Getting Started with the TX2-40/Cs9 Stäubli Robot
SP2 Teach Pendant, VAL 3 Programming

Jean-Louis Boimond
University of Angers

Keywords: Stäubli robot, TX2-40 arm, Cs9 controller, SP2 teach pendant, VAL 3 programming.

This document provides an introduction to the Stäubli TX2-40 robot, equipped with a Cs9 controller, using the SP2 *teach pendant* (and not the *Stäubli Robotics Suite* (SRS) software), by covering the following topics: manual arm movement; access to the *Tool Center Point* coordinates; use of a VAL 3 application (VAL 3 being the programming language used by Stäubli) for automatic arm movement with a brief presentation of the variables (standard types, their creations and initializations); manual arm movement towards a given joint or Cartesian point.

Table of Contents

1) Getting started with the robot

1.1) Starting up the Cs9 controller

The Cs9 controller (used with the robot arm) is started up by switching the main switch on the front of the controller, circled in red in the figure below, to position '1'.

Figure 1: Front of the Cs9 controller.

Wait about 2 mn for the main menu to appear on the *teach pendant*, as shown in the figure below.

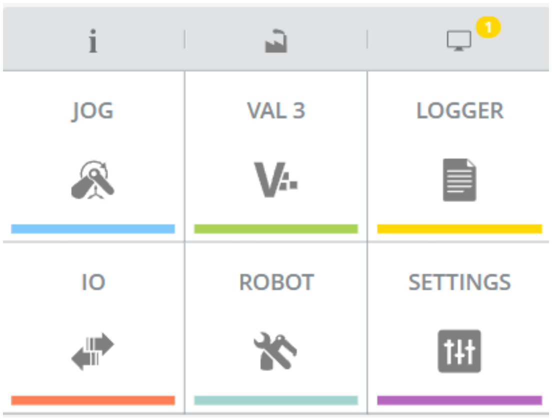


Figure 2: *Teach pendant main menu*.

N.B.:

- The **Home** key , at the top left of the *teach pendant*, takes you back to the main menu,
- The **Back** key , near the top left of the *teach pendant*, takes you back to the previously visited page.

1.2) Arm power with a manual Working Mode

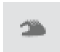
Assumption: The Cs9 controller is in operation (see **1.1**).

The arm power on, which is required to set it in motion, with a manual Working Mode is done through the following two steps:

a) On the Working Mode Selector (WMS9), located near the controller, make sure that the switch (equipped with a key) is set to **manual**  as shown in the figure below.

Figure 3: Working Mode Selector (WMS9) panel.

As a result, the icon  in the bottom right of the *Teach Pendant* appears, indicating the selection of the **manual** Working Mode. In this mode, robot speed is limited to 250 mm/s, allowing the operator to stand close to the robot arm.

Note: An alternative to using the WMS9 is to select the ***manual slow*** item  from the drop-down menu at the bottom right of the *teach pendant*, rather than the ***auto*** item  (given by default), see the figure below:

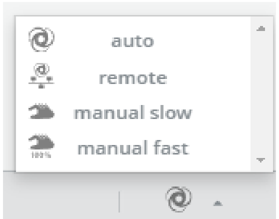


Figure 4: Selecting the Working Mode from the *teach pendant*.

b) Press **Power** key  at the top right of the *teach pendant* to switch the arm power on. This action is only taken into account if the *enabling device*, located on the back of the *teach pendant* (circled in red in the figure below), has been put into its middle position (by pressing the button neither too weakly nor too strongly) in the last 15 seconds (note that the button must be released and pressed again if the arm has not been powered up within 15 seconds). A light around the **Power** key appears (flashing for a few seconds before being steadily) to indicate that the arm power is on.

Figure 5: Back of the *teach pendant*.

N.B.: The arm power is cut off if the *enabling device* is released while the arm is in manual movement. To put the power back on the arm, press the blue **Restart** button located on the WMS9 (see figure 3) to acknowledge the restart before pressing the **Power** key.

N.B.: The **Emergency Stop** button, at the top right of the *teach pendant*, immediately cuts arm power and thus stops the arm from moving (if it was moving).

1.3) Manual movement of the arm

Assumption: The arm is powered up in a **manual** Working Mode (see **1.2**).

Note: It is possible to adjust the speed of the Tool Center Point (TCP) by pressing the **Jog** key [- +], in the right vertical banner of the *teach pendant*. Its percentage value appears at bottom left, with a maximum speed (100%) equal to $250\ mm/s$.

Press the **JOG** button (JOG) on the *teach pendant* main menu (accessed - if you are not already there - by pressing the **Home** key, at the top left) to access the window for manual arm movement.

Once the **JOG** menu has appeared, select one of the three buttons represented in the horizontal banner [Joint | Frame | Tool], at the top of the window, to indicate the space in which you wish to perform the movement:

- the **Joint** button to access the **joint space**,
- the **Frame** button to access the **Cartesian space** associated with the reference frame R_0 of the robot arm,
- the **Tool** button to access the **tool space** associated with the frame R_{Tool} associated with the tool (attached to the robot arm flange).

1.3.a) In the joint space

Pressing the **Joint** button moves the arm through the **joint space** by using angles **J1**, **J2**, ..., **J6**, see the figure below.

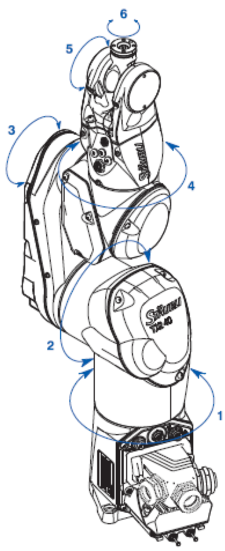


Figure 6: Description of the six joints of the TX2-40 arm.

Press the **Jog** key [- +] (in the right vertical banner of the *teach pendant*), for example, relative to joint **J1** to rotate the arm around the axis of **J1**, either in the negative or positive direction.

N.B.: About the information displayed, by default, in the main window:

- **flange** (equivalent to **flange[0]**) indicates that no tool is selected, so the tool frame R_{Tool} coincides with the frame associated with the flange of the robot arm,
- **world** (equivalent to **world[0]**) indicates that the reference frame (used for *situating* points, frames, etc.) is the world frame (which coincides with the reference frame R_0 of the robot arm.

1.3.b) In the Cartesian space

Pressing the **Frame** button moves the arm through the **Cartesian space** by using **X**, **Y**, **Z** (in mm), **RX**, **RY**, **RZ** (in degree). Press the **Jog** key  (in the right vertical banner of the *teach pendant*), for example, in relation to the **X**-axis so that the arm performs a translation of the TCP along the x_0 axis of the reference frame R_0 of the robot arm. When the **Jog** key is relative to **RX**, **RY**, or **RZ**, the TCP rotates around the axes x_0, y_0 or z_0.

1.3.c) In the tool space

Pressing the **Tool** button moves the arm through the **tool space** by using **X**, **Y**, **Z** (in mm), **RX**, **RY**, **RZ** (in degree). Press the **Jog** key  (in the right vertical banner of the *teach pendant*), for example, relative to the **Y**-axis so that the arm performs a translation of the TCP along the y_{Tool} axis of the frame R_{Tool} associated with the tool (or with the flange if there is no tool). When the **Jog** key is relative to **RX**, **RY**, or **RZ**, the TCP rotates around the axes x_{Tool}, y_{Tool} or z_{Tool}.

1.4) Access to *Tool Center Point* coordinates

The TCP corresponds to the origin of the frame R_{Tool} associated with the tool. Note that the `flange` tool is used by default (in other words, no tool is attached to the flange), which means that the tool frame is confused with the flange frame (by default).

In the **JOG** menu (accessible through the *Teach Pendant* main menu), simply go:

- in the **joint space** (using **Joint** button) to access the TCP angular coordinates (in degree) listed in front of **J1**, ..., **J6** buttons,
- in the **Cartesian space** (using **Frame** button) to access the TCP Cartesian coordinates in the reference frame R_0 of the robot arm listed in front of **X**, **Y**, **Z** (in mm), **RX**, **RY**, **RZ** (in degree) buttons,
- or in the **tool space** (using the **Tool** button) to access the TCP Cartesian coordinates in the tool frame R_{Tool} (whose origin corresponds to the TCP) listed in front of **X**, **Y**, **Z** (in mm), **RX**, **RY**, **RZ** (in degree) buttons.

2) Programming a VAL 3 application

We see in **2.1** how to create a VAL 3 application, entitled `First_steps`, and view the VAL3 code of its `start()` program.

After a brief presentation of standard variables in **2.2**, a first application is performed in **2.3** to move the arm to a vertical posture (defined by a *joint variable* entitled `jDpt`) during two seconds, then the arm moves so that the TCP reaches a point defined by a *Cartesian variable* entitled `pExamplePoint` with values equal to $X = 400, Y = 70, Z = 275$ (mm), $RX = 25, RY = 100, RZ = -25$ (degree).

The way to run an application is described in **2.4**; that for closing an application (deleting it from the controller's RAM) is described in **2.5**.

2.1) Creation of the `First_steps` application and editing of its `start()` program

An application is composed of programs, by default `start()` (called by the system when the application starts) and `stop()` (called by the system when the application is quit). In the following, the application code will be placed only in the `start()` program (the `stop()` program, initially empty, will not be modified).

<u>Creation of the `First_steps` application</u>

From the home page (accessible *via* the **Home** key , at the top left of the *teach pendant*), the creation of an application, entitled `First_steps`, requires the following steps:

- Select the **Val3** menu 

to access the window shown in the following figure:

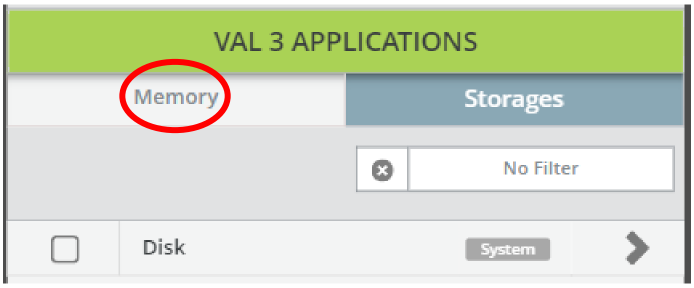


Figure 7: VAL3 applications window with the default selection of the `Storages` tab.

- Select the `Memory` tab (rather than `Storages`), circled in red in the previous figure, to access the controller's RAM. No application is present in the controller's RAM as shown in the following figure:

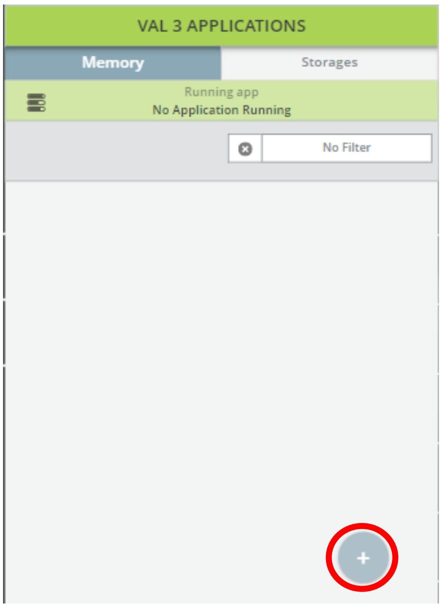
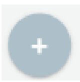


Figure 8: VAL3 applications window when the `Memory` tab is selected.

- Press the  button, circled in red in the previous figure, to create an application. In the window that appears, see the following figure, type `First_steps` in the **Name** field, and then click the **OK** button to confirm.

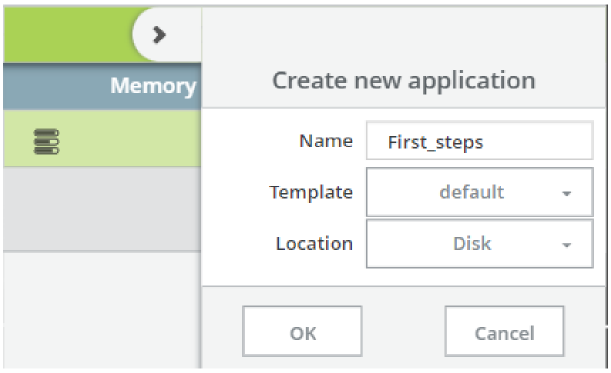


Figure 9: Creation of the `First_steps` application.

This results in the creation of the `First_steps` application in the controller's RAM as shown in the figure below where the `First_steps` application appears in the `Memory` tab of the VAL3 applications window:

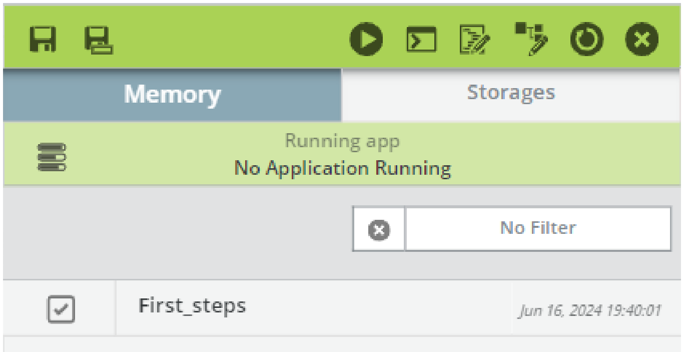


Figure 10: Display of the `First_steps` application in the `Memory` tab of the VAL3 applications window.

This application is also saved on the controller's hard drive, as can be checked in the VAL3 applications window by selecting the `Storages` tab.

Thereafter, **be careful to work only** on the `First_steps` application, so as not to disturb the contents of the controller's hard drive.

<u>Editing of `start()` program</u>

To edit the code of the `start()` program of the `First_steps` application:

- Press the  button, at the top of the menu bar shown in the previous figure. The application programs, that is `start()` and `stop()`, are listed in the **Programs** tab (selected by default) of the window that appears, as shown in the figure below.

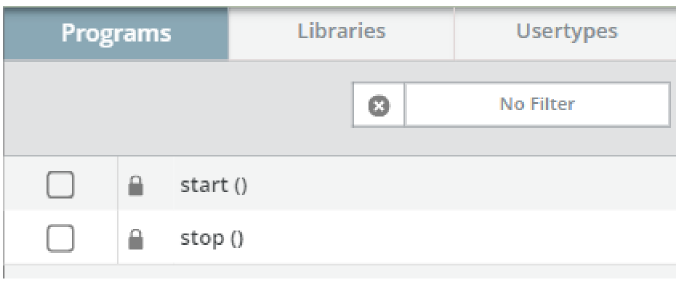


Figure 11: Listing of `start()` and `stop()`) programs of the `First_steps` application.

- Select the `start()` program to display its contents, which are currently empty except for the `begin` and `end` tags delimiting the program's code:

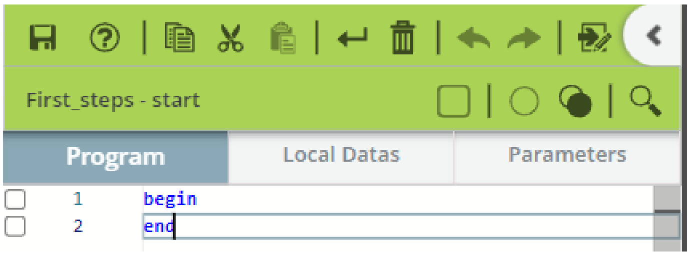


Figure 12: Code of the `start()` program.

2.2) Variables

The main types of VAL3 variables, including those specific to robotics, are briefly described in **2.2.1**. The process for visualizing the variables of an application and declaring a new variable (with values given by default) is described in **2.2.2**, while the method for initializing the variable values is described in **2.2.3**.

2.2.1) Variable types

Several variable types are available in VAL3. There are the classic variables of a programming language such as Boolean (`bool`), numeric (`num`), string (`string`) variables. Some variables are specific to robotics, for example: variables *points* defined in the *joint space* (later called *joint points* (`jointRx`)) or defined in the *Cartesian space* (later called *Cartesian points* (`pointRx`)); *tools* variables (`tool`); variables for defining *Cartesian frames* (`frame`); variables for defining *changes in position* and/or *orientation* (`trsf`).

To make it easier to recognize the type of a variable, it is assumed that the first letters of its name indicate its type, *i.e.*, concerning the types described above:

`bVariable` for a variable of `bool` type,
`nVariable` for a variable of `num` type,
`sVariable` for a variable of `string` type,
`jVariable` for a variable of `jointRx` type,
`pVariable` for a variable of `pointRx` type,
`tVariable` for a variable of `tool` type,
`fVariable` for a variable of `frame` type,
`trVariable` for a variable of `trsf` type.

2.2.2) Variables display of the `First_steps` application, declaration of a new variable

Assumption: The `First_steps` application is loaded in the controller's RAM (see figure 10), which gives access to the menu described in the figure below.

<u>**Variables display of `First_steps` application**</u>

Press the  button, circled in red in the menu bar shown in the figure below, to view the application's variables.

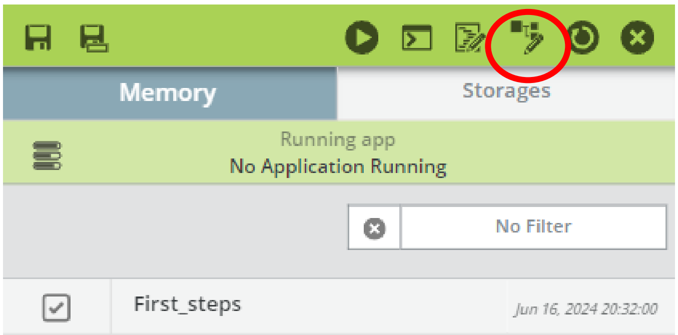


Figure 13: Button allowing access to the variables of the `First_steps` application.

The result is the window, shown in the figure below, listing variables of all types in alphabetical order when the **Data** tab is selected (which is the default case). Variables are listed in a hierarchical manner when the **Geometry** tab is selected.
By default, the `mNomSpeed`[1] variable, of `mdesc` type, is set to indicate the speed of the TCP as the robot arm moves.

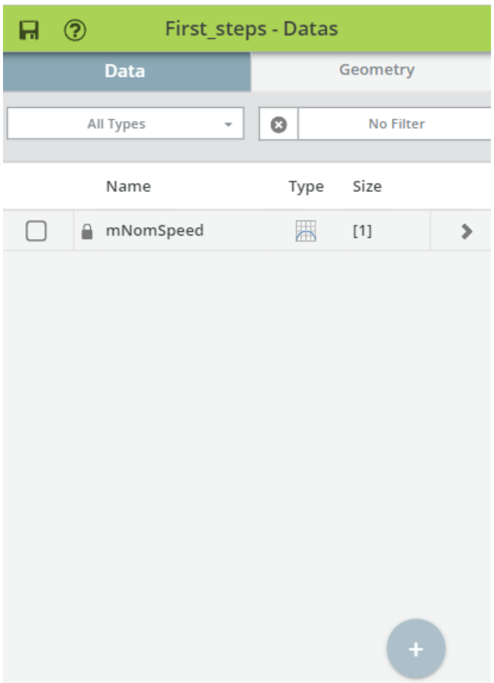
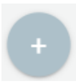


Figure 14: Variables display (using **Data** tab) of `First_steps` application.

Note that the presence of a padlock icon in front of a variable indicates that it is private (which is the case with the `mSpeedName` variable), it is public otherwise.

<u>**Declaring a new variable**</u>

For example, let us define two variables: a joint variable called `jDpt` (of `jointRX` type) and a Cartesian variable called `pExamplePoint` (of `pointRx` type).

The  button, at the bottom right of the previous figure, allows the creation of new variables whose type, name, container, etc. are to be declared in the window that appears after pressing the  button.

[1] `mSpeedName` if you use the English language.

✓ **Creating the `jDpt` joint variable**

After pressing the  button, the `jDpt` variable is created using the contents of the fields described in the figure below:

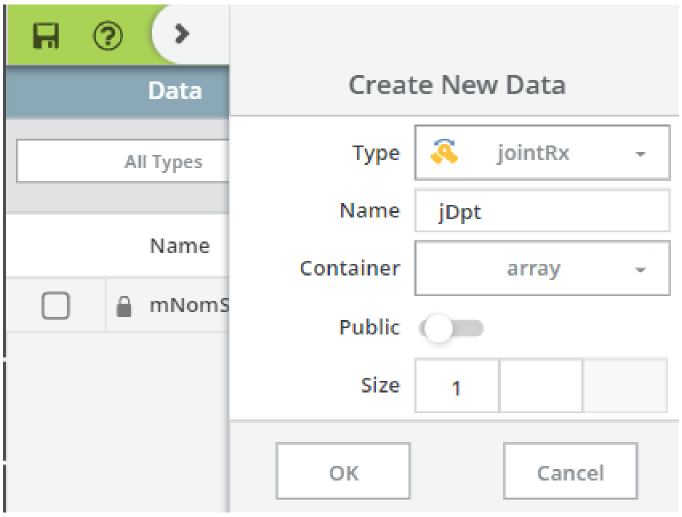


Figure 15: Creation of the `jDpt` variable.

where it is indicated that the `jDpt` variable (`Name` field) is of type `jointRX` (`Type` field), it corresponds to an array (`Container` field) of unit size (`Size` field) (which explains the name `jDpt[0]`) and that its scope is private (`Public` button on off). Remember to validate your data by clicking on the **OK** button. Note that the variable is initialized with default values, access to these values being described in annex **A.2**.

The initialization of this variable is done in **2.2.3** in the `start()` program; note that it is also possible to read, or initialize, a joint or Cartesian variable using the VAL3 menu, see Annex **A.2**.

✓ **Creating the `pExamplePoint` Cartesian variable**

After pressing the  button, the `pExamplePoint variable` is created using the contents of the fields described in the figure below:

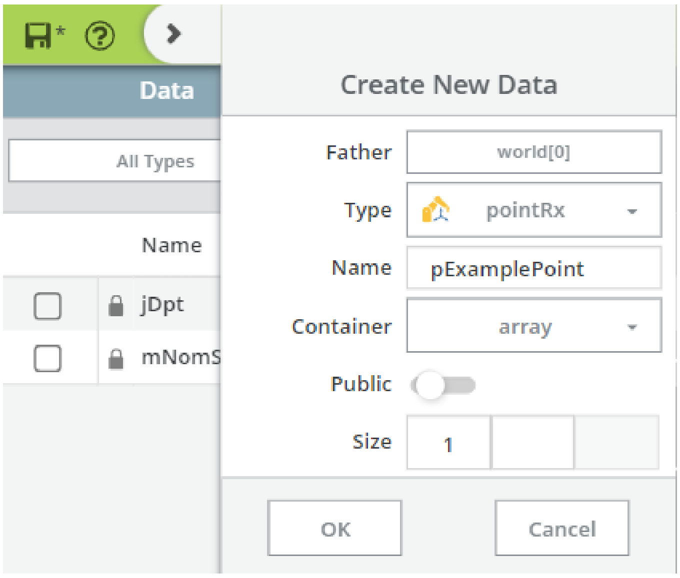


Figure 16: Creation of the `pExamplePoint` variable.

where it is indicated that the `pExamplePoint` variable (`Name` field) is of type `pointRX` (`Type` field), it corresponds to an array (`Container` field) of unit size (`Size` field), and that its scope is private (`Public` button on off). The variable is defined relative to the `world` frame (`Father` field), corresponding to the reference frame R_0 of the robot. Note that it is possible to define points relative to a frame (previously defined) other than the `world` frame. Remember to validate your data by pressing the **OK** button. Note that the variable is initialized with default values, access to these values being described in annex **A.2**.

Note: The variables are not saved if an asterisk appears in the floppy disk icon (*), at top left of the previous figure. Click on this icon to make the recording, the icon will no longer have an asterisk.

2.2.3) Initializing a variable

It is possible to initialize in the `start()` program a variable such as, for example, `jDpt` (of `jointRX` type) or `pExamplePoint` (of `pointRX` type). This is done simply through the following instructions (of course, to be placed before their use in an instruction):

```
jDpt={0,0,0,0,0,0}
pExamplePoint={{400,70,275,25,100,-25},{ssame,esame,wsame}}
```

where:

- `{400,70,275,25,100,-25}` values are the `x,y,z,Rx,Ry,Rz` coordinates indicating the *location* (*i.e.*, the *position* and the *orientation*) of *the* `pExamplePoint` point,
- `{ssame,esame,wsame}` parameters (of type configuration) prohibit a change in the arm configuration during the movement towards the `pExamplePoint` point. In the program described in **2.3**, the arm configuration is the one used to reach the `jDpt` joint point, *i.e.*, left shoulder, positive elbow and wrist.**Note:** Two other methods exist for initializing a variable. Unlike the previous method, which initializes a variable directly in the program code, these methods require manipulation of *teach pendant*.

The method, described in Annex **A.2**, is applied *via* the **VAL3** menu and can be used to initialize any type of variable, including variables: *point* of type `jointRx` or `pointRx`; *frame* of type `frame`; *tool* of type `tool`.

The method, described in Annex **A.3**, is applied *via* the **JOG** menu (usually used to move the robot arm manually towards a given point, see **3**) and can be used to initialize *point*, *frame* and *tool* variables.

2.3) Coding the `start()` program

Assumption: `jDpt` joint variable and `pExamplePoint` Cartesian variable are created, see the procedure described in **2.2.2**.

The first step is to edit the code of the `start()` program, see the procedure described in **2.1**. Complete the `start()` program, initially consisting of `begin` and `end` tags, as follows:

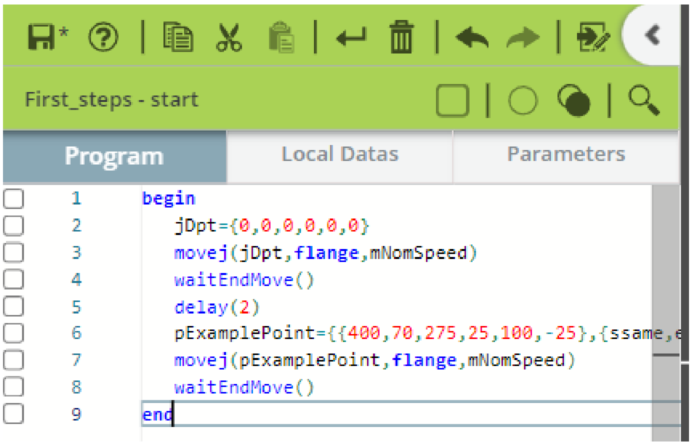


Figure 17: Code contained in the `start()` program.

The joint variable `jDpt`, of type `JointRx` (see **2.2.1** for more details), is such that $J1 = \cdots = J6 = 0$ (see **2.2.3** for initialization of the variable).
The `movej(jDpt,flange,mSpeedName)` instruction moves from the *current point* (the point reached just before the instruction is executed) to `jDpt` *point* (where the robot arm is vertically extended). The `flange` parameter indicates that there is no tool attached to the robot arm, *i.e.*, the frame associated with the TCP corresponds to the frame associated with the robot's flange. The use of `movej` ensures that the trajectory optimizes movement speed (we speak about *point-to-point* movement); note that there are other movement instructions for straight or circular trajectories, but they do not guarantee an optimal movement speed.
The `waitEndMove()` instruction, located after `movej(jDpt,flange,mSpeedName)`, ensures to achieve the movement to `jDpt` *point* before continuing the execution of the program, so there is no smoothing phenomenon of `jDpt` *point* with the next *point* (`pExemplePoint`).
The `Delay(2)` instruction causes a 2-second wait at the *current point*, *i.e.*, `jDpt`.
See **2.2.3** for the initialization of the Cartesian variable `pExamplePoint`.
The `movej(pExamplePoint,flange,mSpeedName)` instruction performs a move to the `pExamplePoint` *point* with a mode of operation similar to the move to `jDpt` *point*.
The `waitEndMove()` instruction, located just before the `End` instruction, ensures to achieve the movement to `pExamplePoint` *point* before the program stops. Also, be sure to always place this instruction just before the `End` instruction of your program.

This program is such that once the arm is vertically extended, 2 seconds elapse, and then the TCP reaches the `pExamplePoint` *point* with coordinates $X = 400, Y = 70, Z = 275, RX = 25, RY = 100, RZ = -25$.

Note: The program is not saved if an asterisk appears in the floppy disk icon (*), at top left of the previous figure. Click on this icon to make the recording, the icon will no longer have an asterisk.

2.4) Running the application

Assumptions:
- The **Off** mode is the one selected in the **JOG** menu (and not **Joint**, **Frame** or **Tool**) (see **1.3**),
- The arm is powered up in a **manual** Working Mode (see **1.2**),
- The `First_steps` application is loaded into RAM's controller (see figure 10).

N.B.: The **Move/Hold** button , at the top right of the *teach pendant*, allows a soft and immediate stop of the robot arm movement (which can be useful during a program development), this type of stop does not cut arm power.

The `First_steps` application being loaded in the controller's RAM, go to the page described in figure 10 (after 2 successive presses of the **Back** key if you come from the menu described in the previous figure). The box corresponding to the application being checked (which sets the application 'at the top' of the execution stack if several applications are loaded into RAM (which is not the case here)), the application is launched by pressing the **Run** button (in the menu bar), then by pressing: the **Move/Hold** button of the *teach pendant* once, then a second time continuously, as well as on the *enabling device* (knowing that the arm movement stops as soon as the **Move/Hold** button or the *enabling device* is released).

2.5) Closing the application

Be sure that the `First_steps` application is stopped as indicated by the message **No Application Running** Running app No Application Running displayed on the *teach pendant*; if this is not the case, the application can be stopped by pressing the **stop** button Running app First_steps , at the top right of the *teach pendant*. Once the box corresponding to the `First_steps` application has been checked (see figure 10), it is closed by pressing the **Close** button , at the top right of the *teach pendant*, which removes it from the RAM (Memory).

3) Manual movement to a given point

During trajectory setting, it may be useful to manually move the robot arm to test access to certain *points*. Such movements are made from the **JOG** menu, accessible *via* the main menu of *the teach pendant* (press the **Home** key to go there).

Assumptions:
- The arm is powered up in a **manual** Working Mode (see **1.2**),
- The `First_steps` application is loaded into RAM's controller (see figure 10).

From the **JOG** menu described in the figure below:

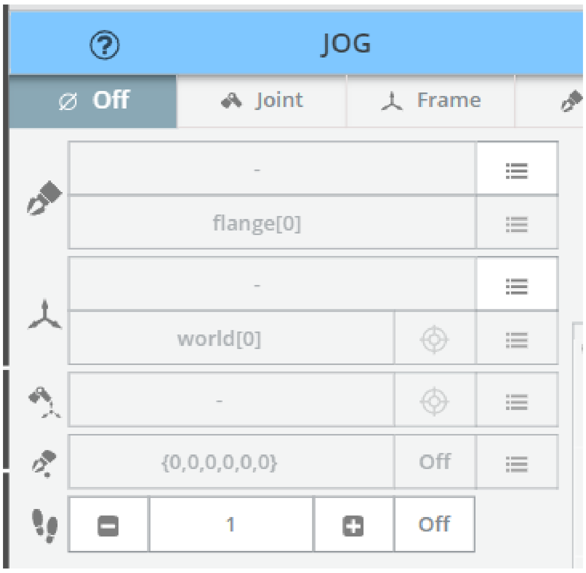


Figure 18: **JOG** menu.

- Press the button circled in red in the figure below to access the `First_steps` application (located at the bottom of the figure below):

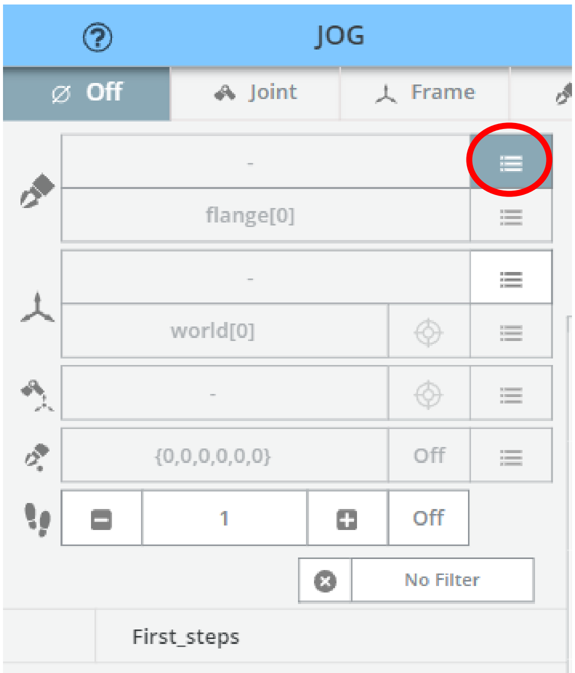


Figure 19: Access the `First_steps` application in the **JOG** menu.

- Select the `First_steps` application to open the window shown in the figure below, where the tool and the base frame used in the application are the default ones, *i.e.*: `flange` (corresponding to the case where there is no tool attached to the robot arm) and `world` (which means that the *points* considered below are defined in the reference frame R_0 of the robot (and not another frame that may have been defined previously)).

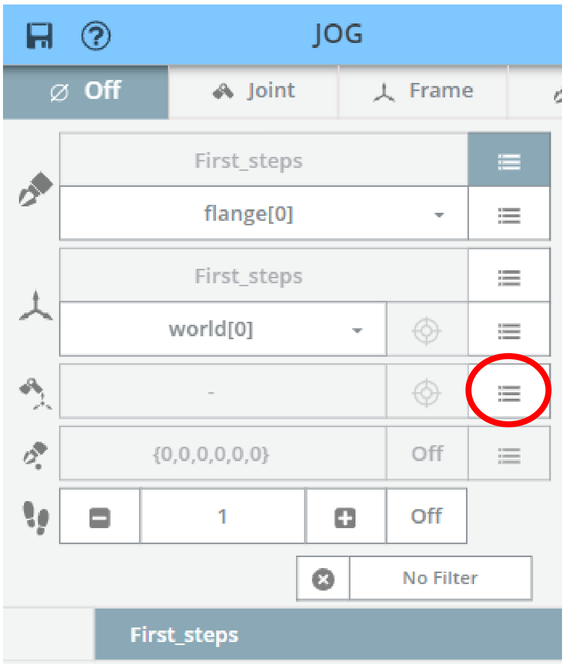


Figure 20: Selecting `First_steps` application in the **JOG** menu.

- Press the button circled in red in the previous figure to view the *joint variables* (due to the selection, by default, of **Joint** in the field circled in red in the figure below) associated with the application: the `jDpt` variable, in the present case, as shown in the figure below:

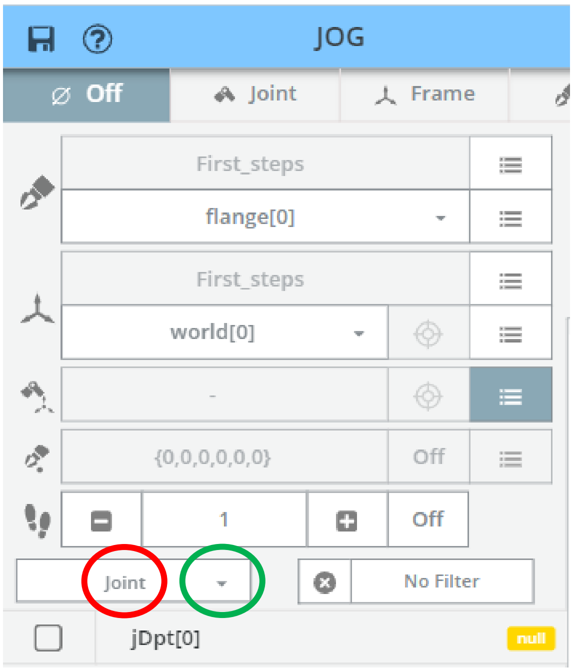


Figure 21: Display of the **joint** variables (`jDpt` in the present case) in the **JOG** menu.

3.1) Manual movement to `jDpt` joint point

In the window described in the previous figure, check the box corresponding to `jDpt` variable to indicate that you want the TCP to move to `jDpt` *point*. Press the **Joint** button , which appears at top right of the window shown in the previous figure, to indicate that the movement will be calculated in the joint space, leading to the window shown in the figure below:

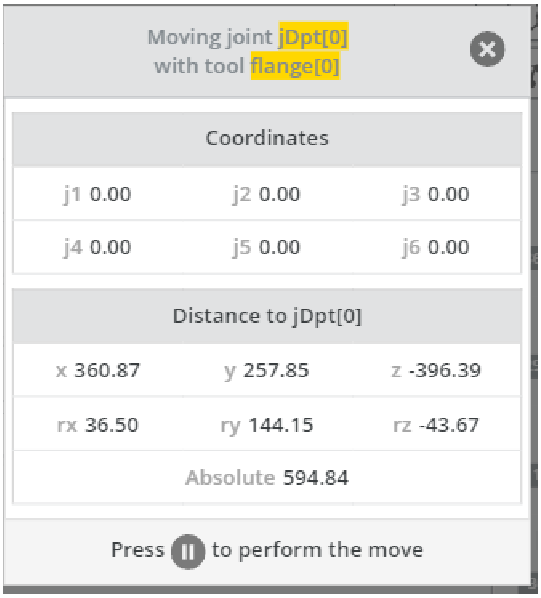


Figure 22: Window showing the `jDpt` variable data before the robot arm moves.

which indicates the distance between the *current point* and the `jDpt` *point* and mentions that the arm will be set in motion by pressing the **Move/Hold** key .

3.2) Manual Movement to `pExamplePoint` Cartesian Point

In the window described in figure 21, select **Point** (instead of **joint** selected by default) from the drop-down menu circled in green in figure 21 to view the *Cartesian variables* associated with the application: the variable `pExamplePoint`, in the present case, as shown in the figure below:

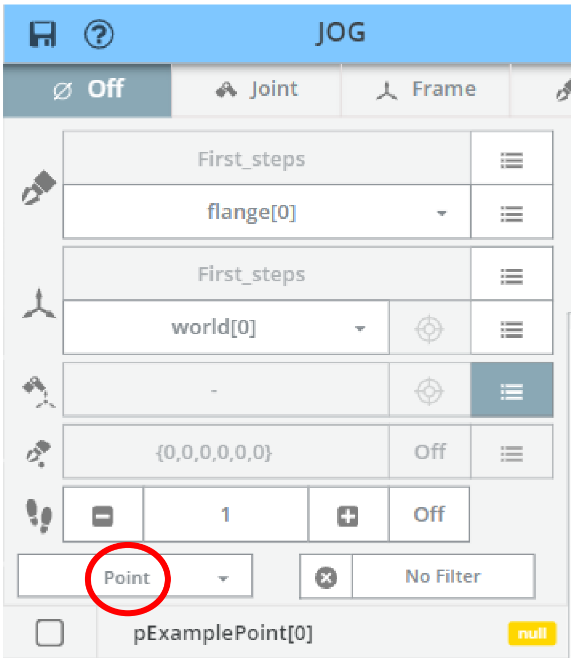


Figure 23: Display of the **point** variables (`pExamplePoint` in the present case) in the **JOG** menu.

Check the box corresponding to `pExamplePoint` variable to indicate that you want the TCP to move to `pExamplePoint` *point*. Two ways to set the robot arm in motion are proposed:

- by pressing the **Joint** button, which appears at top right of the window shown in the previous figure, to indicate that the movement will be calculated in the joint space, leading to the window shown in the figure below:

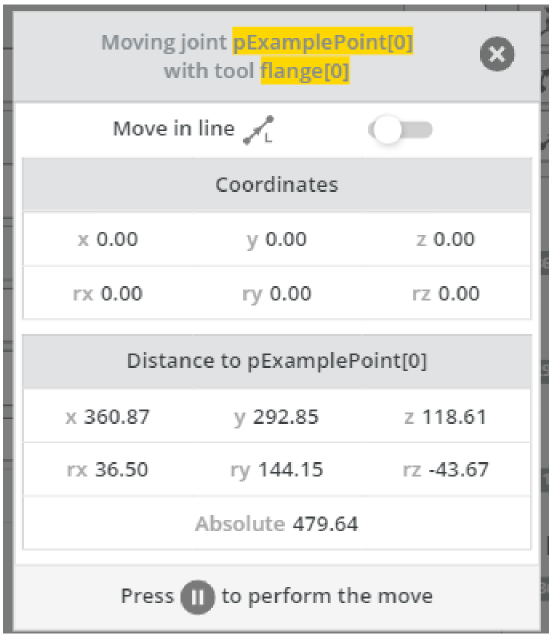


Figure 24: Window showing the `pExamplePoint` variable data before the robot arm moves.

which indicates the distance between the *current point* and the `pExamplePoint` *point* and mentions that the arm will be set in motion by pressing the **Move/Hold** key.

You will notice that the button corresponding to the **Move-in-line** field Move in line is set to off, which means that the active mode is indeed the point-to-point mode (and not the straight line mode).

- or by pressing the **Line** button, which appears at top right of the window shown in the figure 23, so that the TCP joins `pExamplePoint` *point* in **a straight line**. Note that the calculation of the movement is done in Cartesian space, so the movement is not always possible!

ANNEX

A.1) Loading an application stored in the controller into RAM

The following procedure is used to load an application located on the controller's hard disk into RAM, by example, for its execution.

From the home page (accessible *via* the **Home** key, at the top left of the *teach pendant*):

- Access the list of applications located on the controller's hard disk by selecting the **Storages** tab in the **VAL3** menu VAL 3, then pressing the button on the line corresponding to **Disk**, circled in red in the figure below:

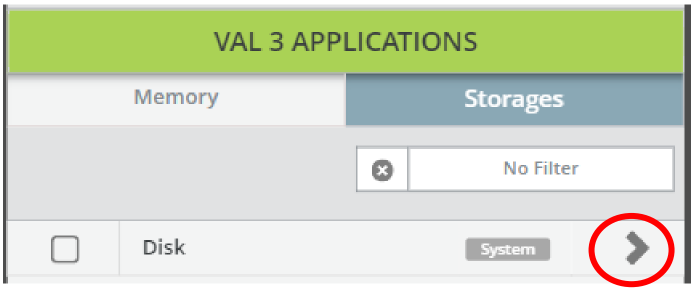


Figure 25: Selecting the controller's hard disk.

- In the window that appears, check the box corresponding to the application you want to load into RAM, which causes it to appear in the **Memory** tab, like the window described in figure 10.

A.2) Reading, initializing a variable using the VAL3 menu

As seen in **2.2.2**, the **VAL3** menu allows the display of all the variables (of all *types*) declared in an application. We'll see that the **VAL3** menu can also be used to initialize the contents of a variable, such as the joint variable `jDpt` used in the `First_steps` application.

Assumption: The `First_steps` application is loaded into the RAM's controller (see figure 10).

Go to the page for viewing variables of the `First_steps` application (for this, see the procedure described in **2.2.2**), which gives rise to the page shown in the figure below:

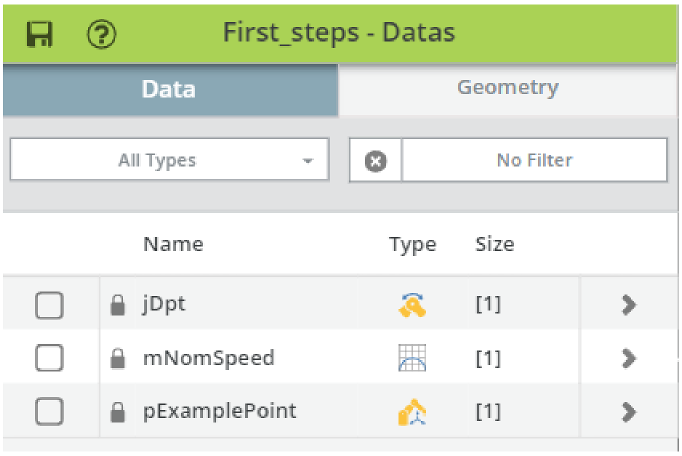


Figure 26: Display of the variables *via* the **Data** tab.

By pressing the > button corresponding to `jDpt` variable, then checking the corresponding box in the window that appears, it is possible to read the contents of the variable, but also to initialize its values by pressing the **Edit** button (at the top of the menu bar), see the figure below:

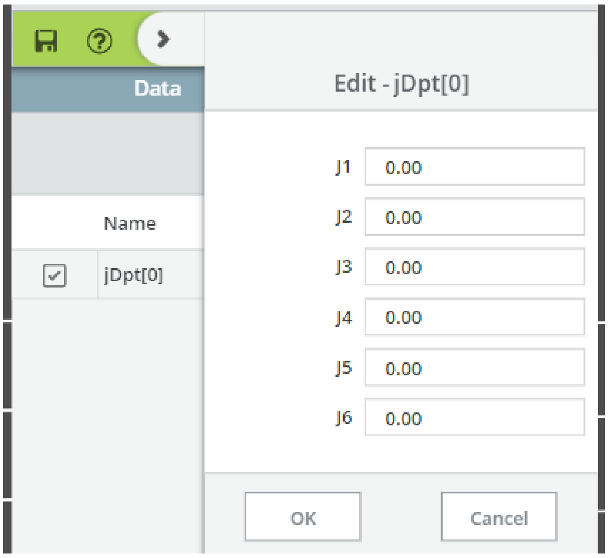


Figure 27: Reading, initializing the contents of `jDpt` variable.

Don't forget to validate with the **OK** button if you modify the variable contents!

A.3) Reading, initializing a variable using the JOG menu

The **JOG** menu is usually used to manually move the robot arm to a given point (see **3**), but it is also possible to use this menu to create a variable of type: point (`jointRX, pointRX`), `frame` or `tool`, and to read or initialized its contents.

As an example, let's read the contents of the `jDpt` joint variable used in the `First_steps` application. Then create a new Cartesian variable called `pOtherPoint`.

Assumption: The application to be edited, in the present case `First_steps`, is in RAM (see figure 10).

✓ **Reading, initializing the `jDpt` joint variable (of type `jointRx`)**

Go to the **JOG** menu to view the joint variables of the `First_steps` application, as shown in **3** and **3.1**, resulting in the page shown in the figure below (identical to figure 21):

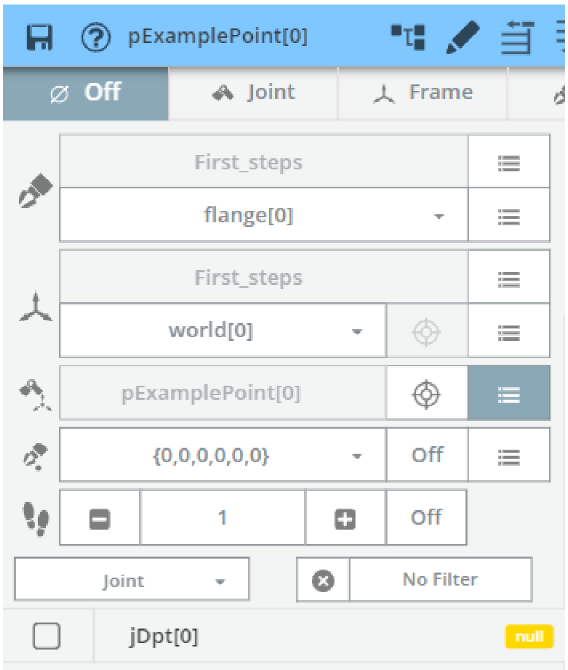


Figure 28: Display of the joint variables (of `jointRX` type), in the present case `jDpt`, in the **JOG** menu.

- Check the box for `jDpt` variable, then press the **Edit** button at the top of the menu bar to access the window described in the figure below, which allows you to read or initialize the contents of `jDpt` variable.

Figure 29: Reading, initializing the contents of the `jDpt` variable.

✓ **Creating a Cartesian variable `pOtherPoint` (of type `pointRx`)**

Go to the **JOG** menu to view the Cartesian variables of the `First_steps` application, as described in **3.2**, resulting in the page shown in the figure below (identical to figure 23):

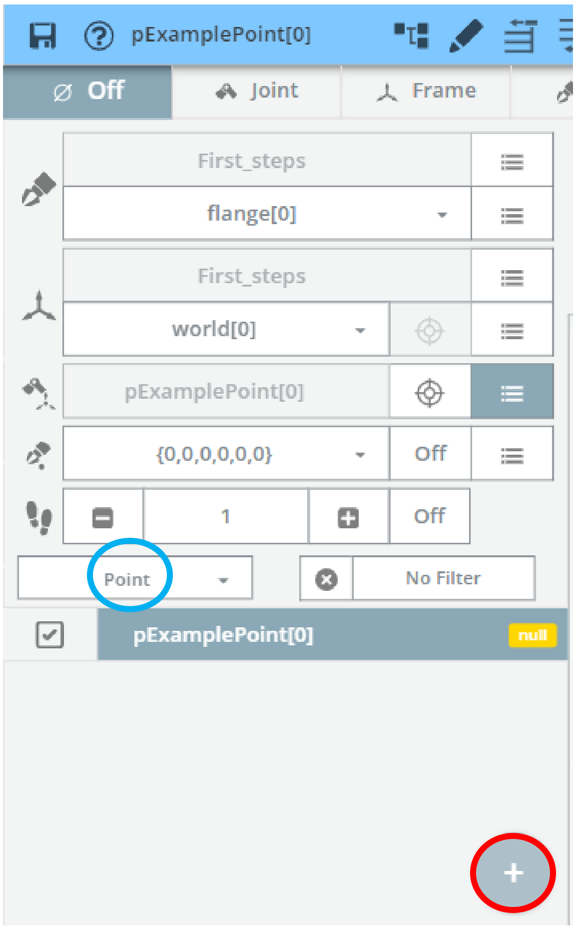


Figure 30: Display of the Cartesian variables (of `pointRX` type), in the present case `pExamplePoint`, in the **JOG** menu.

✓ Note that the point that will be created is defined in the frame `world` (*i.e.*, the reference frame R_0 of the robot), as shown in the figure below.

- Then press the  button circled in red (bottom right) in the previous figure. Select the contents of the fields in the window that appears:
  - **Name** to declare a Cartesian variable (**Type** `pointRx`), named `pOtherPoint`,
  - **Container** to indicate that the variable corresponds to an array (`array`) of dimension 1 (**Size** field),

  specify that its scope is private (**Public** field to off, selected by default), as shown in the figure below.

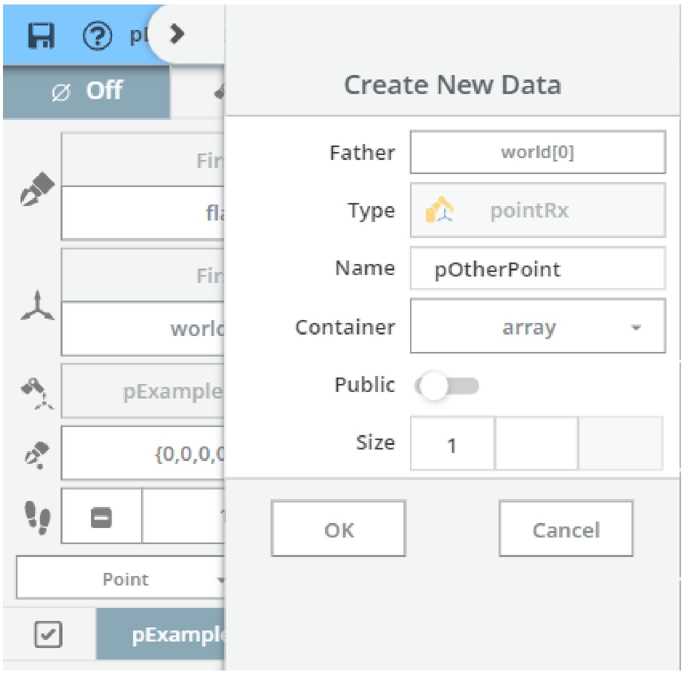


Figure 31: Creating the `pOtherPoint` variable.

which gives (after validation *via* the **OK** button) the result described in the figure below where the `pOtherPoint` variable appears.

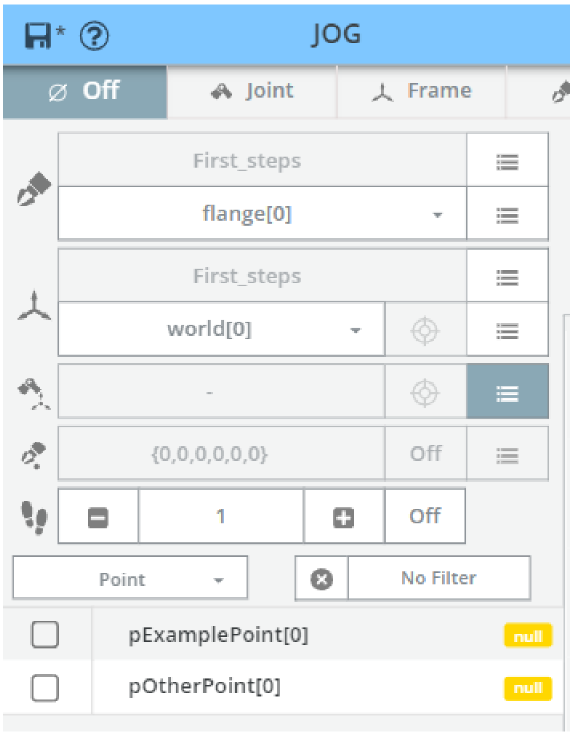


Figure 32: Display of the Cartesian variables (of `pointRX` type), in the present case `pExamplePoint` and `pOtherPoint`, in the **JOG** menu.

- Check the box for `pOtherPoint` variable, then press the **Edit** button at top of the menu bar to access the window described in the figure below, which allows you to read or initialize the contents of `pOtherPoint` variable through the field values X, Y, Z, RX, RY, RZ.

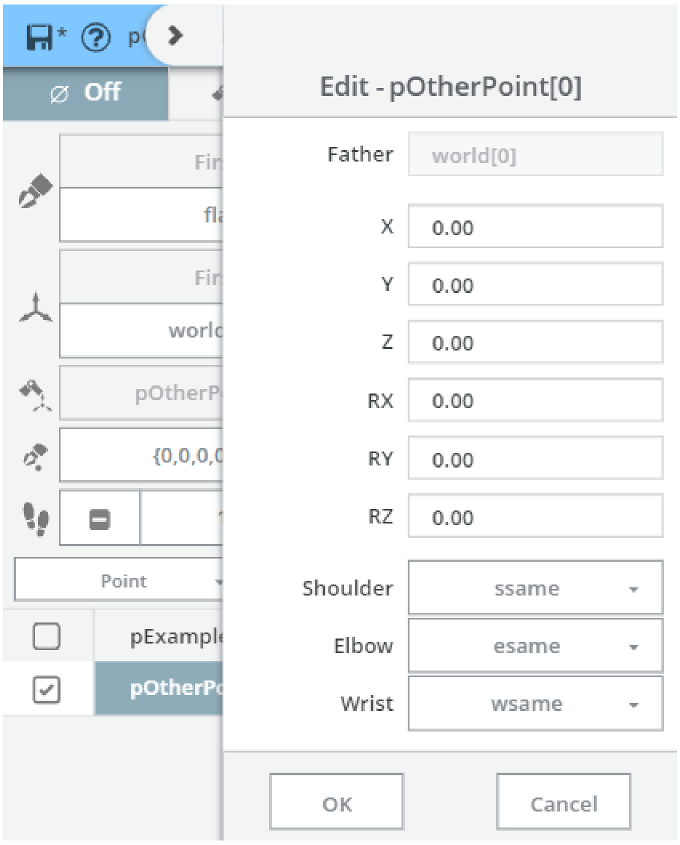


Figure 33: Read, write/initialize the contents of the `pOtherPoint` variable.