ROBÓTICA INDUSTRIAL

PRÁCTICA 2: PROGRAMACIÓN DE APLICACIONES DE PICK & PLACE CON UN ROBOT INDUSTRIAL

Students:

Enrico Maria Marinelli
Arvin Das

Professor:

Ángel Valera Fernández

ACADEMIC YEAR 2023/2024

Contents

List of Figures

1 Introduction

In this report we will explain our approach on how to make a pick and place program for the IRB140 robot provided by ABB.

We were given a simulation environment replicating the real LAB environment, it is composed of:

- a set of cans;
- a set of bricks;
- a conveyor belt;
- two placing stations, one for bricks and one for cans;
- an IRB140 Robot;
- three tables.

Furthermore the bricks could arrive in angles of 0°, 45°, 90° (Ampliation 2) and there are packaging rules to be followed the stacks: the can stacks are in made of two layers of 4 cans each, whereas the brick stacks are made of two layers of 2 bricks each, with the bricks at the top layer oriented at 90° with respect to the Z-axis in world reference frame (Ampliacion 1).

2 Environment Setup

For the simulation of the pick and place robot to run correctly, we had to setup the environment in RobotStudio.

2.1 Inputs and outputs

We started by defining the output and input signals, these signals can be found in Table 1.

In the real laboratory the robot controller uses a computer vision algorithm applied on a camera pointed at the conveyor belt to detect what the incoming object is and what orientation it has, but since we could not use such technology in the simulation.

This problem has been dealt through the FlexPendant interaction: the 4 output buttons has been programmed and cross connect them with the input signals related to the output signal[1]. This had to be done because, with our simulation model, we cannot directly impose these input to the controller.

[1]This explains the necessity of the 4 output signals similar to the 4 inputs, defined in the Table 1

outputs	inputs
$CONVEYOR\_FWD$	$CONVEYOR\_OBJ\_SEN$
$GRIPPER\_CLOSE$	-
sal_Lata	$objeto\_Lata$
sal_Brick	$objeto\_Brick$
$sal\_Brick45$	$objeto\_Brick45$
$sal\_Brick90$	$objeto\_Brick90$

Table 1: Table of outputs and inputs

2.2 RobotStudio Station

We also had to create the right environment in RobotStudio, to successfully simulate the functioning of the robot.

To do this we had to load in the working station, which consisted of a worktable, a conveyor belt, two placing stations (one for bricks and one for cans), the IRB140 robot and the podium, on which we placed the robot, hence modifying its original position. After that we put a sucker tool on the end of the robot. The tool was necessary to pick up the brick and cans. When we have done all of this the final station should look like Figure 1.

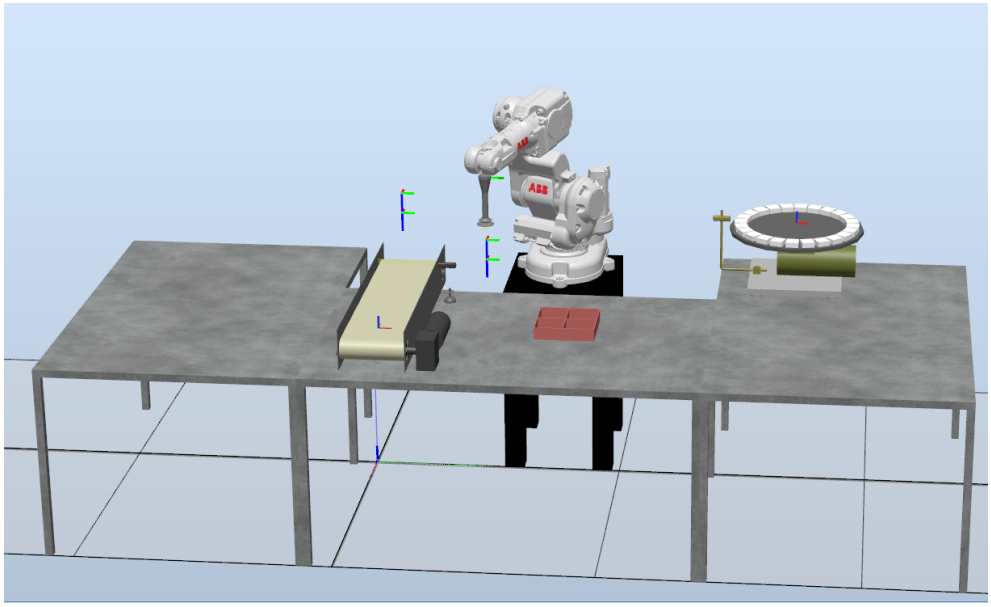

Figure 1: The final station.

Lastly we had to connect all the logistics of the simulation. We did this by going to the simulation window, then logic of station, then in design we had to connect the arrows according to the functionality. In the end the connections the one shown in Figure 2.

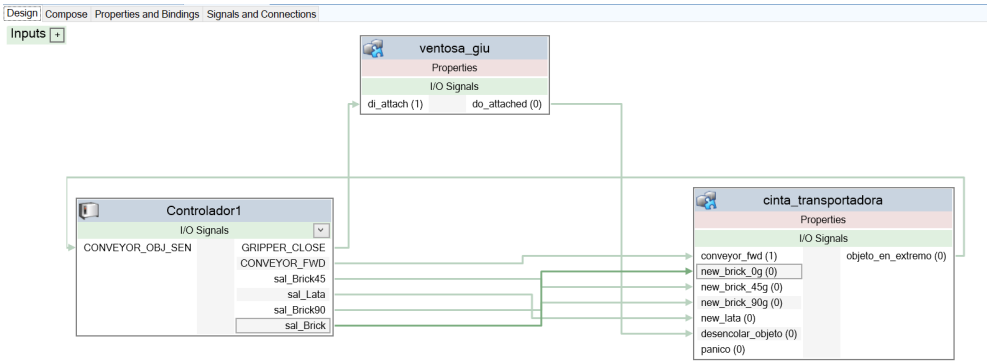

Figure 2: Logistics of the simulation.

In addition to the assigned setup, we modified the component *cinta_transportadora* by slowing down its movement from $-1000\frac{mm}{s}$ to $-100\frac{mm}{s}$(Figure 3), **this choice is only applied on the simulation environment**. The reason behind it is to have a more consistent behaviour of the sensor data of the objects, that heavily affects the picking position of the objects and, subsequently, the placing precision of the objects in the simulation.

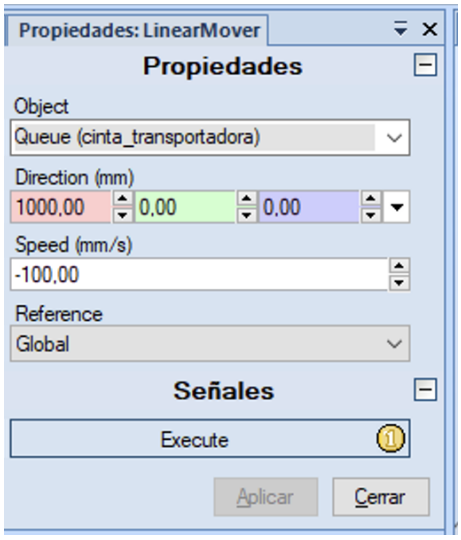

Figure 3: Modification of the conveyor belt speed (in simulation only).

3 RAPID Program Solution

The two modules used in the simulation are **MainModule** and **Module1**, where **Module1** contains only the constant robottarget variables (corresponding to the pose and other arguments) for each location needed in the task accomplishment.
It has been added a new error-type variable *NoSpaceError* used to further handle scenarios where a task is completed but an additional item corresponding to the task items is given.
The MainModule will be explained in the following sections.

3.1 Main program

In the main module of the program we have declared variables and conditional expressions to handle the different behaviors of the robot, for each of the tasks. In this way the IRB140 Controller is able to recognize the tasks completeness status, represented by the number of stacked bricks and/or can. It has been assumed to have one and only one object at the same time on the conveyor belt, for simplicity.
The robot should behave correctly even if the object arrival is not in the right order for one of the tasks., e.g. first item is a can, the second is a brick, then a can again... and every possible combination of the two is feasible, until the according task has been completed.

In the main code, first of all we declare multiple variables set to 0 each time the simulation starts, that indicate the status of the tasks, by counting the number of cans or bricks in their place positions 4.

```
VAR num orientation := 0;
VAR num bricks:=0;
VAR num latas:=0;
VAR num latas_layers:=0;

BookErrNo NoSpaceError;


set GRIPPER_CLOSE;
MoveJ Pos_origen, v300, fine, tool0\WObj:=wobj0;

Reset sal_Brick;
Reset sal_Lata;
```

Figure 4: Initialization of the simulation.

Then a infinite loop is necessary to perform continously the actions of the robot. The conveyor belt is set to always go forward, it is stopped only when an object is detected by the proximity sensor, therefore the two signals $CONVEYOR\_OBJ\_SEN$ and $CONVEYOR\_FWD$ are supposed to never be at the same level, but they switch at the same time.

```
WHILE TRUE DO
    Set CONVEYOR_FWD;
```

Figure 5: Infinite loop.

When an object is detected by the computer vision system (fictitiously substituted in the simulation environment) an input signal is sent, it includes the object type, and orientation if it is a brick.
A conditional statement on the type of signal received is handling the execution of the code, they will be explained in detail in Sections 3.2 and 3.3.

At the end of the while loop it has been set a command $WaitTime$ with 1 second to slightly reduce the frequency of iterations, affecting only the conditional checks of signals, not the operations.

3.2 Can Movement

The signal $objeto\_lata = 1$[2] determines that a can has been detected and the robot should be in charge of dealing with the object. First it is needed to wait for the arrival of the object at the pick position, located where the proximity sensor is.

Once the proximity sensor detects the object $CONVEYOR\_OBJ\_SEN = 1$, the conveyor is stopped $CONVEYOR\_FWD = 0$. Then the program in charge for the movement is applied, giving as argument the status of the station, given by the parameters $latas \in [0, 3]$, $latas\_layers \in [0, 1]$ chosen to make more synthetic the code of PROC $MoveLata$.

```
IF sal_Lata = 1 THEN
    TPWrite "lata detected";
    IF latas_layers = 2 THEN
        TPWrite "There is no space for latas left!";
        RAISE NoSpaceError;
        RETURN;
    ENDIF


    WaitDI CONVEYOR_OBJ_SEN, 1;
    Reset CONVEYOR_FWD;

    MoveLata latas,latas_layers;
    Reset sal_Lata;
    IF latas = 3 THEN
        latas_layers := latas_layers+1;
        latas:=0;
    ELSE
        latas := latas+1;
    ENDIF
ENDIF
```

Figure 6: Conditional operation for the can.

[2]In the code in Figure 6 $sal\_lata = 1$ is equivalent to $objeto\_lata = 1$ thanks to the cross-connection.

Note in Figure 6 the handling of the event which regarding to have more lata than the 8 considered for the task to be accomplished, it immediately stops the operations by giving a warning message to the operator through the TeachPendant and raising the previously defined *NoSpaceError*, and then returning the main[3].

The robot movements are handled by the external PROC *MoveLata* in Figure 7. The picking position is defined by the CONST robtarget defined in the file inside Module1. Instead of using the same position, a low offset has been added in order to pick the can properly[4].

After reaching the can, the gripper is closed and the robot moves towards the origin position.

```
PROC MoveLata(num latas,num layer)
    VAR robtarget preplace := Pos_preplace_lata;
    VAR robtarget place := Pos_place_lata;


    MoveL offs(Pos_prepick_Lata,10,0,0), v300, fine, tool0\WObj:=wobj0;
    MoveL offs(Pos_pick_Lata,10,0,0), v20, fine, tool0\WObj:=wobj0;
    Reset GRIPPER_CLOSE;
    MoveL Pos_prepick_Lata, v20, fine, tool0\WObj:=wobj0;
    MoveJ Pos_origen, v100, z100, tool0\WObj:=wobj0;
```

Figure 7: Part 1 of PROC *MoveLata*.

Once the piece is handled by the robot, it is calculated the placing position based on the status of the task, determined by a conditional operation on the variable *latas*, following the order in Figure 9.

To accomplish the task of having two stacks of 4 cans, with a total of 8 cans, it has been introduced the binary variable *layer*, that is set to 1 only if the first layer of cans it has been completed. It is used to dynamically change the Z-axis coordinate of the preplace and place position by multiplying it with the height of the can, as shown in Figure 8.
In fact giving rise to a stack of them by maintaining, without any further changes, the execution code discussed previously.

```
preplace.trans.z := preplace.trans.z + 80*layer;
place.trans.z := place.trans.z + 80*layer;
```

Figure 8: Part 2 of PROC *MoveLata*.

[3]Here has been supposed that the operation of moving the latas from the robot workstation to another workstation has not have been done yet, hence the system need to be stopped until it is performed.

[4]This is due to the choice of reducing the speed of the conveyor, that causes the piece to be detected "more quickly" by the sensor, hence the belt movement is stopped when the piece is a bit far off the pick position target.

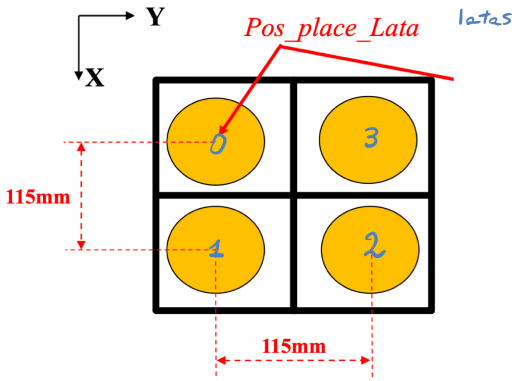

Figure 9: Order of can placing.

```
TEST latas

CASE 1:
    preplace.trans.x := preplace.trans.x + 115;
    place.trans.x := place.trans.x + 115;

CASE 2:
    preplace.trans.x := preplace.trans.x + 115;
    place.trans.x := place.trans.x + 115;

    preplace.trans.y := preplace.trans.y + 115;
    place.trans.y := place.trans.y + 115;

CASE 3:
     preplace.trans.y := preplace.trans.y + 115;
     place.trans.y := place.trans.y + 115;

DEFAULT:
ENDTEST

MoveL preplace, v100, fine, tool0\WObj:=wobj0;
MoveL place, v20, fine, tool0\WObj:=wobj0;

Set GRIPPER_CLOSE;
MoveL preplace, v20, fine, tool0\WObj:=wobj0;
MoveL Pos_origen, v300, fine, tool0\WObj:=wobj0;

ENDPROC
```

Figure 10: Part 3 of PROC *MoveLata*.

3.3 Brick Movement

The signals containing *objeto_brick* has been handled similarly to the signal *objeto_lata*, discussed in Section 3.2 with respect to Figure 6.

The substantial difference consists in the parameter *layer* substituted with *orientation*, that will be used in the PROC *MoveBrick*.

There have been defined three conditional statement, one for each signal received and passing the argument *orientation* according to the signal received, as shown in Figures 11.

```
IF sal_Brick = 1 THEN
    TPWrite "brick detected";
    IF bricks = 4 THEN
        TPWrite "There is no space for bricks left!";
        RAISE NoSpaceError;
        RETURN;
    ENDIF

    orientation := 0;

    ReSet CONVEYOR_FWD;
    Set CONVEYOR_FWD;
    WaitDI CONVEYOR_OBJ_SEN, 1;
    Reset CONVEYOR_FWD;

    MoveBrick bricks, orientation;
    Reset sal_Brick;
    bricks := bricks+1;
ENDIF
```

(a) Brick with orientation 0^{o}.

```
IF sal_Brick45 = 1 THEN
    TPWrite "brick detected";
    IF bricks = 4 THEN
        TPWrite "There is no space for bricks left!";
        RAISE NoSpaceError;
        RETURN;
    ENDIF

    orientation := 45;

    WaitDI CONVEYOR_OBJ_SEN, 1;
    Reset CONVEYOR_FWD;

    MoveBrick bricks, orientation;
    Reset sal_Brick45;
    bricks := bricks+1;
ENDIF
```

(b) Brick with orientation 45^{o}.

```
IF sal_Brick90 = 1 THEN
    TPWrite "brick detected";
    IF bricks = 4 THEN
        TPWrite "There is no space for bricks left!";
        RAISE NoSpaceError;
        RETURN;
    ENDIF

    orientation :=90;

    WaitDI CONVEYOR_OBJ_SEN, 1;
    Reset CONVEYOR_FWD;

    MoveBrick bricks, orientation;
    Reset sal_Brick90;
    bricks := bricks+1;
ENDIF
```

(c) Brick with orientation 90^{o}.

Figure 11: Conditional statement for brick signals.

Then is important to notice that have been required many more variables inside the PROC *MoveBrick* (Figure 12) to handle every scenario, in terms of brick orientation in the picking phase and brick pose[5] in the placing phase.

The orientation of the brick has been chosen to be calculated on the fly by the function *quatProd*, and not to be calculated by hand using the same formulas shown in the function

[5] i.e. position and orientation.

in Figure 13. This can clearly affect the computation time, but is an approach more prone to be flexible with respect to the other one, since defining statically the rotation of the *tool*0 it is sensible to changes of the static reference frame, whereas in the code is calculated starting from *orient oDef*, corresponding to the orientation of the origin *robtarget*.

```
PROC MoveBrick(num bricks, num angle)

    VAR robtarget prepick:=Pos_prepick_Brick;
    VAR robtarget pick:=Pos_pick_Brick;
    VAR robtarget preplace:=Pos_preplace_Brick;
    VAR robtarget place:=Pos_place_Brick;

    VAR orient oDef:=Pos_origen.rot;
    !VAR orient o45:=[0.92388,0,0,0.382];
    VAR orient on45:=[0.92388,0,0,-0.382];
    !VAR orient o90:=[0.7071,0,0,0.7071];
    VAR orient on90:=[0.7071,0,0,-0.7071];

    !o45:=quatProd(oDef,o45);
    on45:=quatProd(oDef,on45);
    !o90:=quatProd(oDef,o90);
    on90:=quatProd(oDef,on90);
```

Figure 12: Part 1 of PROC *MoveBrick*.

```
FUNC orient quatProd(orient o1, orient o2)


    RETURN [
        o1.q1 * o2.q1 - o1.q2 * o2.q2 - o1.q3 * o2.q3 - o1.q4 * o2.q4,
        o1.q1 * o2.q2 + o1.q2 * o2.q1 + o1.q3 * o2.q4 - o1.q4 * o2.q3,
        o1.q1 * o2.q3 + o1.q3 * o2.q1 + o1.q4 * o2.q2 - o1.q2 * o2.q4,
        o1.q1 * o2.q4 + o1.q4 * o2.q1 + o1.q2 * o2.q3 - o1.q3 * o2.q2];
ENDFUNC
```

Figure 13: Function *quatProd*.

The second part of the *MoveBrick* code consists in handling the picking phase with any orientation, and it is performed by using conditional statements on the parameter *orientation* given in input in the main code, as shown in Figure 14.

```
TEST angle
CASE 45:
    prepick.rot:=on45;
    pick.rot:=on45;


    prepick.trans.x := prepick.trans.x + 15;
    pick.trans.x := pick.trans.x + 15;

    prepick.trans.y := prepick.trans.y - 20;
    pick.trans.y := pick.trans.y - 20;

CASE 90:
    prepick.rot:=on90;
    pick.rot:=on90;

    prepick.trans.x := prepick.trans.x - 30;
    pick.trans.x := pick.trans.x - 30;

    prepick.trans.y := prepick.trans.y - 32;
    pick.trans.y := pick.trans.y - 32;

DEFAULT:
    prepick.rot:=oDef;
    pick.rot:=oDef;
ENDTEST

MoveL prepick, v300, fine, tool0\WObj:=wobj0;
MoveL pick, v20, fine, tool0\WObj:=wobj0;

Reset GRIPPER_CLOSE;
MoveL prepick, v20, fine, tool0\WObj:=wobj0;


MoveJ Pos_origen, v100, z100, tool0\WObj:=wobj0;
```

Figure 14: Part 2 of PROC *MoveBrick*.

Note that have been added some offset values to the x and y coordinate depending on the orientation, similarly to the discussion for Figure 7 to achieve a better consistency in gripping the bricks in the same position.

To study the offsets of the coordinates also in for the placing position we have resorted to calculate analytically, starting from the geometry of the objects. In Figure 15, is reported a part of the work done to reach the results. In the picture the first two bricks are already in their place position and the geometry is required to obtain the right offset to place the two bricks on the top in a properly stable stack, ready to be packed.

Also is taken in consideration the orientation of the two, the code is shown in Figure 16, with the usual conditional statement approach.

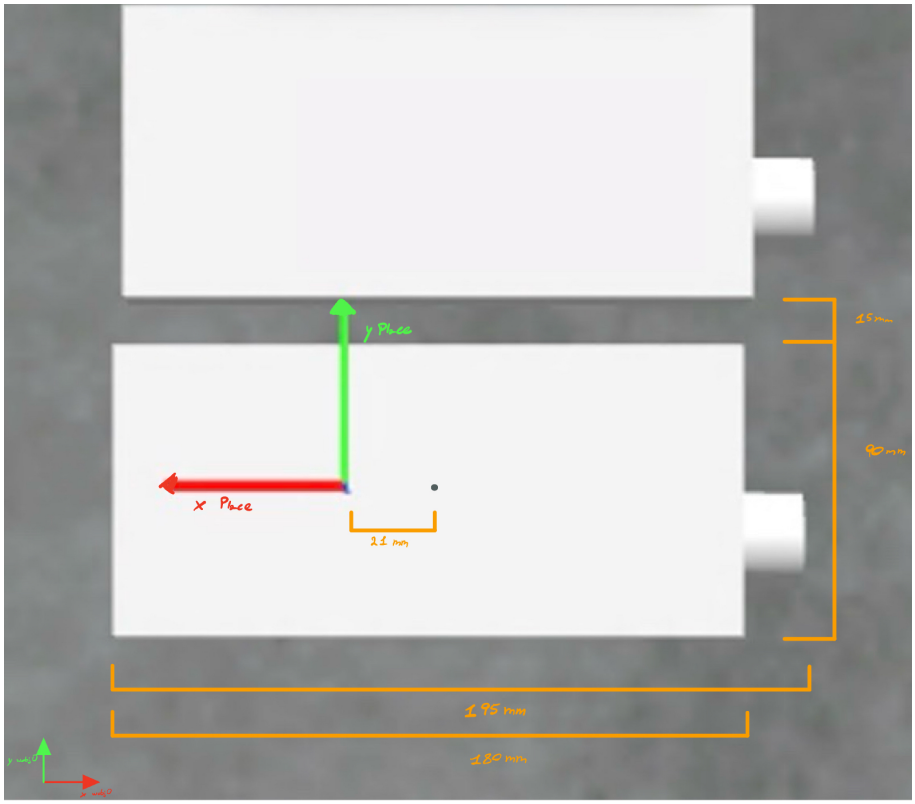

Figure 15: Handwritten geometry of the brick.

```
    TEST bricks

    CASE 1:

        preplace.rot:=oDef;
        place.rot:=oDef;

        preplace.trans.y := preplace.trans.y + 105;
        place.trans.y := place.trans.y + 105;

    CASE 2:

        preplace.rot:=on90;
        place.rot:=on90;

        preplace.trans.z := preplace.trans.z + 60;
        place.trans.z := place.trans.z + 60;

        preplace.trans.x := preplace.trans.x + 21 - 180/2 + 90/2;
        place.trans.x := place.trans.x + 21 - 180/2 + 90/2;

        preplace.trans.y := preplace.trans.y + 90/2 + 15/2 - 21;
        place.trans.y := place.trans.y + 90/2 + 15/2 - 21;

    CASE 3:

        preplace.rot:=on90;
        place.rot:=on90;

        preplace.trans.z := preplace.trans.z + 60;
        place.trans.z := place.trans.z + 60;

        preplace.trans.x := preplace.trans.x + 21 - 180/2 + 90/2 + 15 + 90;
        place.trans.x := place.trans.x + 21 - 180/2 + 90/2 + 15 + 90;

        preplace.trans.y := preplace.trans.y + 90/2 + 15/2 - 21;
        place.trans.y := place.trans.y + 90/2 + 15/2 - 21;

    DEFAULT:
        preplace.rot:=oDef;
        place.rot:=oDef;
    ENDTEST


    MoveL preplace, v100, fine, tool0\WObj:=wobj0;
    MoveL place, v20, fine, tool0\WObj:=wobj0;
    Set GRIPPER_CLOSE;
    MoveL preplace, v20, fine, tool0\WObj:=wobj0;
    MoveL Pos_origen, v300, fine, tool0\WObj:=wobj0;
ENDPROC
```

Figure 16: Part 3 of PROC *MoveBrick*.

4 Conclusion

In conclusion, throughout this report we have presented how we solved the tasks of picking up different objects, with different orientations, and placed them in their designated spots.

To manage to do this we had to simulate the system in Robot Studios. In there we had to set up an environment and declare signals to match the functionality of the real laboratory. We also decided to reduce the speed of the conveyor belt during the simulation[6] to achieve a better repeatability of the results of Figure 17.

Lastly we had to upload our RAPID code to the real robot. To do this, we had to update our main module with new pre-specified robot targets. The reason for this is that the target points in the simulation was not the same ones as for the real robot and, as mentioned before, also the conveyor belt speed is unchanged.
However all the code logic and the orientations that was defined in Sections 3.1, 3.2 and 3.3 are not required to be modified whatsoever to run the real system as the simulated one.

In the end we were able to run successfully[7] our code in the real robot inside the laboratory.

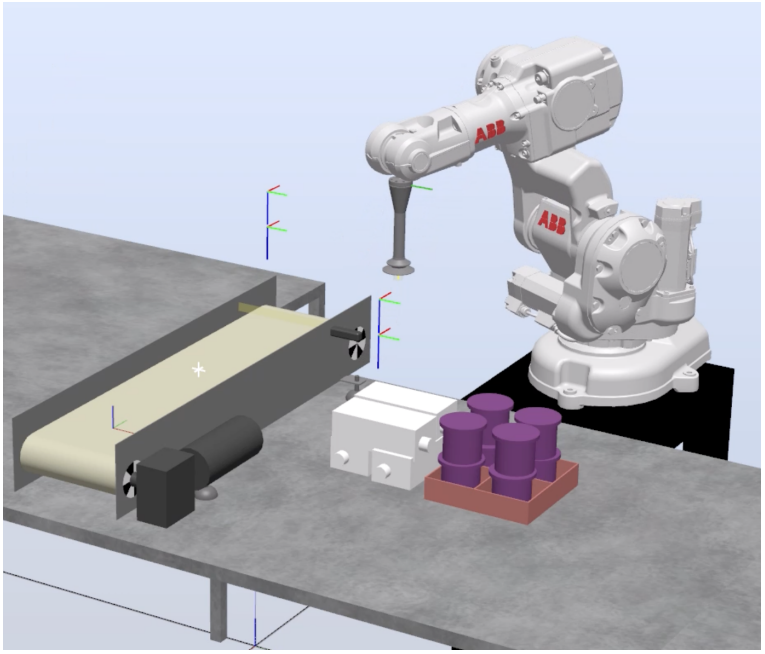

Figure 17: Visualization of the station after tasks completion.

[6] Reported in Figure 3.
[7] Without ampliations specified in Section 1.