Lab 1 - Presentation of the Dobot Magician and the DobotStudio Framework

Jean-Louis Boimond
University of Angers

Objectives of the 4 labs using the Dobot robot:

- Lab 1: Presentation of the robot and its framework DobotStudio (elementary operations, movement instructions, environments: Teaching & Playback, Blockly),
- Lab 2: Calculation of the direct geometric model, representation of the reachability space and simulation of the robot arm,
- Lab 3: Calculation of the inverse geometric model, drawings of a straight line and a figure in the Joint space.
- Lab 4: Use of a camera to position the robot successively at the centers of circles arranged on a A4 sheet.

Table of Contents

Lab 1 introduces the Dobot robot by presenting its features and its DobotStudio framework. Two programming environments (Teachning & Playback, Blockly) are presented, enabling the robot arm to perform trajectories and tasks at specific points. Six questions are asked during this lab.

I. Introduction

Dobot Magician is an educational robot arm equipped with various tools and accompanied by a programming environment. The robot arm can be controlled:

- through the DobotStudio framework, which provides different programming environments such as Teachning & Playback, Blockly, Script,
- or by using more standard languages such as Python, MatLab, C#.

Equipped with the appropriate tool, DobotStudio enables you to perform certain tasks, such as writing/drawing, moving a part, 3D printing.

The robot arm features:
- different bodies in particular: a base, a rear arm, a forearm, a support for attaching a tool/end-effector,
- and 5 joints,

see the following figure in which the tool is a suction cup.

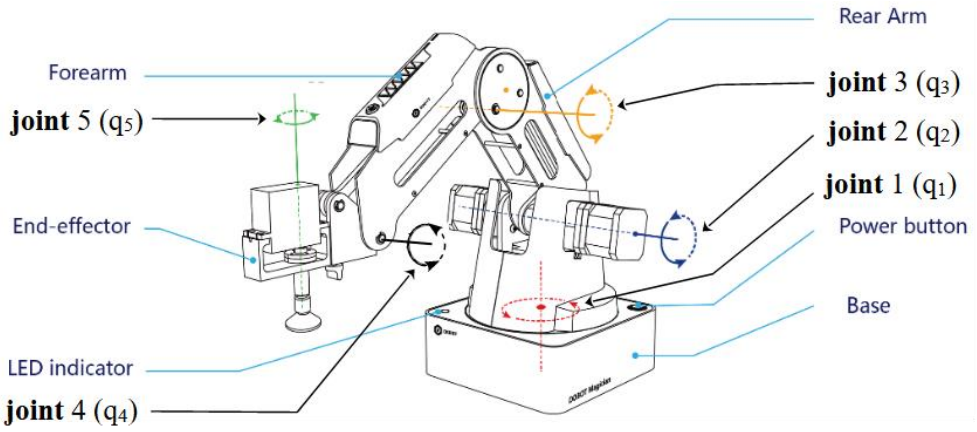

Figure 1: Description of the robot arm equipped with a suction cup.

Locate these different elements on the Dobot robot arm.

When the tool is a suction cup or a gripper, a servo motor is linked with the tool to activate joint 5, for example, to maintain a constant orientation of the tool. Note that this joint is inactive (its angular value is equal to zero) when the tool is a Pen.

Joint 4 is not controllable due to the fact that its value is mechanically fixed according to the angular values applied to joints 2 and 3, so that the axis of joint 5 is always orthogonal to the table/plane on which the robot arm is placed.

Attach the Pen tool (without removing the cap) to the end of the robot arm. Check on the Dobot robot that the Pen tool is parallel to the axis $\overrightarrow{O_0 z_0}$ of the reference (*World*) frame of the robot arm, whatever the posture considered.

Technical data:

Maximum payload	500 g
Maximum reachability	320 mm
Motion rang of joint $q_1 (= J_1)$	$[-90°; +90°]$
Motion rang of joint $q_2 (= J_2)$	$[0°; +85°]$
Motion rang of joint $q_3 (= J_3)$	$[-10°; +85°]$
Motion rang of joint $q_5 (= J_4)$	$[-135°; +135°]$
Maximum speed (with a 250g payload)	Rotational speed of joints 1, 2, 3, 4: 320°/s, Rotational speed of joint 5: 480°/s

Repeated positioning accuracy	0.2 mm

The following figure shows the dimensions of the links of the robot arm and of the Pen tool.

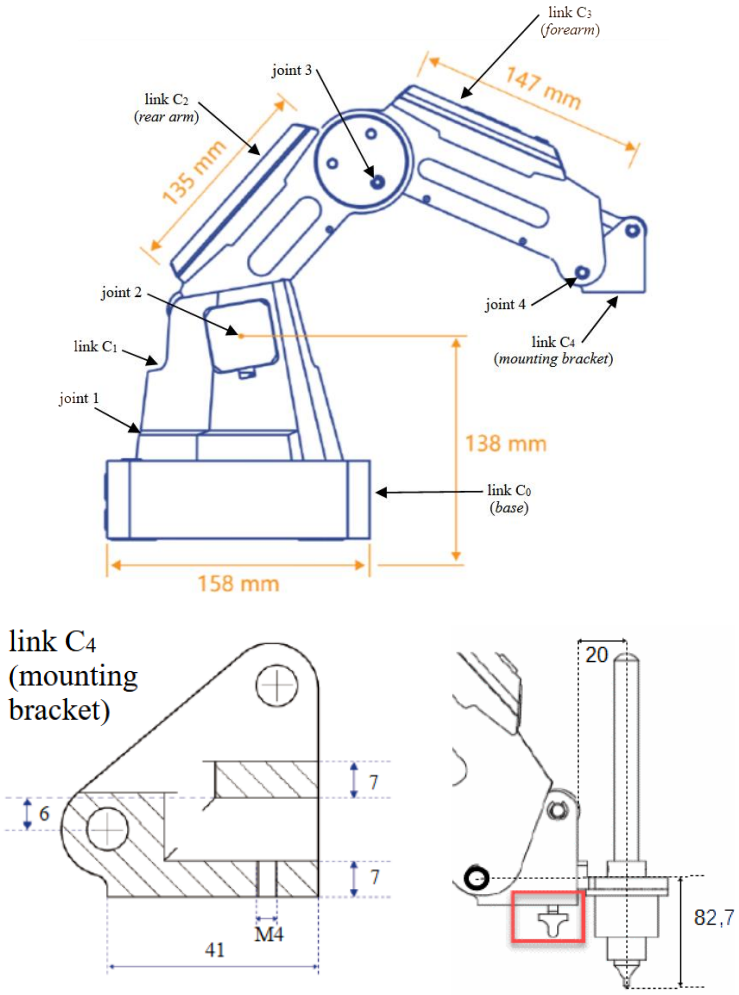

Figure 2: Dimensions of the links of the robot arm, the tool mounting bracket, the Pen tool.

The reference (*World*) frame of the robot arm, x_0, y_0, z_0, and the frame associated with the Tool, x, y, z, are shown in the following figure.

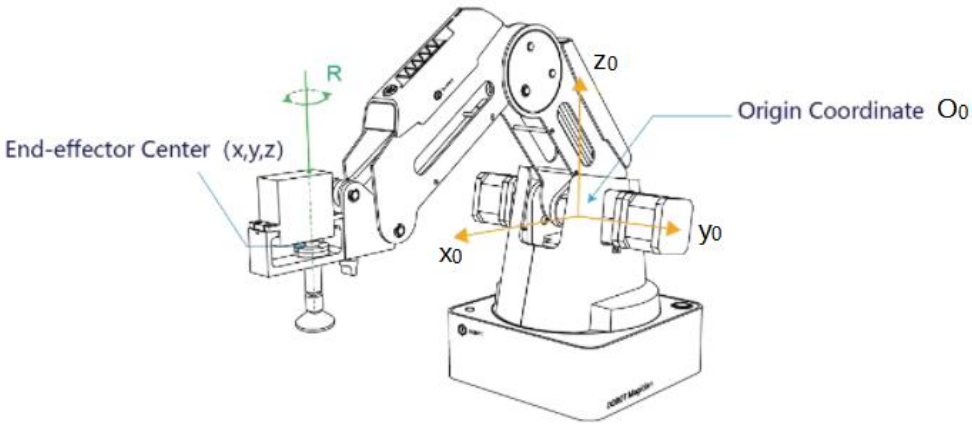

Figure 3: Operational space, *World* and Tool frames.

The following figure represents the robot arm, equipped with the Pen tool, positioned in its initial configuration (*i.e.* when $q_1 = \cdots = q_4 = 0$). The frames associated with the different links of the robot arm are also represented in particular the frames *World* (R_0) and Tool (R_5). Note that point O_5 $(= PF)$ corresponds to the tip of the pen.

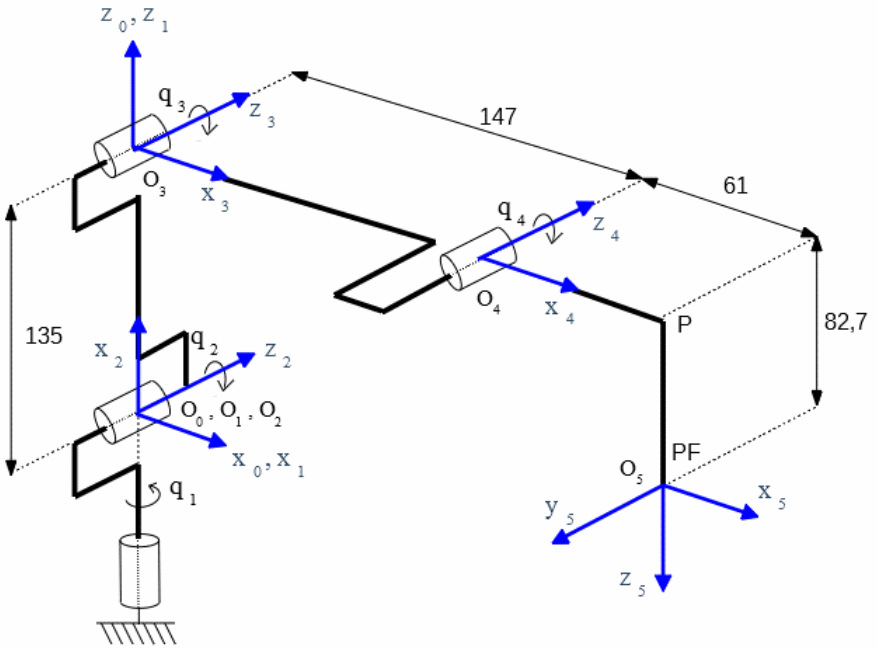

Figure 4: Association of the frames $R_0, \ldots, R_5$ with the links of the robot arm.

Question 1: Locate the different links of the robot arm and the Pen tool in Figure 4. Deduce also from the Figure the coordinates of the points O_3, O_4, P, PF in the frame *World* (R_0) when the robot is in its initial configuration. Locate these points on the robot arm.

With reference to the Denavit-Hartenberg method used later, let us define the angular value $\theta_j(t)$ of the joint j as the value at time t of the rotation angle between the axes $\overrightarrow{O_{J-1}x_{J-1}}$ and $\overrightarrow{O_Jx_J}$ around the axis $\overrightarrow{O_Jz_J}$.

Question 2: Express the values $\theta_1(0), \ldots, \theta_5(0)$ corresponding to the initial configuration of the robot arm (see Figure 4).

We set: $\theta_j(t) = q_j(t) + \theta_j(0), \forall t$, which leads to $q_j(0) = 0, \forall j$ when the configuration is initial.
Note that the angular value $\theta_3(t)$ of joint 3 is actually equal to $q_3(t) - q_2(t) + \theta_3(0)$ which means that it depends on $q_3(t)$ and $\theta_3(0)$ but also (negatively) on the value $q_2(t)$ (applied to joint 2).
The (uncontrolled) value $\theta_4(t)$ of joint 4 is such that the mounting support of the tool is always positioned horizontally (as illustrated in Figures 1, 2 and 3), *i.e.*:

$$\overrightarrow{O_4x_4} = \overrightarrow{O_1x_1}\ ;\ \overrightarrow{O_4z_4} = \overrightarrow{O_1y_1} \text{ (see Figure 4).}$$

Thus, the pen will always be located vertically (along the axis $\overrightarrow{O_5z_5}$) being always orthogonal to the plane $\left(O_0, \overrightarrow{O_0x_0}, \overrightarrow{O_0y_0}\right)$.

Question 3: Show that $\theta_4(t) = -q_3(t)$.

II. Starting/Stopping the robot

- **To start the robot:** position the arm and forearm of the robot so that there is an angle of about 45° between them, then press the **power** button located on the base of the robot. After about 7 seconds, a beep is emitted and the LED indicator, located on the base as shown in Figure 1, changes from yellow to green indicating that the arm is operational.

Note: The LED turns red when the robot arm reaches a limit position. Press (continuously) the unlock button, located on the robot forearm (represented by an open padlock), to move the arm to a desired (and reachable) position; once the button is released, the LED will turn green.

- **Stopping the robot:** press the **power** button on the base of the robot; the forearm then folds (slowly) towards the robot arm.

III. DobotStudio Framework

Installing and launching version 1.9.4 of DobotStudio:

- Copy to your desktop the file DobotStudioSetup_V_1_9_4.zip accessible from the link DobotStudioSetup_V_1_9_4.zip,
- Once the file is unzipped, double-click on the .exe file to install DobotStudio, during this phase, accept the request to install the Device driver,
- To launch DobotStudio, double-click either its icon located on your desktop, or the DobotStudio.exe file, usually located in the C:\DobotStudio directory.

Notation: The values (in degrees) of joints 1, 2, 3 and 5 are denoted in DobotStudio as Joint1, Joint2, Joint3 and Joint4 (rather than Joint5), see the box to the right of the Figure below. They are such as:

$$\text{Joint1}(t) = q_1(t), \text{Joint2}(t) = q_2(t), \text{Joint3}(t) = q_3(t), \text{Joint4}(t) = q_5(t).$$

A. Introduction

DobotStudio enables the robot arm to be controlled through several programming environments such as: Teaching & Playback, Blockly, see the following figure and table.

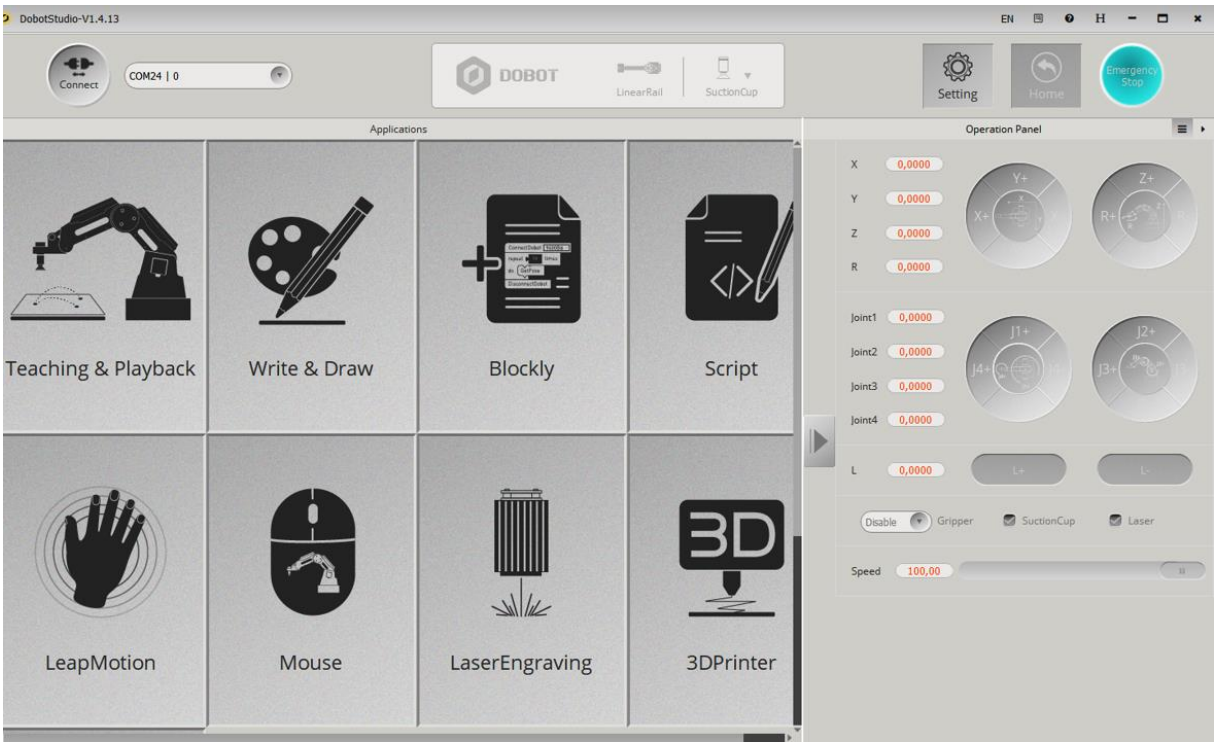

Figure 5: The home page of DobotStudio.

Teaching & Playback	Enables to easily associate the learning/point acquisition phase with movement instructions.
Blockly	Enables through a graphical approach the generation of trajectories and tasks at certain points, by *Drag and Drop* of blocks (the blocks being represented in the form of a puzzle).

B. Connection/Disconnection of the Robot to DobotStudio

Click on the **Connect** button located at the top left of the home page. The robot is connected to DobotStudio when the **Connect** button becomes a **Disconnect** button, see the following figure. In the same way, it is disconnected when the **Disconnect** button becomes a **Connect** button.

Figure 6: Connecting to the DobotStudio application.

Note that it is possible to connect the robot in a quick mode if precision of movement is not required. A dialog box appears on this subject during the connection phase, see the following figure.

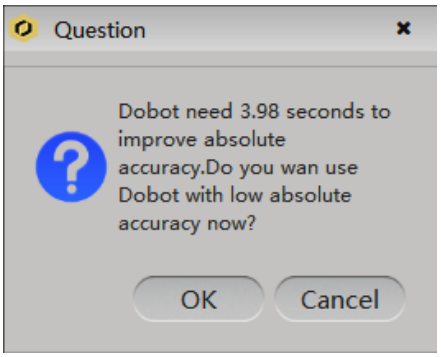

Figure 7: Dialog box for a quick connection (OK) or not (Cancel).

C. Basic Operations

The drop-down menu, positioned in the center of the following figure, is located in the box located at the top right of the home page of DobotStudio. It enables you to select the type of tool (**SuctionCup** in the figure) attached to the end of the robot arm. Possible tools are: **SuctionCup**, **Gripper**, **Laser**, **Pen**, **Advanced**, with **Advanced** enabling another tool to be characterized by coordinates, denoted *xBias*, *yBias*, *zBias.*

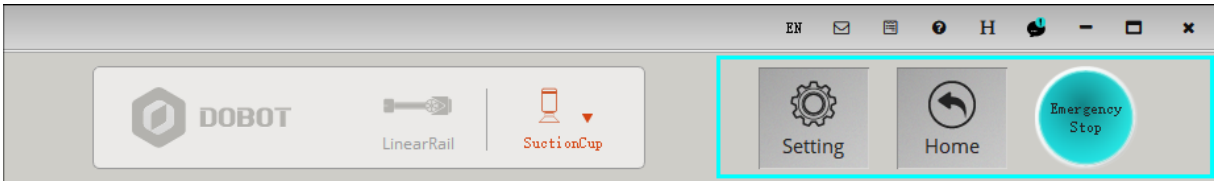

Figure 8: Tool selection, **Setting**, **Home**, **Emergency Stop** buttons.

In this box (see previous figure), three basic operations are also available through **Setting**, **Home**, **Emergency Stop** buttons. The following table briefly describes them.

Setting	To set basic parameters of the robot, see Table 5.2 of the User Guide (located on p. 32-33) for details.
Home	Allows the arm to return to its initial position to obtain a correct reference position. Perform this operation before a movement requiring precision, following a blockage of movement due to an obstacle or a contact of the arm with one of its limit stops.
Emergency Stop	To stop the robot arm in an emergency.

D. Manual Robot Control

The Operation Panel box, located on the right in the DobotStudio home page as shown in the figure below, enables you to:

- move the robot arm manually (not automatically), by considering the Joint space $(\mathrm{Joint1}, \mathrm{Joint2}, \mathrm{Joint3}, \mathrm{Joint4})$ or the Operational space (X, Y, Z, R),
- control a tool such as a Gripper, Suction Cup or Laser.

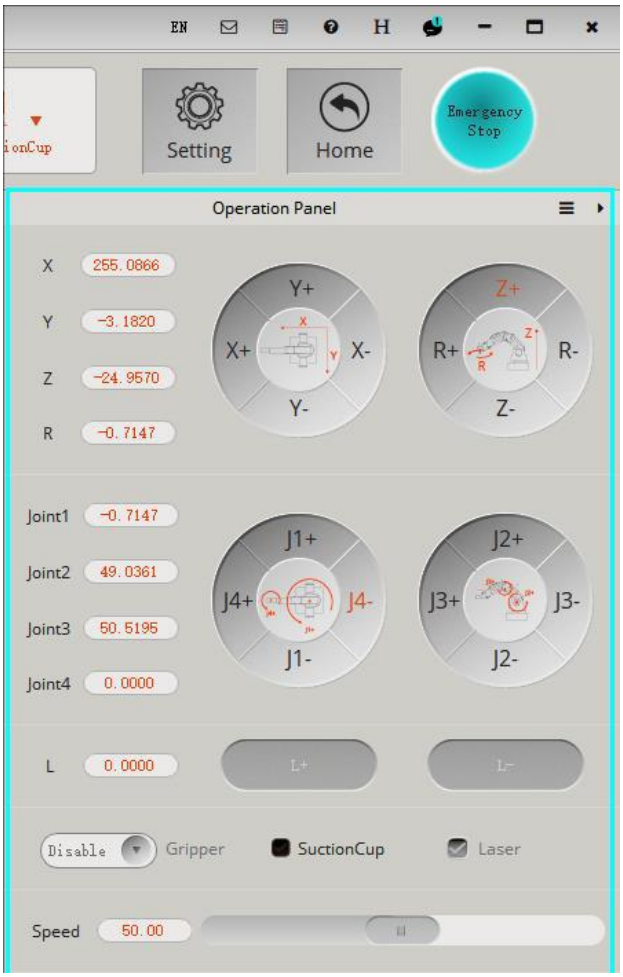

Figure 9: Operation Panel box for manual control of the robot arm and the selection of certain tools.

Note: *Linear rail* control can be disabled by selecting the ≡ symbol located at the top right of the Operation Panel box, and unchecking the *Linear rail* control box.

1. Movement in the Joint Space

The robot arm is set in motion by clicking on ***J1***+/-, ***J2***+/-, ***J3***+/-, ***J4***+/-, see the figure below.

Figure 10: Control of the joints *J1*, *J2*, *J3*, *J4* in the + or - direction.

The movement is axis by axis. The angular values $J1, J2, J3$ refer to joints 1, 2, 3 respectively. The value $J4$ refers to joint 5 (activated by a servo motor attached to the tool when it is a Gripper, Suction Cup or Laser).

For example, joint 1 moves in the positive direction (*i.e.* counterclockwise) by clicking on the *J1+* button; it moves in the negative direction by clicking on the *J1-* button.

Verify that setting joint 2 in motion sets joint 3 in motion, as mentioned in I. Introduction, p. 5.

2. Movement in the Operational Space

The arm is set in motion by clicking on ***X***+/-, ***Y***+/-, ***Z***+/- (*X*, *Y*, *Z* representing the axes of the reference frame (R_0) of the robot arm, *cf.* Figure 3), or on **R**+/- (*R* being the angle between the axes $\overrightarrow{x_0}$ and $\overrightarrow{x_5}$ around the axis $\overrightarrow{z_0}$, that is, $J_1 + J_4$ where J_1 is the angle between the axes $\overrightarrow{x_0}$ and $\overrightarrow{x_1}$ and J_4 is the angle between the axes $\overrightarrow{x_1} = (\overrightarrow{x_4})$ and $\overrightarrow{x_5}$ (around the axis $\overrightarrow{z_0}$)).

The end of the robot arm moves along the *X*, *Y*, *Z* axes, in the positive or negative direction depending on the press of the *X*+/-, *Y*+/-, *Z*+/- buttons.

For example, the end of the robot arm moves along the *X* axis in the positive direction by clicking on the *X1+* button, it moves in the negative direction by clicking on the *X1-* button. Clicking on R+/- button causes the tool to rotate (effective when the servo motor is attached to the Gripper, Suction Cup or Laser tool) around the *R* axis (in the positive or negative direction).

Note that moving along the *Y* axis also causes the *R* axis to move (effective in the presence of the servo motor attached to the Gripper, Suction Cup or Laser tool), to ensure a constant posture of the tool during the movement.

Figure 11: Setting in motion, in the + or - direction, along the *X, Y, Z* axes of the reference frame of the robot arm, or around the *R* axis.

Question 4: With the Pen tool (with its cap) attached to the end of the robot arm, check that the arm, once placed in its initial position/configuration (see Figure 4), is such that:
- the coordinates of the point 0_4, calculated in Question 1, correspond to those (denoted *X, Y, Z*) obtained in the Operation Panel box by selecting beforehand, in the Advanced tool, the appropriate values of *xBias, yBias and zBias*, knowing that these axes relate to the coordinate system (O_4, x_5, y_5, z_5) (**be careful:** these are the axes x_5, y_5, z_5 and not the axes x_4, y_4, z_4!!);
- the coordinates of the point *P*, calculated in Question 1, correspond to those obtained in the Operation Panel box by selecting beforehand the appropriate values of *xBias, yBias and zBias*. Check that these coordinates also match those obtained by selecting the Pen tool;
- the coordinates of the *point PF*, calculated in Question 1, correspond to those obtained in the Operation Panel box by selecting beforehand the appropriate values of *xBias, yBias and zBias*.

E. About Movement : Point To Point, Continuous Path, ARC

Let's take a look at the types of movements you can use:

- Point To Point (PTP) movement: MOVJ, MOVL, and JUMP.

1) MOVJ instruction (joint movement): The calculation of the tool trajectory is done in the Joint space. Each joint evolves simultaneously from its initial/current angular position to that of destination. In the example shown in the figure below, the execution of the MOVJ B instruction leads to the movement of the tool from its initial/current point (denoted A in the figure) to point B.

2) MOVL instruction (rectilinear movement): The calculation of the tool trajectory is done in the operational space to perform linear trajectory. In the example shown in the figure

below, each joint moves in order that the tool moves in a straight line from the initial/current point A to the destination point B (associated with the instruction).

Figure 12: Instructions MOVJ B, MOVL B.

3) JUMP instruction: The movement is similar to the MOVJ instruction with the addition of a move up above the initial/current point (A) and a move down to the destination point B (associated with the instruction). Each joint evolves simultaneously to perform the following sequence of movements, as shown in the figure below:
   - Move up the tool in a straight line at a height h_A above the initial/current point A ($\bar{A}$ being the resulting point),
   - Horizontal movement of the tool (according to a behavior analogous to MOVJ) from the point $\bar{A}$ to a point $\bar{B}$ (located at a height h_B above the point B),
   - Move down the tool in a straight line towards the destination point B.

   About the values of h_A, h_B:
   - if $Z_A = Z_B$ then $h_A = h_B = JumpHeight$ where $JumpHeigh$ in *mm* is defined in the Setting/Playback/JumpParam tab (see Figure 14),
   - if $Z_A > Z_B$ then $h_A = JumpHeight$ (with $h_B > JumpHeight$),
   - if $Z_A < Z_B$ then $h_B = JumpHeight$ (with $h_A > JumpHeight$).

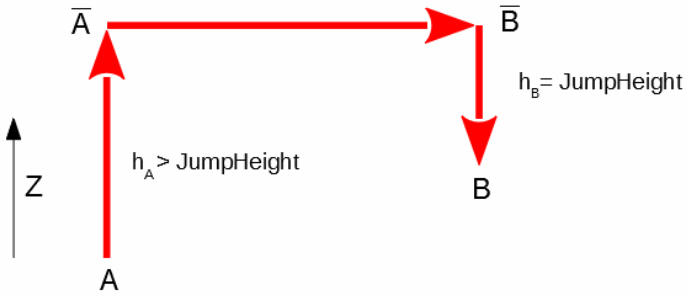

Figure 13: JUMP B when $Z_A < Z_B$.

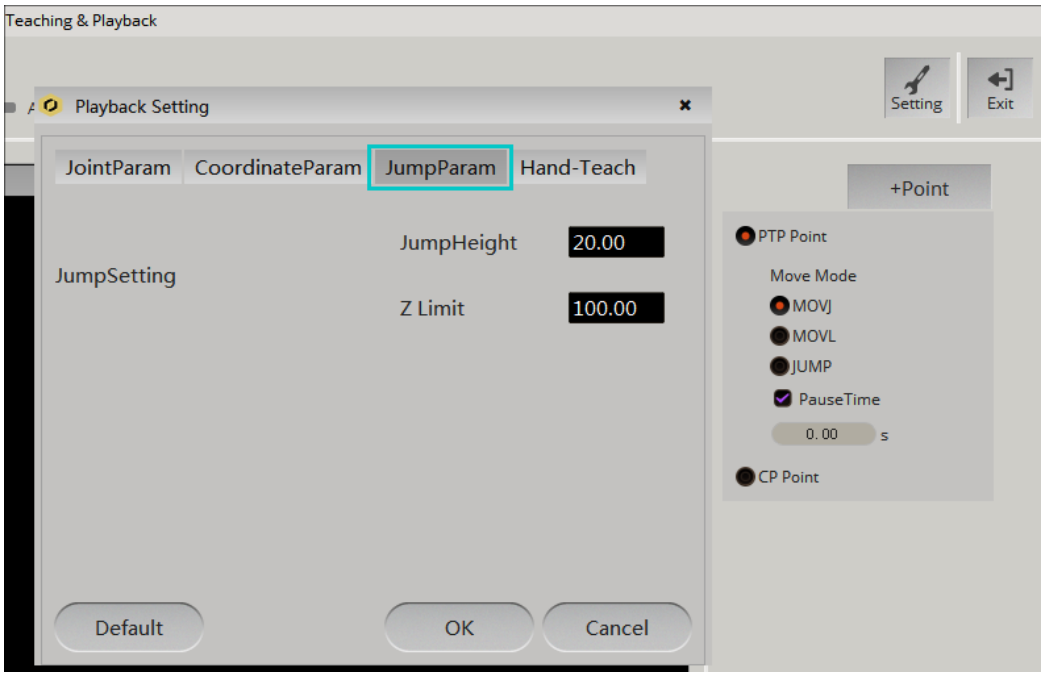

Figure 14: Assignment of the JumpHeight parameter.

- Continuous Path (CP) movement: Such tool movement is performed from the initial/current point to a point (associated with the instruction) without stopping there (unlike the PTP mode), to go, *via* another movement instruction, to another point (without stopping there or destination), which allows a certain fluidity of the movement when passing by this point.

- ARC movement from the initial/current point, denoted A, in the form of an arc defined by 2 other points B, C (an intermediate point (denoted *cirPoint*), an end point (denoted *toPoint*)) not aligned, as described in the figure below, for example, to avoid an obstacle.

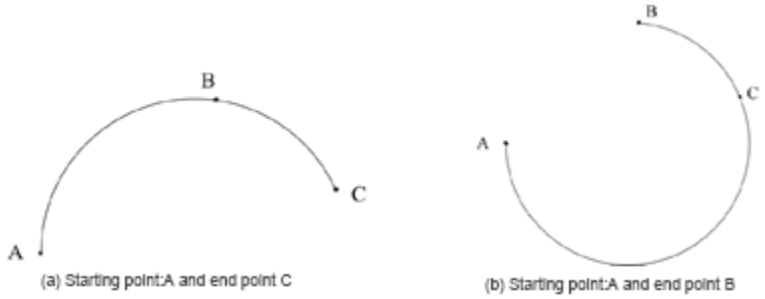

Figure 15: Arc trajectory.

NB: Movements in Continuous Path mode are not accessible in the Teaching & Playback and Blockly environments. Movements in ARC mode are not accessible in the Blockly environment.

F. Teachning & Playback and Blockly Programming Environments

There are several ways to implement a robot arm control program in DobotStudio, especially the Teaching & Playback and Blockly environments described below.

1. Teaching & Playback

After connecting the robot to DobotStudio, the development environment, described in the figure below, appears by clicking on Teaching & Playback on the DobotStudio home page.

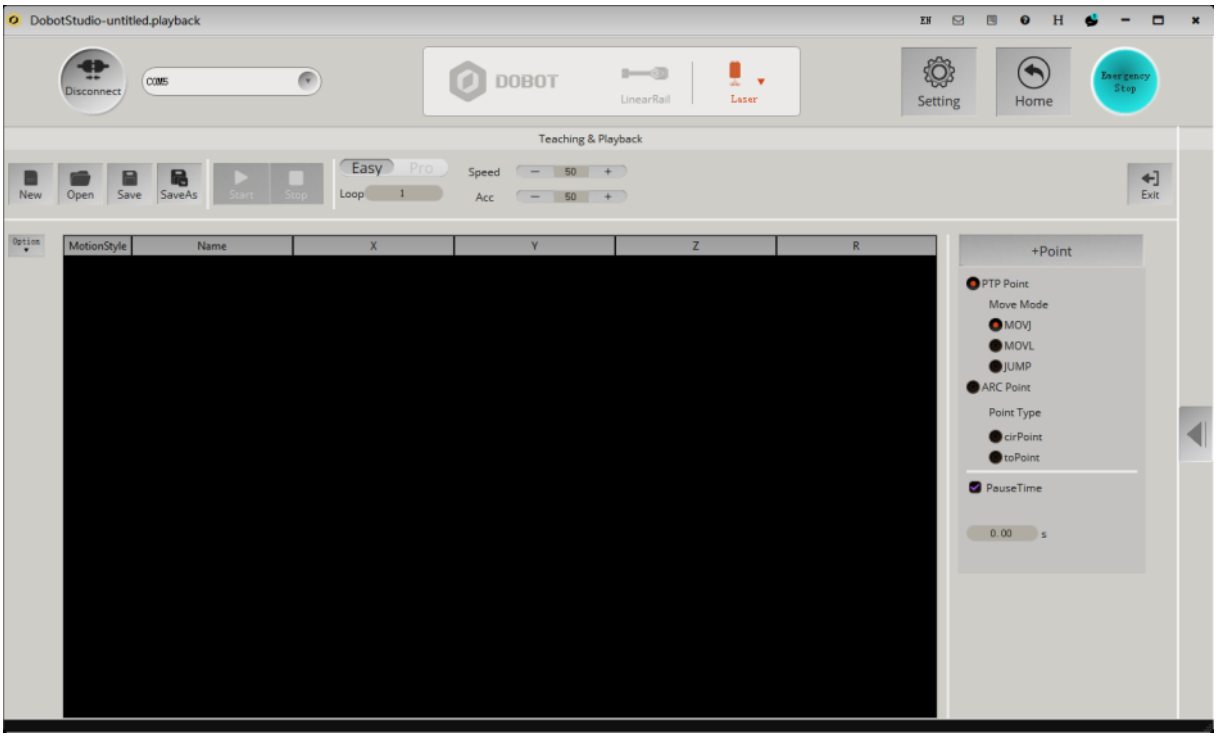

Figure 16: Teaching & Playback page.

- It is possible to: switch between the **Easy** (default) and **Pro** modes (the Pro mode is not of interest in the context of the practical work); assign the number of loops, speed and acceleration (in percentage). See the following figure and descriptive table.

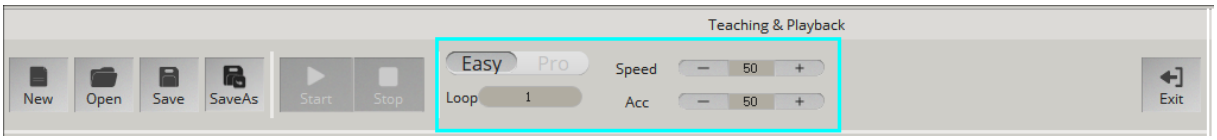

Figure 17: Assignment of Easy/Pro, Loop, Speed et Acceleration.

Items	Description
Easy/Pro	Compared to Easy mode, Pro mode adds features such as offline mode, I/O interface.
Loop	Sets the number of times the robot arm replays the recorded steps (default: 1, value range: 1-999999).
Speed	Sets the speed ratio when playing (default: 50%, value range: 0-100%).
Acceleration (Acc)	Sets the acceleration ratio during playback (default: 50%, value range: 0-100%).
Exit	Back to the DobotStudio home page.

- The command lines are programmed in the development window (see Figure 16):
  - by defining the type of movement knowing that possible movement instructions are MOVJ, MOVL, JUMP, ARC,
  - by saving the points to be associated with the instructions,
  - by setting the pause time associated to certain points,

see the following descriptive table.

Items	Description
Movement mode	Select the movement mode: - Point To point (PTP) with access to MOVJ, MOVL or JUMP instructions, - ARC given an initial point (reached for example *via* a MOVJ statement), an intermediate point *cirPoint* and a destination point *toPoint*.
+Point	Create a new point in the list of saved points.
Pause time	Set the pause time (in seconds) associated with a point.

More precisely, a command line is defined by the following fields:

- *MotionStyle* to select the MOVJ, MOVL, JUMP or ARC movement mode,
- *Name* to associate (if necessary) a name with the command line,
- *X*, *Y*, *Z*, *R* to define the coordinates of the point associated with the instruction (*X*, *Y*, *Z* are the Cartesian coordinates in the reference frame R_0, *R* is the angle of rotation of the servo motor of the joint *J*5). Note the necessity to add the coordinates *X'*, *Y'*, *Z'*, *R'* to define the second point in the case of an ARC instruction. Moving the arm to a desired position is done by pressing the *Unlock* button located on the robot's forearm and represented by an open padlock. Pressing the **+Point** button will record the point in connection with the current movement instruction (note: it is possible to record a point as soon as you release the Unlock button if you first check the *Enable Handhold Teaching* function (by choosing *release the UNLOCK* button from the drop-down menu) located in the *Setting>PlayBack>HandHold Teaching* page).
- *PauseTime* to associate a pause time (in seconds) with the point associated with the movement instruction.

A field on a command line can be modified (copied, pasted, cut, etc.) by highlighting it (by double-clicking), see the figure and the descriptive table that follow.

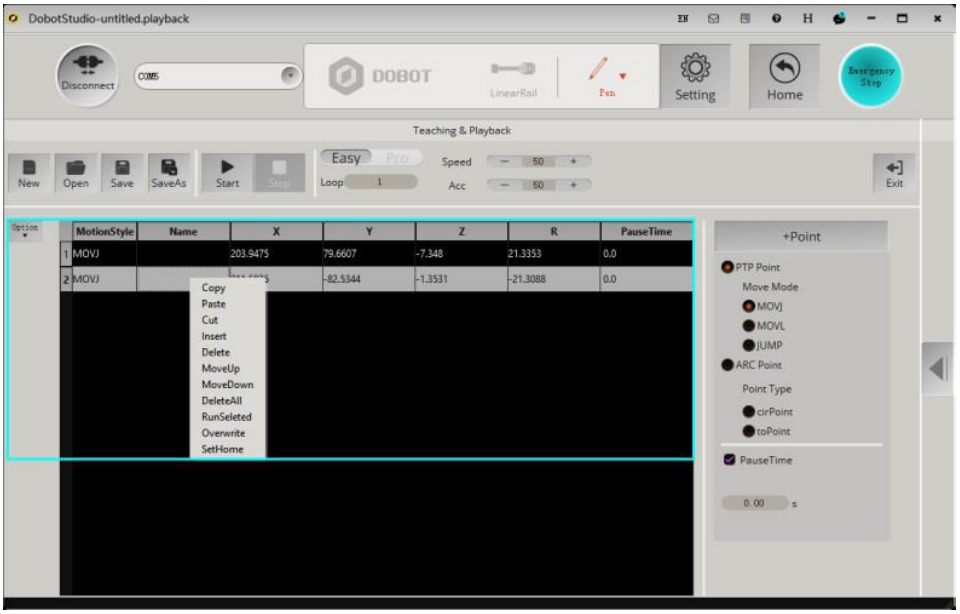

Figure 18: List of instructions.

Items	Description
Right click	In the Popup menu, you can edit a highlighted saved point with copy, paste, cut, etc.

Double-click	Double-click a field to change its value.

Question 5: Create a program in the Teaching & Playback environment that draws a continuous path within the permitted area from point A to point B, then from point C to point D, using the Pen tool on the A4 sheet provided by your supervisor.

2. Blockly

Based on an Google open source project, Blockly is a graphical programming environment that lets you create command lines in a development window by importing blocks (represented by puzzle pieces) from a drag and drop library.

The development environment appears by clicking on Blockly on the home page of Dobot Studio. Consider the example described in the following figure.

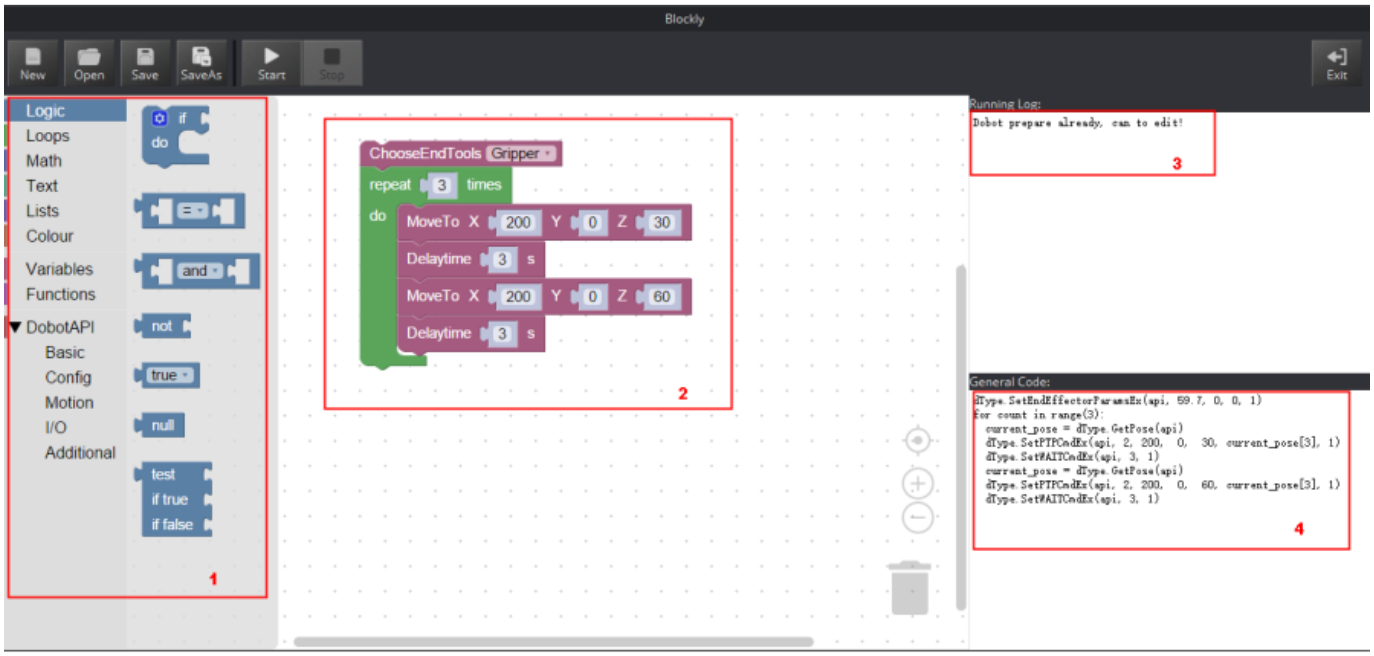

Figure 19: Blockly development environment.

The **New**, **Open**, **Save**, **SaveAs** buttons, located at the top left, allow you to create, open, save a file, knowing that the default path, which can be modified *via* SaveAs, is: C:.. \DobotStudio\config\bystore.

The **Start**, **Stop** buttons, located after the New, Open, Save, SaveAs buttons, allow the start and stop of the current program.

The following table describes the different windows in the development environment.

No.	Window description
1	Window containing the Blockly blocks used to program the robot. To facilitate their use, these blocks are stored in different modules (Logic, Loops, Math, Lists, ... and DobotAPI). Note that DobotAPI is specific to DobotStudio and allows you to: set the speed/acceleration; set up the tool; set the robot arm in movement in the Joint or Operational space; configure I/O interfaces.

2	Development window where the blocks from window 1 are imported, by drag and drop, to form the Blockly program able to realize the trajectory of the robot arm and the tasks to be performed during the trajectory.
3	'Running Log' window where the execution log of the current Blockly program is described.
4	'General Code' window containing the automatically generated lines of API (Application Programming Interface) codes, equivalent to each of the blocks of the current Blockly program.

As an example, the Blockly program described in the figure below allows the gripper attached to the robot arm to perform 3 times a go and back movement along the Z axis with a one-second pause at the 2 points defined by the 'Go to X .. Y .. Z ..' instructions.

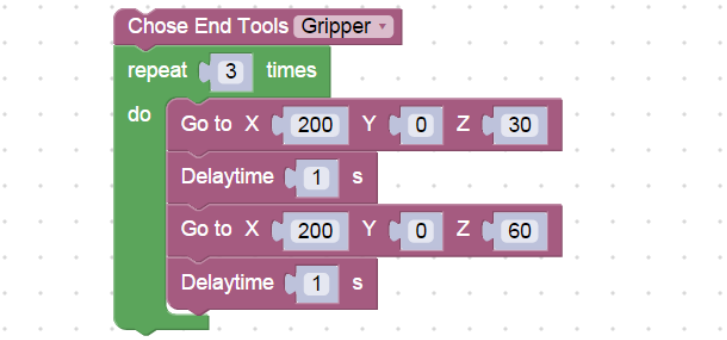

Figure 20: Example of a Blockly program.

Description of the previous program instructions:

+ Chose End Tools Gripper indicates that the considered tool is the gripper; thus the geometric model of the robot will be such that the Tool Center Point is at the end of the gripper,
+ repeat 3 times do repeats the following 4 instructions (in the loop) 3 times:
    + Go to X 200 Y 0 Z 30 sets the robot arm in motion so that the Tool Center Point reaches the coordinate point $(200, 0, 30)$ (defined in the reference frame R_0),
    + Delaytime 1 s causes a pause of 1 second at the current point,
    + Go to X 200 Y 0 Z 60 sets the robot arm in movement so that the Tool Center Point reaches the coordinate point $(200, 0, 60)$,
    + Delaytime 1 s causes a pause of 1 second at the current point.

- **Description of some blocks specific to the robot (contained in the DobotAPI module)**

+ basic blocks of the *Basic* sub-module:

Home	Positions the robot arm in its default posture to obtain a correct position reference (by default $q_2 = 90°, q_3 = 0°$ knowing that these values can be changed *via* "Setting>Initial Pos").
GetTime	
Delaytime 0 s	Causes a pause (here 0) in second between 2 commands.

+ configuration blocks of the *Config* submodule:

ChooseEndTools SuctionCup (Gripper, Laser or Pen)	Indicates the type of tool (SuctionCup, Gripper, Laser, or Pen) is considered to be attached to the end of the arm.
Set End Effector Params XBias 0 YBias 0 ZBias 0	Indicates the (non-standard) tool attached to the end of the arm, through the XBias, YBias, ZBias values relative to the frame (O_4, X_5, Y_5, Z_5) shown in Figure 4.
SetMotionRatio VelocityRatio 20 AccelerationRatio 50	Sets the ratio of velocity (here 20% of 500mm/s) and acceleration (here 50% of 500mm/s^2).
SetJointSpeed Velocity 20 Acceleration 50	Sets the ratio of velocity (here 20% of 500mm/s) and acceleration (here 50% of 500mm/s^2) of joints 1, 2, 3 and 5.
SetCoordinateSpeed Velocity 20 Acceleration 50	Sets the ratio of velocity (here 20% of 500mm/s) and acceleration (here 50% of 500mm/s^2) relative to the Cartesian coordinates *X*, *Y*, *Z* and rotation *R* (joint *J4)*.
SetJumpHeight Height 20	Set the height (here 20mm) used with the JUMP instruction.

+ Motion blocks of the *Motion* submodule:

JumpTo X 200 Y 0 Z 0	Sets the robot arm in JUMP mode to reach the coordinate point (X, Y, Z) equal to (200, 0, 0) in mm.
MoveTo X 200 Y 0 Z 0	Sets the robot arm in motion in MOVL mode to reach the coordinate point (X, Y, Z) equal to (200, 0, 0) in mm.
MoveDistance ΔΞ 0 ΔY 0 ΔZ 0	Sets the robot arm in motion in MOVL mode to reach a coordinate point defined by the increments $\Delta X, \Delta Y, \Delta Z$ relative to the current coordinate point.
SetR 0	Sets the angular value *R*, in this case to 0, without changing the other angles.
SetJointAngle Joint1 0 Joint2 45 Joint3 45	Sets the robot arm in motion in MOVJ mode corresponding to the values *J1*, *J2*, *J3*, here equal in degree to 0, 45, 45 respectively (the value of *J4* remaining unchanged).
GetCurrentCoordinate x (y, z ou r)	Gets the value of the Cartesian coordinate x, y, z or *r* (here *x*).
GetJointAngle Joint1 (Joint2, Joint3, or Joint4)	Gets the value of the angular coordinate $J1, J2, J3$ or $J4$ (here $J1$).
SuctionCup ON (OFF)	Activates the air pump (ON) or OFF.

Gripper Gripper (Release ou OFF)	Closes the gripper (Gripper), releases it (Release), or makes it inactive (OFF).

The blocks contained in the *I/O* submodule allow interaction with the I/O interface. The blocks contained in the *Additional* submodule can specify some additional parameters relating to the Laser and the photoelectric sensor.

Note: The 'set (item) to' block, from the *Variables* module, allows the use of variables in a program. For example, the 2 command lines described in the following figure allow to:
- assign the Z axis coordinate of the current position (see the *Unlock* button, located on the robot forearm, to position the arm) to the 'z' variable,
- move the Tool Center Point to the position defined by the Cartesian coordinates $(200, 0, z)$.

Figure 21: Use of variable 'z' in Blockly.

Question 6: Attach the gripper to the robot arm, see the following figure for wiring. From the cubes "1" and "2" initially positioned respectively on slots *A* and *B* (see the diagram described on the A4 sheet provided by your supervisor), create a Blockly program that can:
- position cube "1" on the slot *C*,
- stack cube "2" on cube "1",
- wait 2 seconds,
- position the cubes "1" and "2" on the locations *D* and *E* respectively.

Figure 22: Connecting the gripper and mini-compressor to the robot arm.

- **Block interpretation in the form of APIs**

Each block has its command lines, based on APIs. For example, the Blockly block:

```
SetJoint Angle Joint1 0 Joint2 45 Joint3 45
```

corresponds to the following 2 lines of code:

```
current_pose = dType.GetPose(api)
dType.SetPTPCmdEx(api, 4, 0, 45, 45, current_pose[7], 1)
```

To allow the execution of the code corresponding to the Blockly blocks, lines of code are placed above et below without appearing in window 4 (shown in the Figure 19) such as:

```
import DobotDllType as dType

CON_STR = {
dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

#Load Dll
api = dType.load()

#Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])
```

for respectively:
- import the *DobotDllType* library by renaming it *dType*,
- load the APIs by obtaining the object, named *api*, of type load,
- connect the robot to the DobotStudio application (by writing the connection status).

The current_pose = dType.GetPose(api) instruction calls the GetPose API (described on p.13 of the Dobot Magician API description), defined by:

```
int GetPose(Pose *pose),
```

to obtain the position of the robot arm (in real time), in the sense that current_pose is a table containing the position and joint values, namely, x, y, z, r, J1, J2, J3, J4.

The dType.SetPTPCmdEx(api, 4, 0, 45, 45, current_pose[7], 1) instruction uses the SetPTPCmdEx API, described on p.36 of the Dobot Magician API description, defined by:

```
int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)
```

and enables the robot arm to move to reach the point, knowing that:
- $PTPMode = 4$ is such that the movement takes place in the joint space with the joint values $J1 = 0, J2 = 45, J3 = 45, J4 = current\_pose[7]$ where $current\_pose[7]$ corresponds to the values *J4* of the current position,
- $isQueued = 1$ to add the command to the "execution queue" (each command placed in the "execution queue" being executed in the order in which it arrived in the queue).