

Lab 1 - Robot Stäubli TX2-40 / Cs9 / SP2: Trajectory Generation

Jean-Louis Boimond
University of Angers

The objective of this lab is to allow a Stäubli TX2-40 robot arm, initially stretched vertically, equipped with a Cs9 controller and an electric gripper Schunk EGP 40-N-N-B:

- to pick up a rectangular part ($22 \times 22 \times 40 \text{ mm}^3$) located on an *initial location*, denoted **E1**,
- move the part to a *work area*, denoted **E2**, to wait for a time delay (to simulate a certain treatment),
- to deposit the part on a *final location*, denoted **E3**,
- to return the robot arm to vertical position,

knowing that the trajectory of the part from one location to the next is constrained by the presence of an obstacle through which the part must move: the first, between **E1** and **E2**, in the form of a corridor, entitled **Corridor**; the second, between **E2** and **E3**, in the form of a quarter circle, entitled **Circle**.

To obtain this, an introduction to programming a Stäubli robot is proposed in part **1** of the document. Part **2** is related to the use of the Schunk gripper. The *crossing points* (also named *relevant points*) of the trajectory to be achieved are acquired in part **3**. The programming of the trajectory from these points is done in parts **3** and **4**.

Table of Contents

1) Introduction to programming a Stäubli robot	1
2) Using the Schunk gripper	2
3) Acquisition of the relevant points of the gripper trajectory	3
4) Programming the robot to situate the TCP at the point <i>P0</i>	5
5) Programming the robot trajectory	5

1) Introduction to programming a Stäubli robot

The aim is to get started with the robot, as well as to introduce you to the basics of its VAL 3 programming with the *Teach Pendant* (called SP2 at Stäubli), using the document entitled 'Getting started with the TX2-40/Cs9 Stäubli robot, SP2 teach pendant, VAL 3 programming' accessible [<here>](#). It is up to you to test the features described in the document by making, among other things, a first program called `First_steps`, **while being careful** when the power is put on the robot arm.

In particular, please reduce the speed of the arm to 25% (see section 1.3, p. 4, of the basic training document of the robot) during this lab.

In the following, you will have to complete an application, called **TP_3A**, located in the hard drive of the controller. This application contains the basic code for the application to be created, including in particular certain variables used (and described) later.

2) Using the Schunk gripper

A Schunk EGP 40-N-N-B gripper is attached to the flange of the robot arm to be able to grasp, or release, the part to be moved. To do this, a variable entitled `tPinceSchunk`, of type `tool`, has been created in the **TP_3A** application. It is characterized by the values $X = 0, Y = 0, Z = 151.7, R_x = 0, R_y = 0, R_z = 0$, and by a link with 2 logical outputs entitled `FastIO/fOut0`, `FastIO/fOut1` allowing the gripper to be closed or opened. The use of this variable, rather than the `flange` variable (used by default), in a motion instruction will allow the *Tool Center Point* (TCP) to be located at the level of the part to be handled. Indeed, the R_{Tool} frame associated with the tool (the gripper) corresponds to that of the R_{Fl} frame (associated with the flange) after a 151.7 mm translation along the z_{Fl} axis, as shown in the following figure. Notice that the O_{Tool} origin of the R_{Tool} frame corresponds to TCP.

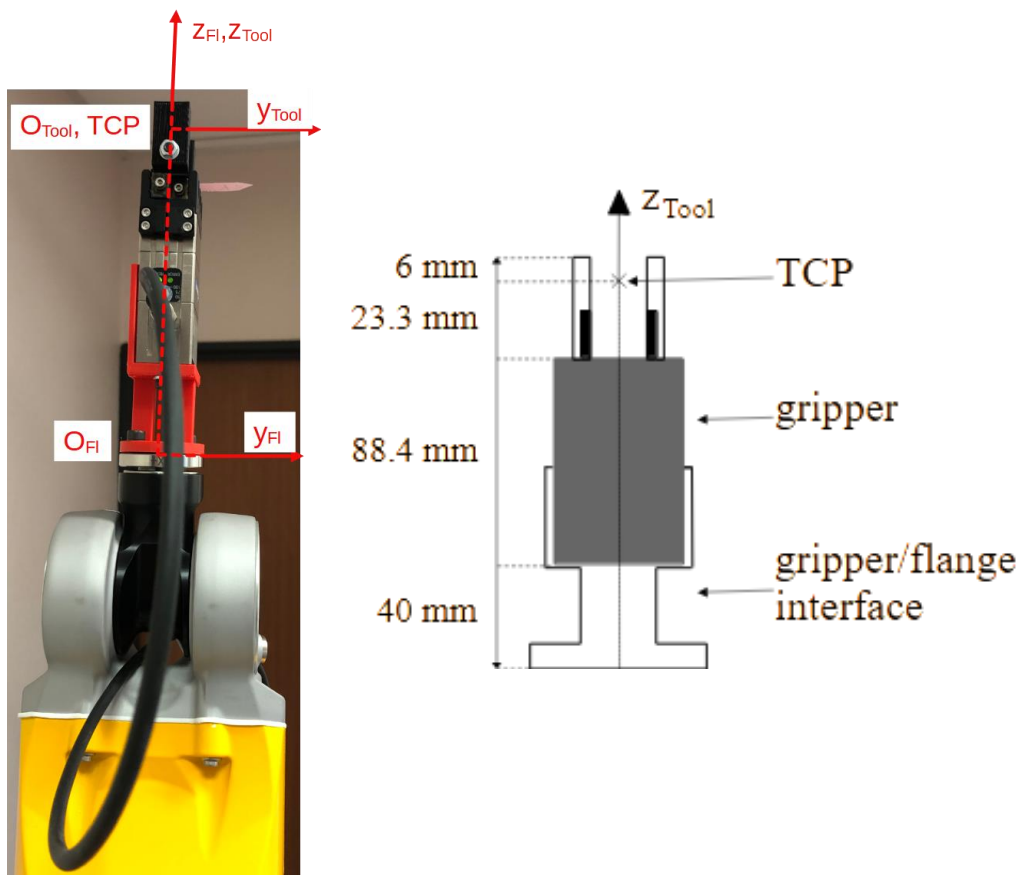


Figure 1: Diagram of the Schunk gripper with a representation of the R_{Fl} frame associated with the robot flange and the R_{Tool} frame associated with the gripper.

Question 1: The gripper being attached to the robot arm and assuming (because it is not the case) that the `First_steps` application (described in the document on how to get started with the robot) contains the `tPinceSchunk` variable (described above), what changes would you observe in the `First_steps` application if the motion instruction:

```
movej (pExamplePoint, flange, mNomSpeed)
```

was replaced by:

```
movej (pExamplePoint, tPinceSchunk, mNomSpeed) ?
```

Manipulation 2: Load the **TP_3A** application into RAM. Verify that the `tPinceSchunk` variable is declared in this application with the characteristics described above.

The instructions for opening or closing the gripper are as follows:

- to open the gripper:



```
// gripper opening
dioSet(diClose,0)
dioSet(diOpen,1)
```
- to close the gripper:

```
// gripper closing
dioSet(diClose,1)
dioSet(diOpen,0)
```

where the `diOpen` and `diClose` variables are connected to the logical outputs `FastIO/fOut0`, `FastIO/fOut1`.

Programming 3: As in the `First_steps` application, declare in the **TP_3A** application a `jDpt` variable, of type `jointRx`, such as $J1 = \dots = J6 = 0$ (see 2.2.2, 2.2.3 of the document on how to get started with the robot), which will allow the arm to be extended vertically. Complete the `start()` program of the application in order to: position the arm in this posture (see 2.3 of the document on how to get started with the robot), close the gripper (if it is not already!), then open it 2 second later to test the functioning of the gripper.

Ask your supervisor to check your code before running the application.

N.B.: It is possible to open or close the gripper by acting from the *Teach Pendant* on the `FastIO/fOut0` and `FastIO/fOut1` logic outputs of the Cs9 controller. To do this, press **IO>Boards>J212 FastIO** and select the **Digital Out** tab. The output status changes by pressing the corresponding key . Make sure that the outputs displayed, *i.e.*, *Fast Output 1* (for `FastIO/fOut0`) and *Fast Output 2* (for `FastIO/fOut1`), are not locked by pressing the key  knowing that this key is operational when the profile is the *maintenance* one (to do this, go to the **Settings>Profiles** window by putting in the **Current Profile** sidebar: *maintenance* (not *default*) in the *Name* field and *spec_cal* (not an empty string) in the *Password* field). The gripper is opened when the outputs *Fast Output 1=Off* and *Fast Output 2=On*; it is closed when the outputs *Fast Output 1=On* and *Fast Output 2=Off*.

3) Acquisition of the relevant points of the gripper trajectory

The trajectory to be realized is based on the data of certain *points*, considered *relevant*, able to *locate* (*i.e.*, *position* and *orientate*) the locations (**E1**, **E2**, **E3**) and the obstacles (**Corridor**, **Circle**) which are fixed on a millimeter plate. These points, represented by crosses in the following figure, will enable the robot arm to move the TCP (corresponding to the point of "contact" with the part, see Figure 1) from one location to the next, taking obstacles into account. Let:

- *P0* the point relating to the *initial location* (**E1**),
- *P1*, resp. *P2*, the point relative to the input, resp., output, of the **Corridor** obstacle,
- *P3* the point relative to the *work area* (**E2**),
- *P4*, resp., *P5*, the point relative to the input, resp., output, of the **Circle** obstacle,
- *P6* the point relative to the *final location* (**E3**).

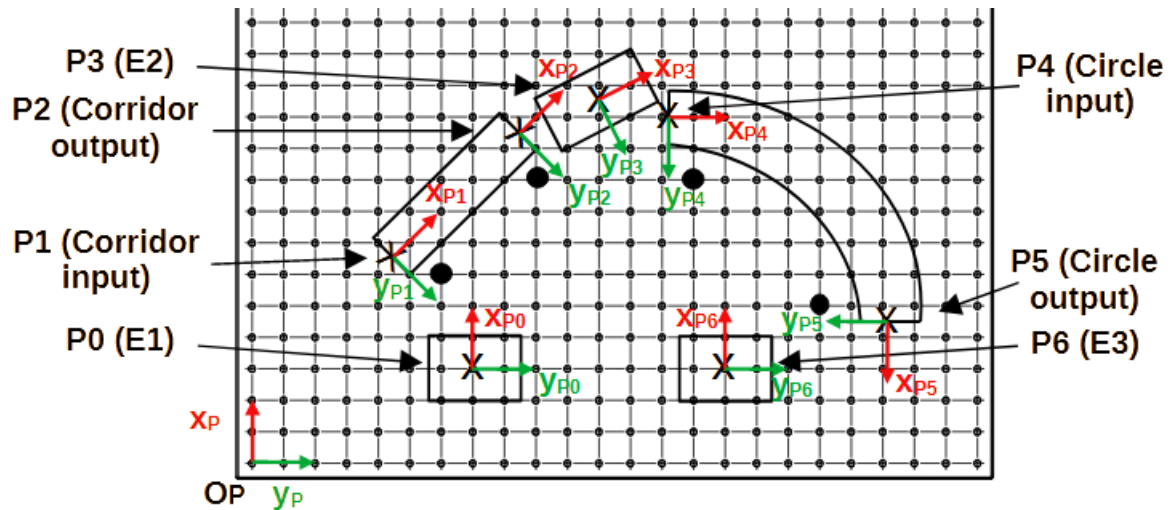


Figure 2: Locations of points P_0, \dots, P_6 on the millimeter plate.

Question 4: Recall why the z axis of these 7 points is opposite the z_0 axis of the reference frame of the robot arm.

The **following manipulation** will allow the acquisition in the application **TP_3A** of these 7 points (so that the TCP is able to be located on these points).

These points are not declared in joint space (*via* a `jointRx` variable), but in **Cartesian space** by giving: 3 coordinates to *position* the point and 3 others to *orientate* it in relation to a reference frame (by default, the reference frame of the robot arm) in a `pointRx` variable.

To have precise measurements, the acquisition of these points will be done while the gripper is holding the part (to be moved). In addition, the posture adopted to reach all the points will be such that the shoulder is on the left with the elbow up.

The z values of these points must be such that the position of the TCP corresponds to the description given in the following figure when the gripper closes, or opens, to grip, or release, the part (of height equal to 40 mm).

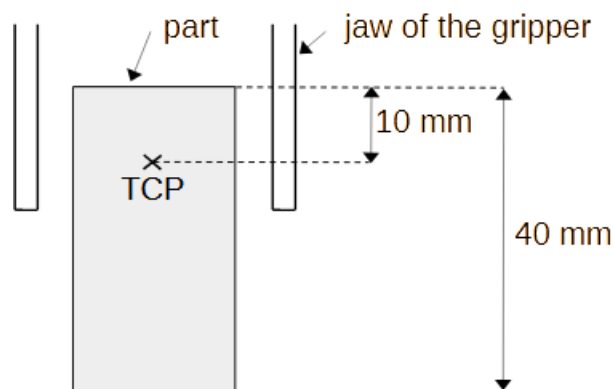


Figure 3: Position of the TCP when the gripper closes, or opens, to grip, or release, the part.

Manipulation 5: The aim is to store the points P_0, \dots, P_6 in the **TP_3A** application done in **Programming 3**. For this, move the robot arm to point P_0 and store this point (corresponding to the current point where the TCP is at the end of the move) in the Cartesian variable `pP0`; do the same thing for the other 6 points, *i.e.* P_1, \dots, P_6 , to store them in the variables `pP1, \dots, pP6`. To do this, see the N.B. at the end of appendix A.3 of the document on getting started with the robot.

More precisely:

- for p_{P0} point, first position the part on the *initial location* (**E1**). Then move the arm slowly (at 1% of its speed) close to **E1** until the gripper is ready to grip the part, then close the gripper (see previous N.B.). This point will then be stored in the variable p_{P0} ;
- **for safety reasons**, the points relative to the two obstacles, *i.e.* p_{P1} , p_{P2} , p_{P4} , p_{P5} , must be such that the bottom face of the part (to be moved) is at a safe distance of 5 mm from the obstacle supports (relative to the point p_{P3} , the part will not be released once located on location **E2**);
- finally, the point p_{P6} must be such that the gripper can open at this point to (carefully) deposit the part on the *final location* (**E3**).

Ask your supervisor to check your results before proceeding.

4) Programming the robot to situate the TCP at the point $P0$

As the approach movement towards $P0$ point is not (yet) managed, the part (to be moved) is not supposed to be positioned on the *initial location* (**E1**) (at the risk of the gripper not gripping it correctly!!). The aim is to complete the **TP_3A** application to enable:

- locate the TCP at j_{Dpt} point to stretch the robot arm vertically, open the gripper and wait 1 second (at j_{Dpt} point);
- locate the TCP at $P0$ point, close the gripper (to grip the part), then open it after 2 seconds (to release the part);
- reposition the robot arm vertically.

Programming 6: Complete/modify the `start()` program in the **TP_3A** application so that the robot arm achieves the objective described above.

Ask your supervisor to check your code before running the app.

5) Programming the robot trajectory

It is assumed that the part (to be moved) is positioned on the base of the *initial location* (**E1**) and that the **TP_3A** application is loaded in RAM.

The trajectory to be achieved is such that the robot arm must:

- Position itself in its initial configuration, *i.e.* locate the TCP at j_{Dpt} point; then open the gripper and wait 1 second;
- Grip the part located at *initial location* **E1**. To do this:
 - locate the TCP at a height of 10 cm above $P0$ point,
 - move the TCP down in a straight line to $P0$ point,
 - close the gripper, then wait 1 second (the part is gripped);
- Move the part to the input of the **Corridor** obstacle. To do this:
 - move the TCP up in a straight line at a height of 10 cm (above $P0$ point),
 - locate the TCP at a height of 10 cm above $P1$ point,
 - move the TCP down in a straight line to $P1$ point;
- Move the part to the output of the **Corridor** obstacle. To do this:

- move the TCP in a straight line to $P2$ point;
- e) Move the part to the *work area E2*. To do this:
 - move the TCP up in a straight line at a height of 10 cm (above $P2$ point),
 - locate the TCP at a height of 10 cm above $P3$ point,
 - move the TCP down in a straight line to $P3$ point,
 - wait for 2 seconds (delay simulating a certain processing of the part);
- f) Move the part to the input of the **Circle** obstacle. To do this:
 - move the TCP up in a straight line at a height of 10 cm (above $P3$ point),
 - locate the TCP at a height of 10 cm above $P4$ point,
 - move the TCP down in a straight line to $P4$ point;
- g) Move the part to the output of the **Circle** obstacle. To do this:
 - move the TCP by doing a quarter circle towards $P5$ point;
- h) Release the part on the *final location E3*. To do this:
 - move the TCP up in a straight line at a height of 10 cm (above $P5$ point),
 - locate the TCP at a height of 10 cm above $P6$ point,
 - move the TCP down in a straight line to $P6$ point,
 - open the gripper, then wait for 1 second (the part is put on the base of the location);
- i) Move the TCP up in a straight line at a height of 10 cm (above $P6$ point); reposition the robot arm in its initial configuration.

In addition to the instructions presented in the document concerning the handling of the robot, such as `movej()`, `waitEndMove()`, `delay()`, let us describe three useful instructions to achieve the desired trajectory.

(a) Approach and release from a point

The `appro` instruction enables you to define a Cartesian *point* at a certain distance from another *point*. For example, the `appro` instruction combined with the `movej` instruction as follows:

```
movej (appro (pP, {0, 0, h, 0, 0, 0}), tPinceSchunk, mNomSpeed)
```

locates the TCP at a distance h (in mm) from a pP *point* along its z_{pP} axis. An illustration is given in the following figure with $h = -100$ where the *point* reached after execution of the previous move instruction is denoted by $pP1$.

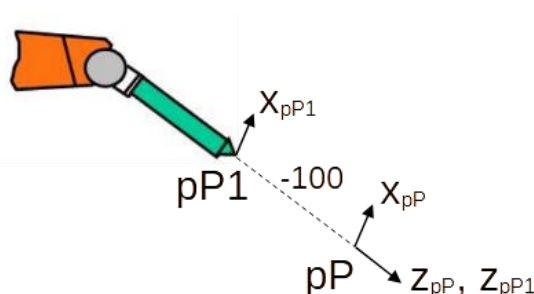


Figure 4: Example of how to use the `appro()` instruction.

Let be a *point* $pP2$ such that its z_{pP2} axis is opposite to the z_0 axis of the reference frame of the robot arm, which is the case of *points* $P0, \dots, P6$. In this case, the following three instructions:

```
movej (appro (pP2, {0, 0, -50, 0, 0, 0}), tPinceSchunk, mNomSpeed)
movej (pP2, tPinceSchunk, mNomSpeed)
movej (appro (pP2, {0, 0, -50, 0, 0, 0}), tPinceSchunk, mNomSpeed)
```

will:

- locate the TCP above $pP2$ *point* at a height equal to 50 mm (1st instruction),

- perform an *approach* movement by positioning the TCP at *pP2 point* (2nd instruction),
- perform a *release* movement by repositioning the TCP above *pP2 point* at a height equal to *50 mm* (3rd instruction).

Note that it is possible to perform movements in Cartesian space (and not in joint space *via* the `movej` instruction) in order to control the trajectory performed by the TCP **at all times**, see sections **b** and **c** below.

b) Straight-line movement

The instruction for moving the TCP in a **straight line** from the *current point* (i.e. the *point* reached by the TCP (just) before executing of the move instruction) to a *pPoint point* (of type `pointRx`) is as follows:

```
move1 (pPoint, tPinceSchunk, mNomSpeed)
```

In the case of the **Corridor** obstacle, such an instruction is particularly interesting with regard to the orientation of the part during its movement within the obstacle, in the sense that the orientation during the movement of the frame associated with the TCP does not change because the orientation of the *current point* (corresponding to *P1 point* located at the input of the **Corridor** obstacle) is the same as that of *P2 point* (located at the output of the obstacle), see Figure 2.

c) Circular movement

The instruction for moving the TCP in a **circular** movement from the *current point* to a *pPointDestination destination point* (of type `pointRx`), passing by a *pPointIntermediaire intermediate point* (of type `pointRx`), is as follows:

```
movec (pPointIntermediaire, pPointDestination, tPinceSchunk, mNomSpeed)
```

The orientation of the frame associated with the TCP during movement is derived from interpolation between the orientations of the *current point*, the *intermediate point* and the *destination point*. This feature is particularly useful for moving the part within the **Circle** obstacle, given the orientation of the *P4, P5 points* (see Figure 2) and the *P4_5 intermediate point* measured below.

Manipulation 7: As you did in **Manipulation 5**, we need to move the robot arm in order to acquire in the **TP_3A** application the *P4_5 Cartesian point* (located on the arc of the circle to be traversed), described in the figure below. Let:

- *X, Y, Z* are the position coordinates of *P4_5 point*;
- *RX, RY, RZ* are the orientation coordinates of *P4_5 point*, knowing that its *x* axis forms an angle of 45° with the *x* axis of *P4 point*, see the figure below.

Let `pP4_5` be the variable of type `pointRx` containing these coordinates. As with the points `pP1`, ..., `pP5`, the point `pP4_5` must be such that the bottom face of the part (to be moved) is at a safety distance of *5 mm* from the **Circle** obstacle.

Ask your supervisor to check your results before proceeding.

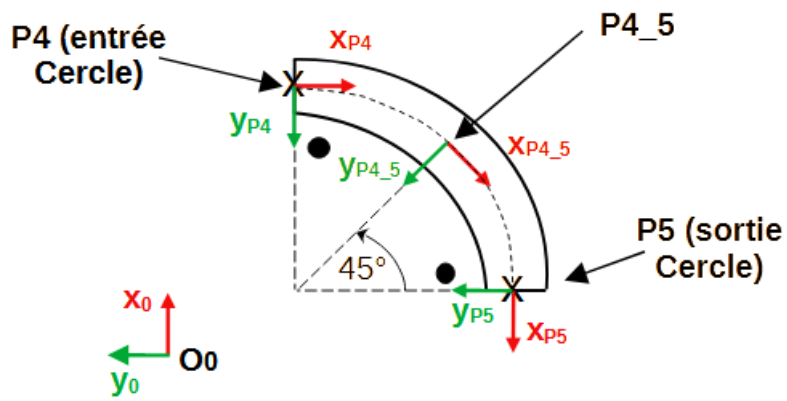


Figure 5 : Situation du point P_4_5 dans l'obstacle **Cercle**.

Programming 8: Complete/modify the `start()` program of the `TP_3A` application so that the robot arm completes the trajectory described through steps a, b, \dots, i at the beginning of section 5.