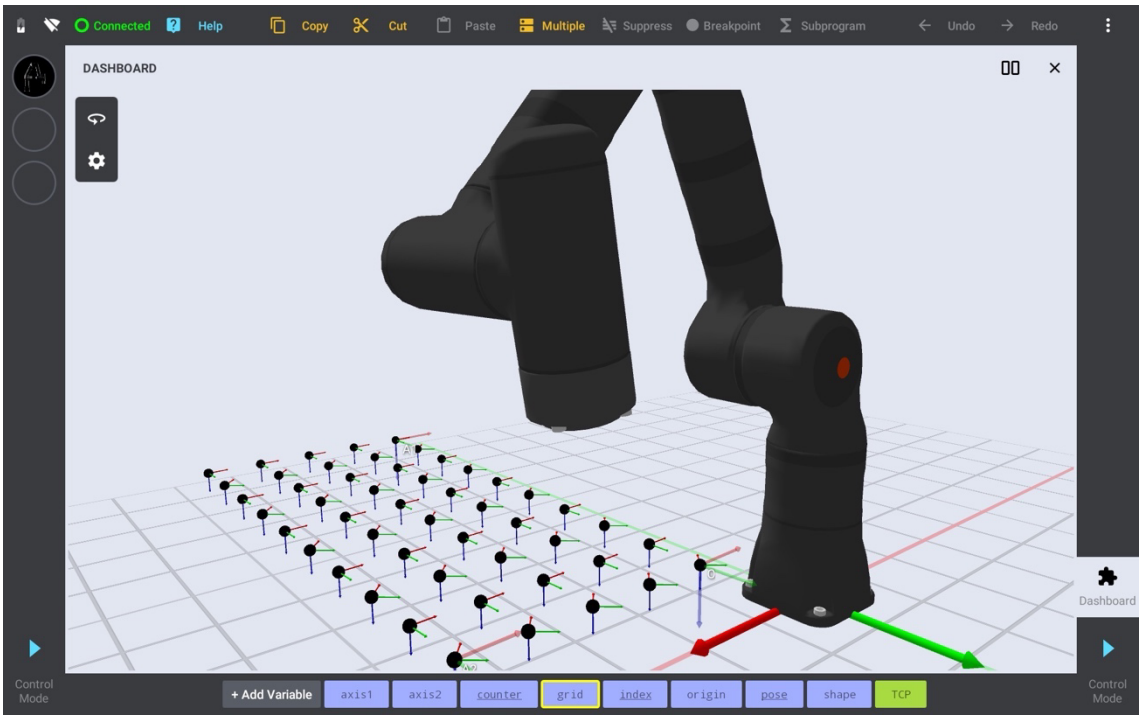


Visualizing a pose variable is straightforward – just select the desired variable. However, it is essential to note that the pose visualization feature is enabled only when a single pose variable is chosen.

Pattern Variable

The Dashboard application streamlines the setup of a pattern variable. When a single pattern variable is selected, the Dashboard application not only displays the origin and axes poses of the pattern but also visualizes the distribution of the pattern's output poses.

Whereas the origin and axes frames are represented by 3D axes models, the pattern's output frames are displayed with black spherical marker accompanied by corresponding 3D axes. This visualization not only represents frame's position and orientation, but also accurately reflects the offset configuration along individual pattern axes. For instance, in the following example, a rotational offset of 45° is applied on every second frame along the pattern's X axis.

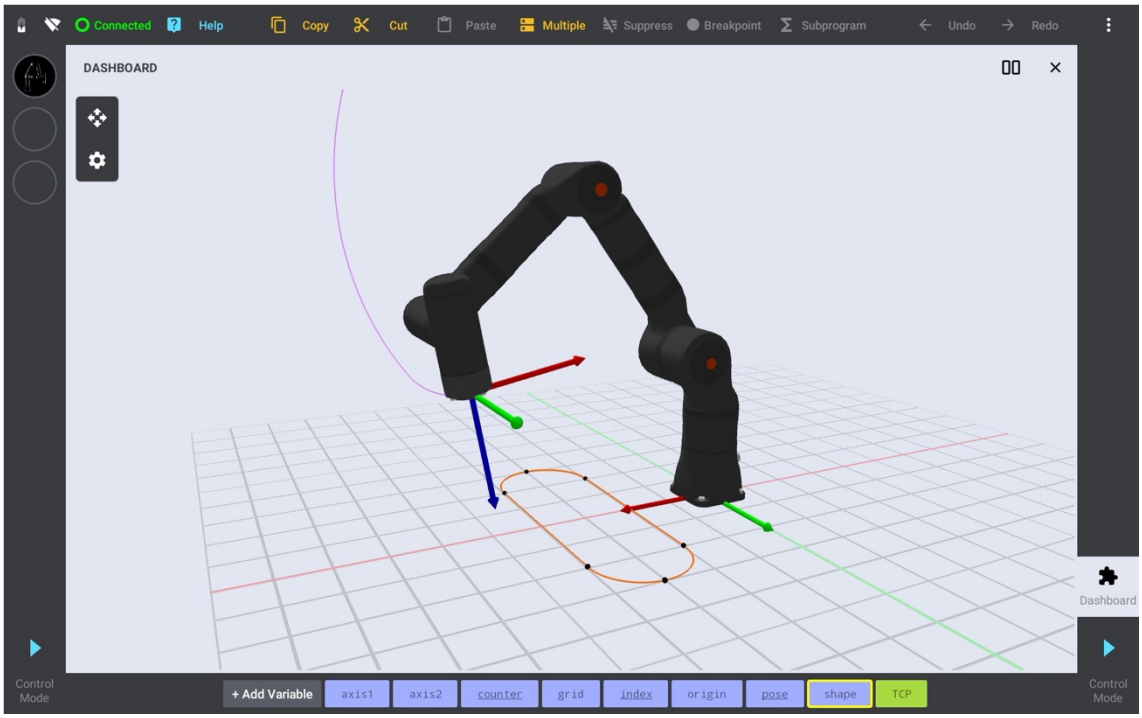


It is important to note, that the maximum number of visualized output frames is limited to 100. All frame markers above this limit won't be visible.

Shape Variable

The Dashboard application enables users to visualize the target trajectory of a selected shape variable. The orange shape path, consisting of line and arc segments, represents the trajectory that a robot will follow during the execution of a corresponding MOVE shape command. The trajectory path is accompanied by black spherical markers, indicating the input pose markers of the shape variable.

Please note that the pose visualization is available only if a single shape variable is selected.



Nitro Tools

The Nitro Tools Cbun provides a suite of tools empowering robot programmers to define pose variables in a more intuitive and sophisticated way. The tools are categorized into two types: Define Pose tools, tailored exclusively for custom pose variables, and Define TCP tools, primarily intended for adjusting TCP system variable.

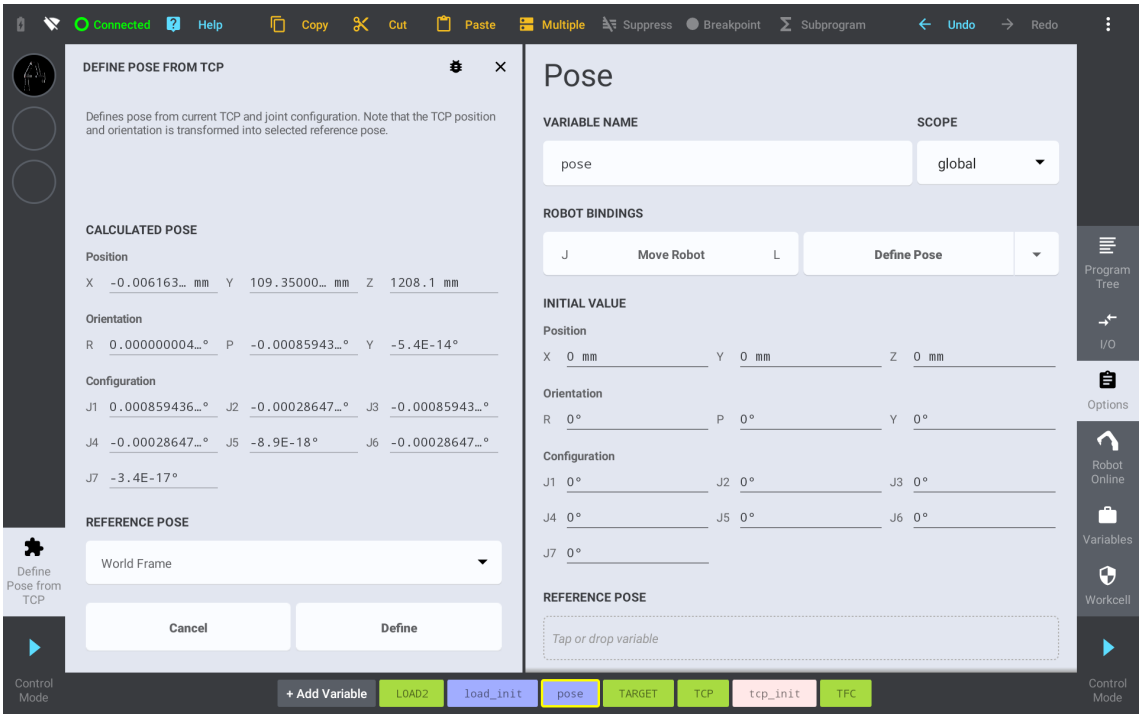
Define Pose from TCP

The Define Pose from TCP tool serves as a valuable feature for assigning the real-time TCP (Tool Center Point) position and orientation into a selected custom pose variable. The position (X, Y, Z) and orientation (roll, pitch and yaw) coordinates of the TCP frame are expressed within a selected reference pose. Furthermore, the Define Pose from TCP tool updates the joint configuration (J1 to J7) of the selected pose variable by using the actual position of individual joints.

This tool plays a vital role during robot programming, allowing users to effortlessly re-define a pose variable that can later serve as a direct target for both Workspace and Jointspace MOVE commands.

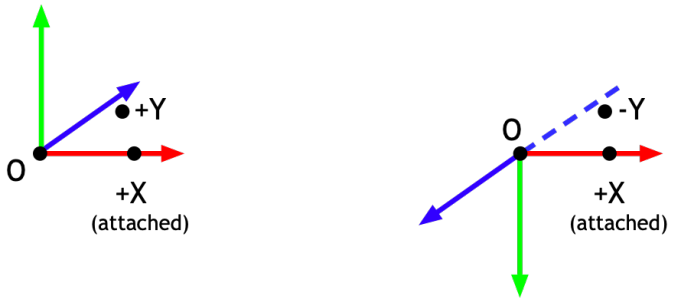
To redefine a selected pose variable from the actual TCP, open variable's options panel, click on the **Define Pose** button and utilize the Define Pose from TCP tool on the opposite screen side. Choose a reference pose, verify the calculated coordinates and update the variable by clicking on the **Define** button.

It is important to note, that the Define Pose from TCP tool is not available for the system pose variables, such as Base or TCP frame.



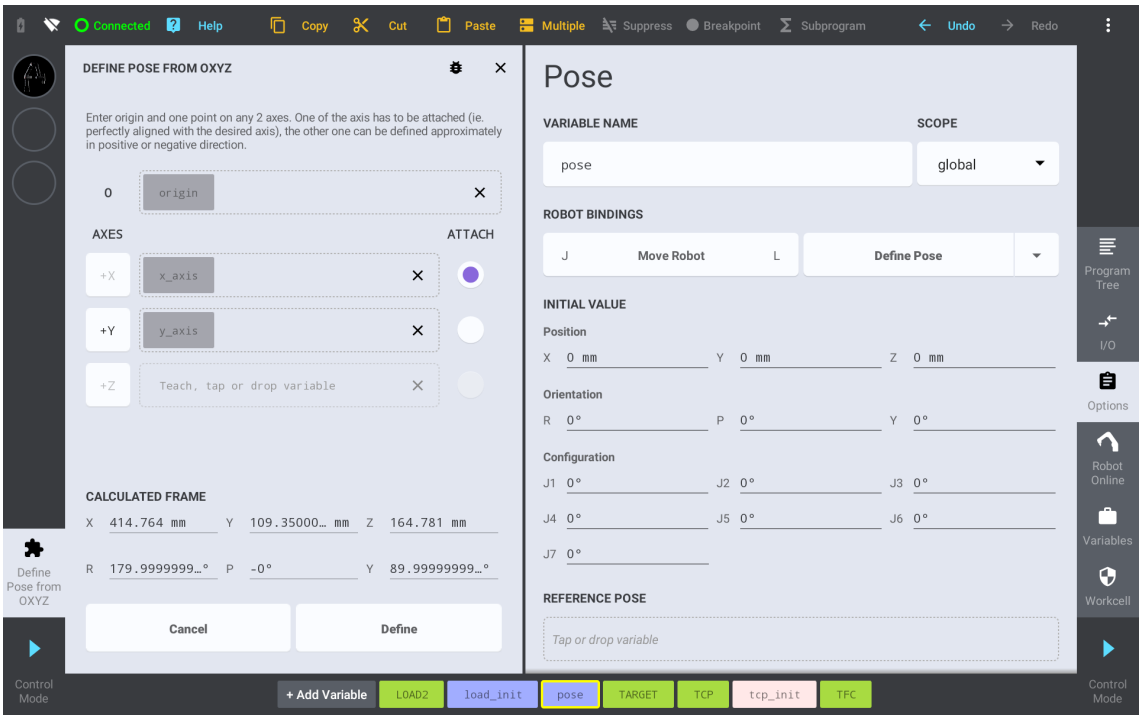
Define Pose from OXYZ

The Define Pose from OXYZ tool empowers the robot programmer to redefine a selected custom pose variable by teaching frame origin along with a point on the XY, XZ or YZ axes. This feature provides a practical and user-friendly method for specifying precise coordinate systems such as palette frame.



Marking an axis as **attached** indicates that the taught point lies directly on the frame axis. The other axis can be defined by teaching an approximate axis point either in positive or negative direction of the axis.

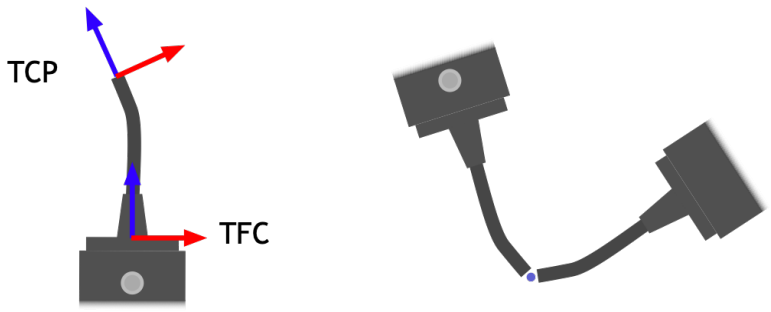
To redefine a selected pose variable as a precise frame, access the variable's options panel, expand the **Define Pose** menu and select the **Define Pose from OXYZ** tool. Specify the origin and axis points by maneuvering the robot's TCP to the desired positions and double-clicking the toggle button to teach each point. Alternatively, points can be defined by dropping pose variables into the corresponding boxes. Choose the attached axis and configure the direction of the other axis. Finally, verify the calculated frame coordinates and update the variable by clicking on the **Define** button.



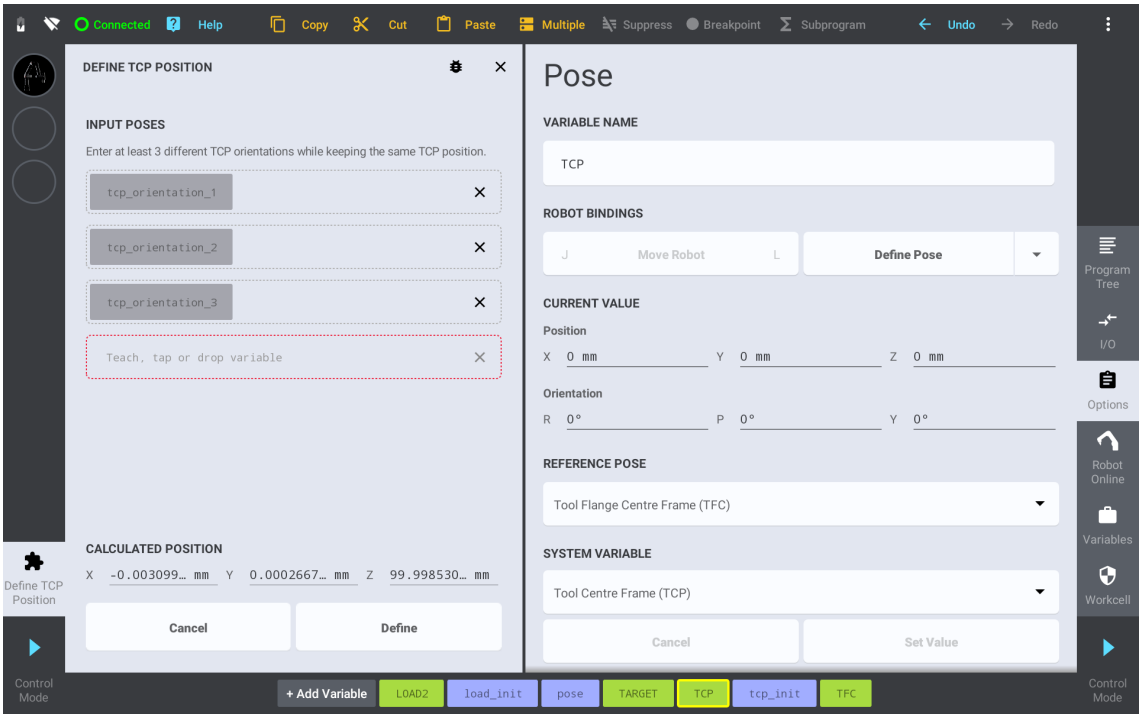
It is important to note, that the Define Pose from OXYZ tool is not available for the system pose variables, such as Base or TCP frame.

Define TCP Position

The **Define TCP Position** tool allows users to automatically calculate TCP of any robot's tool and assign the output into a selected pose variable. To define the TCP position, the robot programmer must teach 3 to 4 poses representing the same TCP position but with different TCP orientations. The calculated TCP position (X, Y, Z) and orientation (roll, pitch, yaw) coordinates are relative to the Tool Frame Center (TFC) frame. This feature is particularly valuable for defining TCP system variable.



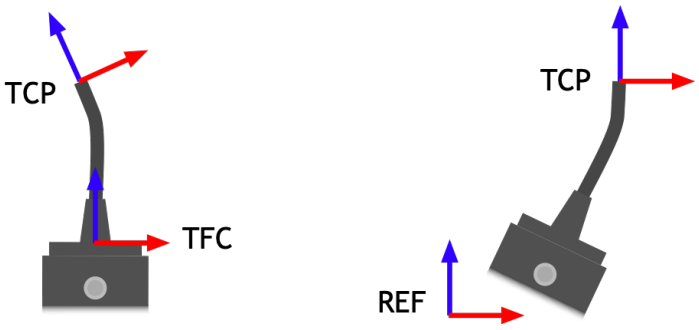
To redefine a selected pose variable as a TCP position, navigate to the variable's options panel, expand the **Define Pose** menu and select the **Define TCP Position** tool.



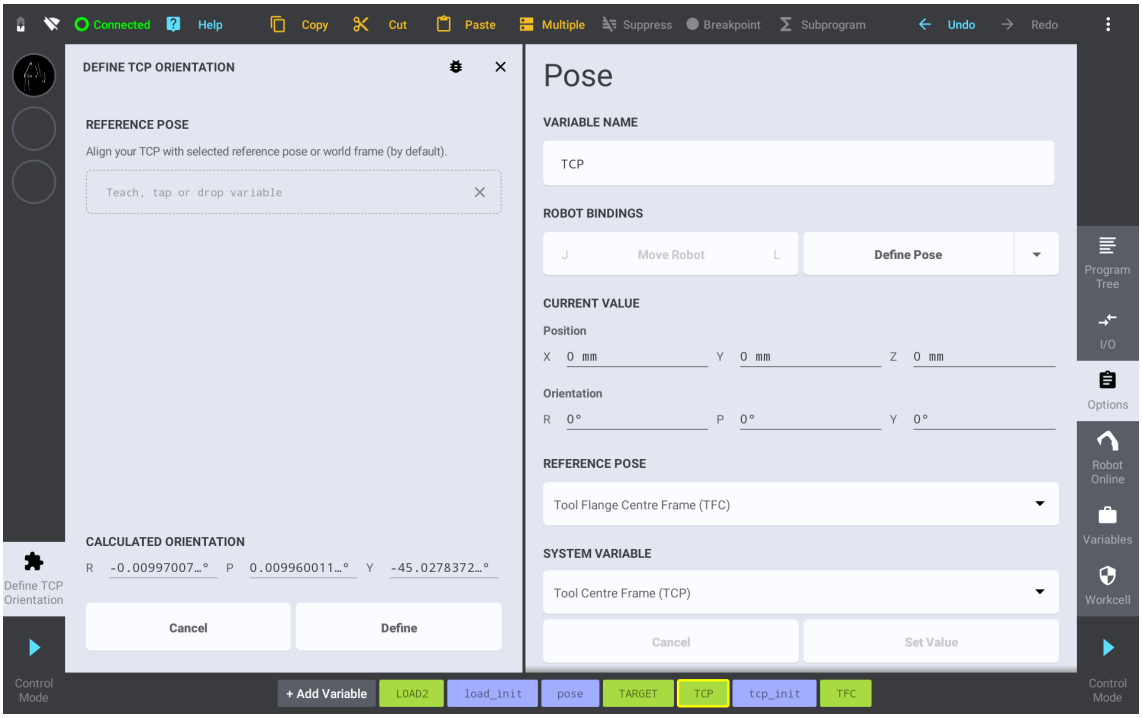
Start by finding some physical reference point in the robot's Workspace, for instance a corner of a table. Move the robot's tool to align the desired TCP position with this the chosen reference point. To teach the pose, simply double-click the toggle button. Teach next pose by keeping the same TCP position but with different TCP orientation. Repeat the same process for 3 to 4 poses. The more poses are used, the higher precision is achieved. Finally, verify the calculated frame coordinates and update the variable by clicking on the **Define** button.

Define TCP Orientation

The **Define TCP Orientation** tool enables robot programmers to automatically calculate the desired TCP orientation (roll, pitch, yaw) by aligning the TCP frame with a chosen reference frame. The resulting orientation is then assigned to a selected pose variable, without impact other pose coordinates.



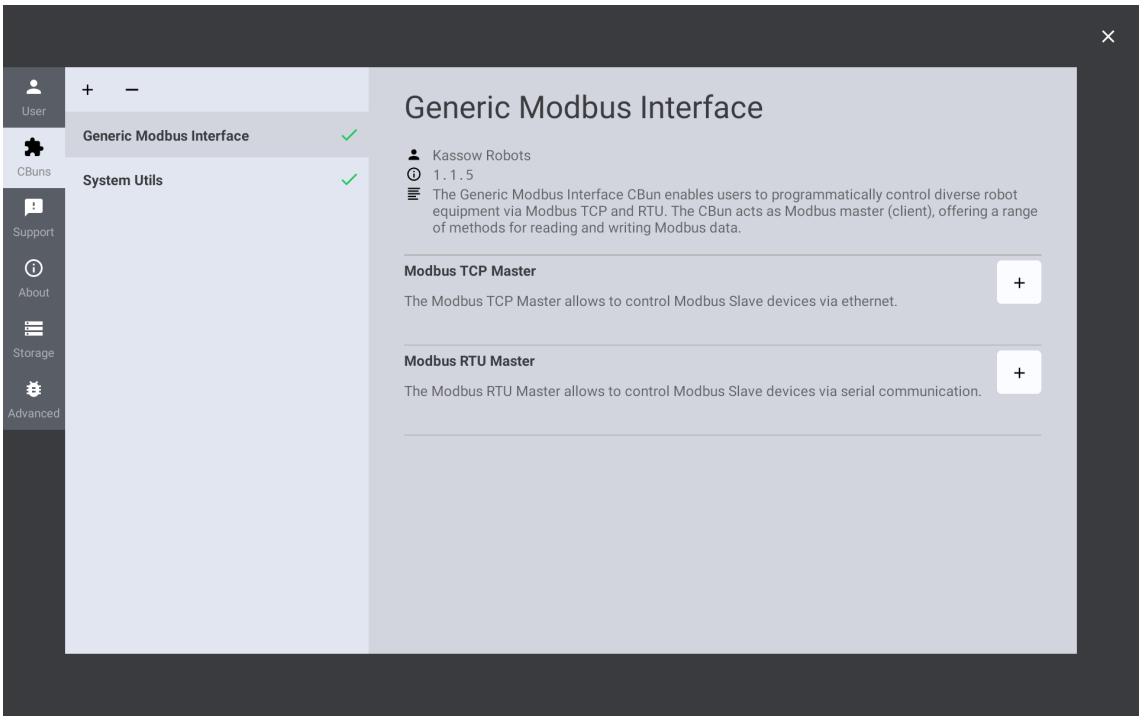
To redefine an orientation of a selected pose variable, navigate to the variable's options panel, expand the **Define Pose** menu and select the **Define TCP Orientation** tool.



Begin by specifying a reference frame with a known orientation. To do this, drop a corresponding pose variable into the designated box. Alternatively, align the robot's TFC frame with the desired reference frame and double-click the toggle button to teach the reference frame. Subsequently, align the robot's TCP frame with the reference frame, verify the calculated orientation coordinates, and update the selected variable by clicking the **Define** button.

Generic Modbus Interface

The Generic Modbus Interface Cbun enables users to programmatically control diverse robot equipment via Modbus TCP and RTU. The Cbun acts as Modbus master (client), offering a range of methods for reading and writing Modbus data.



Modbus TCP Master

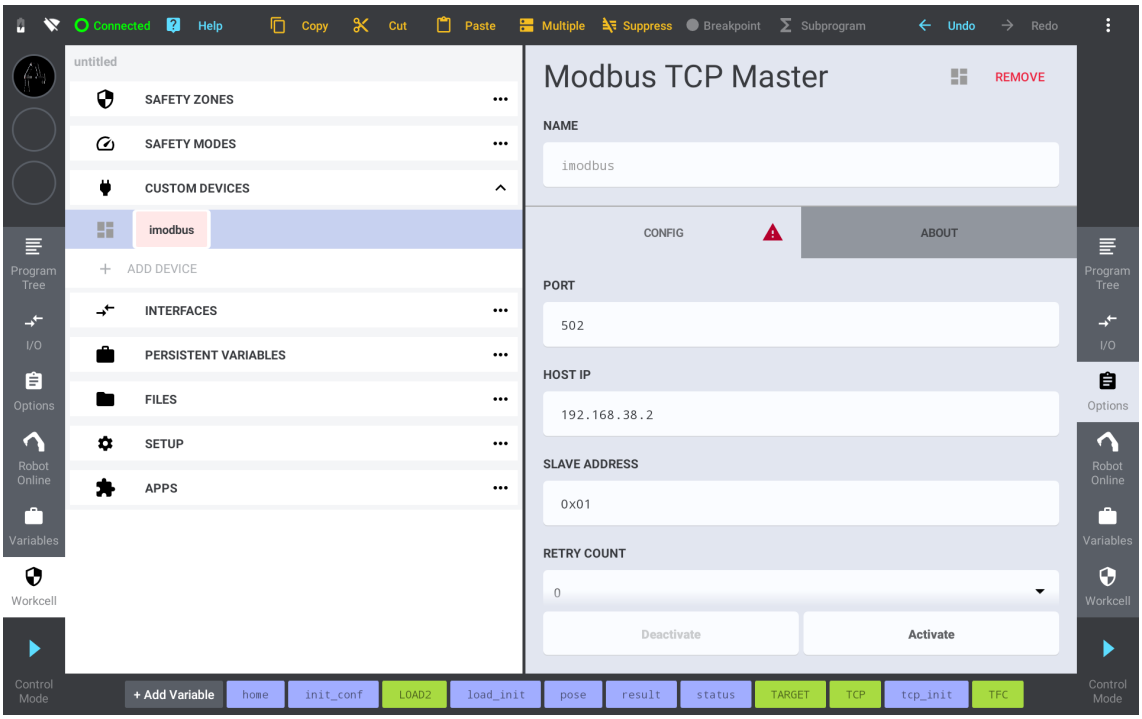
The Modbus TCP Master allows users to control Modbus slave devices via ethernet. The addressing of TCP frames is based on IP address and port number, allowing communication over IP network.

Begin by connecting your robot to an ethernet-based network via its **ethnet** network interface. It is crucial to configure ethnet's IP address and netmask, ensuring that the robot exists within the same subnet as other Modbus TCP devices. Each Modbus slave device typically requires a separate Modbus TCP Master instance. To add a Modbus client into the *Workcell -> Custom Devices* section, click on the Modbus TCP Master **+** button.

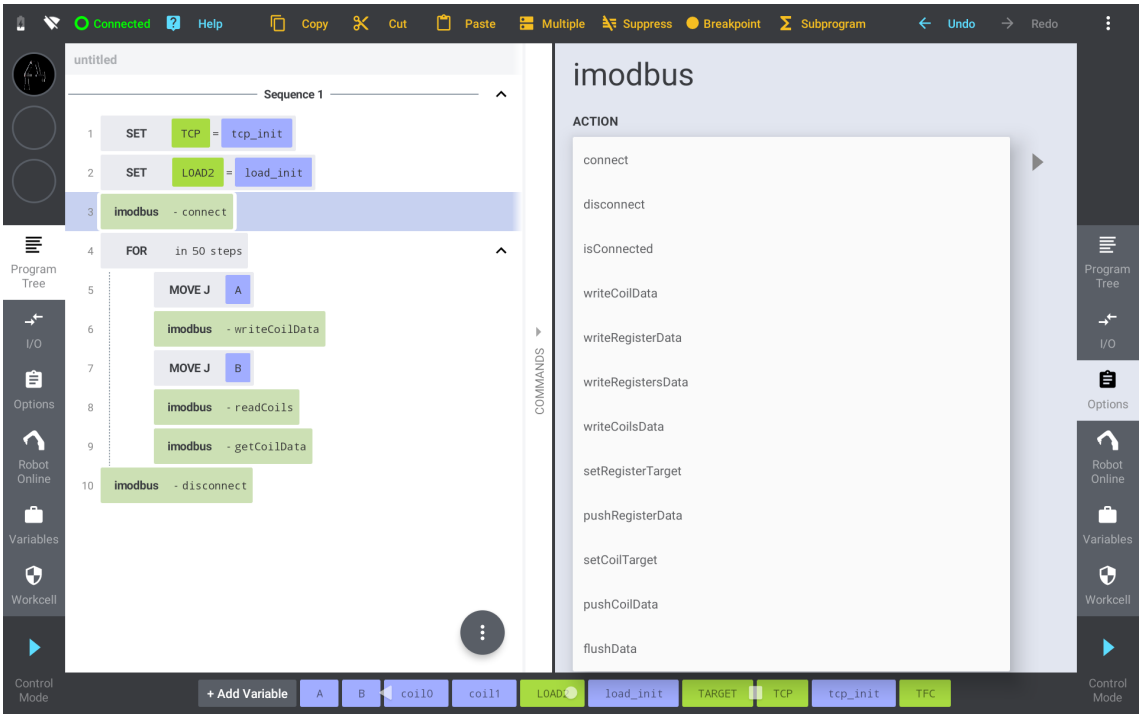
Subsequently, navigate to the Workcell tool, select the desired **imodbus** instance and open its options panel. The options panel provides access to a list of parameters, allowing you to configure the key properties of a Modbus TCP communication, such as **Port**, **Host IP** and **Slave Address** of a desired robot peripheral. Additionally, the user has the option to configure an automatic retry setting.

Furthermore, the user can enable **Connect on Start**, enabling each program sequence to automatically establish a connection with the device. It is crucial to note, that certain Modbus slave devices may not permit multiple TCP/IP sockets. In such instance, the robot programmer must manually manage the connection by using **connect** and **disconnect** Modbus TCP commands, ensuring the maintenance of a single open TCP/IP socket.

Finally, click on the **Activate** button to apply the configuration and enable the Modbus for programming.



To access Modbus interface within a robot program, drag the **imodbus** command from the Commands Box, drop it into a desired position in the Program Tree and select one of the actions.



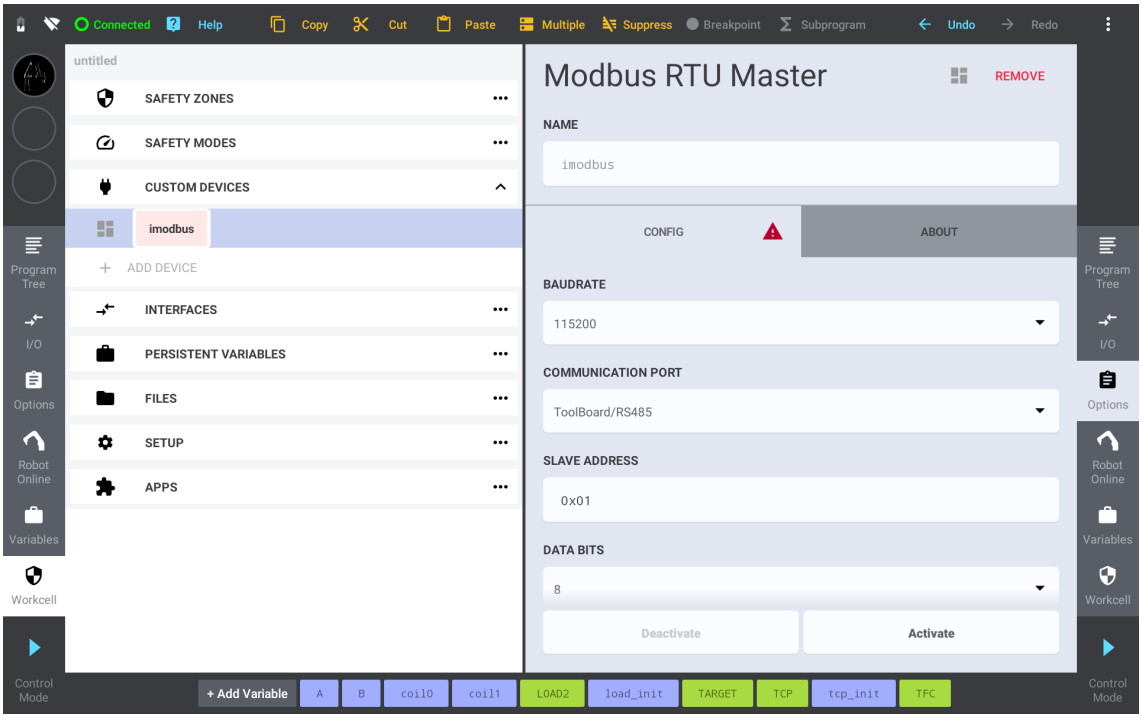
Modbus RTU Master

The Modbus RTU Master allows robot programmers to control Modbus slave devices via serial communication. The addressing of RTU binary frames is based on slave address, allowing to command multiple devices on the same serial communication bus.

Begin by attaching your robot to serial communication bus via corresponding serial ports on IO Board or Tool IO. Alternatively, you can use a USB to Serial convertor to get an additional RS-232 or RS-485 interface. Each Modbus slave device typically requires a separate Modbus RTU Master instance. To add a Modbus client into the *Workcell -> Custom Devices* section, click on the Modbus RTU Master **+** button.

Subsequently, navigate to the Workcell tool, select the desired **imodbus** instance and open its options panel. Specify the robot's **Communication Port**, and configure the serial communication properties, such as **Baudrate**, **Data Bits, Stop Bits** and **Parity**. Proceed by entering the Modbus slave device address. Additionally, you can configure an automatic retry setting.

Finally, click on the **Activate** button to apply the configuration and enable the Modbus for programming.



To access Modbus RTU interface within a robot program, drag the **imodbus** command from the Commands Box, drop it into a desired position in the Program Tree and select one of the actions.

Appendix E – Shape Errors

The Shape variable's options panel provides extensive error reporting system. The following table lists all error messages and their corresponding description.

Message	Description
The marker has an invalid coordinate	Any coordinate of the marker pose is NaN.
The first segment of an open shape must be line	The arc is used as a corner whose position is determined by preceding and succeeding line. Therefore, open shape must start with line.
The last segment of an open shape must be line	The arc is used as a corner whose position is determined by preceding and succeeding line. Therefore, open shape must end with line.
Not enough markers to determine the segment	A line needs at least 2 markers, an arc needs at least 1.
A corner must be followed by a line	The arc is used as a corner whose position is determined by preceding and succeeding line. Therefore, two consecutive corners are invalid.
A closed shape must not both begin and end with corner	If the shape is closed, the last segment is the preceding segment of the first segment. The arc is used as a corner whose position is determined by preceding and succeeding line. Therefore, the one of the first and the last segment must be a line.
The first marker must set the speed (no change is indeterminate)	Set speed to the first marker of the first segment. Every marker except for the first one can have "no change" speed, which means, that the speed is the same as of the previous marker.
The first marker must set the orientation (no change is indeterminate)	Set orientation to the first marker of the first segment. Every marker except for the first one can have "no change" orientation, which means, that the orientation is the same as of the previous marker.
The markers are in wrong order	The order in which the markers appear on shape is different from the order they are mentioned in the shape definition. For example, the definition says that the line consists of point A, B, C, but those points lay on the line in order A, C, B.

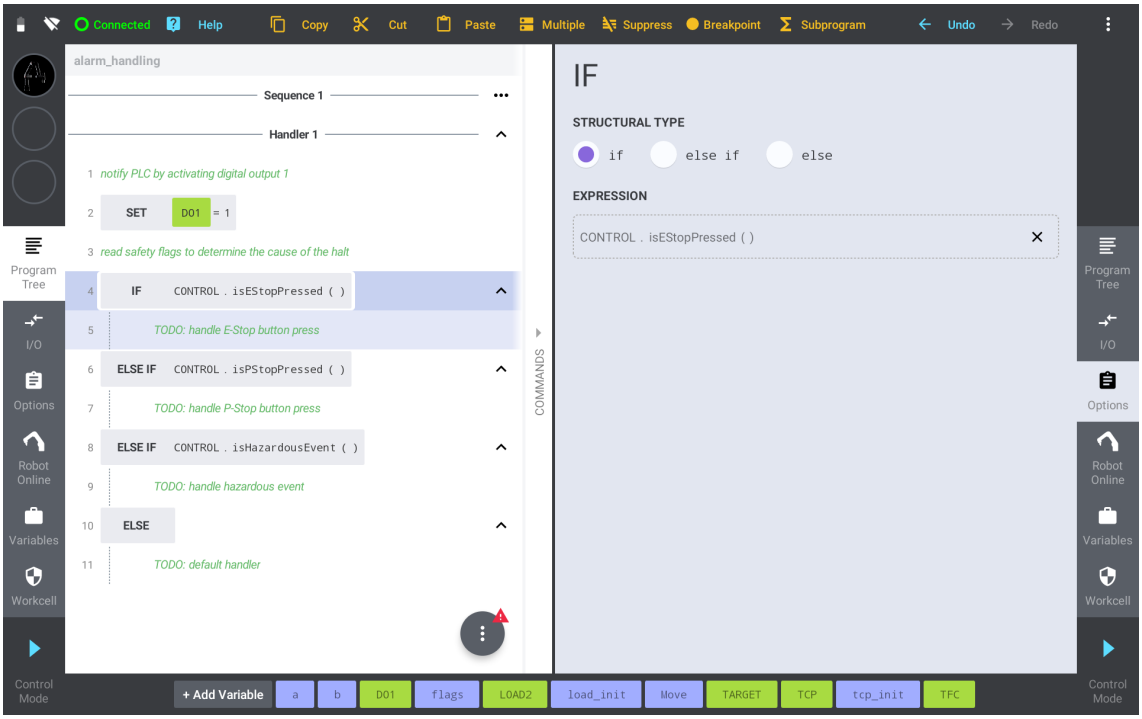
Appendix F – Alarm Handling

The execution of a robot program can halt due to several reasons, such as press of the E-Stop or P-Stop button, safety violation or command exception. By default, the Program Halted dialog appears, prompting the user to resolve the issue manually. Alternatively, the robot programmer can define an alarm handling sequence, allowing the robot program to handle the alarm event automatically.

A typical alarm handling sequence serves to notify a master controller, such as a PLC, of the halt state through robot I/Os. Moreover, it can instruct the robot's equipment to interrupt any ongoing task, for instance, turning off a welding machine. The handler sequence is designed to either autonomously resolve the issue or await user intervention. And finally, the handler sequence is responsible for either terminating the program execution or clearing the halt and resuming the program execution.

The following example demonstrates a typical structure of an alarm handling sequence. The example is highly dependent on the **Program Control** instance provided by the **System Utils** Cbun. The Cbun is accessible in the robot's built-in Cbun repository and must be installed prior using the **CONTROL** instance commands.

To determine a cause of the program halt, use the **isEStopPressed**, **isPStopPressed** and **isHazardousEvent** functions of **CONTROL** instance.

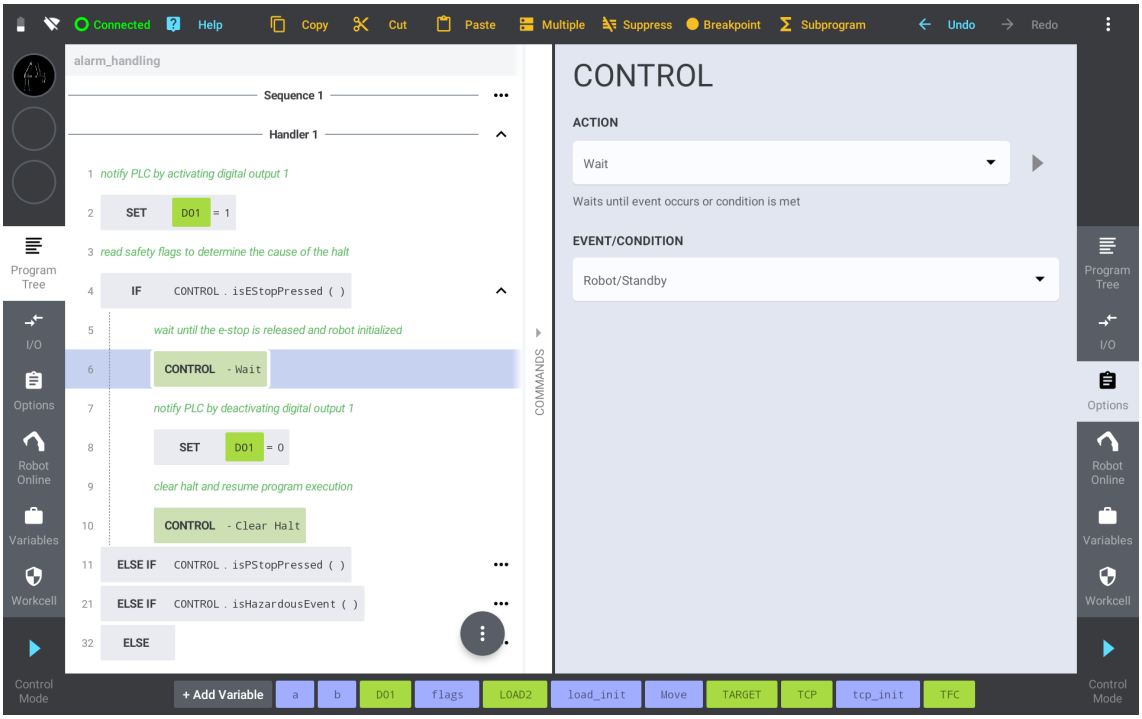


Each cause of program halting requires specific handling. While some halt states can be autonomously resolved, there are instances, such as when the E-Stop button is pressed, where user intervention becomes necessary. This distinction highlights the need for tailored handling procedures based on the nature of the program halt.

E-Stop Handling

The E-Stop safety flag is triggered when any of the E-Stop buttons is pressed, cutting power to the robot arm. Resumption of the robot program is contingent upon releasing the E-Stop button and initializing the robot arm, both of which demand user interaction. The **Wait for Robot/Standby** command is employed to pause the alarm handling routine, awaiting the completion of the robot arm initialization process.

Subsequently, robot I/Os can be utilized to notify any master controller (such as a PLC) about the issue resolution. To complete the process, the **Clear Halt** command is employed, resuming the program execution. Alternatively, you can use the **Terminate Program** command, terminating the program cluster completely.

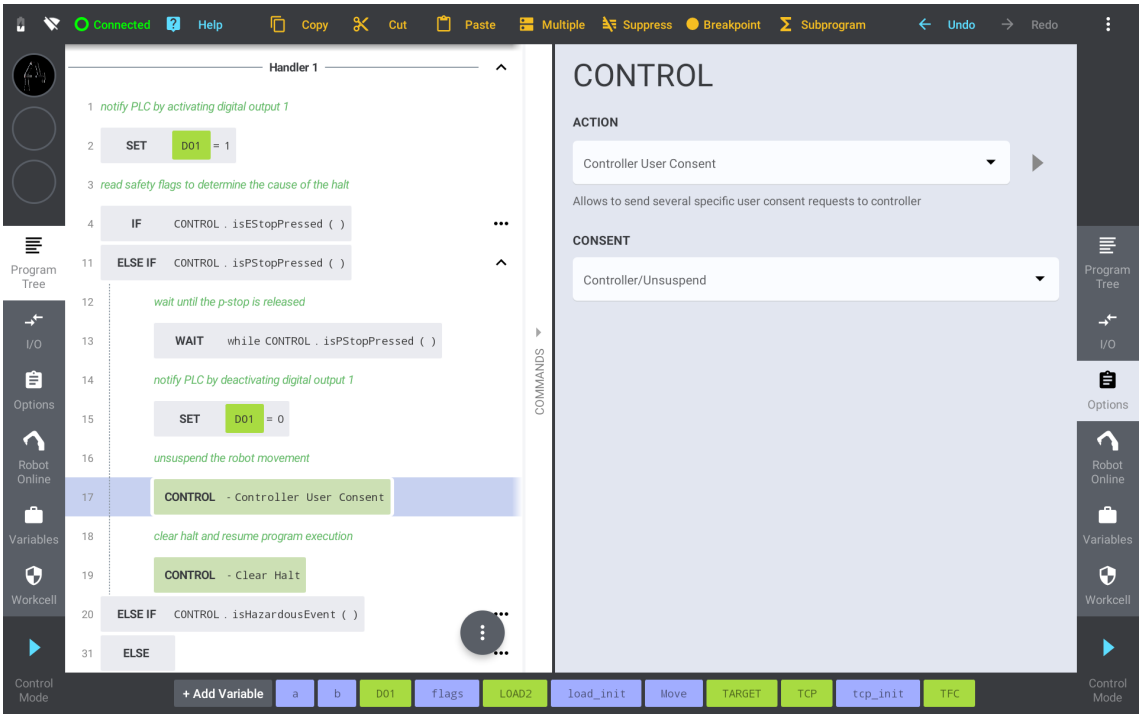


P-Stop Handling

The P-Stop safety flag is activated when any of the P-Stop buttons is pressed, suspending the robot movement, and halting the program execution. Like the E-Stop handling, the user must release the P-Stop button before the program can resume.

Additionally, the, robot I/Os can be utilized to notify any master controller (such as a PLC) about the issue resolution. Subsequently, the P-Stop handling routine must invoke **Controller User Consent – Unsuspend** command to unsuspend any ongoing robot movement.

To complete the process, the **Clear Halt** command is employed, resuming the program execution. Alternatively, you can use the **Terminate Program** command, terminating the program cluster completely.

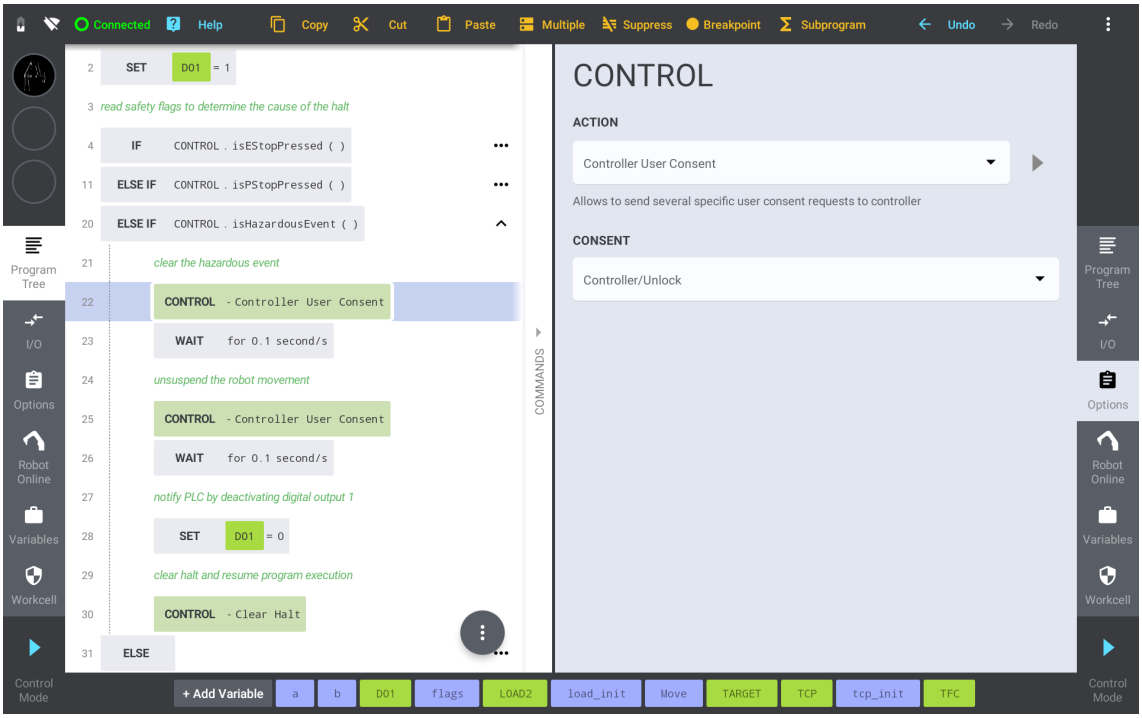


Hazardous Event Handling

The hazardous event is triggered when any of the robot safety conditions are breached, ensuring a safe environment for both robot users and equipment. The hazardous event is typically caused by reaching robot's physical or electrical limits.

To resolve the hazardous event, begin with **Controller User Consent – Unlock** method of **CONTROL** instance, clearing the internal hazardous event flag. Proceed by calling **Controller User Consent – Unsuspend** to resume any ongoing robot movement. It is recommended to **WAIT** command in between these methods, allowing the system to process them.

Subsequently, robot I/Os can be utilized to notify any master controller (such as a PLC) about the issue resolution. To complete the process, the **Clear Halt** command is employed, resuming the program execution. Alternatively, you can use the **Terminate Program** command, terminating the program cluster completely.


Connected
Help
Copy
Cut
Paste
Multiple
Suppress
Breakpoint
Subprogram
Undo
Redo
2 SET DO1 = 1
3 read safety flags to determine the cause of the halt
4 IF CONTROL . isEStopPressed ( )
11 ELSE IF CONTROL . isPStopPressed ( )
20 ELSE IF CONTROL . isHazardousEvent ( )
21 clear the hazardous event
22 CONTROL - Controller User Consent
23 WAIT for 0.1 second/s
24 unsuspend the robot movement
25 CONTROL - Controller User Consent
26 WAIT for 0.1 second/s
27 notify PLC by deactivating digital output 1
28 SET DO1 = 0
29 clear halt and resume program execution
30 CONTROL - Clear Halt
31 ELSE
CONTROL
ACTION
Controller User Consent
Allows to send several specific user consent requests to controller
CONSENT
Controller/Unlock
COMMANDS
Program Tree
I/O
Options
Robot Online
Variables
Workcell
Control Mode
+ Add Variable
a
b
DO1
flags
LOAD2
load_init
Move
TARGET
TCP
tcp_init
TFC