|  |
| --- |
| Universitat Politècnica de Catalunya |
| Escola Superior d’Enginyeria Industrial, Aeroespacial i Audiovisual de Terrassa |
| Màster Universitari en Enginyeria Aeronàutica |
|  |
| The Chocolate Factory |
| |  |  | | --- | --- | | **Students:** | Pol Fontanes Molina | |  | Víctor Martínez Viol | |
|  |
| |  |  | | --- | --- | | **Course:** | Applied Robotics | | **Lab. Group:** | Group 5 | | **Professors:** | Jaume Figueras Jove | |
|  |
| |  |  | | --- | --- | | **Submission Date:** | 3rd April 2019 | |

**Table of Contents**

[1 Objectives 3](#_Toc4929693)

[2 Laboratory Setup 3](#_Toc4929694)

[3 Station Description 3](#_Toc4929695)

[4 Program Documentation 3](#_Toc4929696)

[4.1 Program Execution Strategy 4](#_Toc4929697)

[4.1.1 How the Priority is Handled? 4](#_Toc4929698)

[4.2 Flow Diagrams 5](#_Toc4929699)

[4.3 Code Description 5](#_Toc4929700)

[4.3.1 Interruptions 5](#_Toc4929701)

[4.3.2 Functions and Procedures 6](#_Toc4929702)

[4.3.3 Variables 6](#_Toc4929703)

[5 Conclusions 6](#_Toc4929704)

[6 Bibliography 6](#_Toc4929705)

**Table of Figures**

[Figure 1: Laboratory environment components. IRB140 robot arm and IRC5 controller at the left and the right respectivelly. 3](#_Toc4929706)

**Table of Tables**

[Table 1: Sample’s geometrical data and test conditions 4](#_Toc4929707)

# Objectives

# Laboratory Setup

In the Robotics and CIM laboratory (ESAII department, building TR11) there is an environment conformed by an ABB IRB140 robot arm, the robot controller (IRC5), the teach pendant[[1]](#footnote-1) and a pc connected with the robot controller by means of an Ethernet connection.



Figure 1: Laboratory environment components. IRB140 robot arm and IRC5 controller at the left and the right respectivelly.

# Station Description

The laboratory practice consists on a drying station compound of three main positions:

TBC

# Program Documentation

The following section aim is to explain not only the variables defined and the syntax of the implemented functions but also the program execution flow and how are solved the different issues that may arise when programming a robot arm that work with asynchronous inputs.

The first part of the section covers the strategy applied in order to deal with different robot tasks / movements. Then, the execution flow diagrams of the programs are shown to provide a quick understanding about how the program is structured. Finally, a brief description of the code is done.

## Program Execution Strategy

As it has been shown in section 2, there are several tasks that must be done following a sequential order. Besides, during any step of this sequence, a new chocolate mould can arrive to the input conveyor and thus altering the task sequence.

The methodology followed in this work to deal with a new task appearing while the previous sequence is not yet finished is to define a queue where the different tasks are listed and they are executed following a criterion based on two factors: (a) Time and (b) Priority.

To do so, a unique ID has been assigned to each task and their completion times and priorities (from 0 to 10) have been defined. The following table lists the aforementioned values.

|  |  |  |  |
| --- | --- | --- | --- |
| Task | ID | Completion Time (s) | Priority |
| Pick a chocolate mould from the input conveyor and bring it to the drying warehouse. | 1 | 0 | 0 |
| Take the chocolate mould from the drying warehouse and bring it to the manipulation station. | 2 | 60 \*  120 \*\* | 6 |
| Take the mould from the manipulation station and bring it to the output conveyor | 3 | 5 | 10 |
| No task to do | 0 | 0 | 0 |
| \*if chocolate type is 1, \*\*if chocolate type is 2 |  |  |  |

Table 1: Sample’s geometrical data and test conditions

All this information has to be available for the program to stablish an order of execution for the different tasks. Besides, as there are some tasks that take some time to be completed, the robot must know exactly the time at when the tasks are finished. This information is contained inside a queue matrix which is called taskQueue. This matrix stores the task ordered by descending completion time (considering also the priority) and has the following columns:

* **Task Id:** To Correctly identify the task.
* **Completion time:** Stored in seconds since 00:00[[2]](#footnote-2).
* **Optional parameter 1:** Optional parameter, used to distinguish between chocolate type.
* **Optional parameter 2:** Optional parameter, used to identify the position of the mould in the drying warehouse.
* **Priority:** Priority integer, used when ordering the tasks.

### How the Priority is Handled?

As it will be further explained in section 4.2.X, the code developed uses a for-loop to insert a new task inside the queue. It compares the completion time of the new task with the completion time of the tasks already in the queue. The strategy applied to also consider the priority in this comparison is to convert the priority into an artificial time delay to be added to the completion time of the tasks. Therefore, a task with the maximum priority will use its real completion time whereas a task with less priority will use its completion time plus an artificial increment of time when comparing it. This will guarantee that even though the latter is completed before, it will be situated after the task with higher priority in the queue.

## Flow Diagrams

## Code Description

This section is intended to contain a brief description of the functions and procedures implemented in the code.

### Interruptions

Three interruptions have been developed in this code. One (iStop) is used to deal with an emergency stop and the others (iMove1, iMove2) are used to tell the robot that a chocolate mould has arrived.

Below these lines, two code snippets can be found. The first contains the definition of the interrupt variables and their link with the interruption methods. The second, show the implementation of the interruptions aforementioned.

|  |  |
| --- | --- |
| 62  63  64  65  79  96  97  98  99  100  101  102  130 | !Interrupts  VAR intnum pushInt1;  VAR intnum pushInt2;  VAR intnum pushInt3;  ! […]  PROC main()  ! […]  ! 2. Connect interrupts  CONNECT pushInt1 WITH iMove1;  ISignalDI sensor1,1,pushInt1;  CONNECT pushInt2 WITH iMove2;  ISignalDI sensor2,1,pushInt2;  CONNECT pushInt3 WITH iStop;  ISignalDI sensor3,1,pushInt3;  ! […]  ENDPROC |

|  |  |
| --- | --- |
| 138  139  140  141  142  143  144  145  146  147  148 | TRAP iMove1  triggerSeq2 1, taskQueue, taskTimming;  ENDTRAP    TRAP iMove2  triggerSeq2 2, taskQueue, taskTimming;  ENDTRAP    TRAP iStop  emergencyStop;  ENDTRAP |

The interruptions are connected to the digital signals sensor1, sensor2 and sensor3. These signals have been linked to the robot teach pendant programmable keys[[3]](#footnote-3) 1 to 3 shown in the Figure 2.

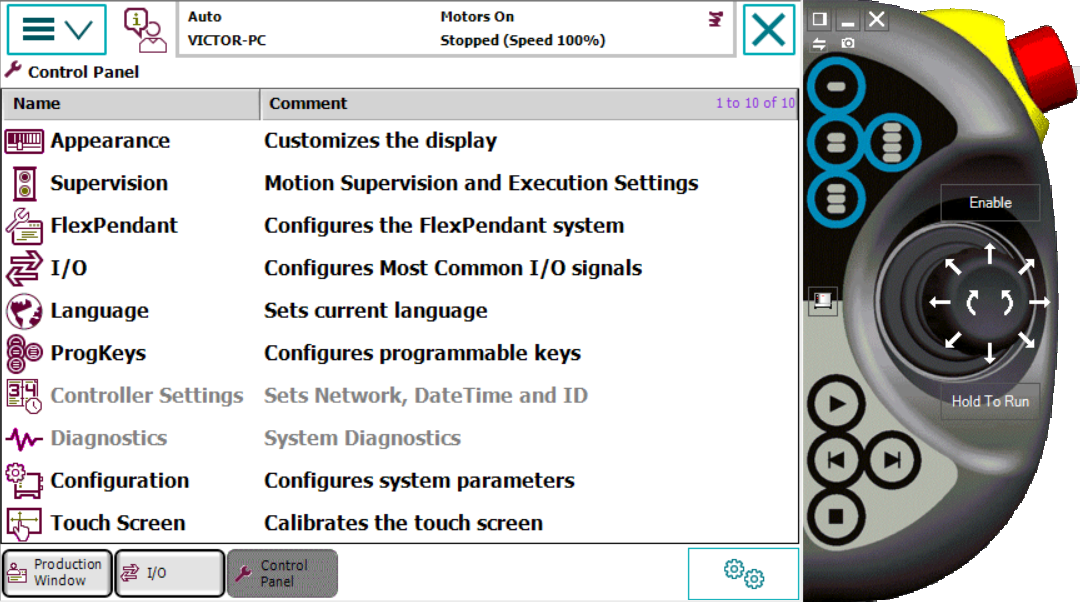


Figure 2: Teach Pendant.

### Functions and Procedures

#### checkPos

|  |  |
| --- | --- |
| **Syntax:** | checkPos() |
| **Description:** | Reads the current position of the arm (using the built-in function CPos() and computes the difference between this position and the reference position of the four key positions of the station. It returns a num code if the difference is smaller than a given threshold.  **Return:** 0: Home; 1: Conveyor; 2: Drying warehouse; 3: manipulation station; 404: The position does not correspond any of the above. |

#### defineOvenPoints

|  |  |
| --- | --- |
| **Syntax:** | defineOvenPts(orig, [x, y, z], [xx, yy, zz], pts) |
| **Description:** | Construct the drying warehouse matrix by taking a reference point called orig, an offset between warehouse positions [x, y, z] and a security offset between the security points and the warehouse points [xx, yy, zz]. This matrix is then stored in a 2x3x3 matrix called pts. |

#### defineConvPoints

|  |  |
| --- | --- |
| **Syntax:** | defineConvPts(orig, [x, y, z], [xx, yy, zz], pts) |
| **Description:** | Construct the input and output conveyors position by taking a reference point called orig, an offset between the two conveyors [x, y, z] and a security offset between the security points and the conveyors [xx, yy, zz]. This matrix is then stored in a 2x2 matrix called pts. |

#### conv2oven

|  |  |
| --- | --- |
| **Syntax:** | conv2oven(p, q, k, i, j) |
| **Description:** | This function is responsible for the movement between the conveyor number k at position p (2x2 matrix) and the position i,j of the drying warehouse whose points are stored in q (2x3x3 matrix). |

#### oven2man

|  |  |
| --- | --- |
| **Syntax:** | Oven2man(p, q, i, j) |
| **Description:** | This function is responsible for the movement between the position i,j of the drying warehouse whose points are stored in q (2x3x3 matrix) and the manipulation station q (2x1 matrix). |

#### man2conv

|  |  |
| --- | --- |
| **Syntax:** | man2conv(p, q, i) |
| **Description:** | This function is responsible for the movement between the manipulation station p (2x1 matrix) and the conveyor number i at position q (2x2 matrix). |

#### triggerSeq2

|  |  |
| --- | --- |
| **Syntax:** | triggerSeq2(type, queue, time) |
| **Description:** | Creates a task of type type and adds it to the queue of tasks queue. It also triggers updateDisp |

#### updateDisp

|  |  |
| --- | --- |
| **Syntax:** | updateDisp(n) |
| **Description:** | Clears the teach pendant display and prints the chocolate counters identified by the matrix n  “CHOCOLATE TYPE 1:  Produced = x  Total = x  CHOCOLATE TYPE 2:  Produced = y  Total = Y” |

#### emergencyStop

|  |  |
| --- | --- |
| **Syntax:** | emergencyStop() |
| **Description:** | Sends a break message to the program thread and prints a message in the teach pendant display. |

#### performTask

|  |  |
| --- | --- |
| **Syntax:** | performTask(queue, occOven, time, n) |
| **Description:** | This function executes the first task on the task queue queue, creates a new task and updates the queue. It uses the drying warehouse occupancy stored at occOven, the task timings time and the chocolate counters n. |

#### moveS

|  |  |
| --- | --- |
| **Syntax:** | moveS(p, vFree, vSec, z, t) |
| **Description:** | For a pair of points (security point and actual point) defined as p, this function performs the following movements with zonedata z and tooldata t:   1. MoveJ to the security point with velocity vFree. 2. MoveJ to the actual point with velocity vSec. 3. MoveJ to the security point with velocity vSec. |

#### movTest

|  |  |
| --- | --- |
| **Syntax:** | movTest(p, q, r) |
| **Description:** | This function executes different movements sequences between the Home position and the positions stored at the matrices p, q and r. |

#### gotoOvenPts

|  |  |
| --- | --- |
| **Syntax:** | gotoOvenPts(p) |
| **Description:** | Move the robot sequentially between the positions defined in the matrix p. |

### Variables

There are different variables that need an

# Conclusions

# Bibliography

Per si necessitem snippets del codi

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523 | MODULE ObelixMov  !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  !  ! Module: ObelixMov  !  ! Description:  ! Move the robot  !  ! Author: pol & victor  !  ! Version: 0.1.0  !  !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    CONST robtarget pHome :=[[507.9,-6.43,715.02],[0.697685,-0.00154316,0.716328,-0.0103435],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];      CONST robtarget pConvRef :=[[-109.516885301,-504.792720334,376.620999376],[0.243054291,0.717735721,0.578717345,-0.301440343],[-2,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];      CONST robtarget pManRef :=[[607.9,-6.43,695.02],[0.697685,-0.00154316,0.716328,-0.0103435],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];  !CONST robtarget pManRef :=[[507.9,-6.43,715.02],[0.697685,-0.00154316,0.716328,-0.0103435],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];  CONST robtarget pOvenRef :=[[-291.029880742,659.908842444,594.297264732],[0.363486279,-0.625201176,0.407577419,0.55756781],[1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];    CONST num convSecOffset{3} := [0, 0, -100]; !security offset [x, y, z]  CONST num manSecOffset{3} := [50, 0, -50]; !security offset [x, y, z]  CONST num ovenSecOffset{3} := [0, 100, 0]; !security offset [x, y, z]    CONST num convOffset{3} := [0, -200, 0]; !offset between 2 conv [x, y, z]  CONST num ovenMatOffset{3} := [100, 0, -100]; !offset between oven positions [x, y, z]    CONST speeddata vSecurity := v1000;    ! Task variables  ! +---------+------------------------------------------------+  ! | Task ID | Task Description |  ! +---------+------------------------------------------------+  ! | 1 | pick choco from conveyor1 & bring to oven |  ! | 2 | take choco from oven & bring to man. station |  ! | 3 | take the mould from ms & throw it to conveyor2 |  ! | 0 | no task |  ! +---------+------------------------------------------------+    VAR num taskTimming{4} := [30, 30, 0, 5]; ! time in seconds to trigger next task  VAR num taskPrior{4} := [0, 6, 10, 0];  VAR num timeDelta := 0; !delta between currTime and next Task  VAR num priorDel := 20; !priority artificial time  VAR num timeMov := 30; !time elapsed during robot movements    VAR num taskQueue{30,5}; ![Task id, completion time, opt\_par1(chocType), opt\_par2, priority]    VAR num currTime; ! var to store current time    VAR bool occOven{9}; ! idx computed as (i-1)\*3+j  VAR bool isHome := TRUE; ! flag to know if the robot is at pHome    ! Points variables  VAR robtarget pConv{2,2};  VAR robtarget pOven{2,3,3};  VAR robtarget pMan{2};    ! Flexpendant vars  VAR num numChoc{2,2}; !{1,\*} num produced; {2,\*} Total to produce  VAR num chocType;    !Interrupts  VAR intnum pushInt1;  VAR intnum pushInt2;  VAR intnum pushInt3;  !Check position  VAR pos pos1;  VAR num place;      !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  !  ! Procedure main  !  ! This is the entry point of your program  !  !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC main()  MoveJ pHome, v1000, fine, tool0;  !-----------------------------------    ! 0. Points definition  pMan := [pManRef, Offs(pManRef, manSecOffset{1}, manSecOffset{2}, manSecOffset{3})];  defineConvPts pConvRef, convOffset, convSecOffset, pConv;  defineOvenPts pOvenRef, ovenMatOffset, ovenSecOffset, pOven;    ! 1. Job Configuration  TPErase;  TPWrite "Welcome to the chocolate factory";  TPReadNum numChoc{2,1}, "How many chocolate 1 items will be produced?";  TPReadNum numChoc{2,2}, "How many chocolate 2 items will be produced?";  updateDisp numChoc;      ! 2. Connect interrupts  CONNECT pushInt1 WITH iMove1;  ISignalDI sensor1,1,pushInt1;  CONNECT pushInt2 WITH iMove2;  ISignalDI sensor2,1,pushInt2;  CONNECT pushInt3 WITH iStop;  ISignalDI sensor3,1,pushInt3;    ! 3. Reescale the task priority  FOR i FROM 1 TO Dim(taskPrior,1) DO  taskPrior{i} := (taskPrior{i} - 0) \* (0 - 10) / (10 - 0) + 10;  ENDFOR    ! 4. Start the job  !while produced < total  WHILE numChoc{1,1}<numChoc{2,1} OR numChoc{1,2}<numChoc{2,2} DO  !get current time  currTime := GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec);    !do some movement  IF taskQueue{1,1} <> 0 AND taskQueue{1,2} - currTime < timeDelta THEN  performTask taskQueue, occOven, taskTimming, numChoc, pConv, pOven, pMan;  isHome := FALSE;  ELSEIF (NOT isHome) AND (taskQueue{1,1} = 0 OR taskQueue{1,2} - currTime > timeMov) THEN  MoveJ pHome, v1000, fine, tool0;  isHome := TRUE;  ENDIF  ENDWHILE    ! TEST. Movement Tests  !movTest pConv, pOven, pMan;    !-----------------------------------  MoveJ pHome, v1000, fine, tool0;  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ! @deprecated  TRAP iMove  triggerSeq chocType, taskQueue, taskTimming;  ENDTRAP    TRAP iMove1  triggerSeq2 1, taskQueue, taskTimming;  ENDTRAP    TRAP iMove2  triggerSeq2 2, taskQueue, taskTimming;  ENDTRAP    TRAP iStop  emergencyStop;  ENDTRAP  !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  FUNC num checkPos()  VAR pos check\_diff;  VAR num diff;  VAR num threshold := 250;  !0-> home 1->conveyor 2->oven 3->man  !check home  pos1 := CPos();    check\_diff.x := pos1.x - pHome.trans.x;  check\_diff.y := pos1.y - pHome.trans.y;  check\_diff.z := pos1.z - pHome.trans.z;    diff := VectMagn(check\_diff);  IF diff <= threshold THEN  RETURN 0;  ENDIF  !check conveyor  pos1 := CPos();    check\_diff.x := pos1.x - pConvRef.trans.x;  check\_diff.y := pos1.y - pConvRef.trans.y;  check\_diff.z := pos1.z - pConvRef.trans.z;    diff := VectMagn(check\_diff);  IF diff <= threshold THEN  RETURN 1;  ENDIF    !check oven  pos1 := CPos();    check\_diff.x := pos1.x - pOvenRef.trans.x;  check\_diff.y := pos1.y - pOvenRef.trans.y;  check\_diff.z := pos1.z - pOvenRef.trans.z;    diff := VectMagn(check\_diff);  IF diff <= threshold THEN  RETURN 2;  ENDIF    !check man  pos1 := CPos();    check\_diff.x := pos1.x - pManRef.trans.x;  check\_diff.y := pos1.y - pManRef.trans.y;  check\_diff.z := pos1.z - pManRef.trans.z;    diff := VectMagn(check\_diff);  IF diff <= threshold THEN  RETURN 3;  ENDIF            !DEFAULT          RETURN 404;    ENDFUNC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC defineOvenPts(robtarget orig, num matOffs{\*}, num secOffs{\*}, INOUT robtarget pts{\*,\*,\*})    !define the security points {1} and the inside oven points {2}  FOR i FROM 1 TO 3 DO  FOR j FROM 1 TO 3 DO  pts{1,i,j} := Offs(orig, matOffs{1}\*(i-1), matOffs{2}, matOffs{3}\*(j-1));  pts{2,i,j} := Offs(pts{1,i,j}, secOffs{1}, secOffs{2}, secOffs{3});  ENDFOR  ENDFOR  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC defineConvPts(robtarget orig, num convOffs{\*}, num secOffs{\*}, INOUT robtarget pts{\*,\*})    !define the security points {1} and the conv points {2}  FOR i FROM 1 TO 2 DO  pts{1,i} := Offs(orig, convOffs{1}\*(i-1), convOffs{2}\*(i-1), convOffs{3}\*(i-1));  pts{2,i} := Offs(pts{1,i}, secOffs{1}, secOffs{2}, secOffs{3});  ENDFOR  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC gotoOvenPts(robtarget pts{\*,\*,\*})  FOR i FROM 1 TO 3 DO  FOR j FROM 1 TO 3 DO  MoveS [pOven{1, i, j}, pOven{2, i, j}], v1000, v200, fine, tool0;  ENDFOR  ENDFOR  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC conv2oven(robtarget pConv{\*,\*}, robtarget pOven{\*,\*,\*}, num iConv, num iOven, num jOven)  !Check position  place := checkPos();  IF place = 2 OR place = 404 THEN  MoveJ pHome, v1000, fine, tool0;  ENDIF    MoveS [pConv{1, iConv}, pConv{2, iConv}], v1000, vSecurity, fine, tool0;    MoveJ pHome, v1000, fine, tool0;    MoveS [pOven{1, iOven, jOven}, pOven{2, iOven, jOven}], v1000, vSecurity, fine, tool0;    ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC oven2man(robtarget pOven{\*,\*,\*}, robtarget pMan{\*}, num iOven, num jOven)  !Check position  place := checkPos();  IF place = 1 OR place = 404 THEN  MoveJ pHome, v1000, fine, tool0;  ENDIF    MoveS [pOven{1, iOven, jOven}, pOven{2, iOven, jOven}], v1000, vSecurity, fine, tool0;    MoveJ pHome, v1000, fine, tool0;    MoveS pMan, v1000, vSecurity, fine, tool0;    ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC man2conv(robtarget pMan{\*}, robtarget pConv{\*,\*}, num iConv)    MoveS pMan, v1000, vSecurity, fine, tool0;    MoveJ pHome, v1000, fine, tool0;    MoveS [pConv{1, iConv}, pConv{2, iConv}], v1000, vSecurity, fine, tool0;    ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ! @deprecated  PROC triggerSeq(INOUT num type, INOUT num queue{\*,\*}, num time{\*})    VAR num newTask{5};  VAR num auxTask{5};    VAR num currTime;  currTime := GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec);    TPReadFK type, "A chocolate figure has arrived to the station. Which type of chocolate is?", "TP1", "TP2", stEmpty,stEmpty,stEmpty;    !Add a task 1 to the queue  currTime := GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec);  newTask := [1, currTime, type, 0, taskPrior{1}];  FOR i FROM 1 TO Dim(queue, 1) DO  IF newTask{2} + priorDel \* newTask{5} + timeMov < queue{i,2} + priorDel \* queue{i,5} OR queue{i,1} = 0 THEN  !backup newTask  auxTask := newTask;    newTask{1} := queue{i,1};  newTask{2} := queue{i,2};  newTask{3} := queue{i,3};  newTask{4} := queue{i,4};  newTask{5} := queue{i,5};    queue{i,1} := auxTask{1};  queue{i,2} := auxTask{2};  queue{i,3} := auxTask{3};  queue{i,4} := auxTask{4};  queue{i,5} := auxTask{5};  ENDIF  ENDFOR  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC triggerSeq2(num type, INOUT num queue{\*,\*}, num time{\*})    VAR num newTask{5};  VAR num auxTask{5};    VAR num currTime;    updateDisp numChoc;    TPWrite "A chocolate figure TYPE" \Num:=type;  TPWrite "has arrived to the station";    !Add a task 1 to the queue  currTime := GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec);  newTask := [1, currTime, type, 0, taskPrior{1}];  FOR i FROM 1 TO Dim(queue, 1) DO  IF newTask{2} + priorDel \* newTask{5} + timeMov < queue{i,2} + priorDel \* queue{i,5} OR queue{i,1} = 0 THEN  !backup newTask  auxTask := newTask;    newTask{1} := queue{i,1};  newTask{2} := queue{i,2};  newTask{3} := queue{i,3};  newTask{4} := queue{i,4};  newTask{5} := queue{i,5};    queue{i,1} := auxTask{1};  queue{i,2} := auxTask{2};  queue{i,3} := auxTask{3};  queue{i,4} := auxTask{4};  queue{i,5} := auxTask{5};  ENDIF  ENDFOR  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC updateDisp(num n{\*,\*})  TPErase;    TPWrite "CHOCOLATE TYPE 1:";  TPWrite " Produced = ", \Num := n{1,1};  TPWrite " Total = ", \Num := n{2,1};  TPWrite "CHOCOLATE TYPE 2:";  TPWrite " Produced = ", \Num := n{1,2};  TPWrite " Total = ", \Num := n{2,2};  !TPWrite " Position = ", \Pos := pos1;  !TPWrite " Position according to us = ", \Num := place;  ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC emergencyStop()    TPWrite "EMERGENCY BUTTON PRESSED";  Break;    ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC MoveS(robtarget p{\*}, speeddata vFree, speeddata vSec, zonedata z, PERS tooldata t)    MoveJ p{1}, vFree, z, t;  MoveJ p{2}, vSec, z, t;  MoveJ p{1}, vSec, z, t;    ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC performTask(INOUT num queue{\*,\*}, INOUT bool occOven{\*}, num time{\*}, INOUT num n{\*,\*},  robtarget pConv{\*,\*}, robtarget pOven{\*,\*,\*}, robtarget pMan{\*})    VAR num iOven;  VAR num jOven;  VAR bool found := FALSE;  VAR num newTask{5}; ![taskID, time, opt1, opt2, priority]    VAR num currTime;    !Switch TaskID  TEST queue{1,1}  CASE 1:  ! CHOCO HAS ARRIVED  !check the 1st empty position on the oven  FOR i FROM 1 TO 9 DO  IF (NOT occOven{i}) AND (NOT found) THEN  found := TRUE;    iOven := i DIV 3;  jOven := i MOD 3;  occOven{i} := TRUE;  !hack: avoid 0s  IF iOven = 0 iOven := 1;  IF jOven = 0 jOven := 3;  !Break; there's no break :(  ENDIF  ENDFOR    conv2oven pConv, pOven, 1, iOven, jOven;  !generate a new task  currTime := GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec);  newTask := [2, currTime+time{queue{1,1}+(queue{1,3}-1)}, queue{1,3}, 3\*(iOven-1)+jOven, taskPrior{queue{1,1}}];    CASE 2:  !PICK FROM OVEN & BRING TO MAN  !get the oven position  iOven := queue{1,4} DIV 3;  jOven := queue{1,4} MOD 3;  !hack: avoid 0s  IF iOven = 0 iOven := 1;  IF jOven = 0 jOven := 3;    occOven{queue{1,4}} := FALSE;    !perform the task  oven2man pOven, pMan, iOven, jOven;  !generate a new task  currTime := GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec);  newTask := [3, currTime+time{queue{1,1}+1}, queue{1,3}, 0, taskPrior{queue{1,1}}];    CASE 3:  !TAKE MOULD AND BRING TO CONVEYOR  !perform the task  man2conv pMan, pConv, 2;  !generate a new task  newTask := [0, 0, 0, 0, taskPrior{queue{1,1}}];    !Update chocolate counters  IF queue{1,3} = 1 THEN  Incr n{1,1};  ELSE  Incr n{1,2};  ENDIF    !Erase the contents of the display and print the numbers of figures completed  updateDisp n;    DEFAULT:  !do nothing and exit the proc  RETURN;  ENDTEST    !update the queue list comparing the completion times          FOR i FROM 2 TO Dim(queue, 1) DO              IF newTask{1} = 0 OR (queue{i,1} <> 0 AND queue{i,2} + priorDel \* queue{i,5} < newTask{2} + priorDel \* newTask{5} + timeMov) THEN                  queue{i-1,1} := queue{i,1};                  queue{i-1,2} := queue{i,2};                  queue{i-1,3} := queue{i,3};  queue{i-1,4} := queue{i,4};  queue{i-1,5} := queue{i,5};              ELSE                  !newTask completes before the queued task                  queue{i-1,1} := newTask{1};                  queue{i-1,2} := newTask{2};                  queue{i-1,3} := newTask{3};  queue{i-1,4} := newTask{4};  queue{i-1,5} := newTask{5};                  RETURN;!Break; There's no break :(              ENDIF          ENDFOR    ENDPROC    !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PROC movTest(robtarget pConv{\*,\*}, robtarget pOven{\*,\*,\*}, robtarget pMan{\*})  ! Perform diferent movement tests  !oven2oven  gotoOvenPts pOven;    !conv2oven  FOR i FROM 1 TO 3 DO  FOR j FROM 1 TO 3 DO  conv2oven pConv, pOven, 1, i, j;  ENDFOR  ENDFOR    !oven2man  FOR i FROM 1 TO 3 DO  FOR j FROM 1 TO 3 DO  oven2man pOven, pMan, i, j;  ENDFOR  ENDFOR    !man2conv  man2conv pMan, pConv, 2;    !oven2home  FOR i FROM 1 TO 3 DO  FOR j FROM 1 TO 3 DO  MoveJ pOven{1, i, j}, v1000, fine, tool0;  MoveJ pHome, v1000, fine, tool0;  ENDFOR  ENDFOR    !conv2home  FOR i FROM 1 TO 2 DO  MoveJ pConv{1, i}, v1000, fine, tool0;  MoveJ pHome, v1000, fine, tool0;  ENDFOR    !man2home  MoveJ pMan{1}, v1000, fine, tool0;  MoveJ pHome, v1000, fine, tool0;  ENDPROC    ENDMODULE |

1. A peripheral device used to control an industrial robot remotely. It is an HMI interface that can be used not only to configure and control the robot but also to program and design new capabilities and features. Figure 2 shows a virtual simulation of the ABB teach pendant. [↑](#footnote-ref-1)
2. Using GetTime(\Hour)\*3600 + GetTime(\Min)\*60 + GetTime(\Sec); to compute the time [↑](#footnote-ref-2)
3. To configure a programmable key as a digital variable it has to be set in *Control Panel/ProgKeys* option. [↑](#footnote-ref-3)