RobotRun User Guide

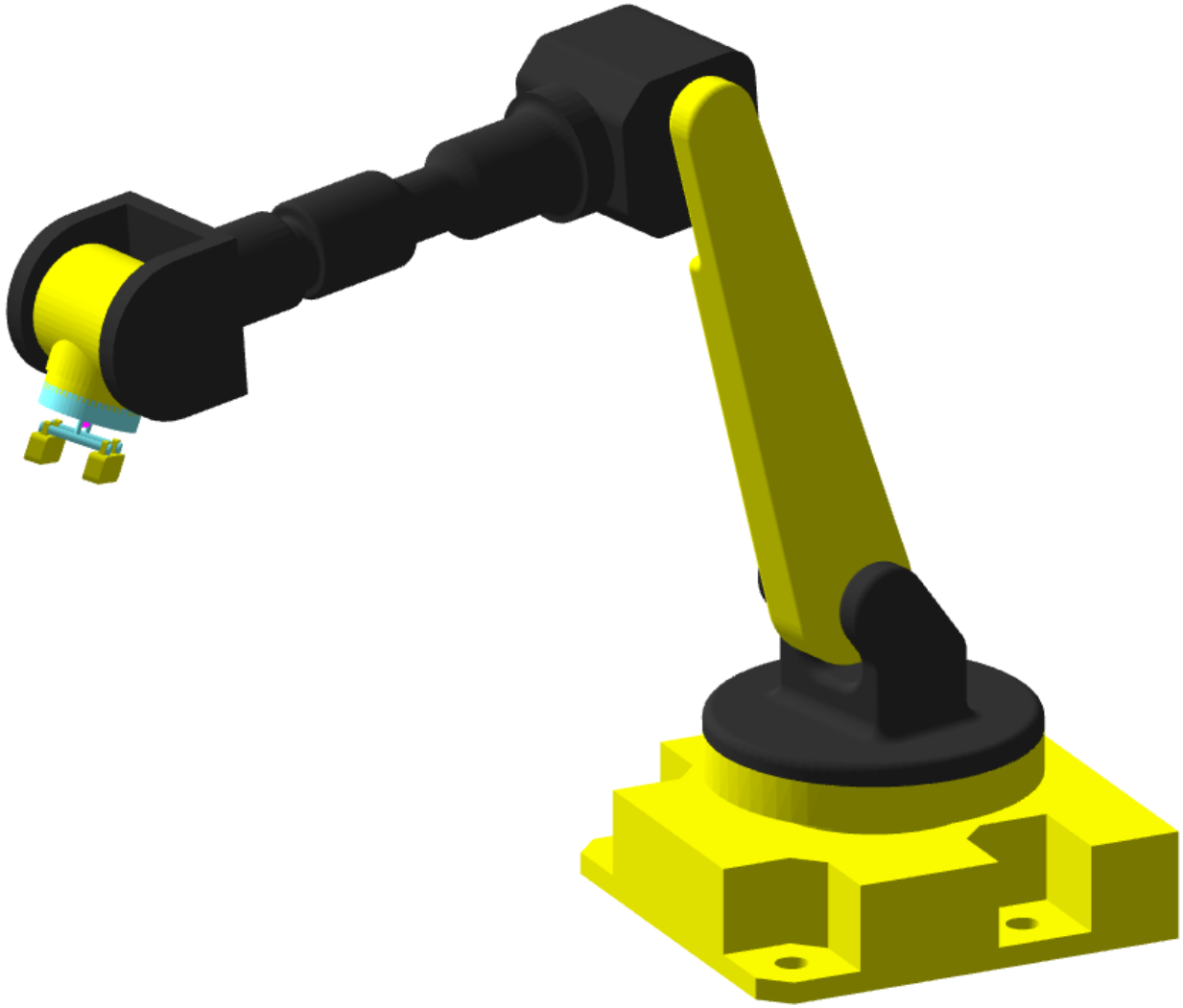


Table of Contents

Disclaimer

For the sake of not writing a novel, I assume that you have had some experience with programming and manipulating a Robotic arm with a Pendant, so this guide will only explain how the concepts of programming a Robotic arm relate to the program, RobotRun.

Installation

The RobotRun project can be accessed on GitHub:

https://github.com/skuhl/RobotRun.git

The most recent build of RobotRun can be found in an archive file (i.e. RobotRun.zip) on the master branch of the repository, though this build may not be the most stable version.

Alternatively, the most recent stable version of RobotRun can be found on this website:

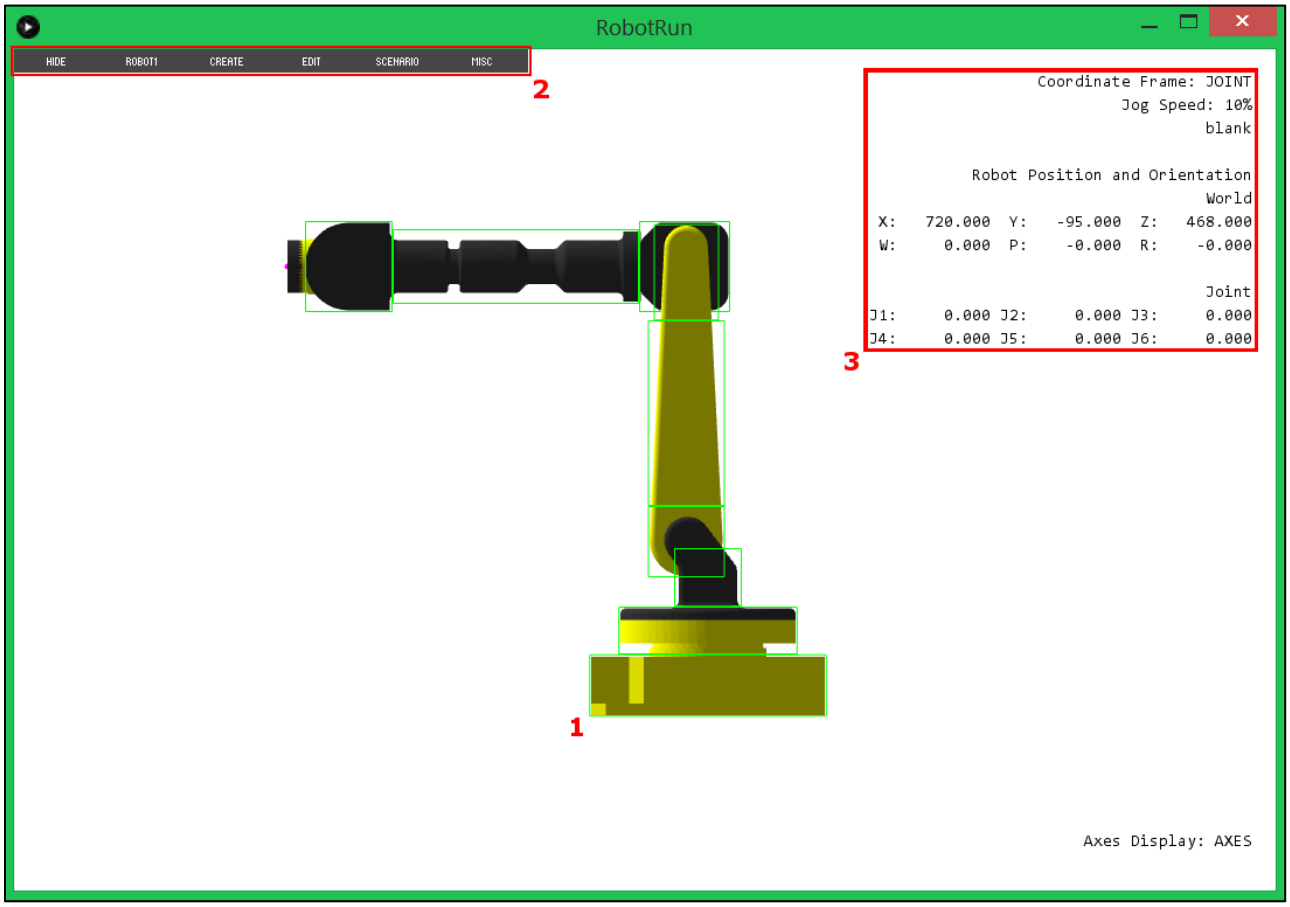
http://www.cs.mtu.edu/~kuhl/robotics/

After downloading the relevant version of the program, unzip it somewhere on your hard drive. If you have another version of the RobotRun software already on your computer, then you can copy the tmp subdirectory from that version of RobotRun and paste it into the new version of the RobotRun software, thus you will have access to all the data from the old version.

In order to use the video/audio recording functionality, you will need to install ffmpeg (https://www.ffmpeg.org/).Running the Program

Navigate to the folder where you unzipped it and “RobotRun.exe” (if on Windows) or just “RobotRun” (if on Linux).

When you start the program, you will see a screen similar to the following:



1. The robot. The active robot begins centered in the screen and can be control via the *Pendant* window. The user has the option to add a second robot (see Miscellaneous Window section), which is independent of the original robot.

2. Window Tabs Bar. The button bar on top left edge of the screen controls what window is currently active within the program. By default, there are four windows available: *Pendant*, the *Object Creation*, *Object Editing*, and the *Scenario*. The *Robot 1*, *Robot 2,* and *Camera* windows are hidden (see miscellaneous window section).

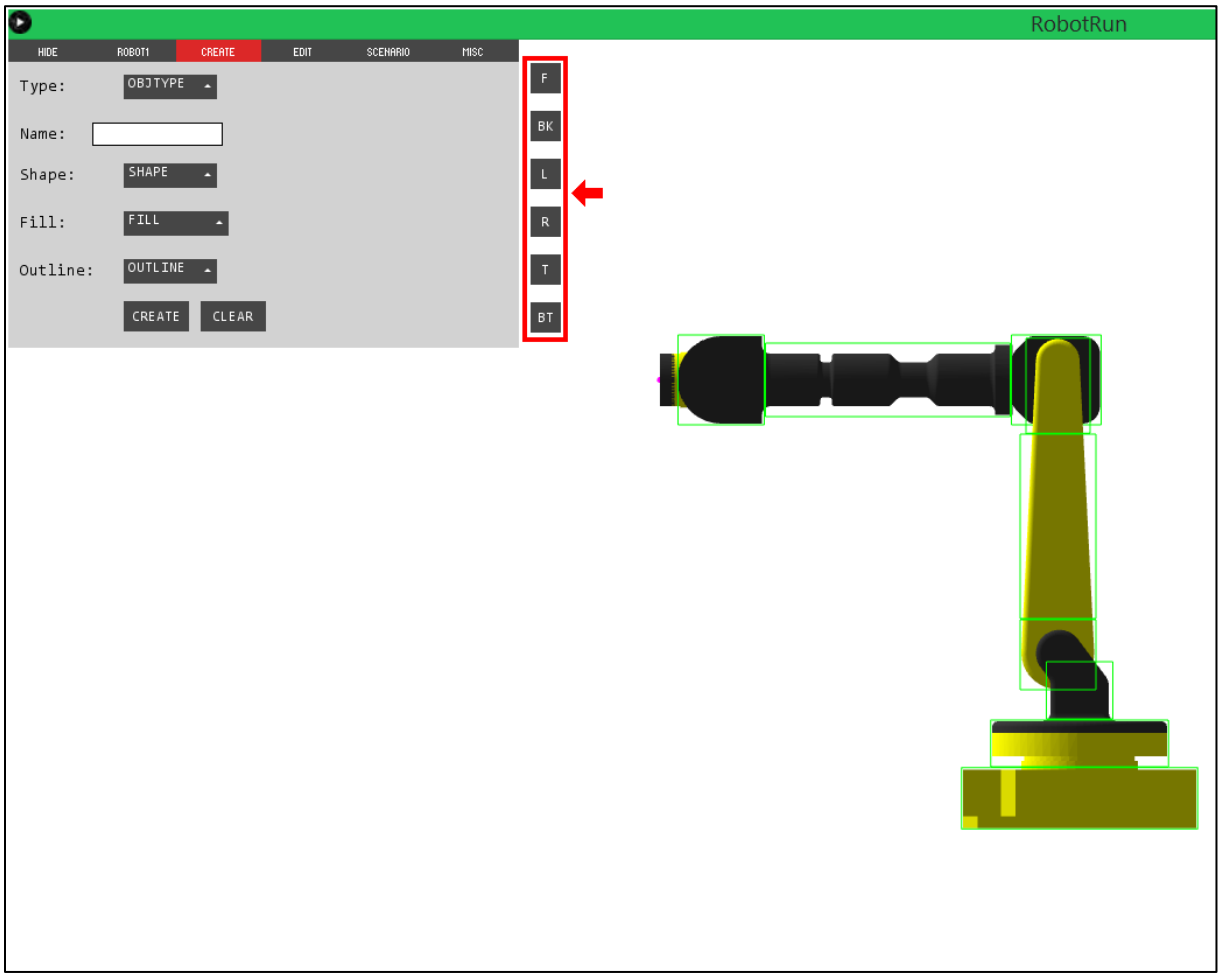
3. Main Display. The far left column of the screen is where a number of fields are displayed. Some common fields are the Coordinate System, the Robot's Jog Speed, the Active Scenario, and the Robot's Joint Angle/Orientation Display. Some fields are variable and are only displayed in certain situations. For example, the Robot's User frame orientation is only displayed when a User frame is active.

Camera Control

Most camera control is done with the mouse:

⇒ Pan the camera by dragging the middle mouse button.
⇒ Rotate the camera by dragging the right mouse button.
⇒ Zoom the camera by moving the mouse wheel.

There are also default views defined by small buttons, which appear alongside the active window's right-hand side.



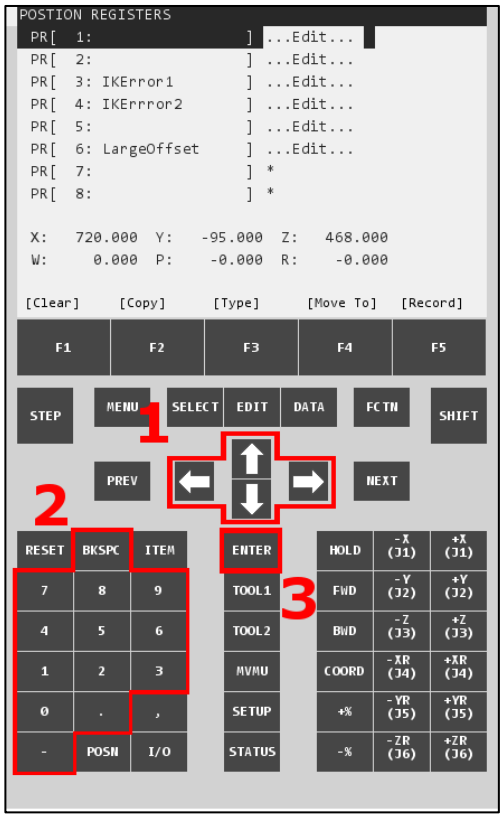
The buttons highlighted in the screenshot above define the six default views:

1. Front View (*F*): the default camera position.
2. Back View (*Bk*): the complement view of the front view.
3. Left View (*L*): facing towards the front of the Robot.
4. Right View (*R*): facing towards the back of the Robot.
5. Top View (*T*): looking down on the Robot.
6. Bottom View (*Bt*): looking up at the Robot.

All of these views' positions are named with respect to the camera's default position (shown in the screen shot above). Though, the descriptions of each view assumes that the *Robot* is in its default position (i.e. all joint angles equals zero).

The Pendant Window

The *Pendant* Window houses the UI used to program and interact with the active robot, teach and edit frames, and save and use *Data* and *Position* Registers. The *active screen* determines the output displayed on the text area at the top of the *Pendant*. The *Pendant* has numerous screens including the program navigation screen, where the list of programs is displayed, and the active frames screen, which displays the current active Tool and User frames.



The Arrow Buttons.

UP, DOWN, LEFT, RIGHT:

The arrow buttons are used to navigate the contents of the active screen, though not all screens implement all arrow buttons.

The Numpad

BKSPC, NUM0 - NUM9, PERIOD, DASH:

The buttons NUM0 - NUM9 are used in combination with PERIOD and DASH to input any numeric input (register values, frames, etc.). BKSPC and the *delete* function (RIGHT, when SHIFT is **on**) are functional for some numeric input as well.

ENTER

ENTER is used to confirm any input as well as confirming many screen transitions.

PREV

Generally returns to the previous screen. Although, some screen transitions invalidate the previous screen function.

When the second robot is added to the application, then the user has the ability to switch between the two robots, although only one robot can be active at a time (the other of which is inactive). The pendant is shared between robots. However, the user interacts with active robot through the pendant, except in the case of robot call instructions.

The Shift and Step Buttons

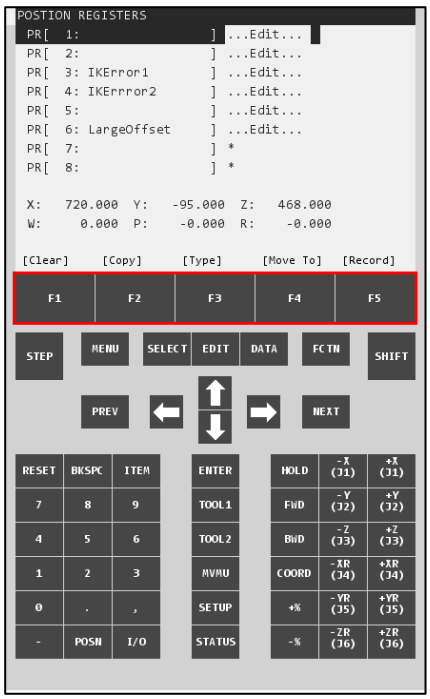
The SHIFT and STEP buttons are toggle able buttons on the far left and right (respectively) of the *Pendant* near the top of the button section. These buttons are highlighted in red when they are **on** and are the normal gray button color when they are **off**.

SHIFT:
In general, any function that would cause the *Robot* to move, or records a position requires SHIFT to be **on** in order to work. The state of SHIFT also influences a number of other miscellaneous things, which will be explained in later sections.

STEP:
Currently STEP has only one function: when it is **on**, running a program will cause the program to execute only the current instruction and move to the next instruction. If STEP is **off**, then running a program will cause the program to run from the currently selected instruction to the end of the program.

The Function Buttons

Five function buttons (F1, F2, F3, F4, and F5) form a bar across the middle of the *Pendant* (as shown in the screen shot below). The behavior of these buttons varies based on the *active screen*. A *function label* will appear above any function keys, which have a role for the *active screen* on the text area above each respective function button, surrounded by square brackets.



Below is a screen shot of the *Pendant* when the *active screen* is data register navigation.

In this instance, F1, F2, and F3 have a purpose, since they each have the *function labels* ([Clear], [Copy], and [Switch], respectively).

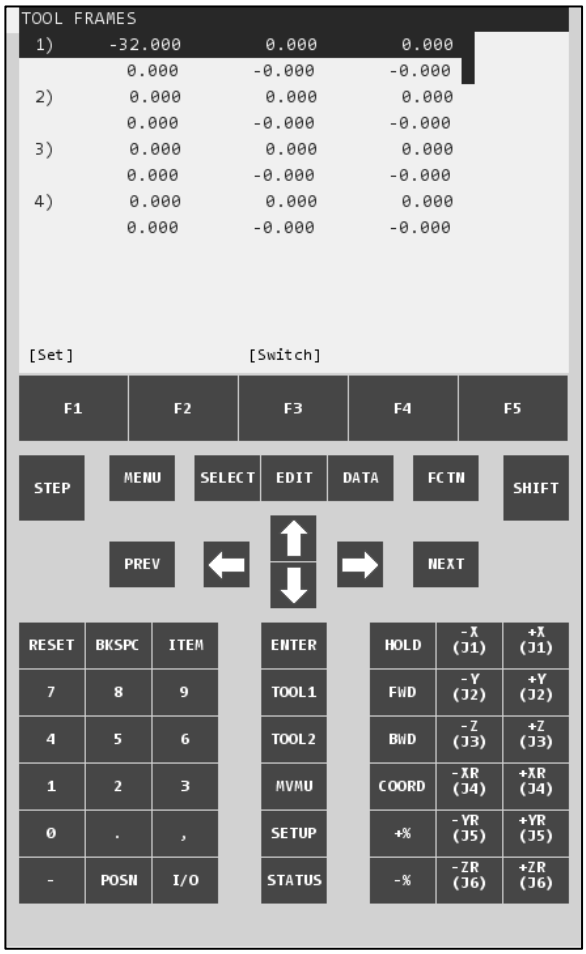
It is important to note that the state of the SHIFT button does influence function buttons in two ways:

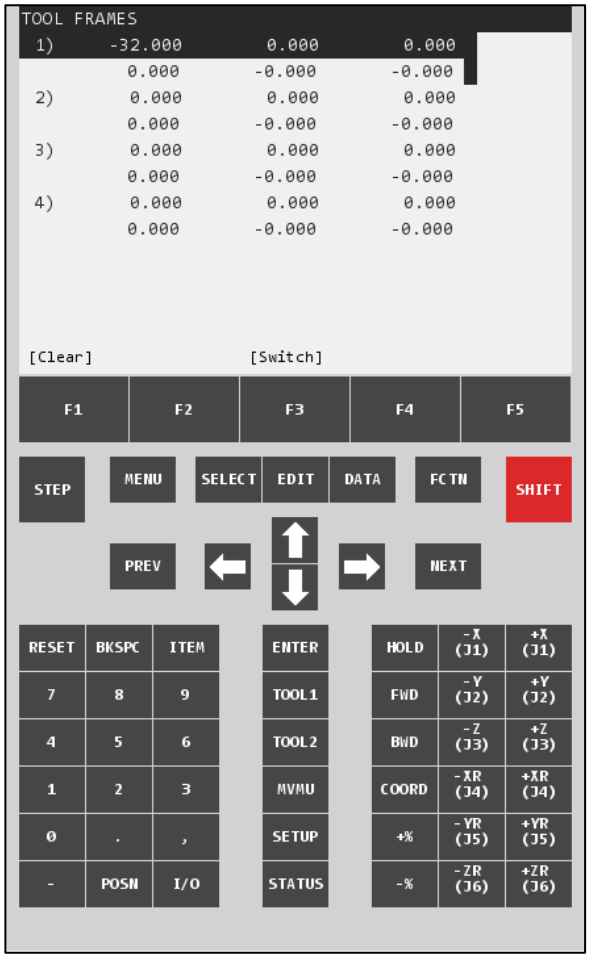
1. Function buttons may have two behaviors: one when SHIFT is **off** and one when SHIFT is **on**.

   For example, when either frame navigation screen is active, the F1 button has two behaviors illustrated by the screen shots below. SHIFT is **off** in the left screen shot and **on** in the right screen shot.

2. Any function that records the position of the *Robot*, or causes the *Robot* to move will only work if SHIFT is **on**.

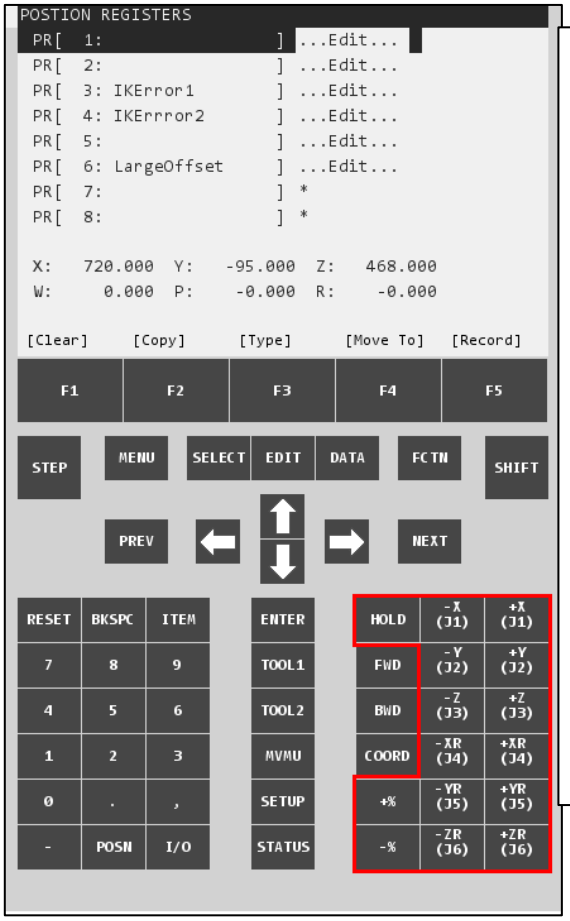
In the tool or user frame teaching screens, the F5 button will move the *Robot* to the currently highlighted *teach point*, if the point is initialized. However, this will only work if SHIFT is **on**. The *function label* is displayed regardless of the state of SHIFT

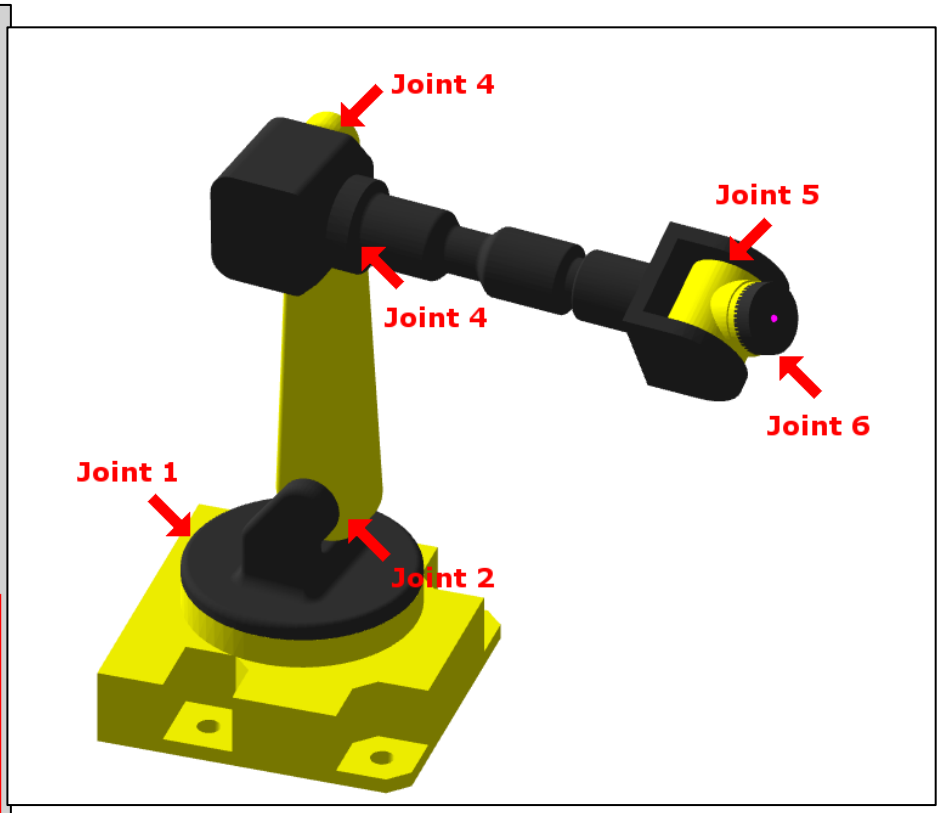




Jogging the Robot

The main buttons involved with jogging the *Robot* are surrounded in red boxes in the left screen shot below and the joints with their respective numbers are displayed in the right screen shot.





The twelve jog buttons form the rightmost column. Each row or pair of buttons corresponds to one axis (or joint) motion. The left button, in the pair, is negative motion and the second is positive motion for that axis (or joint). Each button is labeled based direction of motion as well as the axis (or joint), to which it corresponds. For example, the third row of Jog buttons correspond to the translational motion across the z axis of the active frame (or the motion of the *Robot's* third joint). When the *Robot* is jogging, the jog buttons will be highlighted based on the motion of the *Robot*. So, if the *Robot* is currently moving the +X direction, then the +X/J1 button will be highlighted in red. If the *Robot* is jogging on an axis (or joint) and the button representing that motion or the complement motion is pressed, then the *Robot's* motion on that axis (or joint) will halt.

When the Robot is moving in any other coordinate frame, but Joint, then it is possible that a *motion fault* will occur. The *motion fault* flag indicates that the motion of the *Robot* from some position to another position is undefined by our Inverse Kinematics algorithm. When a *motion fault* occurs, then the Robot will stop moving and not respond to any motion commands until the *motion fault*

flag is disabled. When the *motion fault* flag is active, a message will appear on the right side of the window in red text. Click the RESET button, when SHIFT is **on**, to disable the *motion fault* flag.

-X/J1, +X/J1, -Y/J2, +Y/J2, -Z/J3, +Z/J3:
The top six jog buttons are the *translational jog buttons*, which are involved in linear motion of the *Robot* (or the first three joints' motion) across the x, y, and z axes of the active frame.

XR/J4, +XR/J4, YR/J, YR/J5, ZR/J6, ZR/J6:
The bottom six jog buttons are the *rotational jog buttons*, which are involved in the rotation motion of the *Robot* (or the last three joints' motion) around the x, y, and z axes of the active frame.

HOLD:
The HOLD button stops any and all *Robot* motion.

+%, -%:
The +% and -% buttons control the speed, at which the Robot jogs. The set of possible Robot jog speed values is {1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100}. When SHIFT is **off**, +% and -% will jump to the closest value greater than and less than the current jog speed of the Robot, in set of possible values. When SHIFT is **on**, +% and -% will jump forward and backward respectively to closet of value in the subset {1, 5, 50, 100}.

RESET:
The RESET button will, when SHIFT is **on**, disable the *motion fault* flag and allow the Robot to move again after a *motion fault* occurs.

Main Menu

The main menu is an intermediate screen, which links to the frame, macro, manual function, and I/O register menus. You can navigate the main menu screen with the UP and DOWN buttons. Press ENTER to transition the screen associated with the active line in the main menu screen.

MENU:
Transitions pendant screen to the main menu screen, from which you can access the frame, I/O register, macro, or manual function screens.

I/O Registers

Each robot has a set of I/O registers, one for each end effector. The I/O register value defines the state of the end effector associated with the register. I/O registers can be modified with I/O register instructions, register statements, (see instruction section) or in the I/O register screen. The I/O register screen can be accessed through the main menu screen (see the previous section).

In the I/O register screen, you can navigate between registers with the UP and DOWN buttons. Pressing ENTER will toggle the state of the respective I/O register between ON and OFF. Only the

suction and claw gripper are affected by the state of their respective I/O registers, since they are the only end effectors that interact with world objects (see the World Object section for more details).

Frames

Aside from the Joint and World frames, there are 10 Tool and 10 User frames that you can define. By default all user-defined frame are equivalent to the World frame (indicated by X, Y, Z, W, P, and R equal to zero).

Active frames screen

The active frames screen can be reached by pressing COORD, when SHIFT is **on**, in any screen. In this screen, you may set the active Tool or User frame by navigating to the correct line and using the Numpad to input the corresponding integer associated with the frame you wish to activate. Inputting zero for either one frame line will deactivate the active frame. Additionally, the user frame navigation (*uframe nav*) and tool frame navigation (*tframe nav*) screens can be reached from the active frames screen: navigate to the correct line and press F1.

Here is a screenshot of the active frames screen, in which the line for the active User frame is highlight.

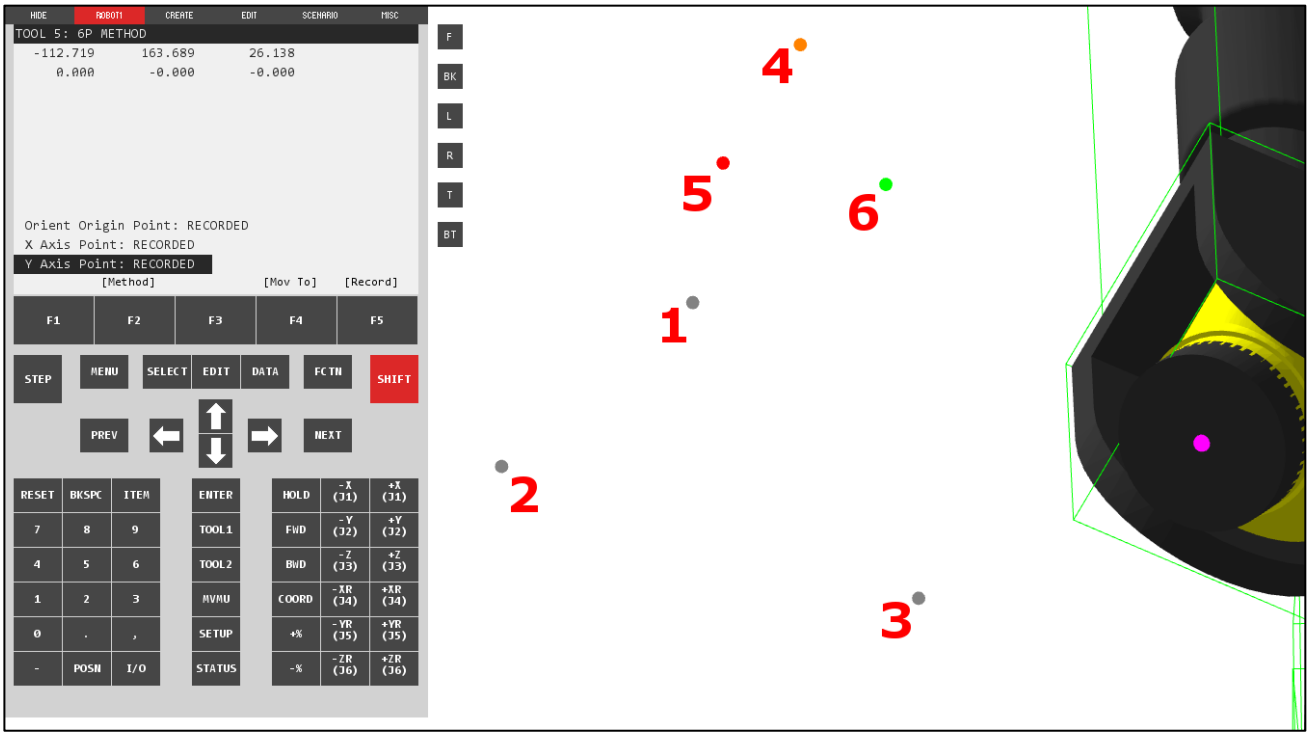
Also, in this example, there are no active Tool or User frames, because the values are zero.

3-Point, 4-Point, and 6-Point teaching methods

To edit a frame you must be in either the *tframe nav* or *uframe nav* screens. Use UP and DOWN to navigate to one of the lines on the *active screen*, which corresponds to the frame you wish to edit and press ENTER. In the next screen press F2 to list the methods of teaching. Depending on which type of frame you are teaching you will have some different options. For this section I will focus on the *teach point methods*: the Tool 3-Point and 6-Point, and the User 3-Point and 4-Point methods.

All the *teach point* methods are taught in the same way. Simply jog the *Robot* to a position and press F5, when SHIFT is **on**, to save the current position of the *Robot*. The *active screen* display for all *teach point* methods will display a list of points and an indication as to whether or not that point has been taught. You can use the UP and DOWN buttons to highlight different points. However, F5 will save the *Robot's* position to the currently selected point replacing the existing value of the point!

Any saved points will also be displayed in the environment as small colored points, when teaching with a *teach point* method. The screenshot below illustrates a fully taught 6-Point Tool frame:



Points 1-3. The three approach points are **all gray** and only apply to the tool frame's end effector offset.
Point 4. The Orient Origin point is **orange**.
Point 5. The X-Axis Direction point is **red**.
Point 6. The Y-Axis Direction point is **light green**.
The Axis Origin point is **blue and only applies to the User frame's 4-Point teach method (hence why it does not show up for the Tool frame's 6-Point method)

The points **4 – 6**, in the screenshot, apply to the User frames teach methods as well, since the User frame only teaches the axes. It is worth noting that for both the 3-Point and 6-Point teach methods of the Tool frame, all three points must be close to orthogonal, or else the frame will not be valid. For the teaching of the axes in either the 3-Point or 4-Point User, or the 6-Point Tool teach methods, the X-Axis and Y-Axis points should form orthogonal vectors, each with respect to the Orient Origin point. Though, as long as the three points are **not** collinear, then the frame will be valid.

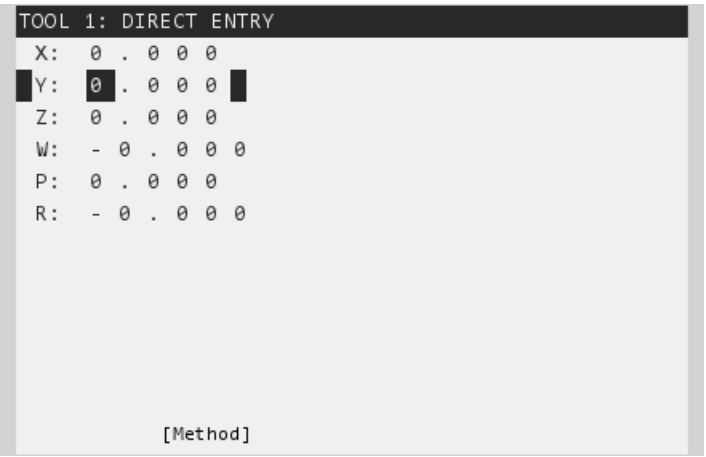
In order for a *teach point* method to be successful:

1. All points must be initialized.
2. For Tool frames, all three approach points must be close to orthogonal.
3. If they exist, the X-Axis, Y-Axis, and Orient Origin points **cannot** be collinear.

Press ENTER to save and update the frame. If the frame is not set to active, or is not updated, then the frame was not valid.

Direct entry teaching method

Alternatively to the *teach point* methods, you may define a frame by specifying the X, Y, Z, W, P, R values, explicitly. Similarly to the *teach point* methods, the direct entry method can be found in each frames view screen by pressing F2. When you enter the direct entry screen, the *active screen* display will look similar to this:



Use UP and DOWN to navigate between the six different input fields for the frame direct entry. Similar to the active frames screen, you will use the Numpad to input each value, except you may input any real number between -9999 and 9999, inclusive. The BKSPC button and *delete* function work in the direct entry screen as well. Once you have specified your desired frame values (all fields must be initialized!), then press ENTER to confirm the direct entry.

I should mention that any points you teach for any *point teach* method as well as the direct entry values of a frame are saved independent of the frame's current active X, Y, Z, W, P, and R values. So, you can teach points for a frame method and not update the frame with the method without fear of losing your taught points. Though, the three approach points are shared amongst the 3-Point and 6-Point methods of a frame. The same is true of the X-Axis, Y-Axis, and Orient Origin points for a User frame.

Setting/resetting frames

Aside from the methods mentioned in the active frames screen section, you can set the active Tool or User frame within the *tframe nav* and *uframe nav* screens respectively. When in the correct in the active screen, highlight either line corresponding the frame you wish to activate and press F1, when SHIFT is **off**. In a similar fashion, you can reset a frame to the World frame (erasing all taught points in the process) by pressing F1, when SHIFT is **on**.

COORD:

There are four types of frames: Joint, World, Tool, and User. When SHIFT is **off**, COORD will transition to the next frame type based on the sequence:

Joint → World → Tool → User → Joint

If no active tool or user frame are set, then the tool and user coordinate frames function the same as the world frame.

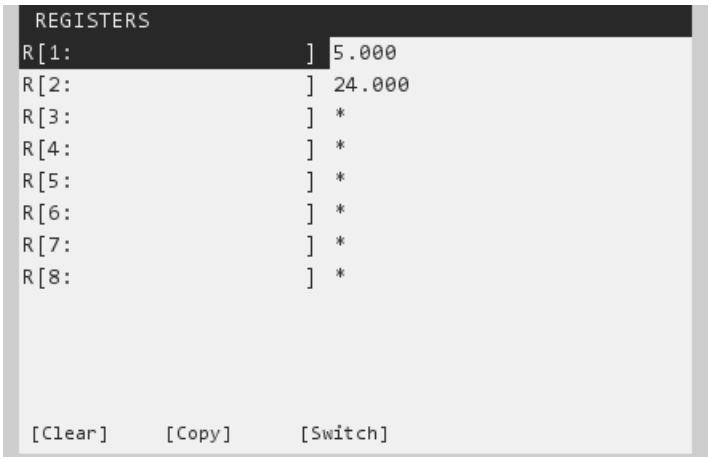
Global Registers

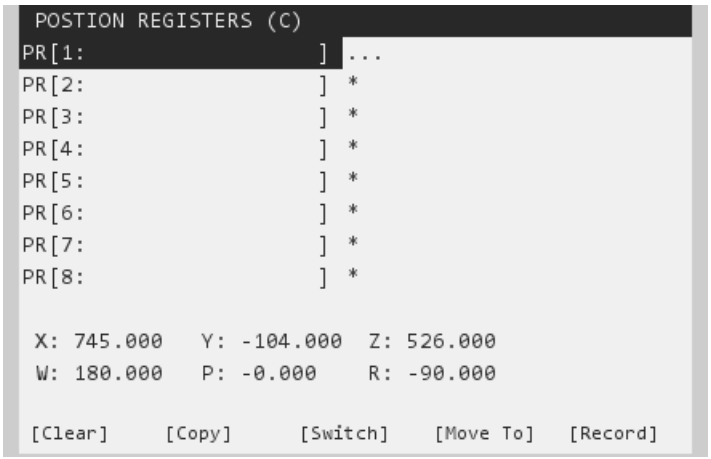
The RobotRun software features 100 data and 100 position registers, which can be manipulated by the user or through program execution. For this section, I will focus on how the user can directly modify global registers. Refer to the Instruction section for information regarding program register instructions.

Navigating to registers

Both register screens can be accessed from the register navigation screen:

The DATA button will transition to the register navigation screen from any other screen. Use UP and DOWN to select the register screen you wish to view and press ENTER. A list of registers will appear in the active screen display similar to the screenshots below:





You can switch between the two register screens by pressing F3, when either screen is active.

Editing registers

To edit the value of a register, use UP and DOWN to select the register you wish to edit, use RIGHT to move to the second column, and press ENTER. For data registers, you will simply be prompted at the bottom of the screen to input a real number value. Use the Numpad buttons to input the value and press ENTER to confirm your input. For position registers, a new screen will appear, with the position values (either joint or Cartesian) displayed with the prefix associated with each value similar to the frame direct entry method. Also, you will input values for a position register in a similar fashion to a frame direct entry as well. Press ENTER to confirm your position entry.

You can also copy the values of an initialized register to another register. Use UP and DOWN to navigate to the register you wish to duplicate and press F2. You will be prompted for the index value of the register you want to copy the selected register into. Use the number buttons (NUM0 – NUM9) to input the register index and press ENTER.

In addition, you can completely clear a register entry (comment included) by navigating to the entry you wish to remove with UP and DOWN followed by pressing F1. There is no confirmation for clearing a register, so if press F1, then the currently highlighted register will be cleared!

Position registers have some special functionality as well. You can move the Robot to the position specified in a position register by navigating to the desired register and pressing F4, when SHIFT is **on**. Moreover, you can save the current position of the Robot into a position register with F5, when SHIFT is **on**. The position registers are treated with respect to the active User frame, or the World frame if no User frame is active for the F4 and F5 functions.

Commenting Registers

You can edit the comment associated with a register by first navigating to the register, of which you wish to edit the comment, with UP and DOWN and then pressing ENTER. The comment edit screen looks similar to this:

You can navigate the comment, character by character, with LEFT and RIGHT. RIGHT will append a blank character to the end of the comment if you move past the last character of the comment until the maximum length of a comment, 20 characters, is reached. Each function button has a sequence of characters associated with it (shown in the function label). When you press a of the function button, the current character you have selected in the comment will be overridden with a value from the character sequence associated with the function button you pressed based on the number of times you pressed that function button in succession. In the case of the example above, the first character in the comment, 'A', is selected. So, if I pressed F3 once, then 'A' would be overridden by 'M'. If I pressed F3 twice in succession, then 'M' would replace 'A', on the first press, and 'N' would replace 'M', on the second press. In this way, you can move to a specific index in the comment string and cycle through the character sequence associated with a function button to build the comment. You can switch between lower and uppercase characters with UP and DOWN and the function labels will change according to the case that is active. Integer values 0 through 9 are also valid characters in the comments. Also, you can use BKSPC and the *delete* function to remove characters from the working comment. Once you have built your desired comment, then press ENTER to confirm the comment.

DATA:

Pressing the DATA button will bring up the register navigation screen, which simply allows you to choose between the two register screens: data and position registers.

Programs

SELECT:
Press SELECT to transition to the program navigation screen, which displays a list of all saved/created programs. Use UP and DOWN to navigate the list of programs.

EDIT:
EDIT normally, will load the program instruction navigation screen for the last edited program. If no program has been selected to edit, then the program navigation screen is loaded instead. Furthermore, in the program navigation screen, EDIT functions the same as ENTER.

Creating a program
Transition to the program navigation screen and press F1 to create a program. Then you must name the program, which is an identical process to commenting registers. Press ENTER to confirm the program name. You can rename a program by navigating to the program you want to edit and pressing F2. In the same way you can delete or copy a program by pressing F4 and F3, respectively. Though, when you copy a program, you must provide a name for the copy and you must confirm program deletion with F4 or cancel with F5.

Editing programs
To edit a program, you must first transition to the instruction navigation screen of a program. This screen can reached by pressing SELECT, navigating to the program you wish to edit and pressing ENTER.

Saving programs
When you open the program navigation screen or press the 's' key on the keyboard all programs are saved. Programs are not saved under any other circumstance!

Running programs

FWD:
When in the instruction navigation screen of a program, you can execute the program, from the currently selected line by pressing FWD, when SHIFT is **on**. If STEP is **off**, then execution of the program will continue on until the end of the program is reached or an error occurs. If STEP is **on**, then only the currently selected instruction will execute.

BWD:
When in the instruction navigation screen, you can execute the previous motion instruction based on the instruction currently selected by the cursor. BWD will only function, when step is set and the instruction prior to the active instruction is a motion instruction.

Instructions

When in a program instruction navigation screen, you can view all the instructions associated with a program and navigate the lines of instructions with UP and DOWN. Some instructions span multiple columns on a single line and you use LEFT and RIGHT to navigate the columns of a specific instruction. Whenever you create a new instruction (either with F1, when SHIFT is **on**, or F2), the new instruction will override the instruction, on the line you currently have selected. So, it is wise to only add new instructions at the line denoted with END. If you want to insert instructions into a specific place in the program, then use the insert option, when can be reached by pressing F5 in the instruction navigation screen of a program.

Motion

To create a new motion instruction set SHIFT to **on** and press F2. A motion instruction has at least one position associated with it, which contains the main point of the motion instruction. A point is comprised of angle values and Cartesian values (X, Y, Z, W, P, and R), which define the orientation of the Robot with respect to its joint angles (angle values) and some frame (Cartesian values). In addition, each position has a register reference (type and index) and a speed and termination value.

Furthermore, a motion instruction has several fields:

* motion type
  There are three motion types: joint, linear, and circular. A joint motion instruction will move the Robot based on the joint values associated with the main point. A linear motion instruction will move based on the Cartesian values associated with the main point. A circular motion instruction has a secondary position. This type of instruction requires a preceding motion instruction definition in order to execute. The previous position defines the start point and the secondary position defines the endpoint of an arc on the circle, whose center point is defined by the main point of the motion instruction.

* register type
  A motion instruction can reference a global position register (PR), which are shared amongst all programs, or a local position (P), which is specific to a program. In addition, there is a third position type, camera object (WO), which references the position of a world object, in the active scenario based on the perspective of the camera.

* register index
  Both the positions of a program and the global position registers are organized in the form of an array. Hence, the register index of a motion instruction references the position associated with its register index. There are 100 global position registers with indices 1 through 100. Also, each program has 1000 local positions with indices 1 through 1000.

* speed

The speed of a motion instruction is defined based on its motion type. Joint motion instructions has a percent speed, which is within the range of integers, 1 through 100. A linear or circular motion instruction has a speed (mm/s) within the range of integers, 5 through 1000.

* termination
  The termination of a motion instruction is an integer value ranging from 0 to 100. A termination value of 0 will be displayed as FINE on the screen and is the default for all motion instructions. This field only applies when executing multiple motion instructions in succession.

* offset
  You can define a position register offset for a motion instruction, which will be applied to the position of the motion instruction's defined position (both points for a circular motion instruction). The offset is not set by default, so it appears as a blank space! To set an offset, simple move to the last column on the first line of a motion instruction in the pendant screen and press F4, then select PR[...]. Now, you can move over another column on the same line and press R4 again to set the index of the position register, which will be used as the offset. If the position register is defined, then its values, should appear at the bottom of the pendant screen.

The '@' will appear on a motion instruction, if the point of the motion instruction's position reference equals the Robot's current orientation. Moreover, you can manually edit a local position of a motion instruction by navigating to the position label column (i.e. 'P[' or 'GP[') and pressing F5. The position edit screen functions identically to the position register edit screen.

I/O

To create a new I/O instruction, when in a program instruction navigation screen, press F2, then ENTER. I/O instructions modify the state of one of the Robot's end effector registers. You can view the state of the I/O registers and what register references what end effector, when you create a new I/O instruction. Each register can either have the value on or off.

Frame

To create a new frame instruction, when in a program instruction navigation screen, press F2 and navigate to the frames line and press ENTER. A frame instruction will change the current active User or Tool frame. The frame type fields TFRAME and UFRAME refer to active Tool frame and active User frame respectively. The frame index value is an integer value within the range 0 through 10, which refers to a specific frame. A value above 0 will set the active frame to another frame. A value of 0 will remove the active frame without replacing it with another frame.

Register statement

To create a new register statement instruction, when in a program instruction navigation screen, press F2 and navigate to the registers line and press ENTER. Here you will be promoted for the register type and index, where the result of the register statement will be stored.

- R[x] = (…)
  The result will be stored in a data register. The index value can only be an integer from 1 through 100.

- IO[x] = (…)
  The result will be stored in an I/O register. The index value can only be an integer from 1 through 5.

- PR[x] = (…)
  The result will be stored in a position register. The index value can only be an integer from 1 through 100.

- PR[x, y] = (…)
  The result will be stored into a specific value of a position register (i.e. X or W). The index mapping is as follows: 1 -> X or J1, 2 -> Y or J2, 3 -> Z or J3, 4 -> W or J4, 5 -> P or J5, 6 -> R or J6. Just like PR, the position index can only be an integer from 1 through 100.

Once you initialize the instruction, the statement will contain a single uninitialized element. Elements can be added to the expression, edited, and removed. To insert an element into the expression, navigate to an element in the expression and press F3. Then, navigate to the desired operand/operator with UP and DOWN and press ENTER. The new element will be inserted to the right of the selected element and will be uninitialized: A '…' element is an uninitialized operand and a '_' is an uninitialized operator. To edit an element in the expression, navigate to an element in the expression with LEFT and RIGHT and press F4. Then, select the desired element; operands cannot be edited to become operators and vice versa. However, an element can be deleted by navigating to the element with LEFT and RIGHT and pressing F5, although, an expression cannot be empty

The operands that can be used are:

- R[x] Register value of index *x*
- IO[x] IO Register of index *x*
- PR[x] Position Register position of index *x*
- PR[x, y] one value of a Position Register position of index *x*
- (…) subexpression
- Const constant value

R[x], PR[x, y], and Const operands are floating point values and therefore can be used in an expressions together. However, operands PR[x] and IO[x] are positions and boolean values, respectively. So, expressions containing PR[x] operands should not contain other types of operands. Likewise for expressions containing IO[x] operands.

Below is a list of operators and what operands, with which each operator should be used:

Operators		Operands*		
Symbol	Operation	R[x]/PR[x, y]/Const	IO[x]	PR[x]
+	Addition	x		x
-	Subtraction	x		x
*	Multiplication	x		
/	Division	x		
|	Integer Division	x		
%	Modulus (Remainder)	x		

*Rows marked with 'x' indicate the operator of the given row works with the operand(s) of the given column

The subexpression operator has a special relationship with operators. The validity of an operator for a subexpression depends on what type of operands are in the subexpression.

If statement

To create an if statement, when in a program's instruction navigation screen, press F2 and select the If/Select option. There are two types of if statements: the atomic and complex statements. The first option "If Stmt" is the atomic statement and the later, "If (…)", is the complex statement. The atomic statement has only one operator and two operands in the if statement expression, whereas the complex statement can have up to 8 elements in the expression. So, the atomic statement is a quick way to initialize a simple if statement, while the complex statement allows for more flexibility in building an expression.

When you define a simple if statement, you must provide a default operator, which can be changed later. You edit if statements in the same manner as register statements, except that there are addition operators included in if statement expressions: =, <>, <, >, <=, >=, AND, OR, and NOT. The arithmetic operators can be used to compute a value, which can be used by these logic operators. An if statement's expression must evaluate to a boolean value, in order to be considered valid.

The other part of an if statement is a sub instruction, which may be a jump, call or robot call instruction. The last columns of an if statement after the ':' are reserved for editing the sub instruction of an if statement. If the statement evaluates to true, then the sub instruction will be executed, otherwise, the next instruction in the program will be executed.

Select statement

To create a new selection statement, when in a program instruction navigation screen, press F2 and navigate to the IF/SELECT line and press ENTER. Then, navigate to the SELECT Stmt line and press ENTER again.
The format of a select statement is as follows:

SELECT [arg] = [case1] [result1]
[case2] [result2]

…

The argument and case parameters can be either Const or R[x]. The result parameter can be either a jump label of call instruction. To edit any of the paremeters, simply navigate to a parameter with UP, DOWN, LEFT, and RIGHT, and press F4. To add a new case to a select statement, navigate to either a case or a result parameter and press F3. To remove a case, navigate to the case or result parameter of a case and press F5.

Label

To create a label frame instruction, when in a program instruction navigation screen, press F2 and navigate to the JMP/LBL line and press ENTER twice. Label instructions can be used in conjunction with Jump instructions in order to control program flow. Each label should be defined with a unique integer value.

Jump

To create a new jump instruction, when in a program instruction navigation screen, press F2 and navigate to the JMP/LBL line and press ENTER. Then, navigate to the JMP line and press ENTER again. A jump instruction will transition to the label instruction, whose id is associated with it instead of moving to the instruction immediately succeeding it. Jump instructions take a positive integer for the label id reference.

Call

To create a call instruction, when in a program's instruction navigation screen, press F2 and navigate to the Call line and press ENTER. Then, select the program to call and press ENTER again.

Robot Call

Similar to a call instruction, except the user specifies a program from the inactive robot instead of the active robot. This instruction is only valid when the second robot is added to the application (see miscellaneous window section).

Options

When editing a program, you have access to some addition options aside from editing the instructions of the program directly. To access the options menu, you must be on the first column of a line in the program instruction navigation pendant screen. To select an option press F5 and navigate to the line corresponding to the option you want and press ENTER.

Undo

Reverts the last edit or deletion made to the active program.

Insert

Inserts blank instructions into the active program preceding the instruction corresponding to the active line in the program instruction navigation screen. To insert blank instructions, enter the number of lines you wish to insert using the NUMPAD buttons or keyboard and press F4.

Delete
Allows the selection of instructions to remove from the active program. In the delete screen you can select instructions by navigating a line and pressing enter. Selected lines will be highlighted in a different color than the active line highlighting. When you have selected the lines you wish to delete press F4.

Cut/Copy
Allows for the selection of instructions to copy or cut, and paste elsewhere in a program's list of instructions. You can select instructions in a similar fashion to selecting instructions for the delete option. When you have the instructions you wish to copy/cut, then press F4 to copy or F3 to cut. Pressing F3 or F4 should add the paste function to the F2 button, if it is not already active.

So, you can navigate to the line at which you want to paste the set of instructions. Press F2 to paste the instructions.

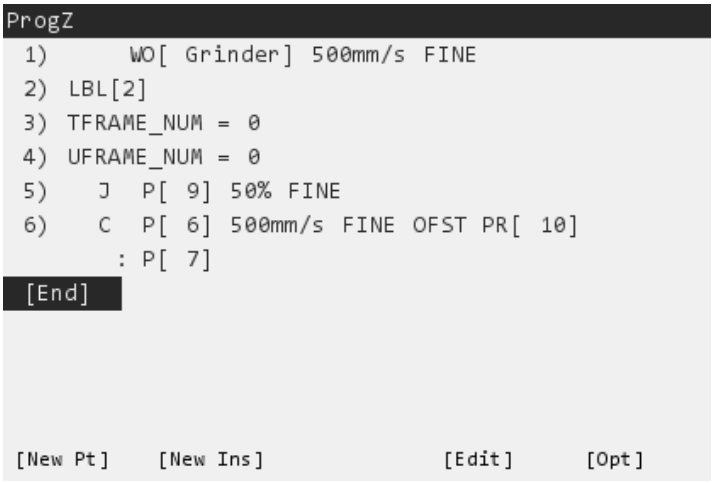
Paste
If you have already copied or cut instructions, then you will be prompted for the way in which to copy the instructions. Each paste type pastes the set of instructions in a slightly different manner:

Type	Effect
Logic	Pastes instructions in normal order, but the pasted motion instruction positions are uninitialized.
Position	Pastes all instructions normally.
Pos Id	Pastes all copied instructions normally, however, the positions are copied to new positions for copied motion instructions.
R Logic	Similar to Logic, but the instructions are pasted in reverse order.
R Position	Similar to Position, but instructions are pasted in reverse order.
R Pos Id	Similar to Pos Id, except the instructions are pasted in reverse order.
Rm Pos Id	Pastes non-motion instructions normally, but reverses the order of motion instruction position IDs amongst all pasted motion instructions.

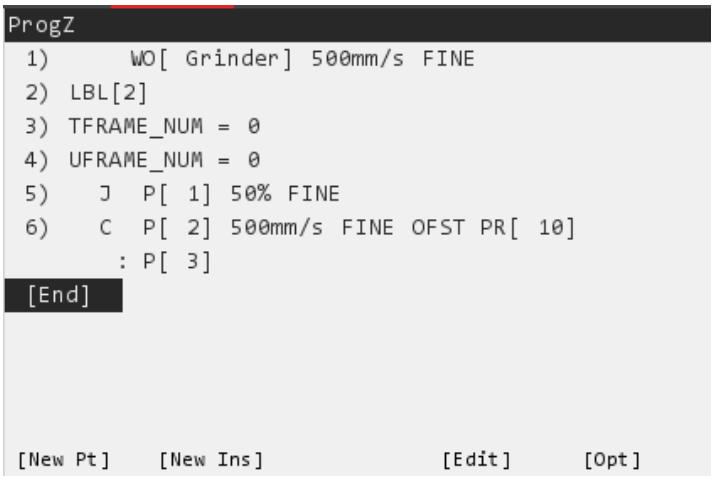
Find/Replace

Renumber
Reorders the position IDs of the active program, in such a way that the position IDs of each motion instruction run in ascending order from the first motion instruction to the last. However, the positions associated with each motion instruction remain the same. For example:

Before renumbering:



After renumbering:



To renumber the position IDs, press F4.

Comment

Allows for commenting/uncommenting of instructions in the active program. You select instructions to comment/uncomment in a similar fashion to selecting instructions for deletion or copy/cut. Press F4 to change the commented state of all selected instructions. Commented appear with a '//' prefix and in a different color then normal instructions. In addition, theses instructions are ignored during the execution of a program.

Macros

You can define macros to execute programs without accessing the program instruction navigation screen of a program. There are two types of macros: user key and manual function macros. Key bound macros associated a program with a specific button on the pendant, while manual function macros are stored in a list, which can be accessed in the manual function screen.

SETUP, STATUS, POSN, I/O, TOOL 1, TOOL 2:
These buttons function as key bound macros. Each button may only have at most one program associated with it at one time.

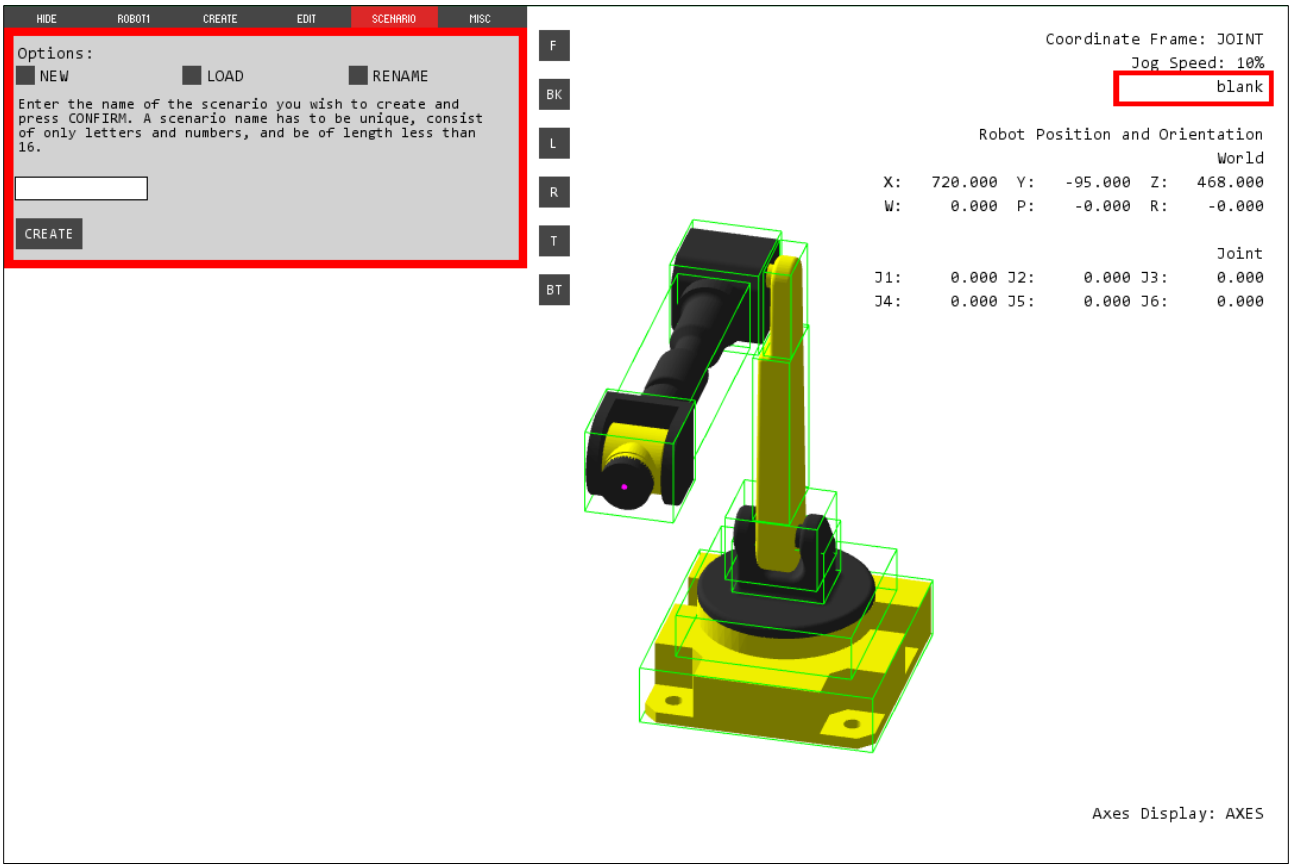
To create a macro, navigate to the macro screen, from the main menu screen and press F1. You will then be prompted for the program for which to define the macro. You can navigate the menu with the UP and DOWN buttons ad press ENTER to select a program. Once you select the program, you will need to decide what type of macro, you want for the program. For user key macros, you will also need to associate one user key with the program. If a macro is already defined for a user key, then you will not be able to define another macro with that user key. You must edit either the program or user key associated with the defined macro.

When a user key macro is defined, then you can execute the program associated with user key button by pressing the button, when SHIFT is **on**. To access a manual function macro, navigate to the manual function screen, through the main menu screen and navigate to the line associated with the manual function. Press ENTER to execute the program.

Marcos can be edited in the macro screen with the F4 button. You can change the program, type (, and button for user key macros) of a macro.

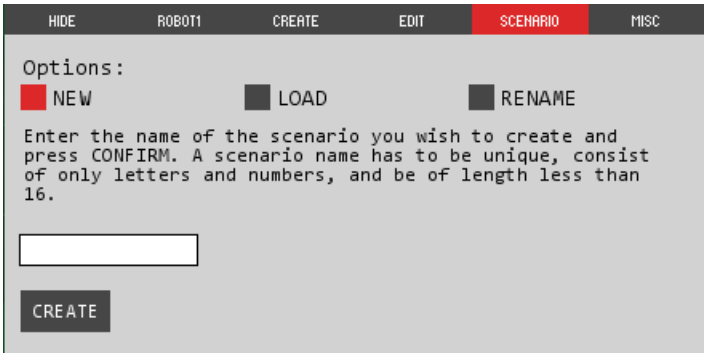
Scenarios

Scenarios define a collections of objects that the Robot can interact with in the environment. The creation and management of world objects will be described in the next section.

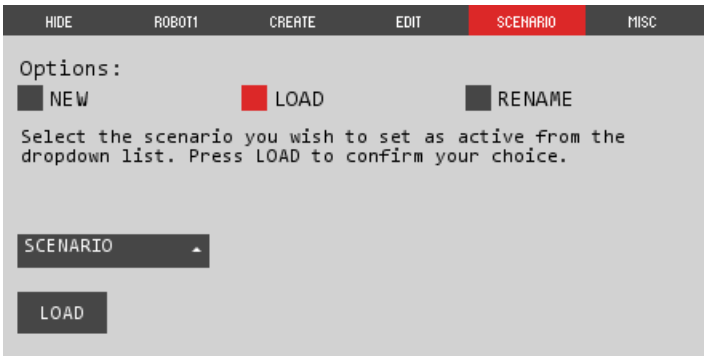


Scenarios can be managed in the scenario window shown in the top left of the screen. There are three variations of the scenario window: new, load, and rename. The radio button row at the top of the window controls, which variant of the screen is active.

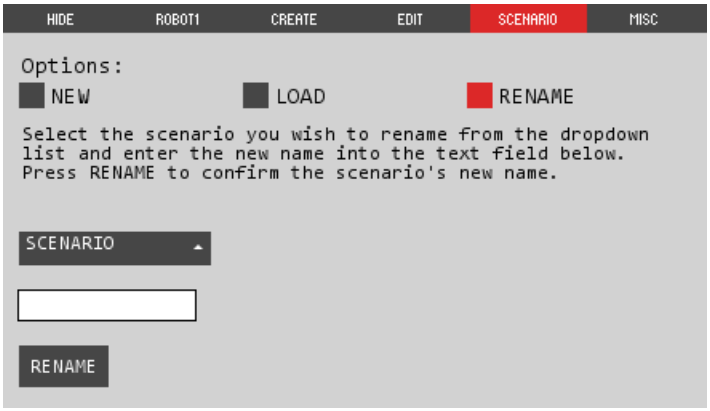
New scenario variant



Load scenario variant



Rename scenario



Simply follow the directions on each variant.

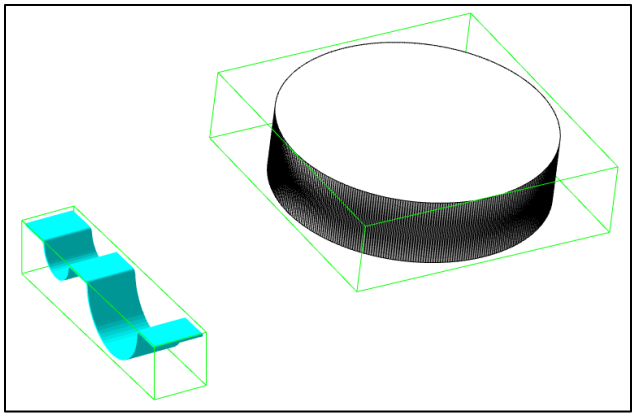
World Objects

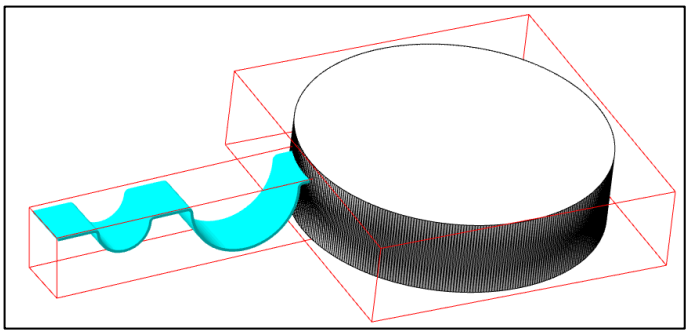
The Robot can interact with world objects in the active scenario. The user has the ability to create a variety of world objects, set their position in the application and even command the robot to pick up, move, and release specific world objects.

Bounding boxes

The two types of world objects are parts and fixtures. Every object have a local coordinate system, which describes the object's position and orientation in terms of its parent coordinate system. By default the world frame is the object's parent coordinate system, however, parts can reference a fixture as its parent coordinate system instead.

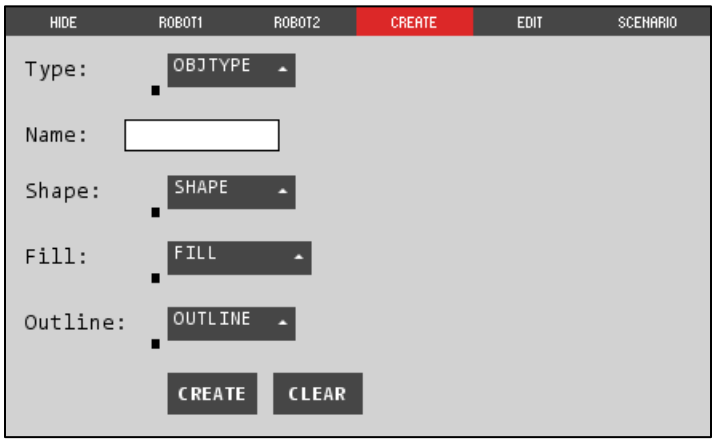
Moreover, parts have a bounding box, which detects collisions with other bounding boxes. A part's bounding box also allows a robot to pick up and move a part in the scenario. Bounding boxes normally appear green. However, when two bounding boxes are colliding, then they appear red (See edit and moving sub-sections for exceptions to this rule). Below shows two examples: one with no world object collision and one with world object collision.





Create a world object

Select the create tab from the button bar on the top left of the application. The window should look similar to this:

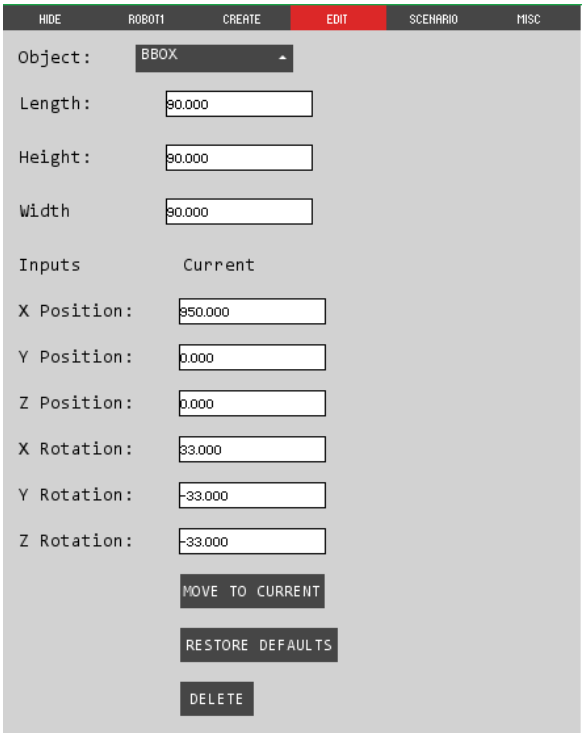


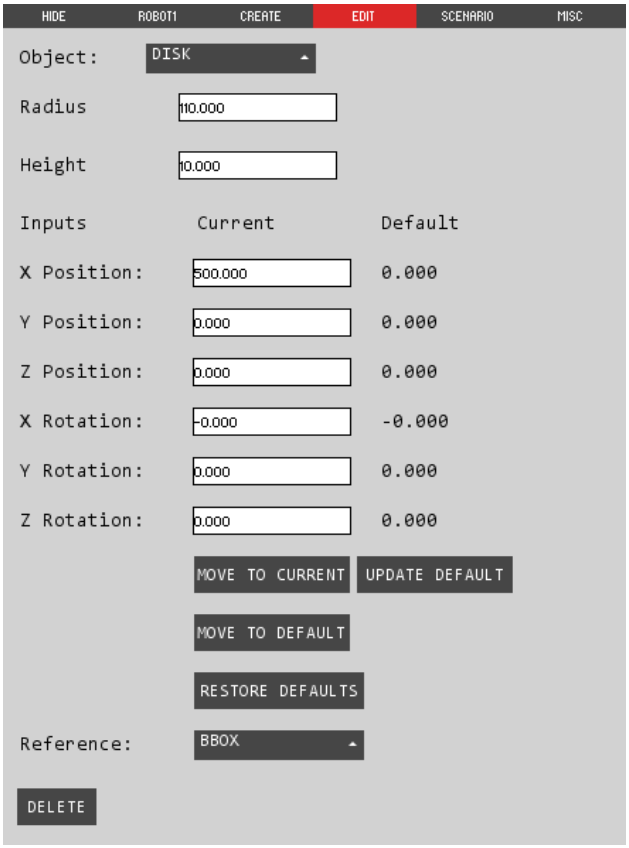
Simply fill out the fields for the object you wish to add to the active scenario (a scenario must be set in order to create a world object). Each world object has a type (part or fixture), shape (box, cylinder, or complex), dimension fields, and color fields. The dimension fields vary based on the shape of the world object. Specifically, the source field of a complex object is defined by a *.stl* source file within the *data/* sub folder of the application's directory. A part also has an optional reference field, where the user can specify a fixture as the part's parent coordinate system. When the world object's required fields have all been specified, then press the Create button to add the world object to the active scenario. The Clear button will reset all fields to their default values.

Position or orient a world object

Initially, all world objects are placed at a default position in the application. With the edit window, the user can position individual world objects in the application. The edit window can be accessed from the button on the top left of the application. The window will resemble either the left of right

screenshot below for parts and fixtures, respectively.



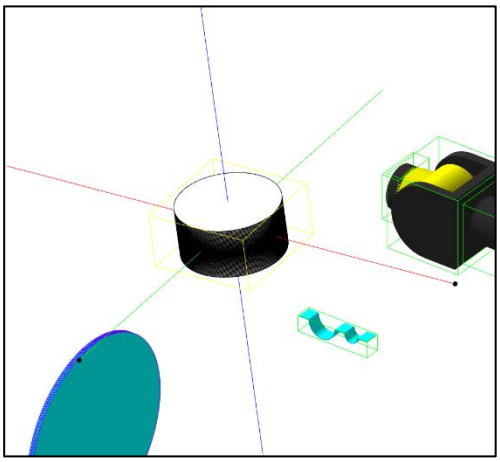


The first dropdown field contains a list of all the names of world objects. When a world object is select, then the dimension fields are updated to reflect the select world object. The X, Y, Z fields pertain to the position of the world object in terms of its parent coordinate system. Likewise, the W, P, R fields refer to the object's orientation in terms of its parent coordinate system.

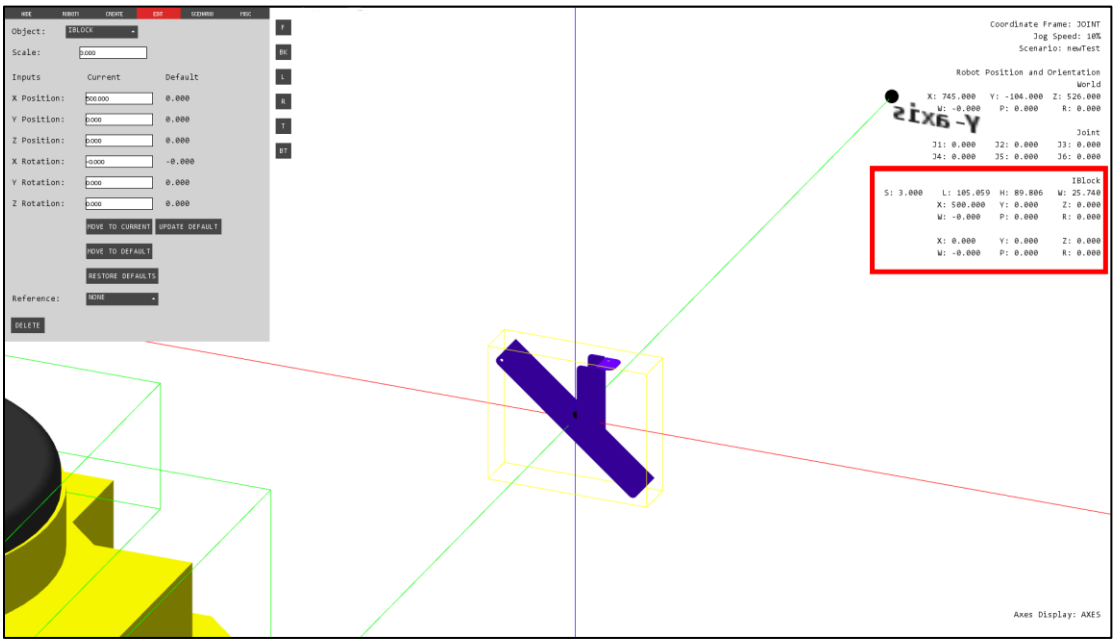
A fixture has only one coordinate system for the user to define. However, a part has two separate coordinate frames that you may define: the current and default coordinate systems. A part will always be rendered at its current coordinate frame. The default coordinate frame acts as a place holder coordinate frame, since there is no way to immediately undo moving an object with the robot. You need only fill out the fields you wish to update for the selected object and press the Move to Current button to update an object's current position. Likewise, the Move to Default button will update the object's current orientation to that of its default orientation. However, the Update Default button will take the input field values and apply them to the part's default orientation.

Furthermore, the Restore Defaults button will move all parts, in the active scenario, to their default orientation.

The coordinate system of the selected world object will appear at the selected world object in the application. In addition, the bounding box of a part will appear yellow when it is selected to be edited.



The red, green, and blue lines represent the x, y, and z axes of the world object's local coordinate system, respectively. Furthermore, the current values of the selected world object's dimensions and local coordinate system will appear on the right hand side of the application.



World object mouse interactions

In addition to the menu interface, you can position and orient a world object with the mouse as well.

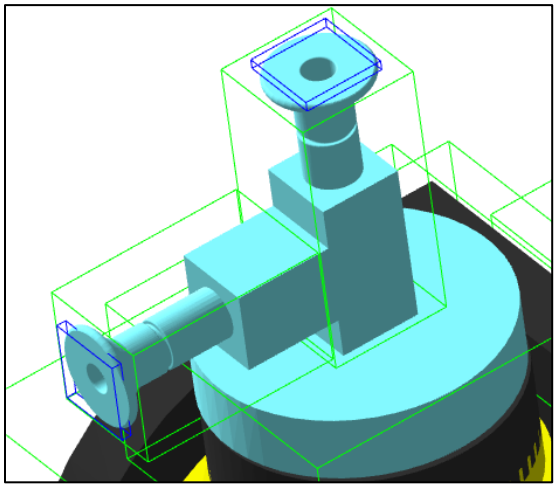
Mouse button	Effect
Left:	select a world object

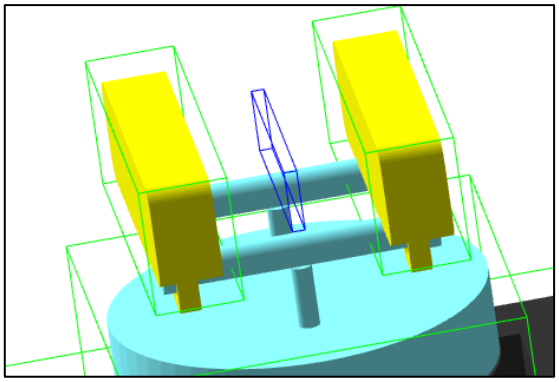
Right:	drag to rotate the selected, world object with respect to the camera's orientation
Middle:	drag to translate the selected world object with respect to the camera's orientation

When the edit window is active, you can select a world object to edit by clicking the object with the left mouse button. If you successfully selected the object, then the fields in the edit window should update according to the world object you selected. Since, the mouse actions for editing world mirror those for moving the camera, you must click on the selected object to edit the object. Clicking anywhere else in the application window will modify the camera.

Move a world object

The claw gripper and suction end effectors grant the robot the ability to pick up, move, and release parts in the active scenario. Below are images of the mentioned end effectors.





These end effectors have unique blue bounding boxes, which link the motion of the robot arm with that of a part. A part can only be linked with the robot's end effector when its bounding box is colliding with a blue bounding box and no other bounding box of the end effector. When the part can be picked up, its bounding box will appear blue. At this point, the user can press p, on his or her computer's keyboard, to pick up the part. While the robot holds a part, the part's bounding box will not appear blue. When a part is held by the robot, the user can again press p to release the object. In addition, if the user switches end effectors, then any part held by the robot is released.

Miscellaneous Window

This window holds a number of functions, most of which change display options in the application.

EE drop down
Changes the active robot's active end effector

Currently there are 6 end effectors: **faceplate** (no end effector), **suction**, **claw gripper**, **pointer**, **glue gun**, **wielder.** The suction and claw gripper end effectors can be used to interact with parts in the active scenario (see world object section).

Axes Display drop down
Changes how the axes are rendered in the WORLD, TOOL, or USER frame.

- Axes Display the x, y, z axes of the active frame
- Grid Displays a grid along the plane spanned by the bottom of the robot's base segment for two of the axes of the active frame
- None Display no axes

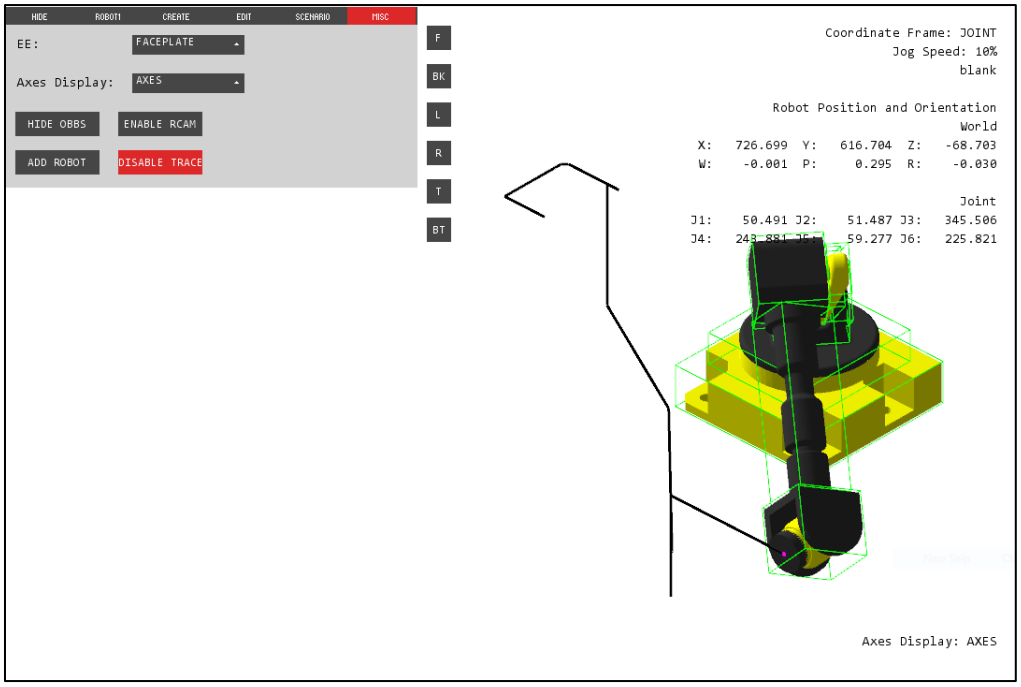
Hide/Show OBBS button
Toggles the display of the robot's and world object's bounding boxes. The button label describes the action will be performed when the button is pressed.

Enable/Disable RCam button
Toggles the robot camera display on/off and shows/hides the respective *Camera* window.

Add/Remove Robot button
Adds or removes a second robot. In addition, the *Robot 2* label will be added to/removed from the top button bar. Moreover, when the second robot is added, then the *Pendant* label becomes *Robot 1*. Both robot's share the same pendant window, but have separate memory spaces and are independent of one another. Furthermore, only the active robot's bounding boxes will be rendering when bounding box rendering is active. Removing the second robot will not erase its memory space. As with the previous button, this button's label describes the action will be performed when the button is pressed.

Enable/Disable Trace button

Toggles the robot trace function on or off. When the trace function is active, then the position of the robot's tool tip will be tracked as the robot moves and rendered in the scene.



The trace is not finite, so after a certain length of tracking, the oldest trace segments will disappear as the active robot continues to move. Also, when the trace is turned off, then the trace buffer will be cleared.

Keyboard Functions

Pendant shortcuts

Key	Button Shortcut
Enter	ENTER
Backspace	BKSPC
*Shift**	SHIFT
*F1**	1
*F2**	2
*F3**	3
*F4**	4
*F5**	5
↑	UP
↓	DOWN
←	LEFT
→	RIGHT
u	-X/J1
i	+X/J1
j	-Y/J2
k	+Y/J2

m	-Z/J3
,	+Z/J3
o	-XR/J4
p	+XR/J4
l	-YR/J5
;	+YR/J5
.	-ZR/J6
/	+ZR/J6
-	-%
=	+%

*only valid when a text entry menu is not active on the pendant

Text, number, point entry

Some screens on the pendant require the user to input text (i.e. naming a program, commenting a register), which can be done with the pendant function keys in addition to the keyboard. The same is true for number entries (i.e. editing a data register) or point entries (i.e. editing a position registers), which can be completed with either the pedant's numpad or the keyboard.

General keyboard functions

Key Combination	Function
Ctrl + c	Change the active robot's coordinate frame: a kin to pressing the COORD button on the pendant.
Ctrl + e	Changes the active robot's end effector following this pattern: faceplate → suction → claw gripper → pointer → glue gun → wielder
Ctrl + p	If the robot is holding a part, then that part is released. Otherwise, if the robot's active end effector is either the claw gripper or suction and a part is colliding with the robot's end effector's blue bounding box, then that part becomes held by the robot.
Ctrl + s	Saves all data for the active robot (programs, frames, registers, and scenarios).
Ctrl + r	Resets all of the active robot's joint angles to 0.
Ctrl + Alt + r	Toggle recording feature on or off (a message is displayed when recording is on).
Ctrl + s	Saves all data for the active robot (programs, frames, registers, and scenarios).
Ctrl + z	Reverts a previous edit or deletion of a world object in the current active scenario.