Universitat de Girona
Escola Politècnica Superior

Treball Final de Grau/Carrera

Realitzat i defensat a UCLeuven-Limburg (nom universitat) de Bèlgica (país)

Estudi: Grau en Eng. Electròn. Ind. i Automàtica Pla 2009

Títol: Robotic Arms

Document: Memòria

Alumne: Juan Salom Bonnin

Director/Tutor: Jaume Puig
Departament: Eng. Química, Agrària i Tecn. Agroalimentària

Convocatòria (mes/any): gener/2018

Index

1. INTRODUCTION

1.1. Background

The University of Leuven-Limburg in their Electro-mechanics courses use KUKA robots to teach the students how a robotic/mechanical arm works, its functionalities, capabilities and its usage in production lines in a real industrial environment. Nowadays they own a KUKA KR-C3, a robot with six degrees of freedom.

They have recently acquired a new FANUC LR Mate 200/iD 4s robot with six degrees of freedom as well as its offline and online software.

Also, they downloaded an ABB software Robotstudio to watch how it performs and if it would be a good choice to buy an actual robot.

1.2. Purpose

The project describes a unique exercise that intention is to teach the students about the different brands of robotic arms on how to utilize these robots for future works in industries.

Precisely, it is a palletizing exercise that grabs sticks and places them into layers in which every even layer the rotation changes resulting in a tower of perpendicular sticks, which size is within the software and hardware limitations of the robot.

The objective is to show the students the main differences in coding/programming and teaching the distinct brands of robots so they will have a wider acknowledgment about it and could be a big difference in their workplaces.

1.3. Scope

This project has required an autonomous learning on two of the three robots and has taken 3 months to finish it. Because the course cannot take that long, the exercise will be simplified for the student, as a meaning, the code and layout of the robot is almost done, the student will have to complete it.

Two courses for each robot should be the appropriate period of learning, so it will be between 3-6 hours a lesson. Taking into account that one of the robots it is only simulated that sums a total of 5 exercises, definitely, that is a total of 15-30 hours of learning, depending on how the teachers want to manage the time.

2. LIST OF FIGURES

3. COMPANY

The company I did the project is the University of Leuven-Limburg, at the Technology faculty, located in Campus Diepenbeek, Hasselt, Belgium.

The ambition of UCLL is to stimulate regional development. This in no way excludes an international scope, to the contrary. The future of the region and the alumni is closely linked to their international competences. UC Leuven-Limburg brings international cases and practices to its regional businesses. By developing international competences amongst their local graduates and by attracting exchange students, the university college supplies the region with much-needed international alumni. For all of this, we turn to the international partners for inspiration, good practices and collaboration.

In my staying, I was in the Robotics Laboratory, working with the real robots they have and the software which lets you practice with the simulated robot that you choose. My purpose was to teach the professors and they consequently, would the students.

4. EXERCISE DESCRIPTION

The exercise is a palletizing exercise involving sticks that have to be picked up with a vacuum tool (as it is the same than in the laboratory) from a vertical structure (Fig. 1) and then placed in a certain order on top of a table (Fig 2).

The student has to think on how it is easier for the robot to do this, spending the less effort on coding/programming the robot and also thinking on the shorter ways for the robot, taking care of singularities and not exceeding axis limits neither. Also, the type of movement has to be in mind (linear, joint or arc) as it can create problems when running the program.

The idea of the exercise is to make a program whose code is simple, clean and clear, by that means, the student has to do a thinking before starting as organization is key to get it nicer.

It is important that the program is not full of taught points as the idea here is to teach the less points possible and after it make some calculations to get a full autonomous program, this way if in the future it is necessary to change something this would be just changing numbers and not have to teach new points.

4.1. Picking

The first part of the program is for the robot to pick the sticks. In this exercise the sticks are laying in a vertical structure on top of each one like in the next figure.

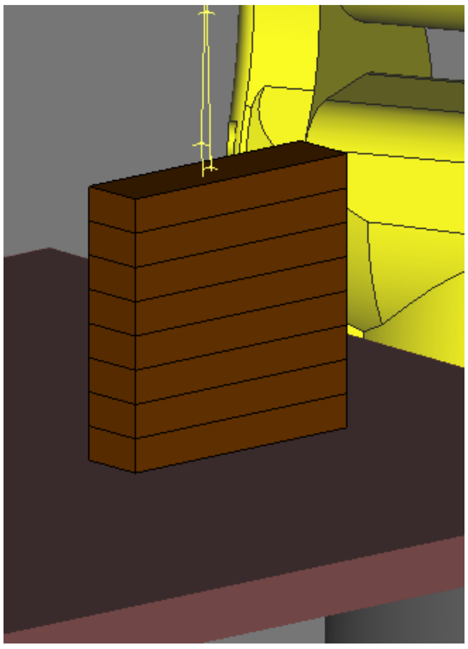


Figure 1. Picking structure

The number of sticks can be variable as in the code it has to be set up as a variable. Then the robot will keep placing them depending on how many sticks there are as it depends on layers of sticks, where each layer contains 4 sticks. If the number of sticks is divided by 4 and the remainder is not 0, then the robot has to leave the sticks that do not form a layer of 4.

The robot has to learn a position over the structure of sticks, its height has to be much higher than the first stick on top (in case there are a lot of sticks), then it starts to go down with the vacuum ON, when it grabs a stick, the vacuum sensor goes 'HIGH' and the down movement has to stop. Then the piece is taken to the placing spot.

4.2. Placing

After picking the stick, the robot takes it to another location; this location is set up in the code and can be wherever but inside the robot axis limits. The target structure has to be like seen in the figure 2.

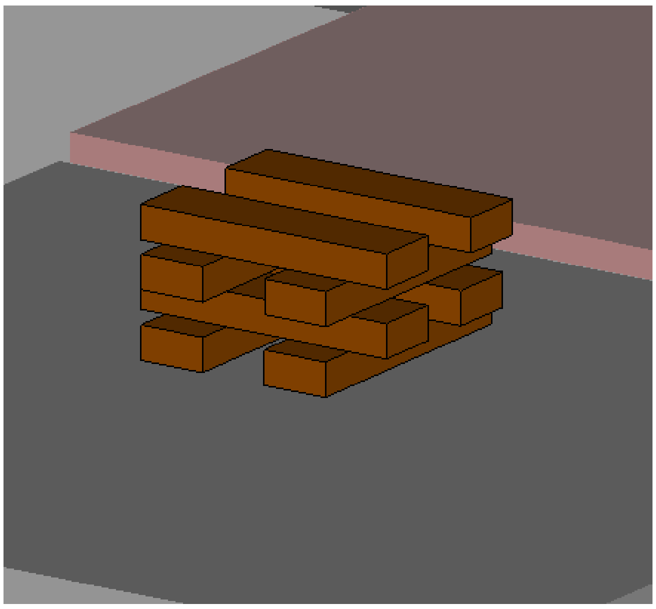


Figure 2. Placing structure

As it can be seen there are layers of 2 sticks in the same orientation, to not make it that difficult for calculations, the layers consist on 2 sticks in a perpendicular orientation and other 2 sticks perpendicular to them, making a total of 4 per layer.

For placing, there is only one point needed to be taught to the robot, as the other points are calculated from that point in the code. The first stick goes to that taught point, then the second just moves in a axis. For the third, it has to move in the three axis and one rotation axis minimum, normally around ‘Z’.

5. CODES

In this chapter the codes used for each exercise will be explained separately for each brand because were some I/O issues between hardware and software.

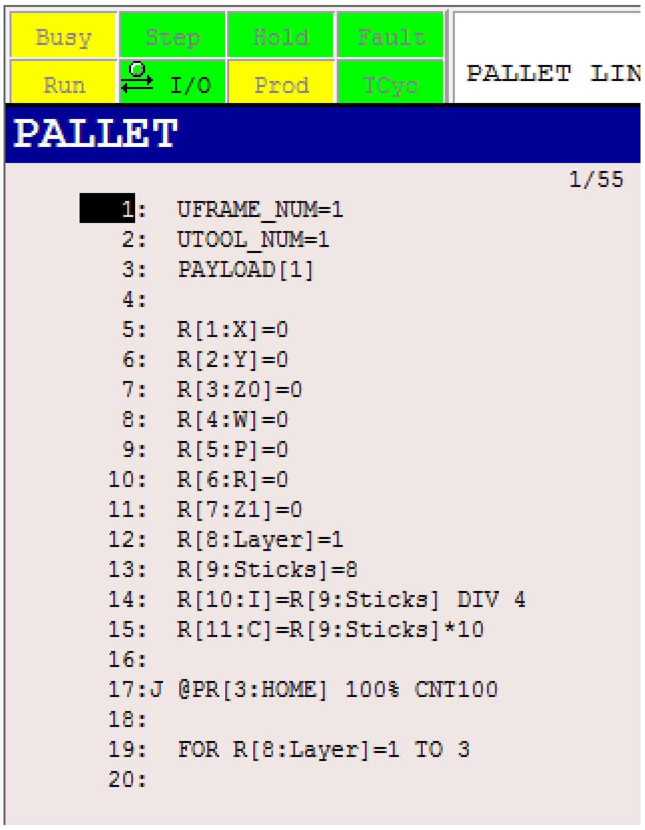
To summarize, there are some common variables in each code, however there could be more or less depending of the language used in the programming screen.

H or C	Variable used to set the height of the stack of sticks
I	Auxiliary variable used to set the maximum layers
Sticks	For setting the total number of sticks
Layer	Counter for the corresponding layer at that moment

5.1. Software

5.1.1. FANUC Roboguide

The FANUC software called Roboguide uses a visual simulation program where the objects are linked to fixtures. It has a lot of configurations for the Robot controller and environment, however, the idea for the exercise is to use the Teach pendant as the operator would use it in the actual robot. The TP has the environment for coding, nevertheless it is not free programming and it is not as intuitive as other coding languages. Its basis is on the language ‘KAREL’.



```
   UFRAME_NUM=1
   UTOOL_NUM=1
   PAYLOAD[1]

   R[1:X]=0
   R[2:Y]=0
   R[3:Z0]=0
   R[4:W]=0
   R[5:P]=0
   R[6:R]=0
   R[7:Z1]=0
   R[8:Layer]=1
   R[9:Sticks]=8
   R[10:I]=R[9:Sticks] DIV 4
   R[11:C]=R[9:Sticks]*10

J @PR[3:HOME] 100% CNT100

   FOR R[8:Layer]=1 TO 3

```

Figure 3. FANUC sim Code 01

The code starts with the setting of the Frame and Tool that is going to be used in this program. This has to be created in the simulation program.

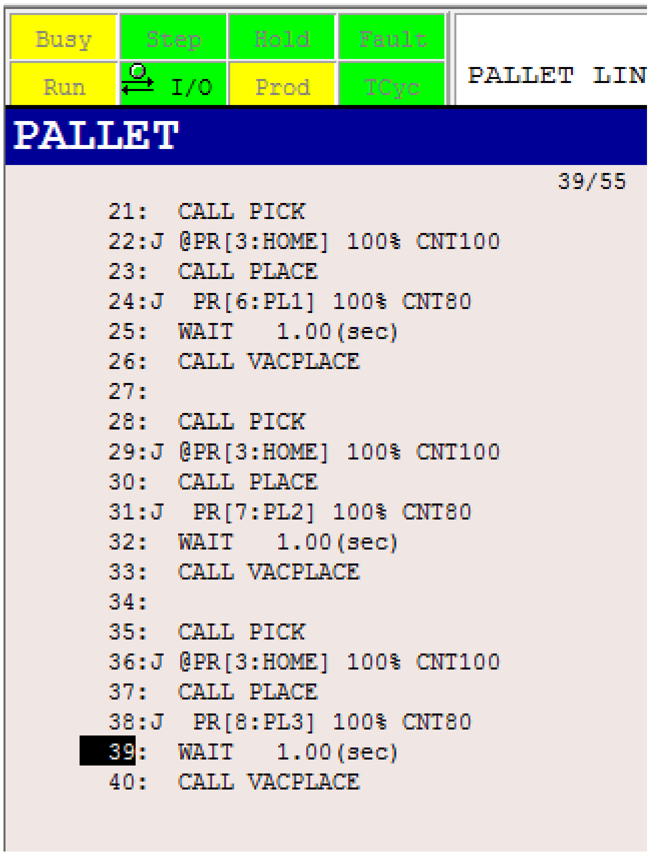
After that, the Payload is set up; the payload indicates how much load the robot is holding. When setting this, the inertia of the robot with and without load has to be calculated as well as the load inertia in every axis. This determines how the robot will accelerate between points.

Then the registers (constants and variables) are started with the value wanted. Registers 1 to 7 are axis registers and registers ‘10:I’ and ’11:C’ will be explained their function later.

A joint movement to the start position ‘Home’ is needed for getting ready the robot and eliminate possible interferences with other programs.

Now, we enter the FOR Loop, from Layer = 1 to 3.

Setting up ‘Layer’ to 0 and the FOR Loop from 0 to 2 seems to not work.



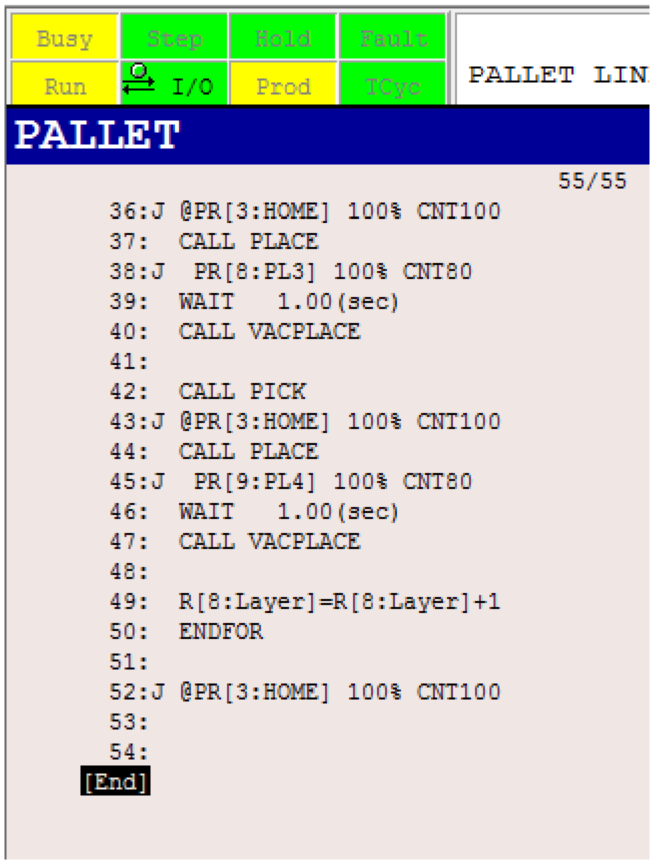
```
21:   CALL PICK
22:J @PR[3:HOME] 100% CNT100
23:   CALL PLACE
24:J  PR[6:PL1] 100% CNT80
25:   WAIT   1.00(sec)
26:   CALL VACPLACE
27:
28:   CALL PICK
29:J @PR[3:HOME] 100% CNT100
30:   CALL PLACE
31:J  PR[7:PL2] 100% CNT80
32:   WAIT   1.00(sec)
33:   CALL VACPLACE
34:
35:   CALL PICK
36:J @PR[3:HOME] 100% CNT100
37:   CALL PLACE
38:J  PR[8:PL3] 100% CNT80
39:   WAIT   1.00(sec)
40:   CALL VACPLACE
```

Figure 4. FANUC sim Code 02

First thing in the loop is to call in the subprogram ‘PICK’ (Fig.6) that will be explained later, but basically it is the program for picking the sticks. Then the robot passes again through position ‘Home’ to avoid going directly to the place point as that could cause collisions.

Then the subprogram ‘PLACE’ is called (will be explained later, Fig. 7) and the robot goes to the corresponding position where it leaves the stick (positions 6, 7, 8 and 9).

The robotic arm waits 1 second, this time control could avoid problems with the outputs switching ON or OFF before or after it is required. Then another subroutine is called, in this case ‘VACPLACE’ is a simulation program not done in the Teach Pendant and will be explained later (Fig.9).



PALLET

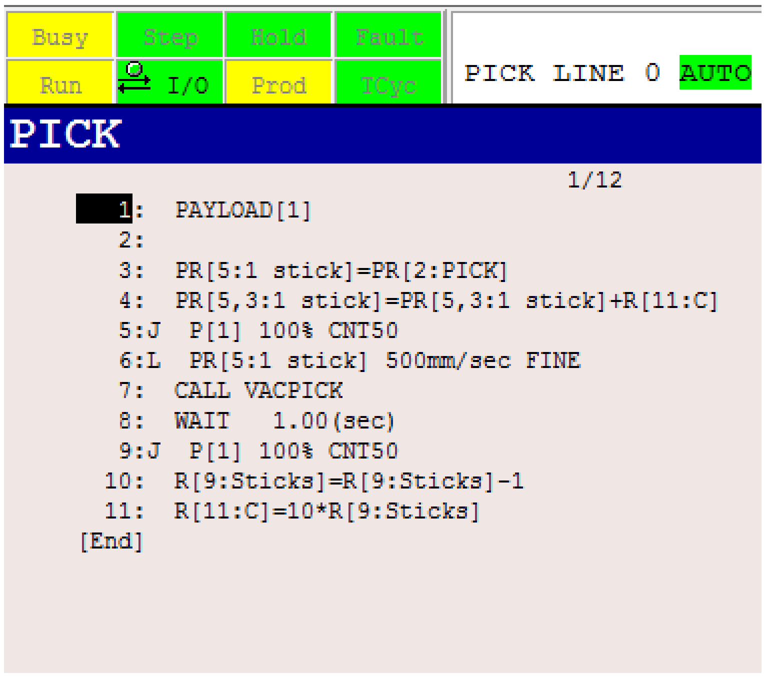
55/55

```
J @PR[3:HOME] 100% CNT100
  CALL PLACE
J  PR[8:PL3] 100% CNT80
  WAIT    1.00(sec)
  CALL VACPLACE

  CALL PICK
J @PR[3:HOME] 100% CNT100
  CALL PLACE
J  PR[9:PL4] 100% CNT80
  WAIT    1.00(sec)
  CALL VACPLACE

  R[8:Layer]=R[8:Layer]+1
  ENDFOR

J @PR[3:HOME] 100% CNT100


[End]
```

Figure 5. FANUC sim Code 03

Before ending the FOR Loop, we increase the layer by 1 as every time this loop is done it creates one layer of 4 sticks.

Then the robot just goes to 'Home' position to finish the program. As said, this program can do many layers as we configure in the code because it is based on calculations of positions.



```
PICK
                                        1/12
    1:  PAYLOAD[1]
    2:
    3:  PR[5:1 stick]=PR[2:PICK]
    4:  PR[5,3:1 stick]=PR[5,3:1 stick]+R[11:C]
    5:J  P[1] 100% CNT50
    6:L  PR[5:1 stick] 500mm/sec FINE
    7:  CALL VACPICK
    8:  WAIT   1.00(sec)
    9:J  P[1] 100% CNT50
   10:  R[9:Sticks]=R[9:Sticks]-1
   11:  R[11:C]=10*R[9:Sticks]
 [End]
```

Figure 6. PICK subprogram

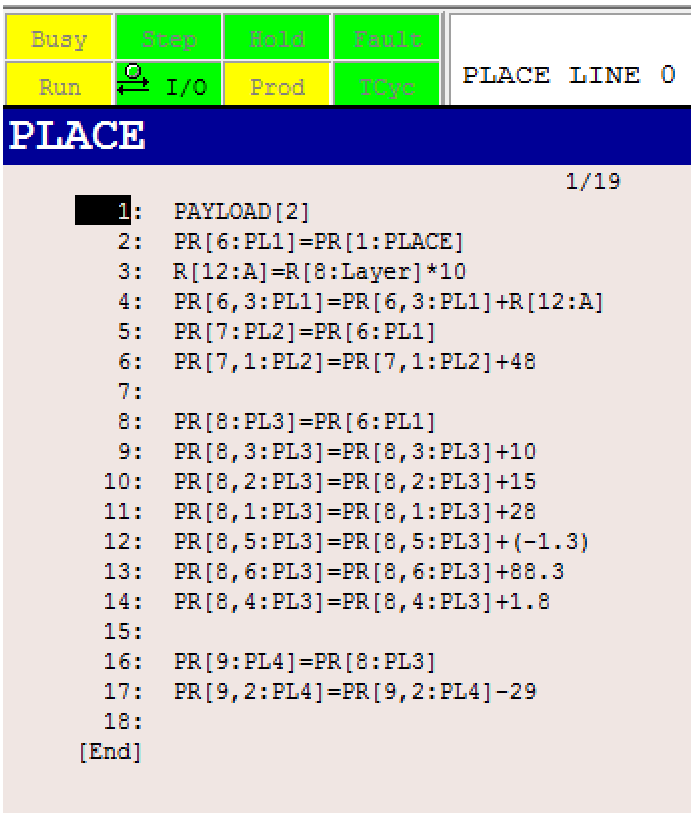
The Payload is set up to the first configuration that it was created for having a stick at the tool as a load.

In the registers a Position Register is created (PR2: PICK), this lays just on top the table where the first stick touches the table. This position must not be changed for calculations as it could mess the entire program. So another P.R. (PR5:1 stick) is set as the same position as PR2, except for the 'Z' axis that is changed and summed with the register R[11:C]. This register is equal to '10 * the number of sticks' so every time it enters the subprogram it goes lower.

Then a P1 is created way higher than the sticks and it moves there with a joint movement, after that it goes to the calculated PR5 in a linear movement.

The subroutine 'VACPICK' (Fig. 8) is a simulation program that basically picks the stick with the vacuum tool, then waits 1 second and goes again to P1.

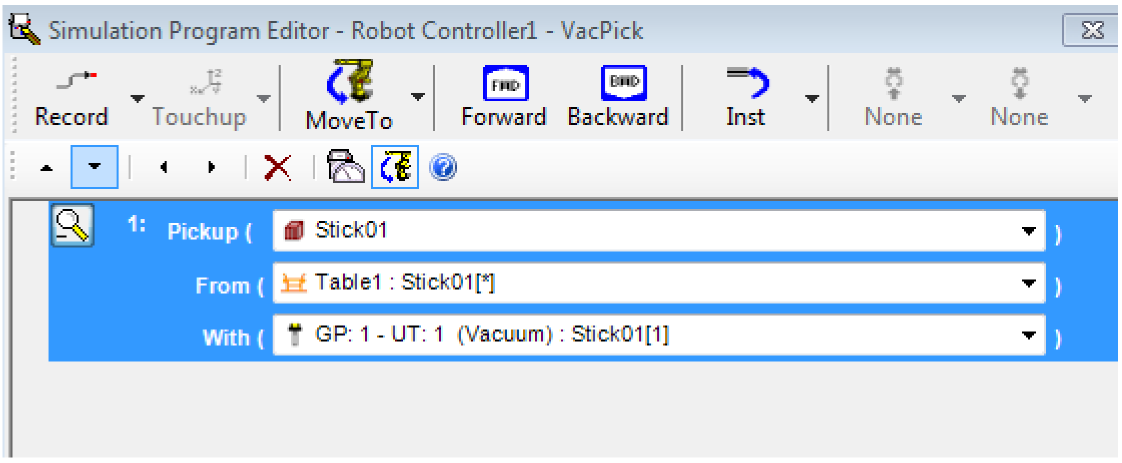
To finish, the registers 'R9: Sticks' and 'R11:C' are calculated again for next loop.



```
PLACE LINE 0
PLACE
                                      1/19
   PAYLOAD[2]
   PR[6:PL1]=PR[1:PLACE]
   R[12:A]=R[8:Layer]*10
   PR[6,3:PL1]=PR[6,3:PL1]+R[12:A]
   PR[7:PL2]=PR[6:PL1]
   PR[7,1:PL2]=PR[7,1:PL2]+48

   PR[8:PL3]=PR[6:PL1]
   PR[8,3:PL3]=PR[8,3:PL3]+10
   PR[8,2:PL3]=PR[8,2:PL3]+15
   PR[8,1:PL3]=PR[8,1:PL3]+28
   PR[8,5:PL3]=PR[8,5:PL3]+(-1.3)
   PR[8,6:PL3]=PR[8,6:PL3]+88.3
   PR[8,4:PL3]=PR[8,4:PL3]+1.8

   PR[9:PL4]=PR[8:PL3]
   PR[9,2:PL4]=PR[9,2:PL4]-29

[End]
```

Figure 7. PLACE subprogram

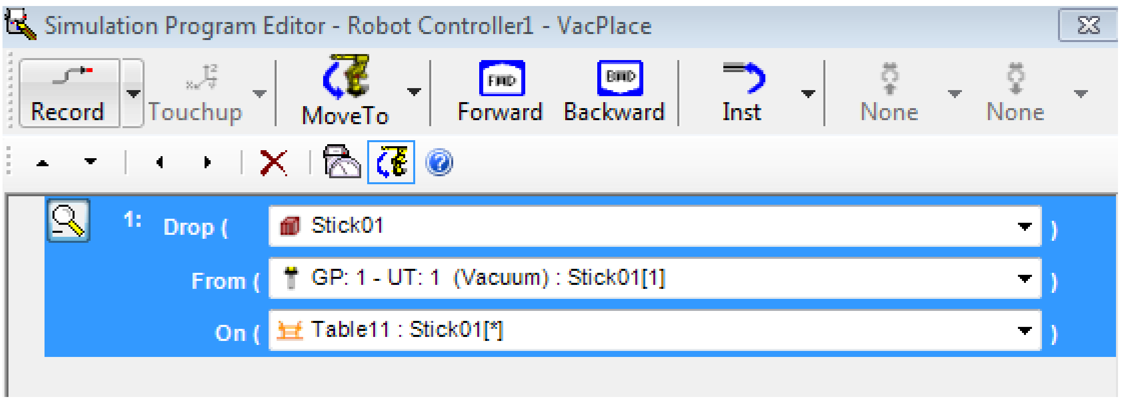
The ‘PLACE’ subprogram starts off with the setting of the Payload that was created for the robot without load at the tool. Then all the next calculations take place to create the points where the sticks are going to land, 4 positions for each layer. As seen, the ‘PR8’ needs changes in every axis, including the rotation ones.



1: Pickup (Stick01)
From (Table1 : Stick01[*])
With (GP: 1 - UT: 1 (Vacuum) : Stick01[1])

Figure 8. VacPick simulation subroutine

This subroutine is inside the simulation program and it is obliged to create it outside the T.P. Every part, tool and fixture has to be configured for picking up.



1: Drop (Stick01)
From (GP: 1 - UT: 1 (Vacuum) : Stick01[1])
On (Table11 : Stick01[*])

Figure 9. VacPlace simulation subroutine

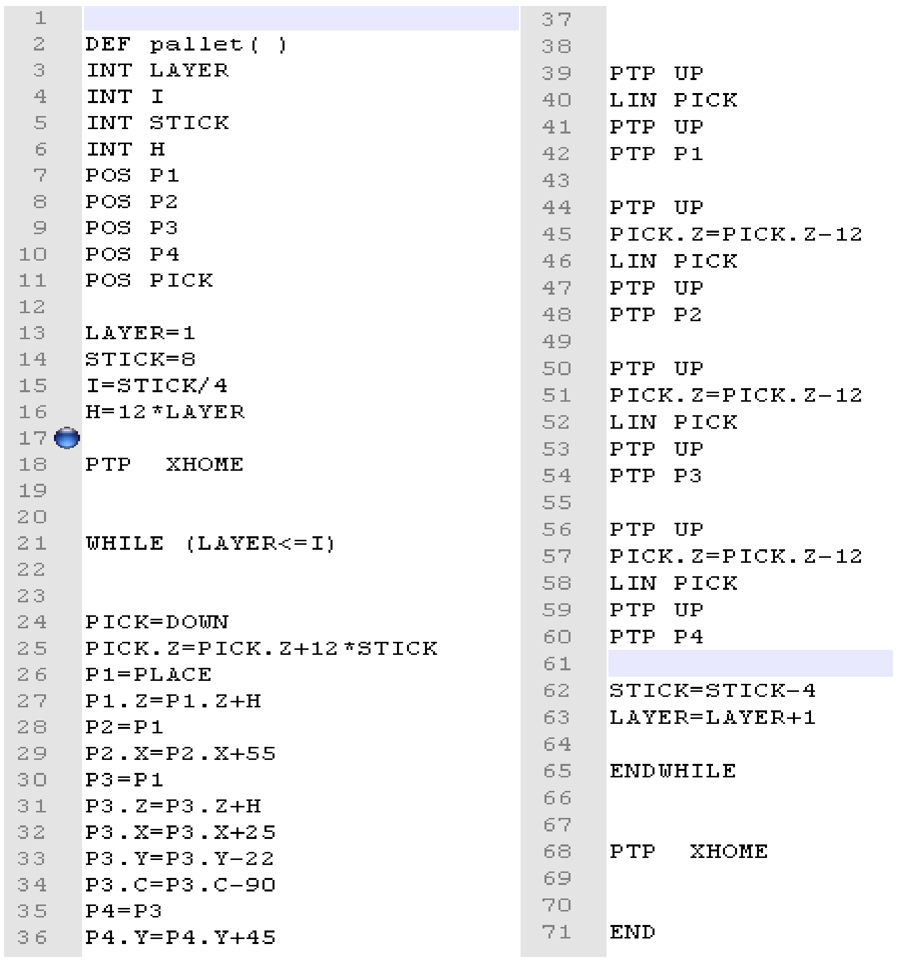
This is exactly the opposite of the subroutine mentioned before, it is used for dropping the sticks in the fixture ‘Table11’. Everything needs to be configured in the main program.

5.1.2. KUKA Sim Tech & OfficeLite HMI

KUKA has its own software called Sim Tech and uses another own program called OfficeLite that lets you connect the virtual T.P. to the simulation program.

Sim Tech is a simple and intuitive program to use for everyone and requires little knowledge on how it works, however this simplicity means that the program has some limitations in software.

The coding language is based on a mix of Pascal and its own language.



```
1
2   DEF pallet( )
3   INT LAYER
4   INT I
5   INT STICK
6   INT H
7   POS P1
8   POS P2
9   POS P3
10  POS P4
11  POS PICK
12
13  LAYER=1
14  STICK=8
15  I=STICK/4
16  H=12*LAYER
17
18  PTP  XHOME
19
20
21  WHILE (LAYER<=I)
22
23
24  PICK=DOWN
25  PICK.Z=PICK.Z+12*STICK
26  P1=PLACE
27  P1.Z=P1.Z+H
28  P2=P1
29  P2.X=P2.X+55
30  P3=P1
31  P3.Z=P3.Z+H
32  P3.X=P3.X+25
33  P3.Y=P3.Y-22
34  P3.C=P3.C-90
35  P4=P3
36  P4.Y=P4.Y+45
37
38
39  PTP UP
40  LIN PICK
41  PTP UP
42  PTP P1
43
44  PTP UP
45  PICK.Z=PICK.Z-12
46  LIN PICK
47  PTP UP
48  PTP P2
49
50  PTP UP
51  PICK.Z=PICK.Z-12
52  LIN PICK
53  PTP UP
54  PTP P3
55
56  PTP UP
57  PICK.Z=PICK.Z-12
58  LIN PICK
59  PTP UP
60  PTP P4
61
62  STICK=STICK-4
63  LAYER=LAYER+1
64
65  ENDWHILE
66
67
68  PTP  XHOME
69
70
71  END
```

Figure 10. KUKA SIM code

To start, declarations of integers and positions variables have to be made before the INI instruction (does not appear but should be at line 17) as well as their values.

The robot goes to 'Home' position and then enters the While (layer has to be equal or less than I).

Position ‘Pick’ takes the coordinates of ‘Down’, which position is just where the first stick touches the table at the picking structure. Then the height of the total of sticks is added to the ‘Z’ axis and the robot goes there to pick it up.

Position ‘P1’ gets the coordinates of ‘Place’ which sits where the first stick at the place structure touches the table, then the next 3 positions ‘P2,P3,P4’ are calculated from ‘P1’.

After that, the robot enters the movement loop where it goes to a position called ‘UP’ which lays on top of the picking structure and after picking the stick (‘Pick’) goes again to ‘UP’ and then to the corresponding place position.

We see that except of the first line, in the others there is a line which changes the height of the picking spot, subtracting 12 mm each time. Also, at the end we know that the robot has already picked up 4 sticks so we subtract it to the total of sticks.

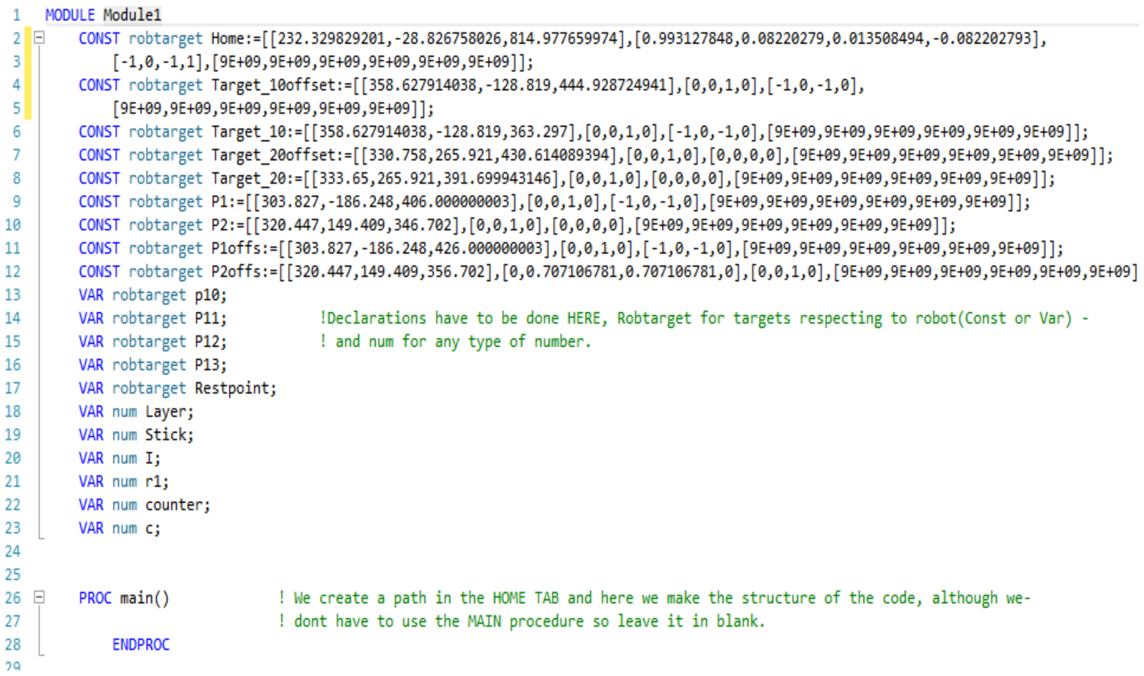
***Remark:** *This is done in an old version of KUKA Sim Tech and OfficeLite and it is not possible to link outputs nor inputs, so this shows only the movements of the robot.*

5.1.3. ABB Robotstudio

ABB has its software called Robotstudio, it is a powerful engine with a lot of features and adjustments that can make a simulation program much similar to the actual robot.

This has a strong drawback which makes the program complex and hard to understand in most parts, nonetheless the coding system called ‘Rapid’ is an interactive and intuitive programming method, as the program helps you with every instruction made, completing it for you if necessary.

The T.P. engine is weak as it has only a few options to edit something about the robot environment, the code cannot be modified and sometimes changing between T.P. and the sim program creates bugs and issues in the software.



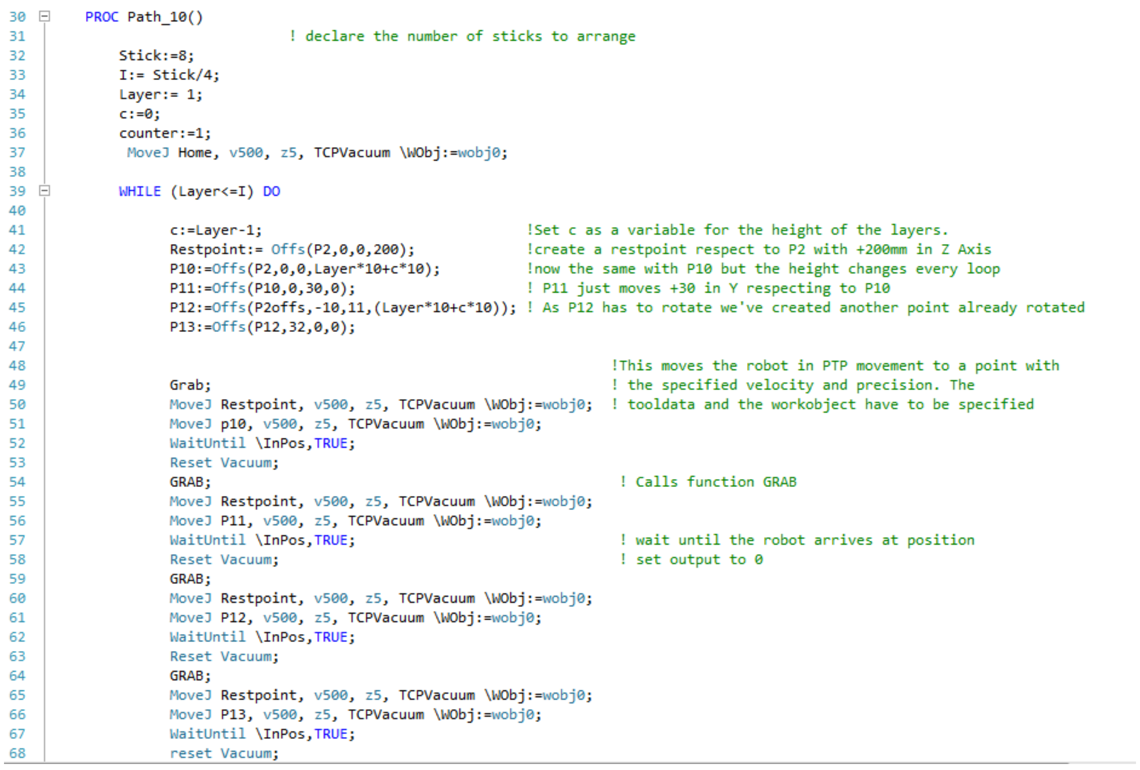
```
MODULE Module1
    CONST robtarget Home:=[[232.329829201,-28.826758026,814.977659974],[0.993127848,0.08220279,0.013508494,-0.082202793],
        [-1,0,-1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_10offset:=[[358.627914038,-128.819,444.928724941],[0,0,1,0],[-1,0,-1,0],
        [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_10:=[[358.627914038,-128.819,363.297],[0,0,1,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_20offset:=[[330.758,265.921,430.614089394],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_20:=[[333.65,265.921,391.699943146],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P1:=[[303.827,-186.248,406.000000003],[0,0,1,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P2:=[[320.447,149.409,346.702],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P1offs:=[[303.827,-186.248,426.000000003],[0,0,1,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P2offs:=[[320.447,149.409,356.702],[0,0.707106781,0.707106781,0],[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]
    VAR robtarget p10;
    VAR robtarget P11;              !Declarations have to be done HERE, Robtarget for targets respecting to robot(Const or Var) -
    VAR robtarget P12;              ! and num for any type of number.
    VAR robtarget P13;
    VAR robtarget Restpoint;
    VAR num Layer;
    VAR num Stick;
    VAR num I;
    VAR num r1;
    VAR num counter;
    VAR num c;


    PROC main()                ! We create a path in the HOME TAB and here we make the structure of the code, although we-
                               ! dont have to use the MAIN procedure so leave it in blank.
        ENDPROC
```

Figure 11. ABB Robotstudio Code 01

The Rapid coding system creates modules and its hierarchy down way are the Procedures, after that there are the instructions.

In the figure 11 we can see how to declare targets for the robot and variables. The easiest way to set targets is to do it in the simulation program as it is much more intuitive and simple; using the code could be useful to change some coordinates or quaternions (axis of rotation).

In this case, the procedure ‘Main’ is not utilized.



```
PROC Path_10()
                         ! declare the number of sticks to arrange
    Stick:=8;
    I:= Stick/4;
    Layer:= 1;
    c:=0;
    counter:=1;
     MoveJ Home, v500, z5, TCPVacuum \WObj:=wobj0;

    WHILE (Layer<=I) DO

          c:=Layer-1;                           !Set c as a variable for the height of the layers.
          Restpoint:= Offs(P2,0,0,200);         !create a restpoint respect to P2 with +200mm in Z Axis
          P10:=Offs(P2,0,0,Layer*10+c*10);      !now the same with P10 but the height changes every loop
          P11:=Offs(P10,0,30,0);                ! P11 just moves +30 in Y respecting to P10
          P12:=Offs(P2offs,-10,11,(Layer*10+c*10)); ! As P12 has to rotate we've created another point already rotated
          P13:=Offs(P12,32,0,0);

                                                          !This moves the robot in PTP movement to a point with
          Grab;                                           ! the specified velocity and precision. The
          MoveJ Restpoint, v500, z5, TCPVacuum \WObj:=wobj0;  ! tooldata and the workobject have to be specified
          MoveJ p10, v500, z5, TCPVacuum \WObj:=wobj0;
          WaitUntil \InPos,TRUE;
          Reset Vacuum;
          GRAB;                                            ! Calls function GRAB
          MoveJ Restpoint, v500, z5, TCPVacuum \WObj:=wobj0;
          MoveJ P11, v500, z5, TCPVacuum \WObj:=wobj0;
          WaitUntil \InPos,TRUE;                           ! wait until the robot arrives at position
          Reset Vacuum;                                    ! set output to 0
          GRAB;
          MoveJ Restpoint, v500, z5, TCPVacuum \WObj:=wobj0;
          MoveJ P12, v500, z5, TCPVacuum \WObj:=wobj0;
          WaitUntil \InPos,TRUE;
          Reset Vacuum;
          GRAB;
          MoveJ Restpoint, v500, z5, TCPVacuum \WObj:=wobj0;
          MoveJ P13, v500, z5, TCPVacuum \WObj:=wobj0;
          WaitUntil \InPos,TRUE;
          reset Vacuum;
```

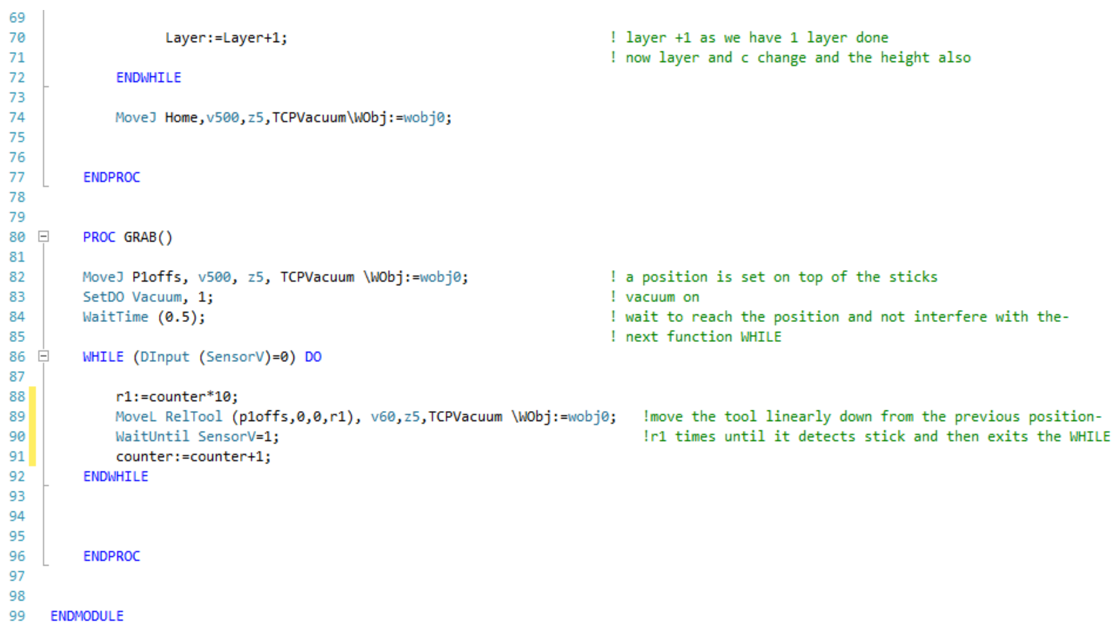
Figure 12. ABB Robotstudio code 02

Inside the procedure the variables have to be set. As always, the common variables are there with a few more that will be explained later.

First thing the robot has to do is go to position ‘Home’, as we see, the positions are in the next form: ***Type of move – Name – Pace – Precision – Tool – Workobject.***

Then the program goes into the While Loop (Layer equal or less than I), then the positions are configured with the instruction ‘Offset’. Next to them there are the comments explaining every point.

After it, calls the function ‘Grab’ (Fig. 13) and moves to ‘Restpoint’ and then to the placing point. A ‘WaitUntil \ InPos’ is needed before setting the vacuum to LOW because the simulation robot resets the vacuum before arriving to the target.



```

              Layer:=Layer+1;                                    ! layer +1 as we have 1 layer done
                                                                 ! now layer and c change and the height also
        ENDWHILE

        MoveJ Home,v500,z5,TCPVacuum\WObj:=wobj0;


    ENDPROC


    PROC GRAB()

    MoveJ P1offs, v500, z5, TCPVacuum \WObj:=wobj0;              ! a position is set on top of the sticks
    SetDO Vacuum, 1;                                             ! vacuum on
    WaitTime (0.5);                                              ! wait to reach the position and not interfere with the-
                                                                 ! next function WHILE
    WHILE (DInput (SensorV)=0) DO

        r1:=counter*10;
        MoveL RelTool (p1offs,0,0,r1), v60,z5,TCPVacuum \WObj:=wobj0;   !move the tool linearly down from the previous position-
        WaitUntil SensorV=1;                                            !r1 times until it detects stick and then exits the WHILE
        counter:=counter+1;
    ENDWHILE



    ENDPROC


ENDMODULE
```

Figure 13. ABB Robotstudio Code 03

At the end of the loop 1 layer is added to the counter condition. This procedure ends going to position ‘Home’.

The subroutine ‘Grab’ moves to ‘P1offs’ which lays on top of the sticks, the vacuum is set to HIGH and a ‘WaitTime’ is strictly necessary to not interfere with the While Loop.

To enter the While Loop, the digital input ‘SensorV’ (detects if there is stick) has to be LOW. Then ‘r1’ is used as height for grabbing the sticks, as we can see, if there is no stick, the robot goes 10mm lower with the instruction ‘Move RelTool’ which moves the tool respecting to position ‘p1offs’ in the axis wanted. There ‘r1’ is set up with the first and last equation that involves ‘counter’. In the move, the ‘Z’ axis is positive as in the simulation program ‘Z’ of the tool is negative.

The ‘WaitUntil SensorV = 1’ is there to space the time between actions and to make sure that the sensor detects the stick. A ‘WaitTime’ should work the same.

5.2. Hardware

5.2.1. KUKA KR-C3

The KUKA KR-C3 is the robot used in the Robotics Lab and the one that has to be taught to the students, it is almost the same as the previous version KR-C2 but the controller and robot is lighter and more compact. Designed for small spaces and little loads.

It is a small robotic arm with 6 degrees of freedom, stands out for his speed and low weight and has a maximum payload of 3 kg. Also, it has a maximum reach of 635 mm and a repeatability of ± 0.05 mm.

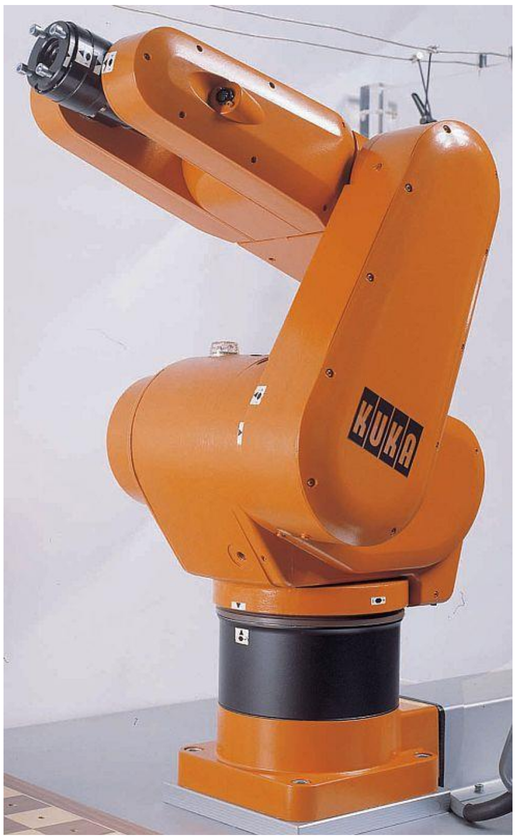


Figure 14. KUKA KR-C3

The T.P. lets you edit the code and control the robot in an easy and safe way.

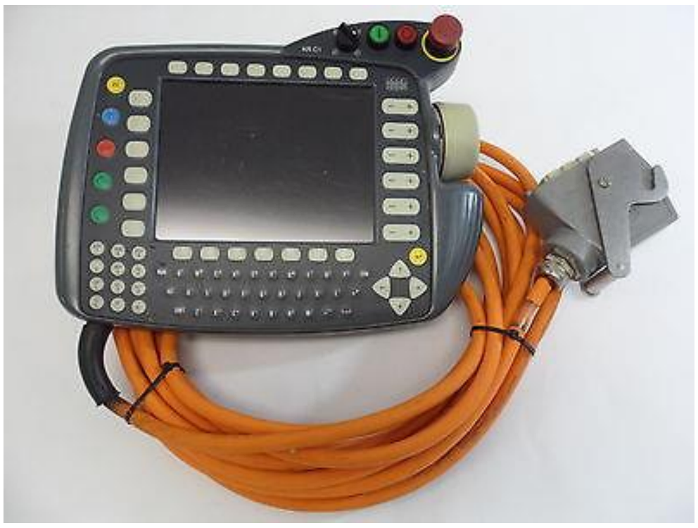
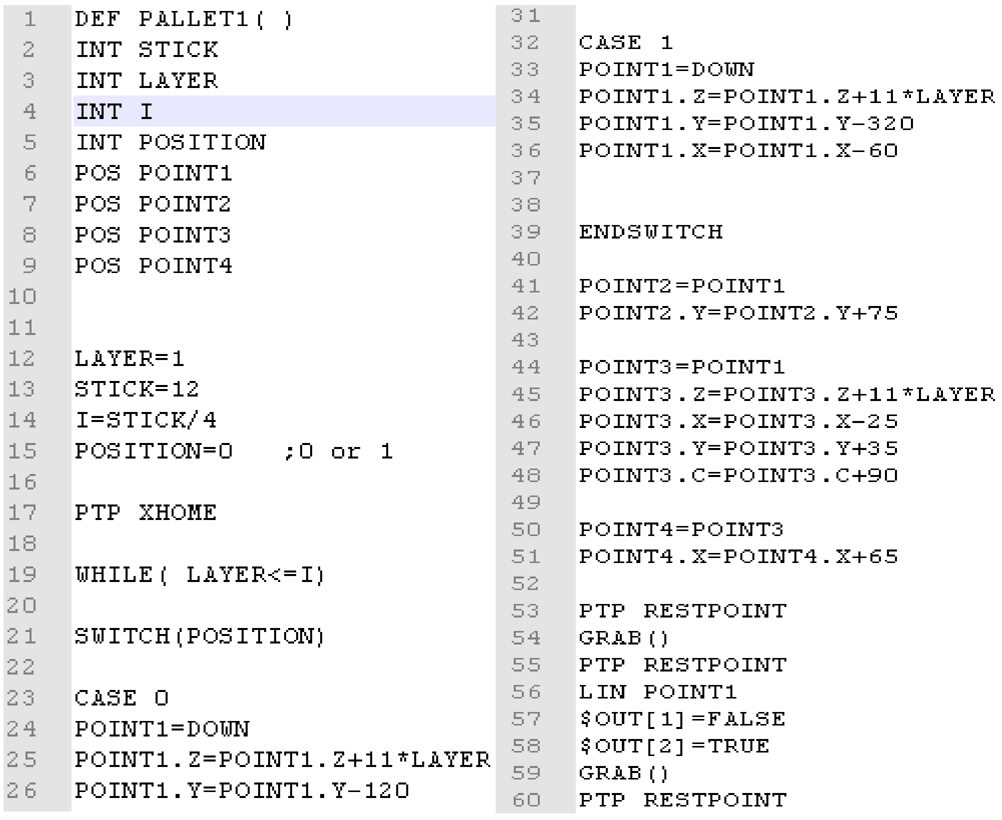


Figure 15. KUKA Teach pendant

The program made for the actual robot differs from the simulation one because in the actual robot there are the inputs and outputs needed, so the code changes.



```
DEF PALLET1( )
INT STICK
INT LAYER
INT I
INT POSITION
POS POINT1
POS POINT2
POS POINT3
POS POINT4


LAYER=1
STICK=12
I=STICK/4
POSITION=0    ;0 or 1

PTP XHOME

WHILE( LAYER<=I)

SWITCH(POSITION)

CASE 0
POINT1=DOWN
POINT1.Z=POINT1.Z+11*LAYER
POINT1.Y=POINT1.Y-120
```

```

CASE 1
POINT1=DOWN
POINT1.Z=POINT1.Z+11*LAYER
POINT1.Y=POINT1.Y-320
POINT1.X=POINT1.X-60


ENDSWITCH

POINT2=POINT1
POINT2.Y=POINT2.Y+75

POINT3=POINT1
POINT3.Z=POINT3.Z+11*LAYER
POINT3.X=POINT3.X-25
POINT3.Y=POINT3.Y+35
POINT3.C=POINT3.C+90

POINT4=POINT3
POINT4.X=POINT4.X+65

PTP RESTPOINT
GRAB()
PTP RESTPOINT
LIN POINT1
$OUT[1]=FALSE
$OUT[2]=TRUE
GRAB()
PTP RESTPOINT
```

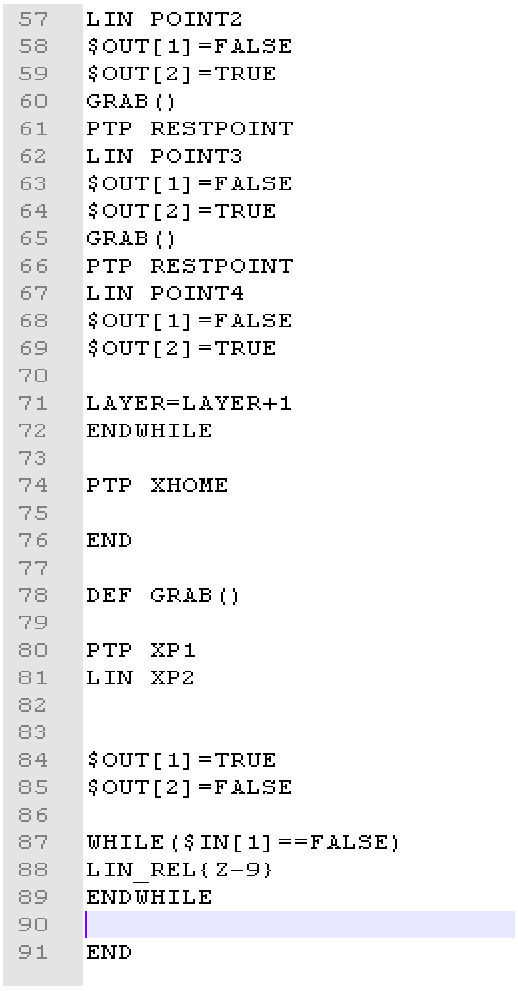
Figure 16. KUKA robot code 01

To start off, always before the INI instruction (does not appear in the code, but should be in Line 16), declarations must be done and after declarations values can be assigned to variables or constants.

The same as in the FANUC code we have similar variables that means the same, we have ‘Layer’, ‘Stick’, ‘I’ and ‘Position’. The last one is an optional function that works with the Switch case and defines where the structure is going to lay.

First the robot goes to ‘Home’ position, and then using a While Loop (While Layer is equal or less than I) the program starts the calculation of the points for placement of sticks. Same as before, we have 4 sticks per layer.

Position ‘Restpoint’ lays in between ‘Home’ and the stick structure, then the subprogram ‘GRAB’ is called (Figure 11) and then it goes with a linear movement to ‘Point1’ which is the first point of the place structure. Then Output 1 is shut OFF and output 2 is set ON, this is for stopping the vacuum.



```
LIN POINT2
$OUT[1]=FALSE
$OUT[2]=TRUE
GRAB()
PTP RESTPOINT
LIN POINT3
$OUT[1]=FALSE
$OUT[2]=TRUE
GRAB()
PTP RESTPOINT
LIN POINT4
$OUT[1]=FALSE
$OUT[2]=TRUE

LAYER=LAYER+1
ENDWHILE

PTP XHOME

END

DEF GRAB()

PTP XP1
LIN XP2


$OUT[1]=TRUE
$OUT[2]=FALSE

WHILE($IN[1]==FALSE)
LIN_REL{Z-9}
ENDWHILE

END
```

Figure 17. KUKA robot code 02

As we see, the process repeats. Calculations are made for each point and require linear axis moves, except for ‘Point3 ’ that also requires a 90 degree rotation in ‘Z’ axis.

At Line 78, the ‘GRAB’ subprogram is defined: the robot goes to ‘XP1’ (on top of the sticks) and then to ‘XP2’ (even closer). The vacuum is activated with output 1 to ON and output 2 to OFF, then the programs enters a while loop meaning that while the vacuum sensor does not detect any stick it does a linear movement in ‘–Z’ direction until the sensor notices it, only after that exits the subprogram.

5.2.2. FANUC LR Mate 200 iD/4s

The FANUC was recently acquired by the university and with it is required to teach the students on a new brand. This model has 6 axis or degrees of freedom. It is more modern than the KUKA and also smaller and lighter.

Has a protection of IP67, has a maximum reaching of 550 mm and a maximum payload of 4 kg; also repeatability is ± 0.013 mm.

Weighs only 20 kg and the controller R30iB is remarkable for its weight and minor size, perfect for compact places and little cabins.

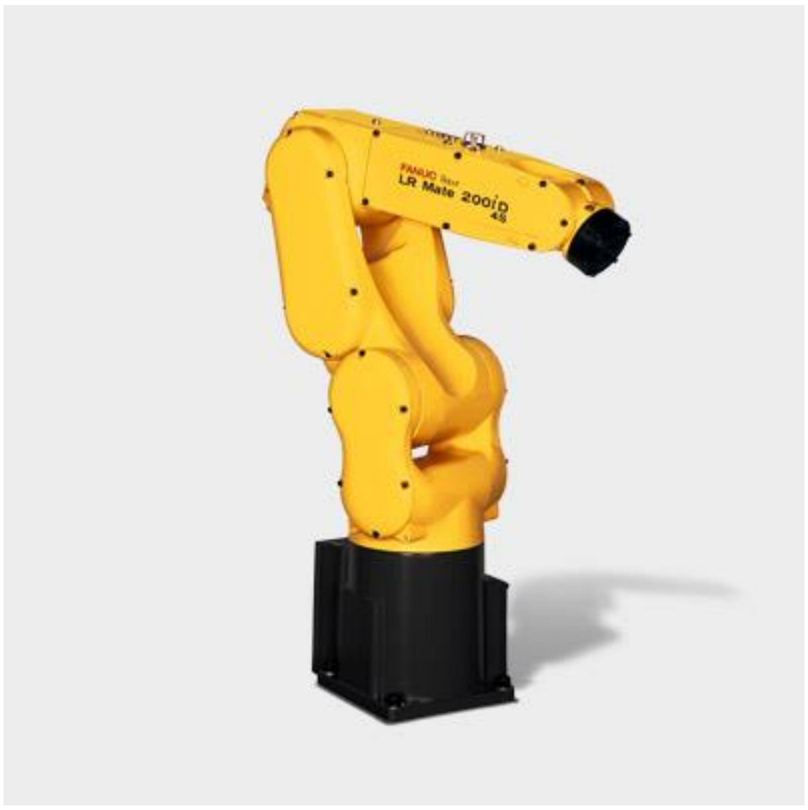


Figure 18. FANUC LR Mate 200iD /4s

The T.P. is called ‘iPendant Touch’; it is lightweight, ergonomic and the touchscreen is resistive. It has space for USB stick and reserved internal options for two more axes.

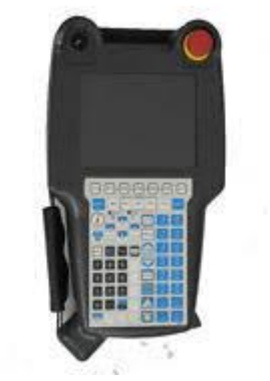
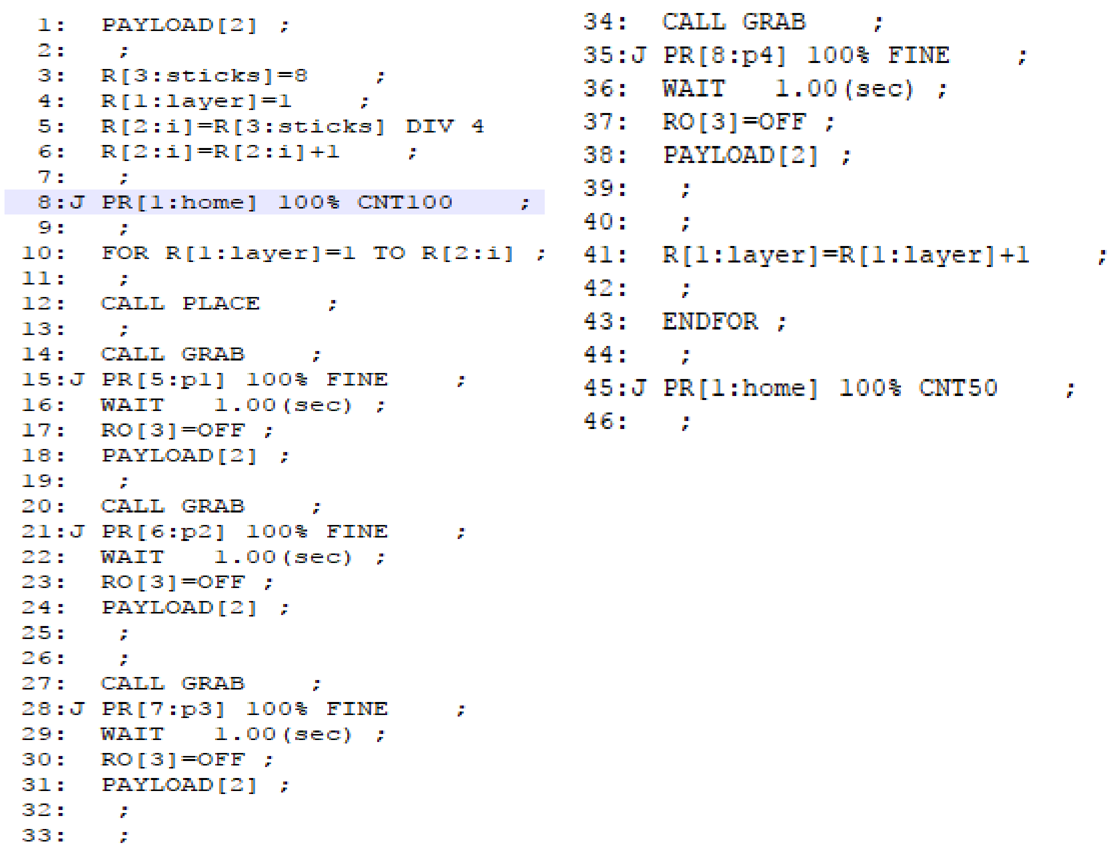


Figure 19. iPendant Touch from FANUC

The program made for hardware and software are different because in the actual robot program actually there is an input sensor that can detect if the output (vacuum) has picked something.



```
 1:   PAYLOAD[2] ;
 2:    ;
 3:   R[3:sticks]=8    ;
 4:   R[1:layer]=1    ;
 5:   R[2:i]=R[3:sticks] DIV 4
 6:   R[2:i]=R[2:i]+1    ;
 7:    ;
 8:J  PR[1:home] 100% CNT100    ;
 9:    ;
10:   FOR R[1:layer]=1 TO R[2:i] ;
11:    ;
12:   CALL PLACE    ;
13:    ;
14:   CALL GRAB    ;
15:J  PR[5:p1] 100% FINE    ;
16:   WAIT    1.00(sec) ;
17:   RO[3]=OFF ;
18:   PAYLOAD[2] ;
19:    ;
20:   CALL GRAB    ;
21:J  PR[6:p2] 100% FINE    ;
22:   WAIT    1.00(sec) ;
23:   RO[3]=OFF ;
24:   PAYLOAD[2] ;
25:    ;
26:    ;
27:   CALL GRAB    ;
28:J  PR[7:p3] 100% FINE    ;
29:   WAIT    1.00(sec) ;
30:   RO[3]=OFF ;
31:   PAYLOAD[2] ;
32:    ;
33:    ;
34:   CALL GRAB    ;
35:J  PR[8:p4] 100% FINE    ;
36:   WAIT    1.00(sec) ;
37:   RO[3]=OFF ;
38:   PAYLOAD[2] ;
39:    ;
40:    ;
41:   R[1:layer]=R[1:layer]+1    ;
42:    ;
43:   ENDFOR ;
44:    ;
45:J  PR[1:home] 100% CNT50    ;
46:    ;
```

Figure 20. FANUC robot code

As it was done in the simulation program, the payload has to be set up, in this case, payload [2] is movement without load and payload [1] motion with 1 stick on the tool.

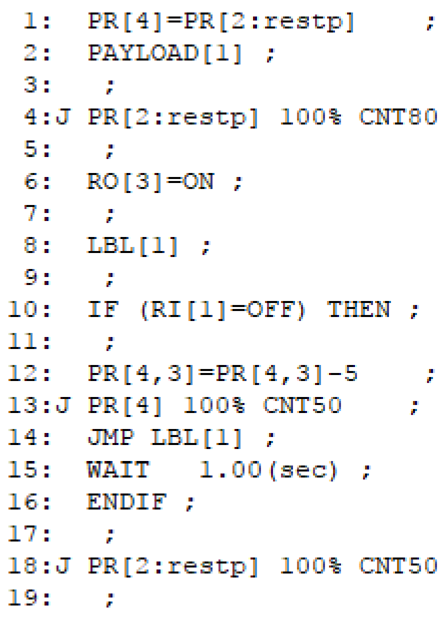
The variables are configured as registers for ease and recognition of the name. We see that the register 'R[2:i]' has two calculations as the coding system does not let you incorporate two different operands in the same line and because the 'For loop' cannot be entered with a variable its value is 0 so it is needed to sum 1 up to 'R[2:i]'.

First off, the robot goes to 'Home' position, then the subprogram 'PLACE' is called, only once and at the beginning (will be explained later). After that comes the subprogram 'GRAB'.

The layer is composed of the four positions registers: '5:p1, 6:p2, 7:p3 and 8:p4'.

The robot waits 1 second for assuring to reach the corresponding position, then the vacuum is turned off (RO[3]) and the payload is set again as without load.

To finish, one unity is added to layer and if reaches the maximum value for the For loop it ends; with the robot to 'Home' position the program ends.



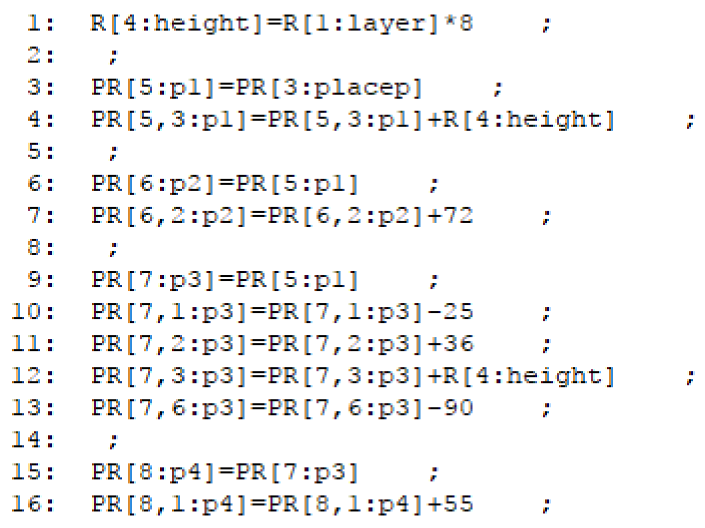
```
  PR[4]=PR[2:restp]    ;
  PAYLOAD[1] ;
   ;
J PR[2:restp] 100% CNT80
   ;
  RO[3]=ON ;
   ;
  LBL[1] ;
   ;
  IF (RI[1]=OFF) THEN ;
   ;
  PR[4,3]=PR[4,3]-5    ;
J PR[4] 100% CNT50    ;
  JMP LBL[1] ;
  WAIT   1.00(sec) ;
  ENDIF ;
   ;
J PR[2:restp] 100% CNT50
   ;
```

Figure 21. Subprogram 'Grab' FANUC

A position register 'PR[4]' is equalized with 'Restpoint' position, this way it is possible to work with a position without changing any coordinates. After, the payload[1] is set on.

Then the robot goes to ‘Restpoint’ which is higher on top of the pick structure and the vacuum is turned on. With an ‘IF’ condition that means if the input ‘RI[1]’ is off, 5 mm of the ‘Z’ axis are subtracted to that position so it starts to go down until the input detects piece, this is achieved because of the instruction ‘LBL and JMP LBL’.

A ‘Wait’ is put on there as it is needed to make sure it grabs the stick. Then, it just goes to ‘Restpoint’ again.



```
R[4:height]=R[1:layer]*8    ;
 ;
PR[5:p1]=PR[3:placep]    ;
PR[5,3:p1]=PR[5,3:p1]+R[4:height]    ;
 ;
PR[6:p2]=PR[5:p1]    ;
PR[6,2:p2]=PR[6,2:p2]+72    ;
 ;
PR[7:p3]=PR[5:p1]    ;
PR[7,1:p3]=PR[7,1:p3]-25    ;
PR[7,2:p3]=PR[7,2:p3]+36    ;
PR[7,3:p3]=PR[7,3:p3]+R[4:height]    ;
PR[7,6:p3]=PR[7,6:p3]-90    ;
 ;
PR[8:p4]=PR[7:p3]    ;
PR[8,1:p4]=PR[8,1:p4]+55    ;
```

Figure 22. Subprogram ‘Place’ FANUC

This subroutine is only meant for calculations, there are no motion actions on the robot.

First off, the height (R[4]) is set up as the ‘Layer’ multiplied by 8 so it goes higher with every layer. As in every program made, the 4 positions are calculated the same.

‘P1’ takes the coordinates of ‘Placep(oint)’, which lays where the first stick touches the table. Then ‘P2 and P3’ just equalize to ‘P1’ and ‘P4’ takes the ‘P3’ coordinates. For the orientation of the perpendicular layer, a turn of -90 degrees in the ‘Z’ axis is obliged.

6. BRAND DIFFERENCES

6.1. Robots

It is important to know which robot has the best features and characteristics for knowing which to buy or use in a production line. In the next table, a comparison is made. The ABB robot is a similar model than the other two brands.

MODEL	KUKA KRC3	FANUC LRMATE 200iD/4s	ABB IRB 120
MAX. LOAD	3 kg	4 kg	3 kg
REACH	635 mm	550 mm	580 mm
WEIGHT	30 kg	20 kg	25 kg
IP	IP40	IP67	IP30
REPEATABILITY	± 0.05 mm	± 0.013 mm	± 0.01 mm
ARM SIGNALS(I/O)	4/4	6/2	6/4
J1 VELOCITY	240 º/s	460 º/s	250 º/s
J2 VELOCITY	210 º/s	460 º/s	250 º/s
J3 VELOCITY	240 º/s	520 º/s	250 º/s
J4 VELOCITY	375 º/s	560 º/s	320 º/s
J5 VELOCITY	300 º/s	560 º/s	320 º/s
J6 VELOCITY	375 º/s	900 º/s	420 º/s
J1 MOTION RANGE	±180º	360º	+165º to -165º
J2 MOTION RANGE	-45º/135º	230º	110º to -110º
J3 MOTION RANGE	-225º/45º	402º	70º to -110º
J4 MOTION RANGE	±180º	380º	160º to -160º
J5 MOTION RANGE	±135º	240º	120º to -120º
J6 MOTION RANGE	± ∞	720º	400º to -400º
PRICE	7.000 €	14.092 €	18.638 €

We can conclude that the best option would be the FANUC as its light weight, protection, joint velocity and motion ranges are better than the other two models. The only important drawback between the other is the maximum reach.

Also, taking in account the price, the KUKA is a good option for quality at cheap cost.

6.2. Software

As for the software, a table like the previous is not understandable for this subject, so the software aspects will be explained objectively and with remarks from the author of the project too.

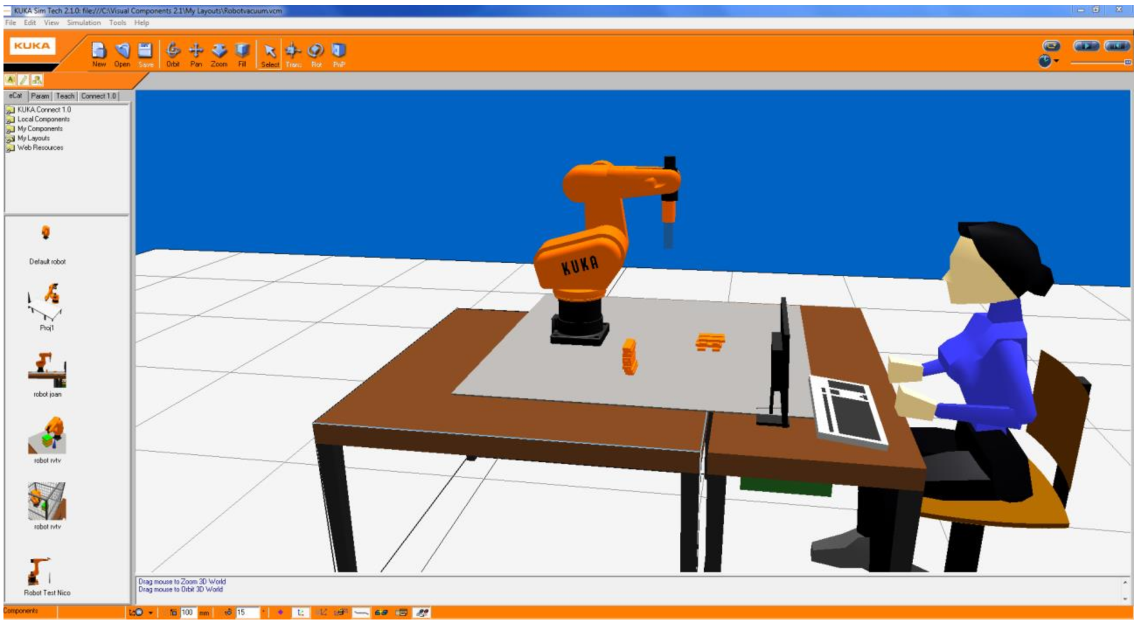
6.2.1. Simulation

The simulation program will be rated on how easy or complex, its overall functions and special features.

To start off with the easiest program, that would be the **KUKA** Sim Pro. To make a layout of a robot with a tool configured standing on a table and doing a simple task is relatively simple and requires low effort of learning. It is an intuitive program for offline programming and it is good to learn about the environment near a robot.

It has a wide variety of components thanks to the 'eCatalog' from a KUKA server that lets you download new components and resources from it.

Nonetheless, the simplicity of the program makes it sloppy and unclear to work in a serious way with it. The teach function does not let many options for making programs. Also the view functions are slow as there are no shortcuts for the mouse, only a few of the keyboard.

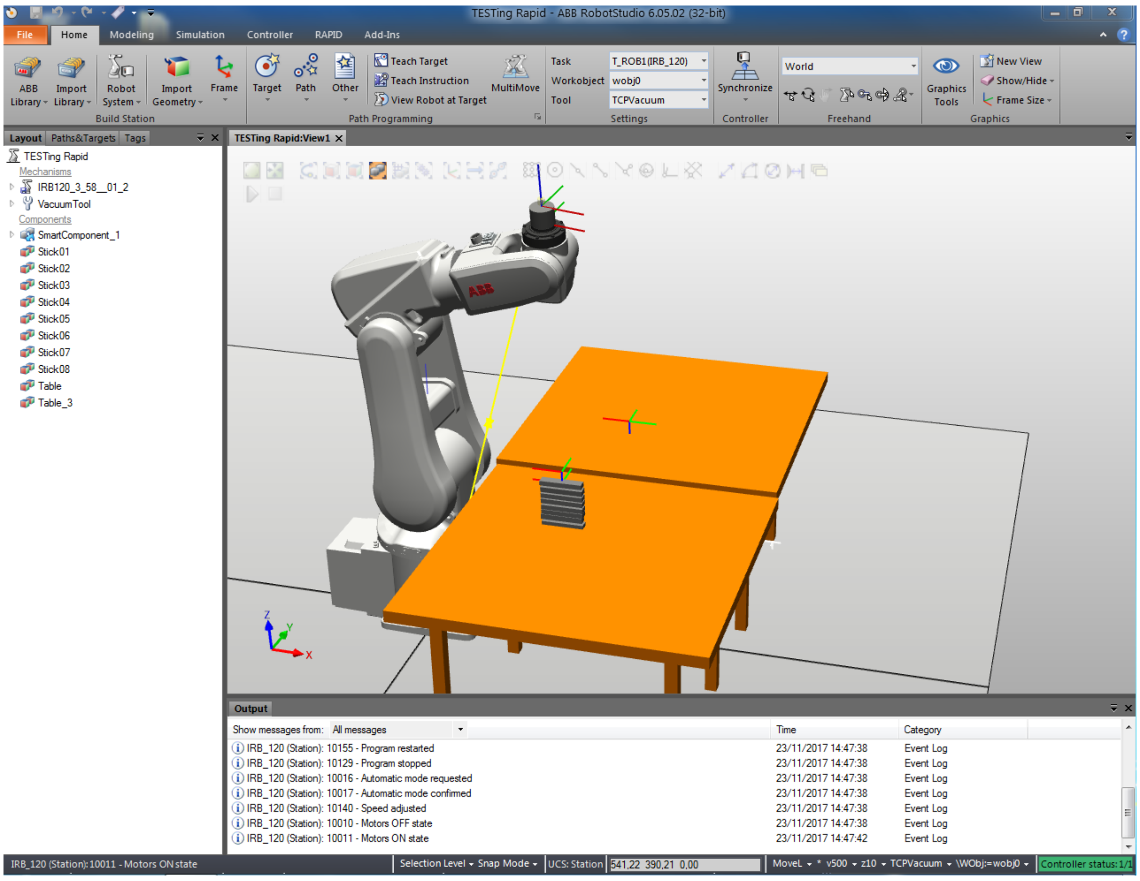


Drag mouse to Zoom 3D World
Drag mouse to Orbit 3D World

Figure 23. KUKA SIM Tech

As for the **ABB** Robotstudio, that would be the most complex sim program out of the three. However, its difficulty on making a simulation program is compensated with the enormous aspects and feature that has. Creating and running a successful program means spending a good amount of time learning but then the capabilities that has are seen and going further away with does not seem already tough.

However, there are aspects (like the constant manual synchronization it is needed between coding and simulation, the amount of configurations you have to do for tools and also the axis orientation that creates problems sometimes) that could be better done.



Task: T_ROB1(IRB_120)
Workobject: wobj0
Tool: TCPVacuum
Settings: World

```
Show messages from: All messages                       Time                  Category
IRB_120 (Station): 10155 - Program restarted           23/11/2017 14:47:38   Event Log
IRB_120 (Station): 10129 - Program stopped             23/11/2017 14:47:38   Event Log
IRB_120 (Station): 10016 - Automatic mode requested    23/11/2017 14:47:38   Event Log
IRB_120 (Station): 10017 - Automatic mode confirmed    23/11/2017 14:47:38   Event Log
IRB_120 (Station): 10140 - Speed adjusted              23/11/2017 14:47:38   Event Log
IRB_120 (Station): 10010 - Motors OFF state            23/11/2017 14:47:38   Event Log
IRB_120 (Station): 10011 - Motors ON state             23/11/2017 14:47:42   Event Log
```

```
IRB_120 (Station):10011 - Motors ON state   Selection Level ▾ Snap Mode ▾ | UCS: Station | 541,22  390,21  0,00 | MoveL ▾ * v500 ▾ z10 ▾ TCPVacuum ▾ \WObj:=wobj0 ▾ | Controller status: 1/1
```

Figure 24. ABB Robotstudio

Then, the **FANUC** Roboguide as for difficulty it would be between the two, it is capable of many things but it is hard to understand how it works because it uses connections between the objects for running it, the objects are not 'free' in space like in the other programs. That utilization of the objects creates a lot of configurations that can be made such as create and destroy delays, approach and retreat automatic points, simple tool creation… also there are a good amount of objects that can be created: parts, fixtures, obstacles, workers, machines, cables…

and every type of object has its own function in the program and this is where it comes the complex part, knowing which object it is needed for an especial function and knowing how to set it up correctly.

This also creates a problem being the program not intuitive and using the virtual TP only for programming is not possible unlike in the actual robot. Some of the functions do not work properly and the program crashes occasionally and corrupts the file.

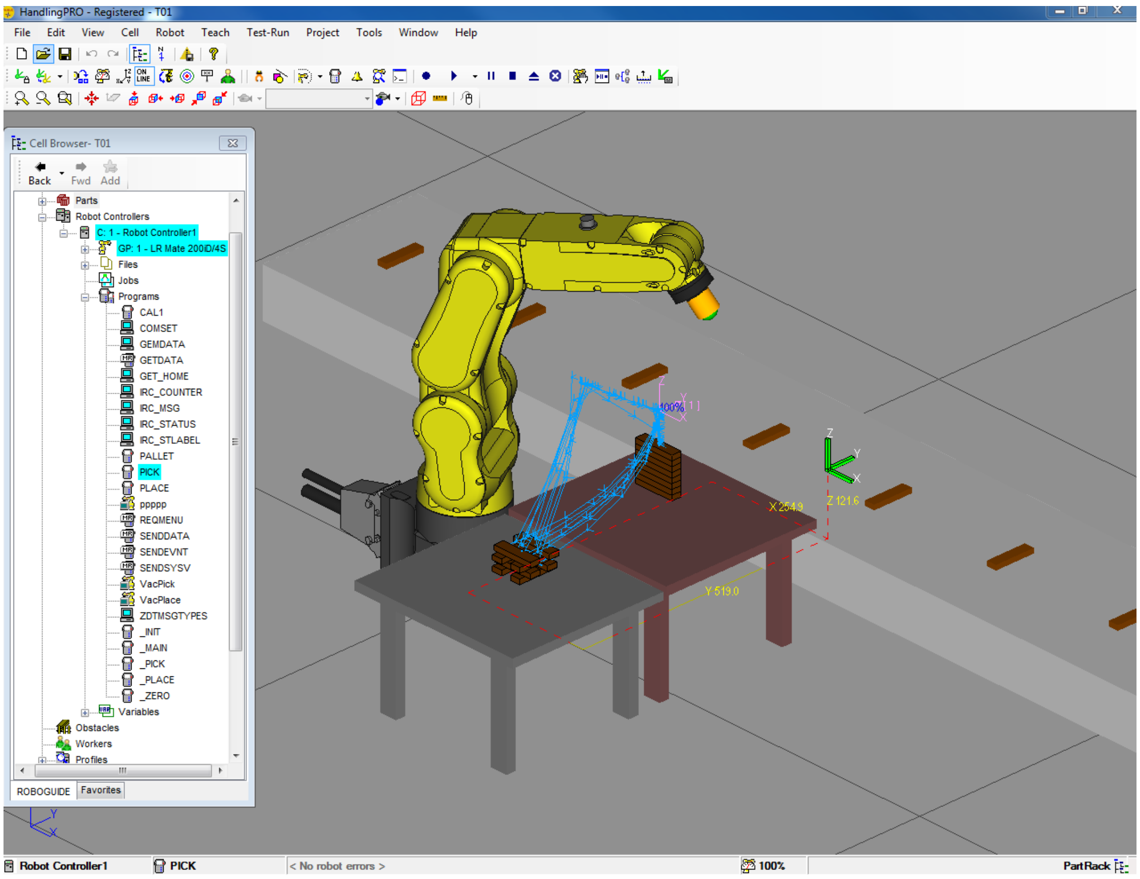


Figure 25. FANUC Roboguide

6.2.2. Coding

KUKA Sim pro for coding has to work side a side with OfficeLite which acts as the TP. The coding in KUKA is pretty straightforward and easy to understand. Uses a similar language to Pascal called KRL, like a mix between Pascal and its own KUKA language. It is a modular language with 4 types of data and two types of movement can be programmed.

Also if a mistake is made the error log is clear and lets you know where the actual issue is at. Overall, it is a good coding system for beginners and a good point to start to work with robots due to its clearness and ease of use.

Robotstudio uses the ABB own coding language which was influenced by the predecessor ARLA and also by C. Includes procedures, trap routines, functions, automatic error handling and can multi task.

Unlike the complexity of the sim program, the Rapid system helps you code, showing all the options that can be typed in the instruction and this does not waste time of learning how to code that specific language.

Like it was mentioned before, the Rapid system in the software works separately from the sim program unless you set it up manually. Also, some arithmetic instructions are hard to understand and unclear, i.e. rotations for a point are in quaternion system.

FANUC Roboguide does not have a free programming system but is basis is on language KAREL, it is possible to code in a PC and then run it in the robot but any mistake appearing in the code will have to be fixed in the PC again. The program uses the TP for coding. Its principal difference it is that utilizes various types of registers (normal registers, position registers, palletize registers…), this registers have to be set up before start coding as if that is not done, correcting the program in case of error could be a great mess.

The TP has a lot of good functions: resistive touchscreen, easy jogging, USB port, lightweight, option to window the screen until 4 windows… but at the same time some functionalities seem older.

Referring to instructions, the programming does the job but with less options than the competence, besides, the not freedom coding makes it hard for the programmer to be imaginative and creative.

	SIM Pro & OfficeLite	**ABB Robotstudio**	**FANUC Roboguide**
Advantages	Straightforward coding and simulating. Instructions are clear and manageable.	Simulation features and options are huge. Automatic coding system and clear error handling.	Registers do not need data type specification. TP has options which ease working.
Drawbacks	Simplicity limits the options for the robot. The sim program is old and tedious for working, i.e. cannot link I/O to program.	Complexity means to spend quite time learning. Synchronization between coding and simulation is not automatic.	Coding system is not free and instructions are limited. It is not possible to use I/O in TP programming.

7. OVERVIEW OF EXERCISE DIFFERENCES

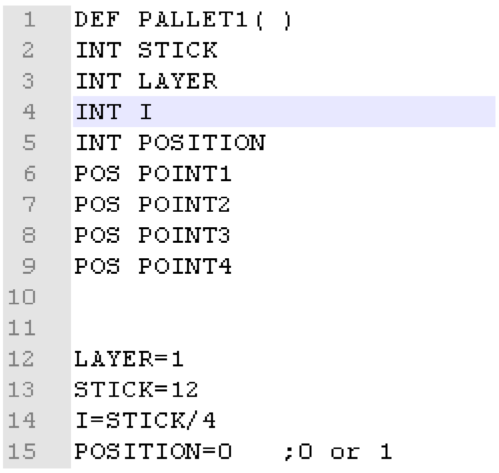
To the reader to understand better the differences between the three coding language systems in this chapter, it will be shown how coding changes referring to instructions to accomplish an objective for the robot.

This will be explained summarized, if the reader needs more information it is more detailed in the chapter 5.Codes.

7.1. Declaring variables

The first thing in the program is to declare the different variables and constants and set their value.

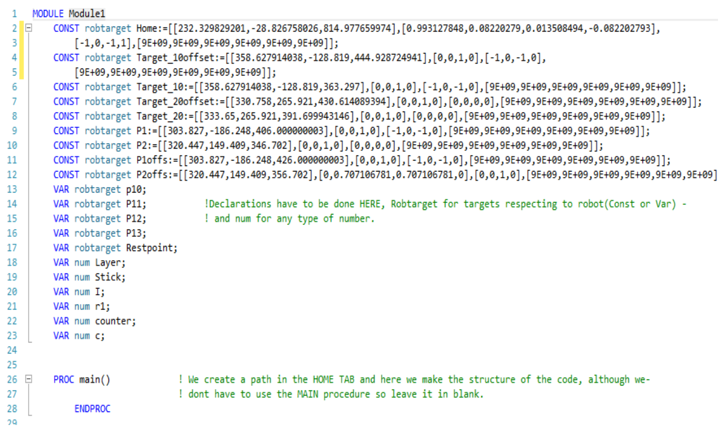
- KUKA: Declarations are made at the top before the INI system instruction. There are four types and must be declared directly. After that, the value has to be given below it.



```
DEF PALLET1( )
INT STICK
INT LAYER
INT I
INT POSITION
POS POINT1
POS POINT2
POS POINT3
POS POINT4


LAYER=1
STICK=12
I=STICK/4
POSITION=0    ;0 or 1
```

Figure 26. KUKA declarations

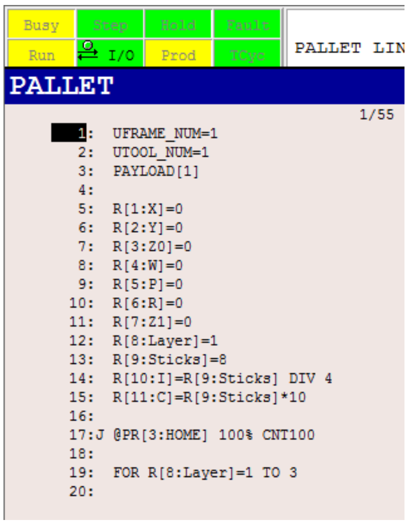
- ABB: Also, declarations must be done before the Procedure routine and the value can be given directly there.



```
MODULE Module1
    CONST robtarget Home:=[[232.329829201,-28.826758026,814.977659974],[0.993127848,0.08220279,0.013508494,-0.082202793],
        [-1,0,-1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_10offset:=[[358.627914038,-128.819,444.928724941],[0,0,1,0],[-1,0,-1,0],
        [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_10:=[[358.627914038,-128.819,363.297],[0,0,1,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_20offset:=[[330.758,265.921,430.614089394],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_20:=[[333.65,265.921,391.699943146],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P1:=[[303.827,-186.248,406.000000003],[0,0,1,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P2:=[[320.447,149.409,346.702],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P1offs:=[[303.827,-186.248,426.000000003],[0,0,1,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P2offs:=[[320.447,149.409,356.702],[0,0.707106781,0.707106781,0],[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]
    VAR robtarget p10;
    VAR robtarget P11;          !Declarations have to be done HERE, Robtarget for targets respecting to robot(Const or Var) -
    VAR robtarget P12;          ! and num for any type of number.
    VAR robtarget P13;
    VAR robtarget Restpoint;
    VAR num Layer;
    VAR num Stick;
    VAR num I;
    VAR num r1;
    VAR num counter;
    VAR num c;


    PROC main()             ! We create a path in the HOME TAB and here we make the structure of the code, although we-
                            ! dont have to use the MAIN procedure so leave it in blank.
        ENDPROC
```

Figure 27. ABB declarations

- FANUC: Declarations can be done, either on the register tab or inside the program. Data type just has to be according the five types of registers existing. The values can be changed also inside or in the register tab.



```
UFRAME_NUM=1
UTOOL_NUM=1
PAYLOAD[1]

R[1:X]=0
R[2:Y]=0
R[3:Z0]=0
R[4:W]=0
R[5:P]=0
R[6:R]=0
R[7:Z1]=0
R[8:Layer]=1
R[9:Sticks]=8
R[10:I]=R[9:Sticks] DIV 4
R[11:C]=R[9:Sticks]*10

J @PR[3:HOME] 100% CNT100

FOR R[8:Layer]=1 TO 3

```

Figure 28. FANUC declarations

7.2. Grab routine

Here, how every robot has to approach, pick a stick and retreat will be seen in code.

- KUKA: A point is set higher than the sticks' structure, the vacuum gets on and with the instruction 'While'; while the input is LOW starts going down with the instruction LIN_REL{Z -9}, which makes the robot descend linearly 9 mm until it detects the stick.

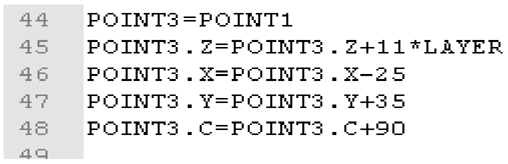
- ABB: The instructions are nearly the same except for a few; a 'WaitTime' is needed before the 'While', inside the 'While' two variables more are needed as the instruction for descending is not the same. 'MoveL RelTool' is for moving the tool linearly but respecting to a point, as the height has to change two variables are needed to change the height every time goes inside the loop.

- FANUC: Because there is no 'While' loop, two mixed instructions are needed: 'IF' and 'JMP & LBL'. A position register is equalized to the one that lays higher of the structure, then the LBL starts before the 'IF'; if the input is LOW 5 mm are subtracted of the position and the robot descends that, after it there is the 'JMP LBL' and the process repeats until input is not LOW anymore.

7.3. Placing routine

Here the process for calculating the placement points and the routine of how the robot does the job will be summarized.

- KUKA: A point where the first stick to be placed touches the table is taught, then from this point the others are calculated, P1 and P2 in the same orientation and height and P3 and P4 higher and 90 degrees turned. The instructions in KUKA for this are as shown in the figure:

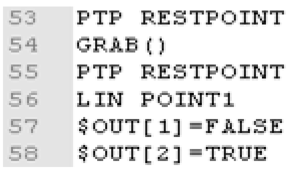


```
POINT3=POINT1
POINT3.Z=POINT3.Z+11*LAYER
POINT3.X=POINT3.X-25
POINT3.Y=POINT3.Y+35
POINT3.C=POINT3.C+90
```

Figure 29. KUKA calculation process

As we see, these are the instructions for it, it is possible to change the three axis and its rotation of a point (X, Y, Z, A, B, C). Units are in millimetres and degrees.

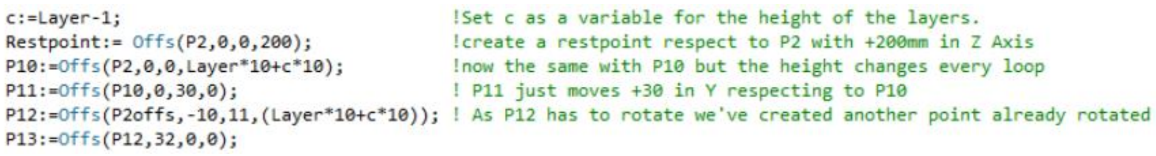
To do the action of placing, a call to subroutine 'Grab' and deactivation of vacuum are needed.



```
PTP RESTPOINT
GRAB()
PTP RESTPOINT
LIN POINT1
$OUT[1]=FALSE
$OUT[2]=TRUE
```

Figure 30. KUKA placing action

- ABB: In this code, the points have to be calculated with an offset from a point. Also rotation is only possible in this way using quaternions which are hard to calculate, so the solution for rotating P3 and P4 is to add a point already rotated to make it simpler.

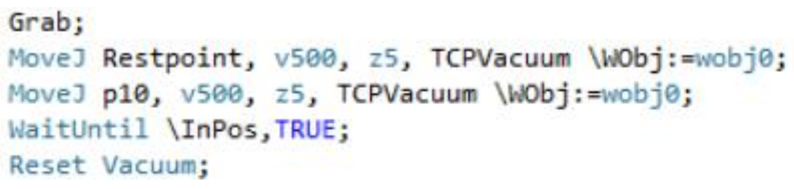


```
c:=Layer-1;                                !Set c as a variable for the height of the layers.
Restpoint:= Offs(P2,0,0,200);              !create a restpoint respect to P2 with +200mm in Z Axis
P10:=Offs(P2,0,0,Layer*10+c*10);           !now the same with P10 but the height changes every loop
P11:=Offs(P10,0,30,0);                     ! P11 just moves +30 in Y respecting to P10
P12:=Offs(P2offs,-10,11,(Layer*10+c*10)); ! As P12 has to rotate we've created another point already rotated
P13:=Offs(P12,32,0,0);
```

Figure 31. ABB calculation process

In the comment section we can see the explanation given for the calculations. Also, the variable 'c' is needed as in this case P1 and P3 are not equalized and the height has to be set up like this.

To make the placement process it works like KUKA except for one added instruction.

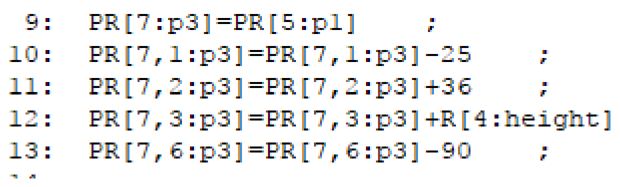


```
Grab;
MoveJ Restpoint, v500, z5, TCPVacuum \WObj:=wobj0;
MoveJ p10, v500, z5, TCPVacuum \WObj:=wobj0;
WaitUntil \InPos,TRUE;
Reset Vacuum;
```

Figure 32. ABB placing action

Before turning off the vacuum it is important to add the ‘WaitUntil \InPos’ as the program tries to execute the instructions as fast as possible and it deactivates the vacuum before arriving at the point.

- FANUC: This system works quite similar to KUKA, the instruction does the same, it only changes its syntax.

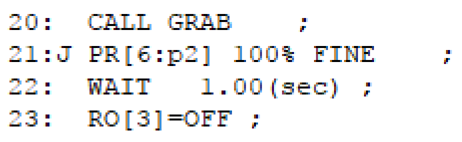


```
PR[7:p3]=PR[5:p1]    ;
PR[7,1:p3]=PR[7,1:p3]-25    ;
PR[7,2:p3]=PR[7,2:p3]+36    ;
PR[7,3:p3]=PR[7,3:p3]+R[4:height]
PR[7,6:p3]=PR[7,6:p3]-90    ;
```

Figure 33. FANUC calculation process

As seen, now the axes are called ‘1, 2, 3, 4, 5 and 6’ referring to ‘X, Y, Z’ and its rotations ‘W, P, R’. To work like used in the code, the position registers have to be saved as Cartesian coordinates.

The process of placing is exactly the same as KUKA.



```
  CALL GRAB    ;
J PR[6:p2] 100% FINE    ;
  WAIT   1.00(sec) ;
  RO[3]=OFF ;
```

Figure 34. FANUC placing action

Inside the ‘Grab’ subroutine there are the approach and retreat positions, also the wait is to make sure it reaches the position before turning off the vacuum.

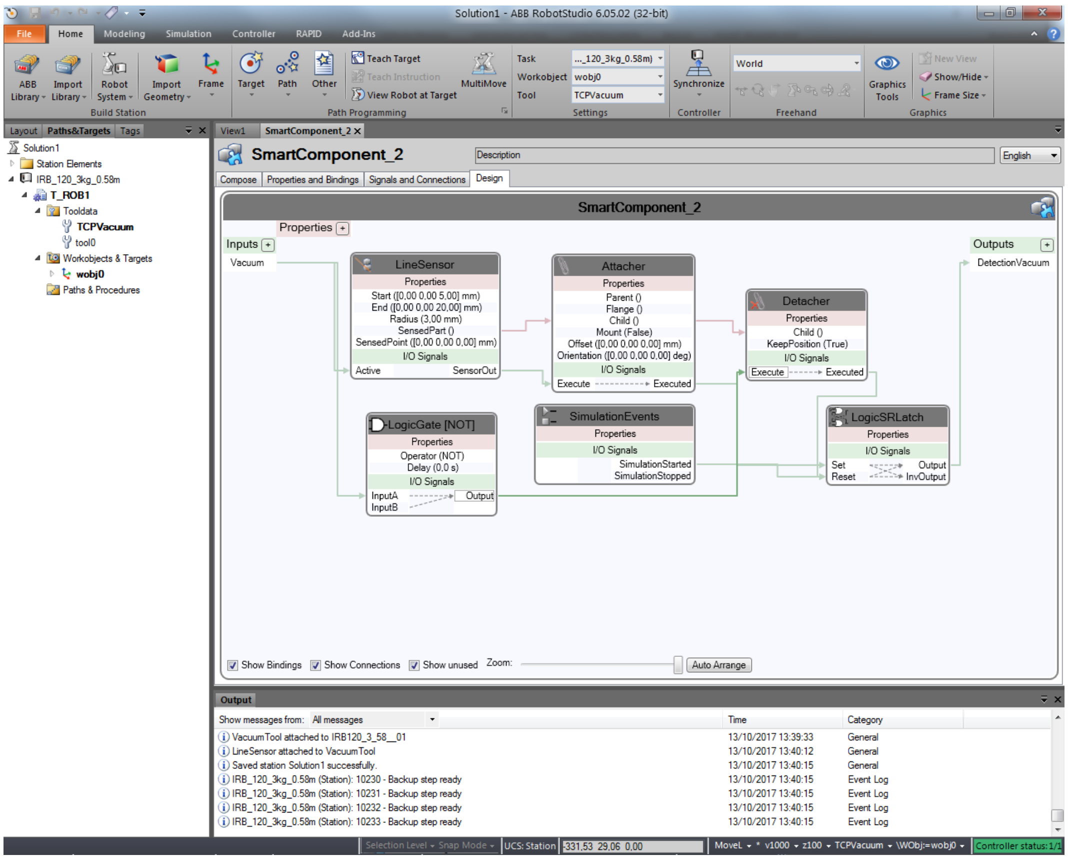
8. CONFIGURATION MANUAL

This chapter is a manual for the lecturers on how to set up correctly the robot and the environment on ABB Robotstudio and FANUC Roboguide, as in KUKA Sim Tech is not possible to link inputs or outputs.

8.1. ABB Robotstudio

To set up the layout like in the figure 24:

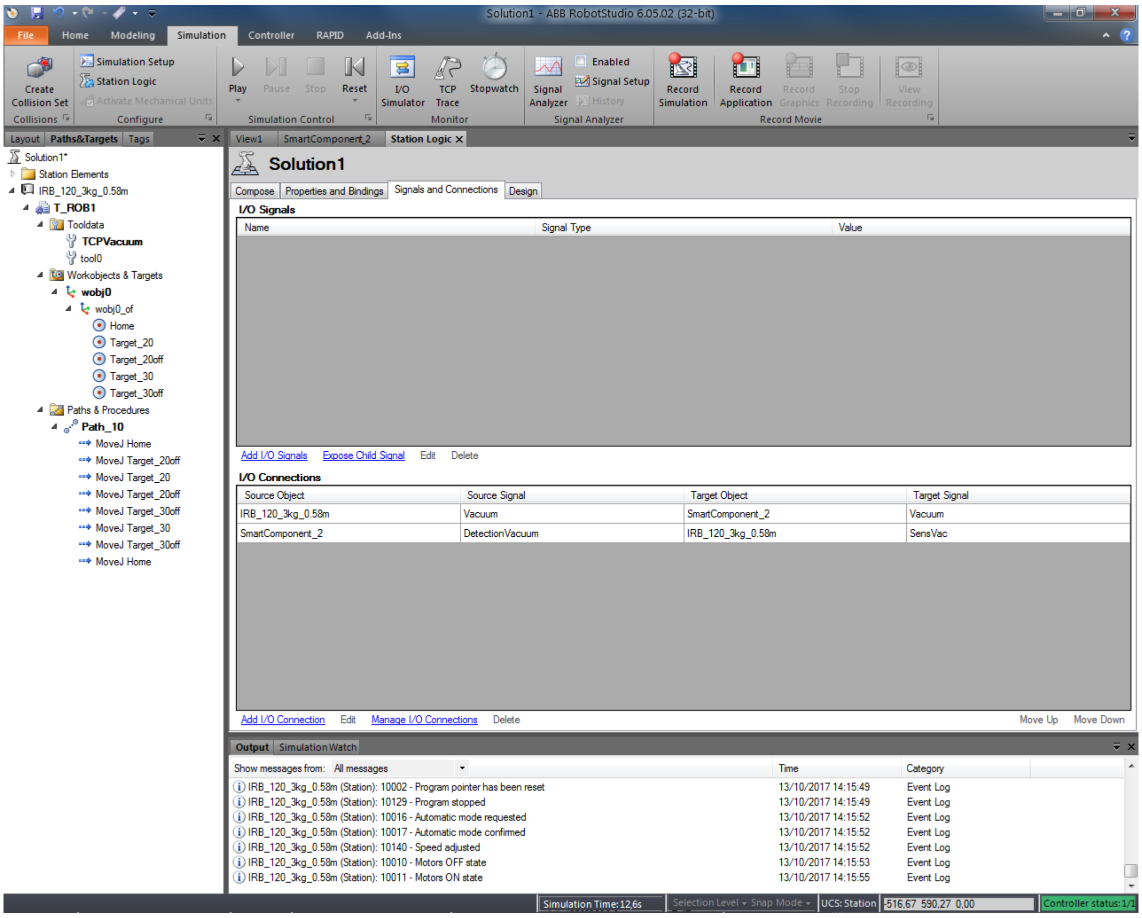
- Add the robot you want.
- Go to create Geometry and make a cylinder at coordinates {0, 0, 0}.
- Go to create a Tool and fill the requests.
- Drag the tool to the robot at the Layout tab.
- Then, in modelling tab, go to smart sensor and add component.
- Add a LineSensor, create it with enough length and small radius. Drag it to the tool at the layout Tab.
- Go to Signals and connections and then add I/O signals and add 1 D.I. and 1 D.O.
- Double click on the created SmartComponent, then add components and connect them this way.



```
Output
Show messages from: All messages                              Time                  Category
VacuumTool attached to IRB120_3_58__01                        13/10/2017 13:39:33   General
LineSensor attached to VacuumTool                             13/10/2017 13:40:12   General
Saved station Solution1 successfully.                         13/10/2017 13:40:15   General
IRB_120_3kg_0.58m (Station): 10230 - Backup step ready        13/10/2017 13:40:15   Event Log
IRB_120_3kg_0.58m (Station): 10231 - Backup step ready        13/10/2017 13:40:15   Event Log
IRB_120_3kg_0.58m (Station): 10232 - Backup step ready        13/10/2017 13:40:15   Event Log
IRB_120_3kg_0.58m (Station): 10233 - Backup step ready        13/10/2017 13:40:15   Event Log
```

Figure 35. SmartComponent connections

- Then you go to Controller, I/O system configuration, and click add signals, 1 D.I. and 1 D.O. Both with value = 1.
- Then go to Station Logic, Signals & Connections and create the next connections:



I/O Connections

Source Object	Source Signal	Target Object	Target Signal
IRB_120_3kg_0.58m	Vacuum	SmartComponent_2	Vacuum
SmartComponent_2	DetectionVacuum	IRB_120_3kg_0.58m	SensVac

```
                                                                      Time                  Category
IRB_120_3kg_0.58m (Station): 10002 - Program pointer has been reset   13/10/2017 14:15:49   Event Log
IRB_120_3kg_0.58m (Station): 10129 - Program stopped                  13/10/2017 14:15:49   Event Log
IRB_120_3kg_0.58m (Station): 10016 - Automatic mode requested         13/10/2017 14:15:52   Event Log
IRB_120_3kg_0.58m (Station): 10017 - Automatic mode confirmed         13/10/2017 14:15:52   Event Log
IRB_120_3kg_0.58m (Station): 10140 - Speed adjusted                   13/10/2017 14:15:52   Event Log
IRB_120_3kg_0.58m (Station): 10010 - Motors OFF state                 13/10/2017 14:15:53   Event Log
IRB_120_3kg_0.58m (Station): 10011 - Motors ON state                  13/10/2017 14:15:55   Event Log
```

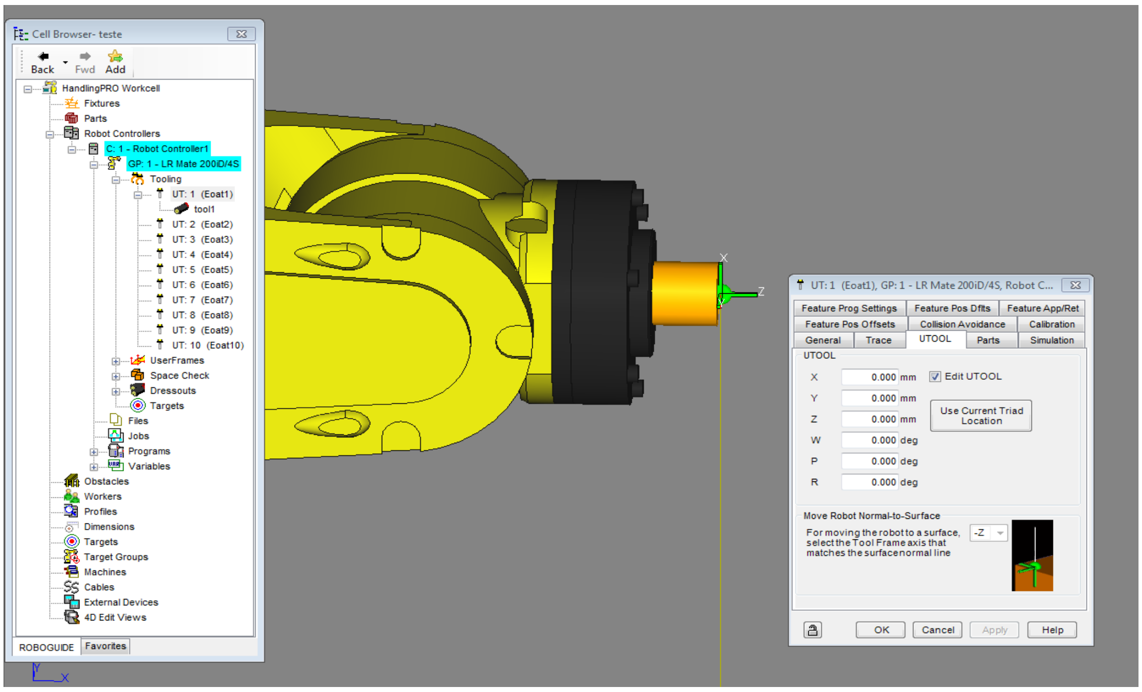
Figure 36. Station Logic I/O connections

- After that, the tool should grab and drop objects and the D.I. sensor should detect if there is an object at or near the tool.
- Now, the environment just has to be created. Place the robot, add some models for objects like tables or cubes, set up correctly the axes of every object with frame and create the targets and paths.
- Then the procedure is created and you have to synchronize with RAPID, so click synchronize to RAPID and the code generated would be in the RAPID tab.
- There you can code and go much further with the actions that the robot can do.

8.2. FANUC Roboguide

For setting up the layout like in figure 25:

- Create new cell and follow the wizard: select create robot with default configuration; select the latest version; click on handling tool H552; select the robot to use, in our case order number H754; then just hit next to everything else and finish the wizard.
- First thing is to add the tool: at the Cell Browser, open Robot controllers > robot > tooling > right-click on UT > add link. If we do not like the tools' models we can add an own model: box, cylinder, sphere or CAD file.
- Double click on the tool and set the location of it and the mass, and go to UTOOL tab, edit it and drag it to the end of the tool. Click 'use current triad location.



UTOOL

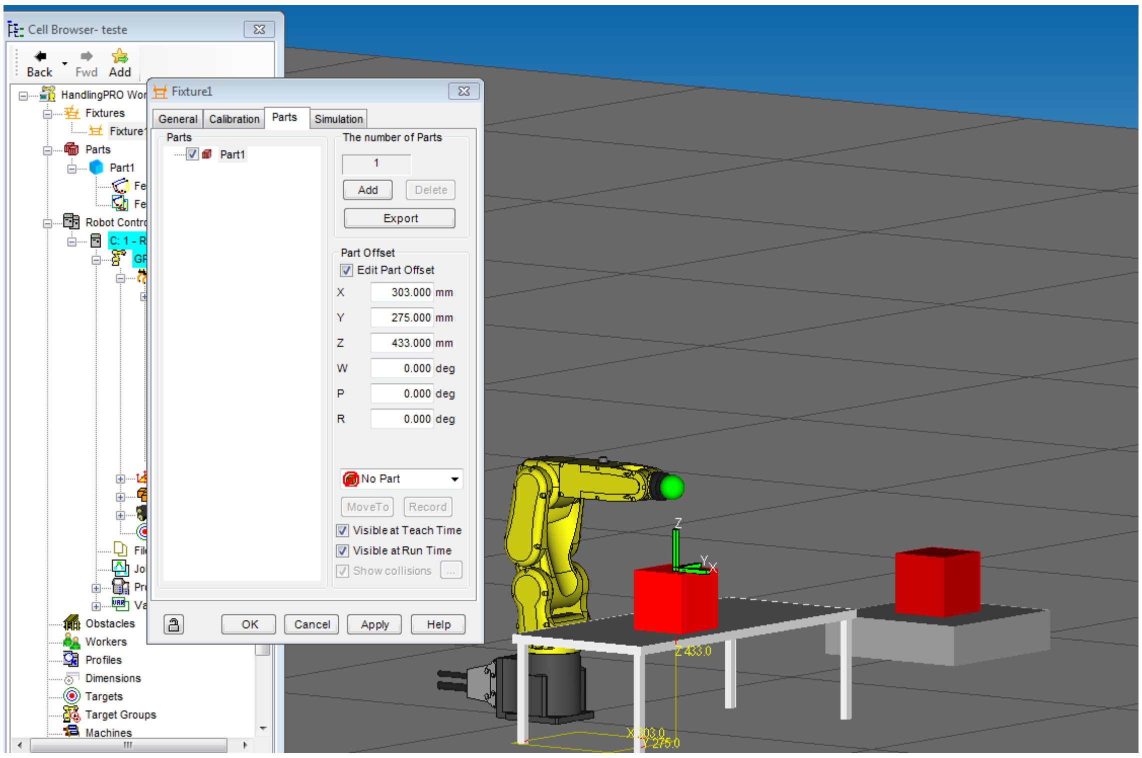
X	0.000	mm	[✓] Edit UTOOL
Y	0.000	mm	
Z	0.000	mm	
W	0.000	deg	
P	0.000	deg	
R	0.000	deg	

Move Robot Normal-to-Surface

For moving the robot to a surface, select the Tool Frame axis that matches the surface normal line: -Z

Figure 37. Utool Configuration

- Then, to create an environment, go to fixtures and create some tables for example. Fixtures do not interact with the robot, only on collision.
- When adding anything, the size and location can be changed.
- Also go to parts and create the parts you want to work with. Remember that the mass should not be greater than the payload. The part will be created on top of a Part Rack, only the rack can be moved.
- After that, go to the fixture created and double click, go to parts tab and select the part you want it to be linked. Select where you want it to be using the part offset.



Parts: ✓ Part1

The number of Parts: 1

Part Offset

✓ Edit Part Offset

X	303.000	mm
Y	275.000	mm
Z	433.000	mm
W	0.000	deg
P	0.000	deg
R	0.000	deg

No Part

✓ Visible at Teach Time

✓ Visible at Run Time

✓ Show collisions

Figure 38. Parts configuration

- Then double click on the tool and do the same in the parts tab. Go to simulation tab and at Gripper settings select Vacuum.

9. GLOSSARY

In this chapter the acronyms appeared in the project are given its meaning.

CAD	Computer Aided Design
D.I.	Digital Input
D.O.	Digital Output
I/O	Inputs & Outputs
IP	Ingress Protection
P.R.	Position Register
T.P.	Teach Pendant
U.C.L.L	University of Leuven-Limburg

10. BIBLIOGRAPHY

http://www.fanuc.eu/es/en/robots/robot-filter-page/lrmate-series/lrmate-200id-4s

http://new.abb.com/products/robotics/industrial-robots/irb-120

http://www.globalrobots.com/product.aspx?product=24933

https://en.wikipedia.org/wiki/KUKA_Robot_Language

https://en.wikipedia.org/wiki/RAPID

https://en.wikipedia.org/wiki/Karel_(programming_language)

http://robot.zaab.org/wp-content/uploads/2014/04/KRL-Reference-Guide-v4_1.pdf

https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf

http://www.aip-primeca.net/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=147&PortalId=1&TabId=150