

Circular arc motion programming – Application on UR3e robot

Jean-Louis Boimond
University of Angers

Keywords: UR3e robot, circular arc segment, Polyscope, G-code file

We aim to understand how circular motions, more precisely, arc segment motions, are performed by the *Tool Center Point* (TCP) such a point indicating the *situation* (i.e., the *position* and *orientation*) of the tool attached to the (free) end of the robot arm. The circular arc is specified through three points defined in the reference frame, noted R_0 , of the robot (frame attached to its base), that is:

- A *start point*, noted $Pt1$, (also called initial point) which indicates the position of the TCP (just before the realization of the circular arc,
- An *auxiliary point*, noted $Pt2$ (also called intermediate point) through which the TCP passes during the circular arc motion,
- An *end point*, noted $Pt3$ (also called destination point) which corresponds to the point reached by the TCP (just) after the realization of the circular arc.

Arcs are oriented. For example, the arc depicted in the following figure is in the clockwise direction.

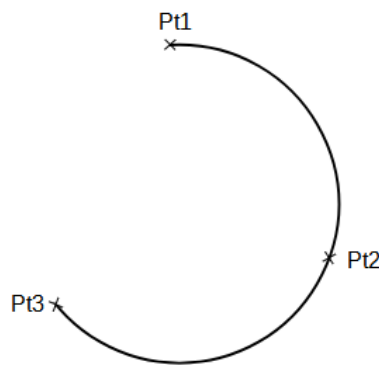


Figure 1: An example of circular arc defined by points $Pt1, Pt2, Pt3$.

Different options are usually proposed by manufacturers of industrial robots concerning the **orientation** of TCP during a circular motion; by example, for UR robots:

- i*) A *constant* orientation, equal to the one of the *start point* $Pt1$ (the orientations of points $Pt2, Pt3$ are disregarded);
- 2i*) A *variable* orientation allowing a gradual transition from the orientation of point $Pt1$ to the one (different) of point $Pt3$ (the orientation of point $Pt2$ is disregarded). Such a movement minimizes the rotation of the tool, see TP4 [Génération de trajectoires rectilignes du Tool Center Point dans l'espace des tâches](#) for details;
- 3i*) An orientation such that the movement of the longitudinal axis of the tool (i.e., the axis Z of the TCP) is established relative to the plane and tangent of the arc, which allows, for example, a welding operation, see <https://www.youtube.com/watch?v=jWhsL74CL9c>.

The orientation control defined by option *i* (resp., *2i*) is *base-related* in the sense that it depends on the orientation of point $Pt1$ (resp., points $Pt1, Pt3$) defined in base frame (R_0). The orientation control

defined by option 3i is *path-related* in the sense that it directly depends on the shape of the circular arc.

Here we are interested in the calculation of points (in $R^6: R^3$ for the *position* + R^3 for the *orientation*), noted P , that allow the TCP to move approximatively along a circular arc (defined by points $Pt1, Pt2, Pt3$) with a TCP orientation defined by the option 3i (described below). More precisely:

- The **position** of points P will be such that they belong to the circular arc with a TCP trajectory corresponding to the passage *in straight line* from one point P to the next point P . The number, noted Nb , of points P must be large enough so that the generated motion is judged to be close enough to the desired circular arc, see the following figure.

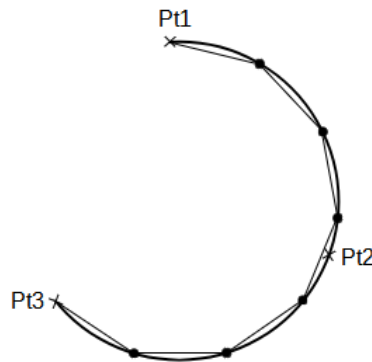


Figure 2: Approximation of a circular arc through 8 points P linked by straight lines.

- The **orientation** of points P along the circular arc will be such that the frame X, Y, Z linked to the TCP verifies that:
 - The axis X is directed towards the center of the arc,
 - The axis Y is tangent to the circular arc,
 - And so, due to the orthogonality of the frame, the axis Z is normal to the radius of the circular arc,

see the following figure:

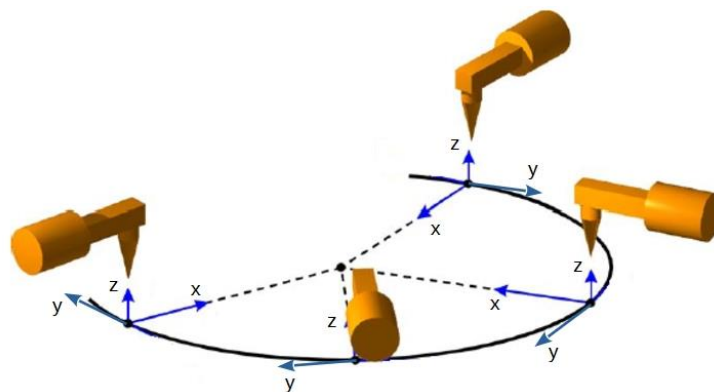


Figure 3: Circular motion with an orientation built from the shape of the circular arc (option 3i).

- ✓ Above all let us load in your working directory the zip file [Tools Circular Arc.zip](#) containing two MatLab functions (Angles_Euler_convention_ZYX_en_fct_matrice_Cosinus_Dir.m, Fct_Creation_file_Gcode.m) and a script file (File_Toolpath.script) written in a pseudo Python executable by UR controllers.

Steps to build the circular arc motion applied to the UR3e robot:

A) About the *position* of points P

A.1) Computation of the center and radius of the circle defined by the points $Pt1, Pt2, Pt3$

A.2) Computation of the arc travel direction

A.3) Computation of the *position* of points P

B) Computation of the *orientation* of points P

C) Programming the motion with Polyscope

C.1) Short description of Polyscope

C.2) Implementation of the Polyscope program

A) About the *position* of points P

A.1) Computation of the center and radius of the circle defined by the points $Pt1, Pt2, Pt3$

Let us consider the particular (simpler) case where the 3 points (of course not aligned) belong to a same plane X, Y , *i.e.*, have the same coordinate, noted Z_{cst} , along the axis Z of the reference frame (R_0) of the robot. We note:

- x_C, y_C the coordinates, noted C , of the circle center in the plane X, Y where the circle is situated; R its radius,
- x_1, y_1 the coordinates of point $Pt1$; x_2, y_2 the ones of point $Pt2$; x_3, y_3 the ones of point $Pt3$.

➤ **Question 1:** Calculate, in a literal form, the values of x_C, y_C and R . Test your result on a simple example.

➤ **Realization 2:** Code a MatLab script that computes the values x_C, y_C, R from the coordinates of points $Pt1, Pt2, Pt3$. Display in a figure the circle (see instruction `viscircles`), its center, and the points $Pt1, Pt2, Pt3$ (see instructions `hold` and `plot`), remark that a representation in R^2 (instead of R^3) is sufficient due to the same Z coordinate of points. Verify that a circle defined with the following coordinates x, y for points $Pt1, Pt2, Pt3$:

$$Pt1 = (100 - \sqrt{1500}, -40), Pt2 = (100 + \sqrt{1500}, -40), Pt3 = (100 + \sqrt{1200}, -70),$$

has a center equal to $(100, -50)$ and a radius equal to 40.

➤ **Question 3:** Which geometric shape should be used to intersect the plane (through which the 3 points pass) to obtain the equation of the circle when the plane is arbitrary (not necessarily orthogonal to the axis Z)?

A.2) Computation of the arc travel direction

Once the equation of the circle has been obtained, we need to calculate the direction (indicated by an arrow) of the arc of the circle between points $Pt1$ and $Pt3$. This direction depends on the position of the *auxiliary* point $Pt2$ and will determine the portion of the circle to travel, as illustrated in the following figure.

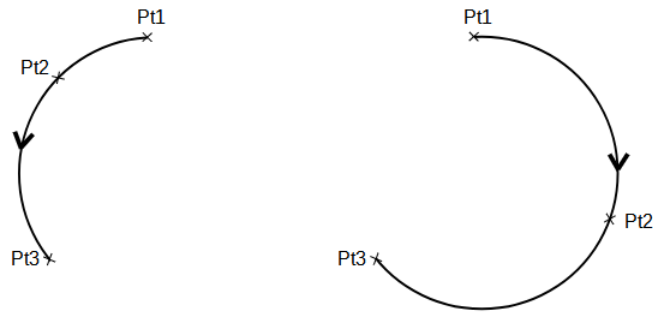


Figure 4: The direction of the arc is: counterclockwise in the figure on the left, clockwise in the figure on the right (note that points $Pt1$ et $Pt3$ are the same for both figures).

The direction of the arc to be travelled can be deduced from the sign of the (directed) angle, denoted φ , between the half-lines $Pt1Pt2$ et $Pt1Pt3$ (note that $\varphi \neq 0$ due to $Pt2 \neq Pt3$): the direction is counterclockwise when φ is positive, it is clockwise when φ is negative, as illustrated in the following figure.

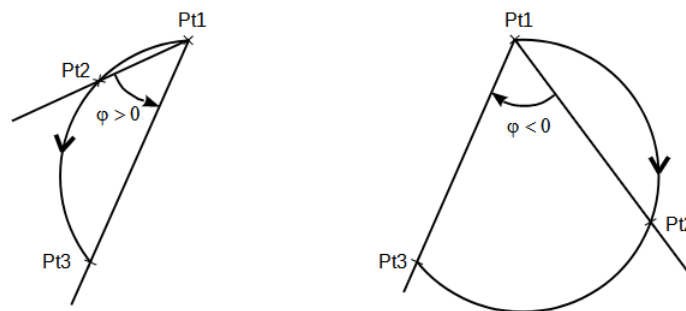


Figure 5: Counterclockwise direction in the figure on the left, clockwise direction in the figure on the right.

➤ **Question 4:** Show that the MatLab instruction:

```
phi=wrapTo180 (atan2d(Pt3(2)-Pt1(2), Pt3(1)-Pt1(1)) - atan2d(Pt2(2)-Pt1(2), Pt2(1)-Pt1(1)))
```

allows the calculation of angle φ in degree given x, y coordinates of points $Pt1, Pt2, Pt3$ (contained in variables $Pt1, Pt2, Pt3$), the value of φ being between -180° and 180° .

➤ **Realization 5:** Complete the script corresponding to 'Realization 2' by inserting the variable `phi`. Test the validity of the result through some examples.

A.3) Computation of the position of points P

As a reminder, Nb is the number of points P (belonging to the circular arc) used to achieve the trajectory. Be sure to take $Nb \geq 3$ so that you have at least one point in addition to points $Pt1, Pt3$; for example, Nb is equal to 8 for the example considered in Figure 2. Let be $P_x(n), P_y(n), P_z(n)$ the coordinates of the n^{th} point, denoted $P(n)$.

Now we need to calculate the coordinates of the points $P(1), \dots, P(Nb)$. The coordinates of *start point* ($Pt1$) and *end point* ($Pt3$) are known, we have:

$$P_x(1) = Pt1(1), P_y(1) = Pt1(2), P_z(1) = Z_{cst},$$

$$P_x(Nb) = Pt3(1), P_y(Nb) = Pt3(2), P_z(Nb) = Z_{cst}.$$

$Nb - 2$ points remain to be calculated, that is, the points $P(2), \dots, P(Nb - 1)$. The coordinates of point $P(n)$, for $2 \leq n \leq Nb - 1$, are calculated as follows:

$$\begin{aligned} P_x(n) &= x_C + R \times \cos(\varphi_n), \\ P_y(n) &= y_C + R \times \sin(\varphi_n), \\ P_z(n) &= Z_{cst}, \end{aligned}$$

where φ_n is the angle between the half-line $CP(n)$ and that carried by the vector \vec{i} , as shown in the following figure where the point $P(n)$ is considered with $n = 3$.

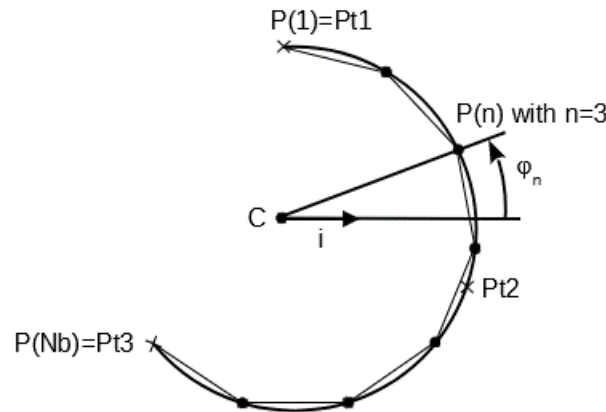


Figure 6: Calculation of coordinates of point $P(n)$, for $2 \leq n \leq Nb - 1$.

➤ **Question 6:** Write the MatLab instructions allowing the calculation of angles φ_1 and φ_{Nb} (used in the following to calculate points $P(2), \dots, P(Nb - 1)$).

Three cases are to be considered to calculate φ_n , for $2 \leq n \leq Nb - 1$, according to the sign of the angle φ (which indicates the direction of the arc) and the one of angle, noted $\varphi_{1,Nb}$, in the interval $[-180^\circ, 180^\circ]$, that there is between the half-lines $CP(1)(= CPt1)$ and $CP(Nb)(= CPt3)$. Indeed, we have:

i) $\varphi_n = \varphi_1 + \varphi_{1,Nb} \times \frac{n-1}{Nb-1}$ when $(\varphi > 0$ and $\varphi_{1,Nb} > 0)$ or when $(\varphi < 0$ and $\varphi_{1,Nb} < 0)$, see the following figure:

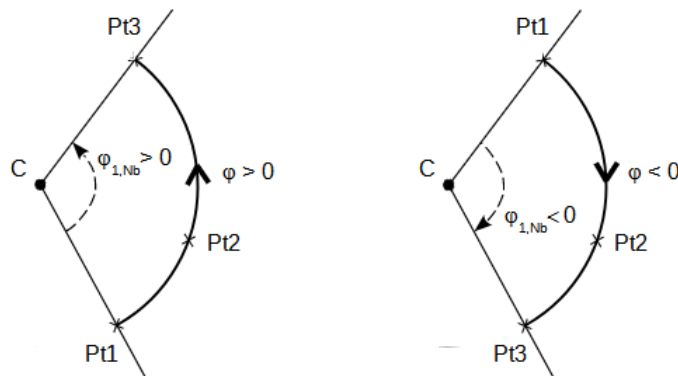


Figure 7: Case when $(\varphi > 0$ and $\varphi_{1,Nb} > 0)$ or $(\varphi < 0$ and $\varphi_{1,Nb} < 0)$.

2i) $\varphi_n = \varphi_1 + (360 + \varphi_{1,Nb}) \times \frac{n-1}{Nb-1}$ (in degree) when $\varphi > 0$ and $\varphi_{1,Nb} < 0$, see the following figure:

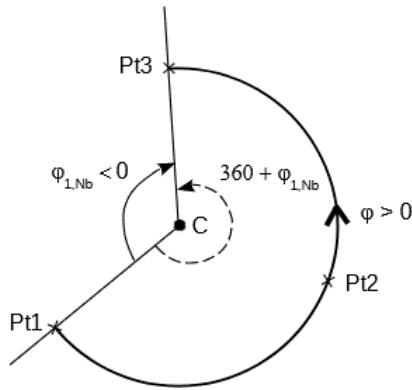


Figure 8: Case when $\varphi > 0$ and $\varphi_{1,Nb} < 0$.

3i) $\varphi_n = \varphi_1 + (-360 + \varphi_{1,Nb}) \times \frac{n-1}{Nb-1}$ (in degree) when $\varphi < 0$ and $\varphi_{1,Nb} > 0$, see the following figure:

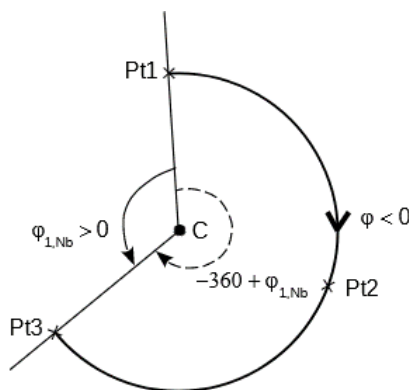


Figure 9: Case when $\varphi < 0$ and $\varphi_{1,Nb} > 0$.

➤ **Realization 7:** Complete the script corresponding to 'Realization 5' by constructing three column vectors, noted P_x , P_y , P_z , containing the coordinates x, y, z of points $P(1), \dots, P(Nb)$ with $Z_{cst} = 10$ (mm) and $Nb = 10$.

B) Computation of the orientation of points P

Like the example shown in Figure 3, the frame $(P(n), \vec{x}(n), \vec{y}(n), \vec{z}(n))$ defining the orientation of point $P(n)$, for $1 \leq n \leq Nb$, is such that:

- $\vec{x}(n)$ is oriented towards the center of the circular arc,
- $\vec{y}(n)$ is tangent to the trajectory,
- $\vec{z}(n)$ is normal to the radius of the circular arc,

as illustrated in the following figure:

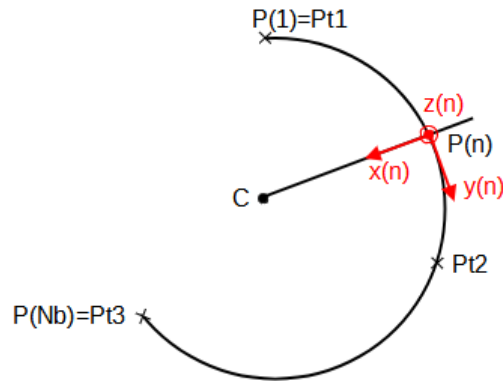


Figure 10: Frame associated with point $P(n)$.

- **Question 8:** Express the vectors $\vec{x}(n), \vec{y}(n), \vec{z}(n)$ as a function of the coordinates of the points C and $P(n)$; deduce the expression of the corresponding Direction Cosine matrix, noted $A(n)$.
- **Realization 9:** Complete the script corresponding to 'Realization 7' by constructing the matrix $A(n)$ for each of the points $P(1), \dots, P(Nb)$.

In order to create a script for the cobot UR3e, the orientation of frames associated with points $P(1), \dots, P(Nb)$ is defined from Euler angles Ψ, θ, φ according to the convention ZYX .

- **Realization 10:** Complete the script corresponding to 'Realization 9' by constructing three column-vectors, noted `psi`, `theta`, `phi`, containing, resp., the values of angles Ψ, θ, φ for each of the points $P(1), \dots, P(Nb)$. To do this, you can use the function:

```
'[psi,theta,phi]=Angles_Euler_convention_ZYX_en_fct_matrice_Cosinus_Dir(A) .m'
```

(contained in the zip file [Tools Circular Arc.zip](#)) that allows the computation of Euler angles Ψ, θ, φ (according to the convention ZYX) corresponding to the Direction Cosine matrix A (of dimension 3×3) given as an input parameter to the function.

To validate your script, consider that $Nb = 10$ and the coordinates x, y, z of points $Pt1, Pt2, Pt3$ are:

$$Pt1 = (100 - \sqrt{1500}, -40, Z_{cst}), Pt2 = (100 + \sqrt{1500}, -40, Z_{cst}), Pt3 = (100 + \sqrt{1200}, -70, Z_{cst}) \text{ with } Z_{cst} = 10 \text{ (mm)},$$

which corresponds to a circular arc with a center equal to $(100, -50)$ and a radius equal to 40 (mm), situated in the plane XY set at a height of 1 cm from the plane X_0Y_0 . Verify that you obtain:

```
psi=[-14.4775 -36.2022 -57.9270 -79.6517 -101.3764 -123.1011 -144.8258 -166.5506 171.7247 150.0000]',  
theta = phi = [0 0 0 0 0 0 0 0 0 0]'
```


Explain why vectors `theta` and `phi` are null.





C) Programming the motion with Polyscope

Before coding a program with Polyscope, we have to power up the robot controller, then to put the robot in *Normal mode* ('mode *Normal de fonctionnement*') to allow the arm to move:

- 1) (Power up the robot controller) The robot controller is powered up by pressing the power button (⏻) located at the top of the *Teach Pendant*. Wait for the main Polyscope screen, including tabs (located at the top of the screen): the **Run** tab to run programs; the **Program** tab to create/modify programs; the **Installation** tab to configure robot and Polyscope settings (including TCP, payload,

I/O, security); the **Move** tab to move (jog) the robot arm, either by moving its joints individually (in the joint space) or by translating/rotating the TCP (in the operational space), in order to learn points; the **I/O** tab to define and scan robot inputs/outputs; the **Log** tab to display warning/error information about the robot arm; the **Program and Installation Manager** referring three icons allowing the creation (**New...**), the loading (**Open...**) or the saving (**Save...**) of programs; **Manual** to indicate that the operational mode of the robot is set to Manual (rather than in Automatic or Remote Control modes); the **Safety Checksum** tab to display the active safety configuration, including maximal joint ranges, position/speed of joints, tool position, I/O ; the **Hamburger Menu** (**☰**) tab for help, settings, and to shutdown the robot controller (once the robot arm is powered down).

2) (Put the robot in *Normal mode*) After powering up the robot controller, the robot arm is still powered off, which is indicated by the red color of the  **Power off** ('*Mise hors tension*') button located in the footer (bottom left) of the screen. To power up the arm and therefore the robot in *Normal mode* ('*mode Normal de fonctionnement*')

- Tap this red button to cause the appearance of an *Initialize* screen; tap the  **ON** button to put the robot in *Active mode* (the arm is under tension but still unable to move due to brakes) which is indicated by the  **Idle** ('*Veille*') button located in the footer,
- Tap the  **START** ('**DÉMARRER**') button to release the brake system, therefore the robot is in *Normal mode* ('*mode Normal de fonctionnement*'), indicated by the  **Normal** button located in the footer,
- Tap the **Exit** button to return to the main screen.

N.B.: The posture corresponding to the arm raised vertically (which corresponds to its resting position) is said to be *singular*. This posture can be left by manually moving the arm, to do this: select the **Move** tab, then tap the **Freedrive** ('**Operation**') button (located at the bottom of the screen). It is then possible to manually pull the robot arm (in a non-vertical posture) as long as you press the **Freedrive – Press & Hold** ('**Fonctionnement libre – Maintenez appuyé**') button.

C.1) Short description of Polyscope

Polyscope uses the touch screen of the *Teach Pendant* as a graphical user interface. Its functioning is detailed [here](#) in English (version 5.11) or [here](#) in French (version 5.0.0). We limit ourselves here to giving the essential concepts needed to create and run a program like the one you are about to realize.

For example, let us consider the realization of a movement defined by 2 points, entitled `Waypoint_1`, `Waypoint_2`. The creation of the program enabling such a movement is as follows:

1. Select the **New...>Program** tab to create a program (initially without name and instruction), situated in a screen located on the left of the screen, in which instructions can be inserted (note that the tab selected in the screen positioned on the right side of the screen is **Command**, rather than **Graphics** or **Variables**).
2. The instructions that can be selected are the (basic) ones listed in the **Basic** ('**De base**') menu (menu selected by default in the banner on the left of the screen). Tap **WayPoint** ('**Pointpassage**') button to add a movement instruction to the program tree, entitled **Movej** ('**DéplacementA**'), with an associated point entitled **Waypoint_1** (by default), see figure below.

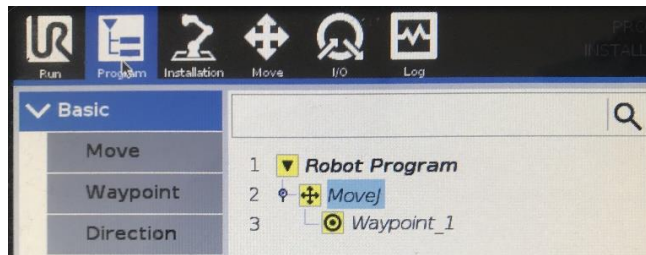


Figure 11: Instruction **Movej**.

This movement instruction, calculated in Joint space, enables rapid arm movement to the **Waypoint_1** point (to be defined).

3. Select this waypoint from the program tree, then in the screen of the **Command** tab (situated on the right of the screen), tap **Set Waypoint** ('**Régler le point de passage**') button, which displays the screen of the **Move** ('**Déplacement**') tab. Be careful to select from the drop-down menu **Feature** ('**Fonction**') the **Base** item (rather than **View** item) to display the coordinates in the robot base frame (R_0), see ① in the following figure.







Figure 12: Setting point **Waypoint_1**.

4. Move the robot arm knowing that several methods are possible:
 - By pressing the arrow keys (on the left) in the *TCP Position* ('*Position PCO*') and *TCP Orientation* ('*Orientation PCO*') screens, see ① in the previous figure,
 - By pulling (manually) the robot arm into desired posture, to do that: tap **FreeDrive** ('**Fonctionnement**') button; then press and hold **FreeDrive – Press & Hold** ('**Fonctionnement libre – Maintenez appuyé**') button, as stated in the previous *Nota Bene*,
 - By selecting in the *Tool Position* ('*Position de l'outil*') screen, situated at top right of screen (see ② in the previous figure), a coordinate **X, Y, Z, RX, RY** or **RZ**, which displays a screen enabling you to modify its contents, validated with the **OK** button,

- By selecting in the *Joint Position* ('*Position d'articulation*') screen, situated at bottom right of screen (see ③ in the previous figure), an arrow linked to one of the six joints of the arm robot.
5. Once the robot arm is in the desired posture, tap **OK** button (see ④ in the previous figure) to finalize the definition of the **Waypoint_1** point (a return to the screen displaying the program follows).
 6. Repeat steps 2 to 5 to define the second point, entitled **Waypoint_2** (by default). Note that the arm movement is performed *via* a single motion instruction (that is, **Movej** ('**DéplacementA**')), which is based on two points.
 7. After moving away from the robot arm, select **Graphics** tab to simulate the evolution of the arm as it (really) moves. Then tap **Play** (▶) button, situated at bottom right of screen; then press **Play from beginning – Robot Program** ('**Lecture depuis le début – Programme de robot**') button. In the screen that appears, press and hold **Move robot to: Waypoint_1** ('**Déplacer le robot à PointPassage_1**') button to move the arm - under your control (knowing that it can be in any position at the start) - to the first point of the trajectory, that is **Waypoint_1**. You can then execute (in a loop) the program (performing the movement from point **Waypoint_1** to point **Waypoint_2**) by tapping **Play from: Robot Program** ('**Lancer à partir de Programme de robot**') button.
 8. Tap **Pause** (⏸) button to pause arm movement or **Stop** (■) button to stop it.

N.B.: The toolbar of the program tree, situated at bottom of the screen where the program is listed, facilitates the construction of the current program, indeed:

- Buttons ↑ and ↓ change the position of a node, previously selected, in the program tree,
 - Buttons ↶ and ↷ are used to cancel/redo command changes,
 - Buttons ✂ and 📄 are used to cut/copy a program node to be used for other actions *via* button **Paste** 📄,
 - Button 🗑 deletes a node from the program.
- **Saving and loading of a program:** You can save the current program by tapping on **Save...>Save Program As...** tab which opens a screen entitled *Save Program* ('*Enregistrer le programme*'). Enter a file name, then validate it by tapping **Save** button (the file will have the *.urp* extension as a Universal Robots program file). The **Open...>Program** tab lets you load a file in a similar way as you save a file.
 - **Other instructions:** There are of course other instructions than *Movej*, in particular:
 - In '*Basic*' menu: **Popup** to display a customized message, with program execution stopped until the user presses **OK** in the Pop-up dialog box; **Halt** to stop program execution;
 - Standard instructions such as **Loop**, **SubProg**, **Assignment**, **If**, are available in the '*Advanced*' menu.
 - **Robot shutdown:** The robot is stopped through the following steps:
 - (Moving the arm to its vertical position (where all joint values are zero)) Select the **Move** tab, then tap **Home** ('**Départ**') button. A screen appears in which the robot arm can be moved to its vertical position as long as you press the **Move robot to: New position** ('**Déplacer le robot à: Nouvelle position**') button. The **Continue** button enables to come back in the screen corresponding to the **Move** tab,

- (Power down the arm) Tap the  **Normal** button, situated at bottom right of screen, to access a screen where you can tap the  **OFF** button. The **Exit** button enables to come back to the initial screen,
- (Robot stop) Select the **Hamburger menu** () tab and tap the **Shutdown Robot** () button, then tap the **Power off** and **Discard changes** buttons to stop the robot controller after a few moments.

C.2) Implementation of the Polyscope program

We are now interested in creating a program allowing the TCP to reach points $P(1), \dots, P(Nb)$ by connecting them with straight lines, like the circular arc shown in Figure 2 and with an orientation illustrated in Figure 10.

To do this, the MatLab script resulting from 'Realization 10' will transmit the coordinates of the points $P(1), \dots, P(Nb)$ to the Polyscope program (to create) by using a G-code file (understandable by Universal Robots). The G-code language was originally developed to program numerically-controlled machine tools by producing (in particular) motion instructions to the controller to enable the tool (attached to the machine tool) to realize lines, arcs, splines, etc. Today, this language is used for other types of applications such as 3D printing and robotics.

The G-code file to be used, entitled 'File_Gcode.nc' (nc being the extension of G-code files), consists of a header (indicating the processing mode of the motion instructions) followed by the motion instructions in charge of *situating* the TCP at points $P(1), \dots, P(Nb)$ by connecting them with straight lines.

- **Realization 11:** The file 'File_Gcode.nc' is created by incorporating, at the end of the MatLab script corresponding to 'Realization 10', the function 'Fct_Creation_file_Gcode.m' (contained in zip file [Tools Circular Arc.zip](#)) with the MatLab variables: Nb, P_x, P_y, P_z, phi, theta, psi as input parameters of the function.

In the case of a UR robot such as UR3e, the use of a G-code file in a Polyscope program is done *via* a 'script' written in a language close to Python; this script, entitled 'File_Toolpath.script' (contained in zip file [Tools Circular Arc.zip](#)), is as follows:

```
(L1)      global Waypoint_1_p=p[0.2,0.1,0.2,0.0,0.0,0.0]
(L2)      global Waypoint_1_q=[1.09,-1.33,-2.57,-0.81,1.57,-0.48]
(L3)      set_tcp(p[0.0,0.0,0.04,3.14159,0.0,0.0])
(L4)      while (True):
(L5)          movej(get_inverse_kin(Waypoint_1_p,qnear=Waypoint_1_q),a=1.5,v=1.0)
(L6)          sleep(3.0)
(L7)          mc_initialize(0,p[0.0,0.0,0.04,3.14159,0.0,0.0],6)
(L8)          mc_set_pcs(p[0.3,0.1,0.0,0.0,0.0,0.0])
(L9)          id_pa=mc_load_path("/programs/File_Gcode.nc",use_feedrate=False)
(L10)         id_1_1=mc_add_path(id_pa,1.0,0.05,0.0)
(L11)         mc_run_motion(id_1_1)
(L12)      end
```

where:

- Line L1 defines the point coordinates, entitled `Waypoint_1_p`, in the operational space as:

$$X = 0.2, Y = 0.1, Z = 0.2 \text{ (in m)}, RX = 0, RY = 0, RZ = 0 \text{ (in rd)},$$

the aim of this point is to bring the TCP close to the X, Y plane where the circular arc will be located (more precisely, close to the PCS defined in line L8);

- Line L2 defines a posture in joint space, by using six joint coordinates (in rd), to reach the previous point. The corresponding point, entitled `Waypoint_1_q`, is used in line L5 to indicate the

posture to adopt (among eight potentially possible solutions) when the TCP reaches point `Waypoint_1_p` through a movement generated in operational space (due to `movej`);

- Line `L3` defines the TCP coordinates X, Y, Z, RX, RY, RZ with respect to the frame associated with the robot flange whose X axis is represented in red, Y axis in green, Z axis (not represented) being oriented towards the outside of the robot flange. Note that the TCP is positioned at 4 cm along the Z axis of the frame associated with the robot flange, which corresponds to the tip of the pencil, see the following figure:

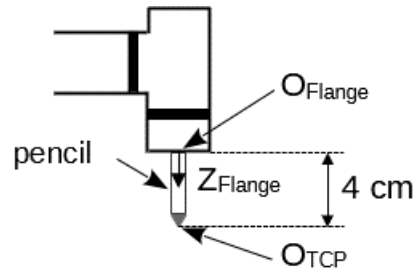


Figure 13: Position of the TCP origin (O_{TCP}) in the frame associated with the robot flange.

- Lines `L4` and `L12` form an infinite loop that allows the movement to repeat itself until the operator decides to stop it;
- Line `L6` pauses the movement for 3 seconds to allow time for the operator to pause the movement to check the *situation* at the point `Waypoint_1_p` (this line is useful for answering Question 14);
- Line `L7` initializes a motion sequence by using the `mc_initialize` function with the following input parameters:
 - the value 0 in the first parameter to indicate that the TCP is attached to the (free) end of the robot arm,
 - in the second parameter, the point defined by the coordinates X, Y, Z, RX, RY, RZ (in the operational space) corresponding to the *situation* of the TCP with respect to the frame associated with the robot flange,
 - the value 6 in the third parameter to indicate that the points described in the following are defined in R^6 , enabling the TCP to be *situated* (that is, *positioned* and *oriented*);
- Line `L8` *situates* the *Part Coordinate System* (PCS), the latter represents a *reference frame* defined as follows:

$$X = 0.3, Y = 0.1, Z = 0 \text{ (in m)}, R_x = 0, R_y = 0, R_z = 0 \text{ (in rd)},$$

with respect to the robot base frame (R_0), for the points defined in the G-code file;

- Line `L9` enables the loading of the G-code file entitled 'File_Gcode.nc', the identifier (`id_pa`) returned by the function `mc_load_path` is used in line `L10` (by function `mc_add_path`) to add a motion as programmed in the G-code file. The parameter `use_feedrate` set to `False` indicates that the speed and acceleration (if any) indicated in the G-code file are not taken into account;
- Line `L11` causes the execution of the movement *via* the identifier (`id_1_1`) (returned by the function `mc_add_path`).

- **Question 12:** Relative to line L3, graphically represent the *situation* of the TCP frame with respect to the frame associated with the robot flange. Graphically represent in the robot base frame (R_0): the *situation* of the frame associated with the point Waypoint_1_p, and the one of the PCS frame. Explain why the TCP takes the *situation*, shown in the following figure, when it reaches the point Waypoint_1_p (reached at the end of the execution of line L5):

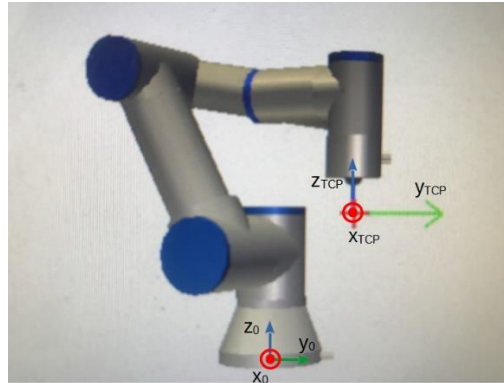


Figure 14: TCP *situation* at point Waypoint_1_p.

- **Realization 13:** Verify that the G-code file (readable *via* Notepad), generated by your MatLab script with $Nb = 10$, $Pt1 = (100 - \sqrt{1500}, -40, Z_{cst})$, $Pt2 = (100 + \sqrt{1500}, -40, Z_{cst})$ and $Pt3 = (100 + \sqrt{1200}, -70, Z_{cst})$ with $Z_{cst} = 10$ (mm) (assumptions considered in 'Realization 10'), corresponds to:

```
(L1)          G90 G21

(L2)          (About the coordinates of points:)
(L3)          (X, Y, Z [in mm] are defined with respect to PCS frame)
(L4)          (A, B, C [in degree] correspond resp. to the angles Phi, Theta, Psi)
(L5)          (knowing that the rotation matrix R = Rz[C]*Ry[B]*Rx[A])

(L6)          G0 X0 Y0 Z30 A0 B0 C0
(L7)          G4 P2 (2 secondes wait)
(L8)          G1
(L9)          X61.2702 Y-40.0000 Z10.0000 A-0.0000 B-0.0000 C-14.4775
(L10)         G4 P2

(L11)         X67.7225 Y-26.3745 Z10.0000 A-0.0000 B-0.0000 C-36.2022
(L12)         G4 P2

(L13)         X78.7600 Y-16.1051 Z10.0000 A-0.0000 B-0.0000 C-57.9270
(L14)         G4 P2

(L15)         X92.8147 Y-10.6506 Z10.0000 A-0.0000 B-0.0000 C-79.6517
(L16)         G4 P2

(L17)         X107.8901 Y-10.7859 Z10.0000 A0.0000 B-0.0000 C-101.3764
(L18)         G4 P2

(L19)         X121.8447 Y-16.4917 Z10.0000 A0.0000 B-0.0000 C-123.1011
(L20)         G4 P2

(L21)         X132.6962 Y-26.9574 Z10.0000 A0.0000 B-0.0000 C-144.8258
(L22)         G4 P2

(L23)         X138.9030 Y-40.6965 Z10.0000 A0.0000 B-0.0000 C-166.5506
(L24)         G4 P2

(L25)         X139.5835 Y-55.7572 Z10.0000 A0.0000 B-0.0000 C171.7247
(L26)         G4 P2

(L27)         X134.6410 Y-70.0000 Z10.0000 A0.0000 B-0.0000 C150.0000
(L28)         G4 P2

(L29)         M30
```

where:

- Line L1 indicates that:

- the following coordinates are absolute (not incremental) due to instruction G90,
- the units are defined in mm (not inches) due to instruction G21,
- Lines L2-L5 are comments (identifiable by parentheses) relating to movement instructions,
- Lines L6-L28 concern movement instructions. The G0 instruction performs a fast movement (generated in the joint space), the G1 instruction performs a straight-line movement (generated in the operational space). The movements are made by:
 - *positioning* the TCP, relative to the reference frame corresponding to the PCS, through its parameters X, Y, Z (in mm), the corresponding values of the N_b points being contained in the MatLab variables P_x, P_y, P_z,
 - *orienting* the TCP, relative to the robot base frame (R_0), through its parameters A, B, C (in degrees), the corresponding values of the N_b points being contained in the MatLab variables phi, theta, psi.

The instruction G4 P2 pauses the movement for 2 seconds to allow time for the operator to view the *situation* of the N_b points (used to approach the circular arc),

- Line L29 indicates the end of the G-code program.

➤ **Question 14:** Graphically represent in the R_0 frame the point reached by the TCP after executing line L6. What does this point correspond to in relation to the PCS? Check that the G-code file contains the correct number of points to complete the circular arc.


➤ **Realization 15:** Now it's time to write and run the Polyscope program enabling the TCP to perform the approximate movement along the desired arc of a circle.

1) A first step is to load/import the G-code file File_Gcode.nc (generated by your MatLab script) into the robot controller. To do this:

- Copy the file File_Gcode.nc to the root directory of a USB key, then insert the key into the USB socket of the cable coming out of the robot controller,
- In order to register this program in the robot controller:
 - Select the **Open>Program** menu,
 - In the screen that appears, select **usbdisk_0** directory (corresponding to the USB key you connected) and select *All Files* from the *Filter* drop-down menu to view the file File_Gcode.nc,
 - Select this file, then tap **Copy** (📄) button; select **Home** (🏠) icon to access the directory of the robot controller where programs (urp, script, nc, ...) are stored, then tap **Past** button,
 - Tap **Cancel** button to return to the previous screen;

2) Make sure that the URcaps *Remote TCP & Toolpath* (which allows the use of G-code files) is in running status. To do this: select **Installation>URcaps>Remote TCP & Toolpath** menu; in the screen that appears, check that the *Controller Status* is **RUNNING**: otherwise, tap **Start** button to start the controller and wait for the *Controller Status* to be **RUNNING**.

3) The program (to be written in Polyscope) must enable the TCP to approximate the circular arc through rectilinear movements between points $P(1), \dots, P(Nb)$. In fact, it simply consists in using a `script` instruction to execute the script `File_Toolpath.script` (the latter uses the file `File_Gcode.nc` to generate the arm movements). To create this Polyscope program, proceed as follows:







- Copy the file `File_Toolpath.script` to the root directory of a USB key, then insert the key into the USB socket of the cable coming out of the robot controller;
- Select the **Program>Advanced>Script** menu which displays a '*Script*' command line to be configured. To do this, in the **Command** screen (displayed by default): select from the drop-down menu, located at the top right, the '**File**' item (rather than '**Line**' item set by default); tap **Edit** button. In the *File Editor* screen that appears, tap **Open** button to select the file `File_Toolpath.script` situated in the **usbdisk_0** directory (corresponding to the USB key you connected); tap **Open** button (the contents of the file are displayed on the screen). In order to save this program in the robot controller, tap **Save as** button, then select **Home** () icon to access the directory of the robot controller where programs (`urp`, `script`, `nc`, ...) are stored and tap **Save** button. Tap **Exit** button to return to the previous screen (**Command**). The command line must be as follows:

```
Script: File_Toolpath.script
```

the fact that this command line is selected displays the contents of the file;

- Select the **Save>Save All** menu to save your Polyscope program under the name "**TP**", in the **File Name** field (remember to tap **Save** button).

4) Run your Polyscope program (TP.urp) to verify that it works correctly. To do this:

- Select the **Program** menu, then the **Graphics** tab (rather than **Command** tab), which should display the UR3e robot in its initial configuration (all joint values are at 0 degrees) (which assumes that the arm is in *normal mode* ('mode *Normal de fonctionnement*'));
- Let us simulate the movement before its execution on the real arm to avoid damaging the arm, to do this:
 - Activate the simulation by shifting the white circle, located at the bottom right of the screen, from left to right (it follows that the button at the bottom left of the screen changes from '*Normal*' to '*Simulation*'); tap **Play** () button, located at the bottom right of the screen, then select '*Play from beginning – Robot Program*' to simulate the arm movement. Remember that buttons **Pause** () and **Stop** () enable to control the movement progression;
 - Once the movement has been validated by simulation, disable the simulation (by shifting the white circle from right to left);
- Move the robot arm close to point `Waypoint_1_p` with a posture close to that of point `Waypoint_1_q` (as shown in the simulation). Execute the movement on the (real) arm by tapping **Play** () button, located at the bottom right of screen, then by selecting '*Play from beginning – Robot Program*' (**Pause** () and **Stop** () buttons enable to control the movement progression).

➤ **Realization 16:** Consider the move instruction listed on line L9 of the `File_Gcode.nc` file (see Realization 13) which allows the first point used to plot the circular arc (that is, $Pt1$) to be reached.

We wish to have the X, Y, Z, RX, RY, RZ coordinates of this point with respect to R_0 frame. To display these coordinates: tap **Play** (▶) button to execute the movement; tap **Pause** (⏸) (located at the bottom of the screen) when the TCP reaches point $Pt1$ (you have 2 seconds to do this due to instruction `G4 P2`); and select **Move** ('Déplacement') tab. Beforehand, remember to:

- Select from the drop-down menu **Feature** ('Fonction') the **Base** item (rather than **View** item) to display the coordinates in R_0 frame,
- Verify that the **Active TCP** field effectively contains the TCP untitled '**TCP_jlb**' which defines the frame associated with the TCP with respect to the frame associated with the robot flange, that is: $X = 0, Y = 0, Z = 40$ (mm), $RX = 3.1416, RY = 0, RZ = 0$ (rd).

Recover the value of X, Y, Z, RX, RY, RZ by using in particular the values X, Y, Z, A, B, C listed in line L9, that is:

```
X61.2702 Y-40.0000 Z10.0000 (mm) A-0.0000 B-0.0000 C-14.4775 (°) .
```