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

# Table of Figures



# Table of Tables

# 1 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] 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.*

# 2 Station Description

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

## 2.1 Drying Station

Consisting of a drying oven which has 9 slots for drying the chocolate moulds in a configuration of 3 by 3.

## 2.2 Conveyors

Integrated by two conveyors, one from where the chocolate moulds arrive and the other where the emptied chocolate moulds are returned.

## 2.3 Manipulation Area

A manipulation stations where chocolates are removed from its mould.

# 3 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

[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 5 shows a virtual simulation of the ABB teach pendant.

provide a quick understanding about how the program is structured. Finally, a brief description of the code is done.

## 3.1 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].
- **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.

[2] Using `GetTime(\Hour)*3600 + GetTime(\Min)*60 + GetTime(\Sec);` to compute the time

### 3.1.1 How the Priority is Handled?

As it will be further explained in section 4.2., 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.

## 3.2 Flow Diagrams

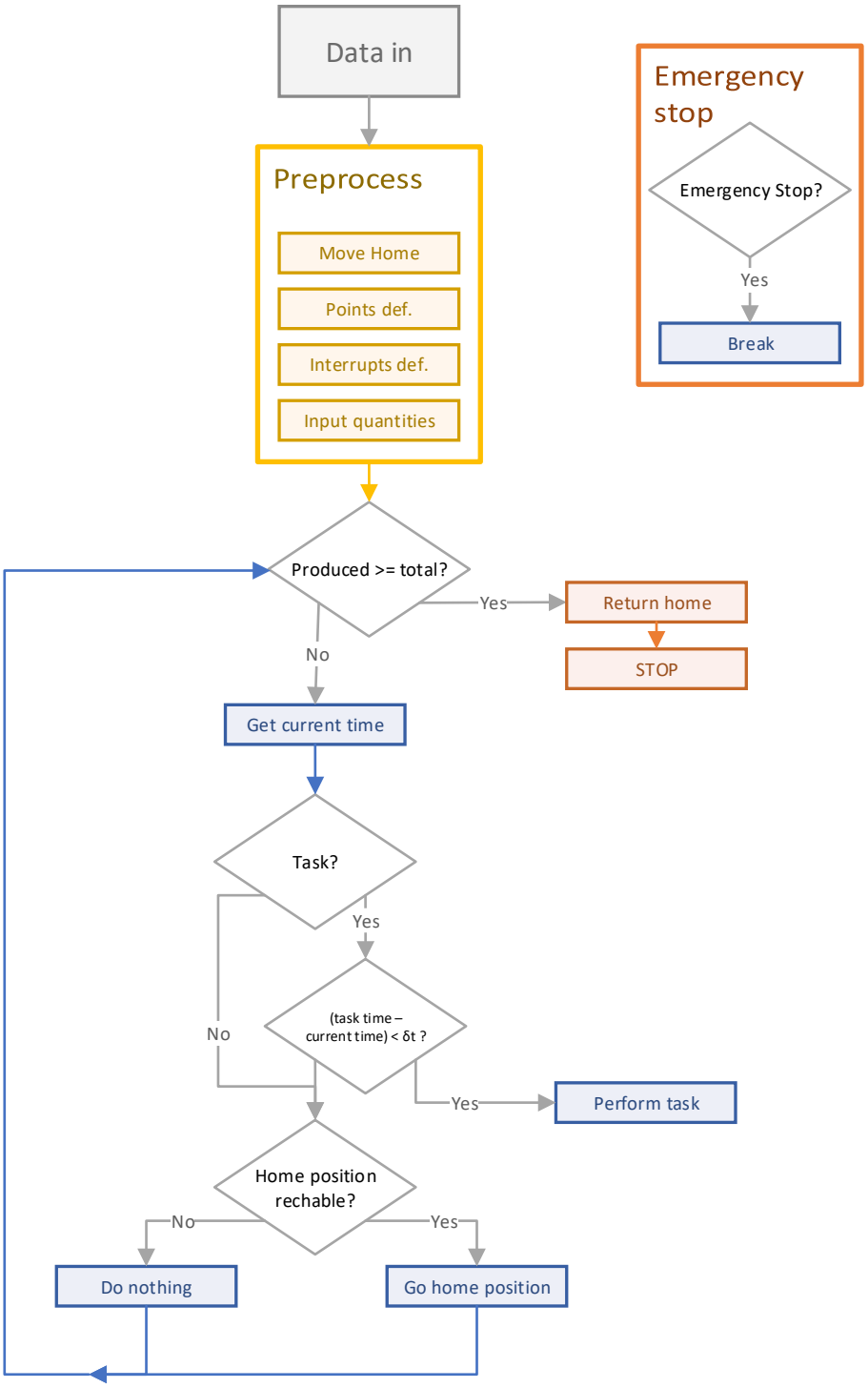


*Figure 2. Main program flow diagram*

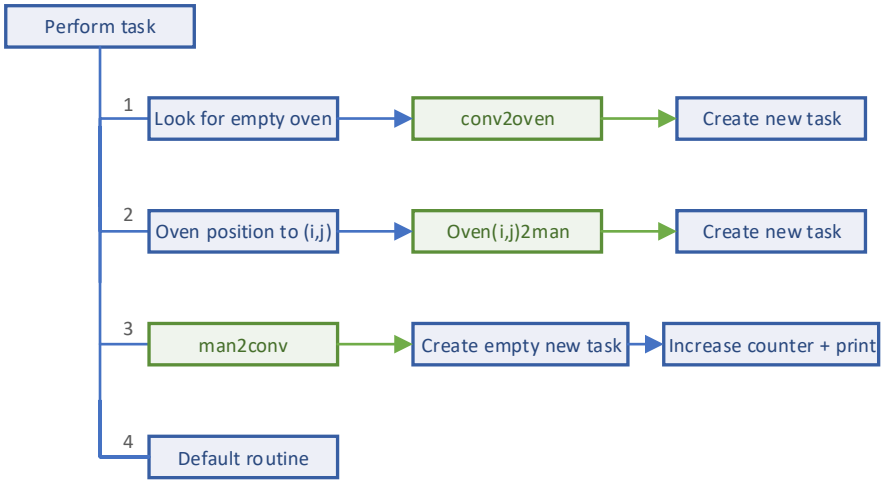


*Figure 3. Perform task switch diagram*

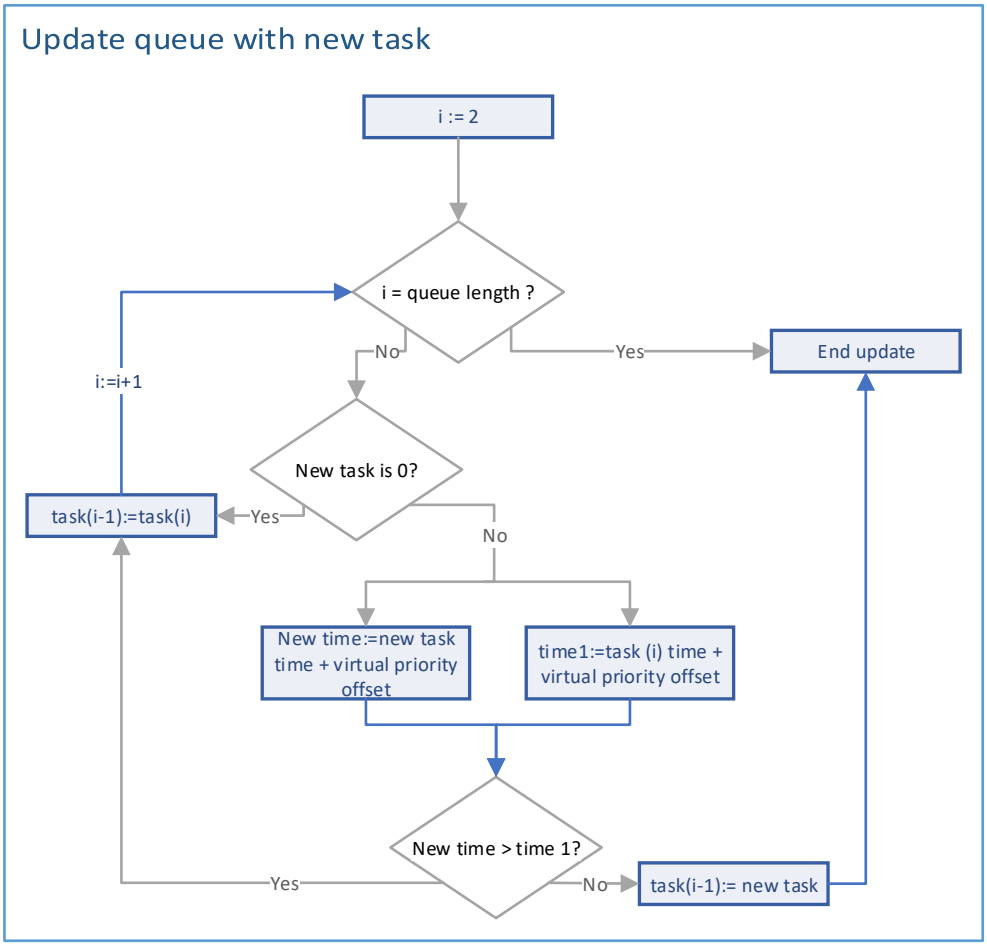


*Figure 4. Tasks queue diagram*

## 3.3 Code Description

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

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

```
138     TRAP iMove1
139         triggerSeq2 1, taskQueue, taskTimming;
140     ENDTRAP
141
142     TRAP iMove2
143         triggerSeq2 2, taskQueue, taskTimming;
144     ENDTRAP
145
146     TRAP iStop
147         emergencyStop;
148     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] 1 to 3 shown in the Figure 5.

[3] To configure a programmable key as a digital variable it has to be set in *Control Panel/ProgKeys* option.

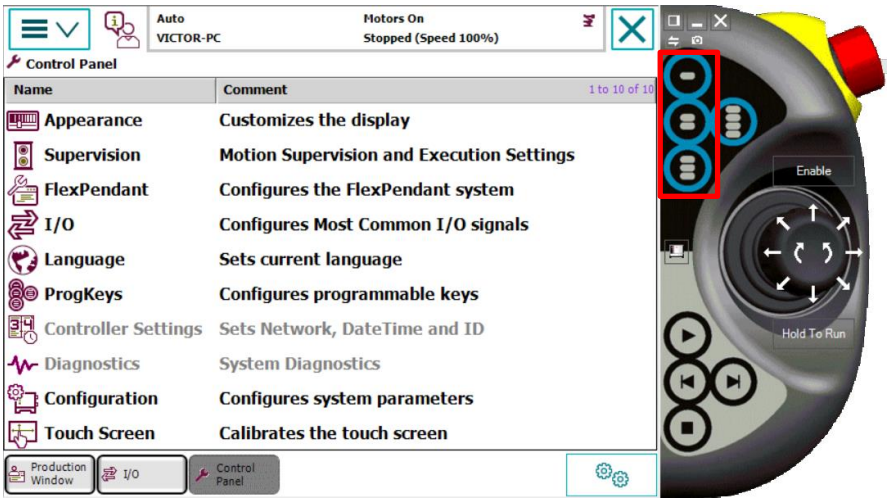


*Figure 5: Teach Pendant.*

### 3.3.2 Functions and Procedures

#### *3.3.2.1 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.

#### *3.3.2.2 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`.

#### *3.3.2.3 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`.

#### *3.3.2.4 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).

#### *3.3.2.5 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).

#### *3.3.2.6 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).

#### *3.3.2.7 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`

#### *3.3.2.8 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"
```

#### *3.3.2.9 emergencyStop*

**Syntax:** `emergencyStop()`

**Description:** Sends a break message to the program thread and prints a message in the teach pendant display.

#### *3.3.2.10 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`.

#### *3.3.2.11 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`.

#### *3.3.2.12 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`.

#### *3.3.2.13 gotoOvenPts*

**Syntax:** `gotoOvenPts(p)`

**Description:** Move the robot sequentially between the positions defined in the matrix `p`.

### 3.3.3 Variables

There are different global variables used on the program, listed below:

| Type | Variable | Function |
|---|---|---|
| robtarget | pHome | Home reference coordinates |
| robtarget | pConvRef | First conveyor reference coordinates |
| robtarget | pManRef | Manipulation station reference coordinates |
| robtarget | pOvenRef | First oven slot reference coordinates |
| num | convSecOffset{3} | Conveyor security offset |
| num | manSecOffset{3} | Manipulation station security offset |
| num | ovenSecOffset{3} | First oven slot security offset |
| num | convOffset{3} | Offset between the two conveyors |
| num | ovenMatOffset{3} | Offset between oven positions |
| speeddata | vSecurity | Security movement velocity |
| num | taskTimming{4} | Time in seconds to trigger next task |
| num | timeDelta | Time difference between currTime and next Task |
| num | timeMov | Time elapsed during robot movements |
| num | currTime | Current execution time |
| bool | occOven{9} | Oven slots ids computed as (i-1) *3+1 |
| bool | isHome | Flag variable to know if arm is at home position |
| robtarget | pConv{2,2} | Vector of coordinates for the conveyor (offset and final position) |
| robtarget | pOven{2,3,3} | Vector of coordinates for the oven slots (offset and final position) |
| robtarget | pMan{2} | Vector of coord. for the manipulation area (offset and final position) |
| num | numChoc{2,2} | Vector of produced and to produce chocolate chips |
| num | chocType | Chocolate type identifier |
| intnum | pushInt1 | Interrupt variable 1 |
| intnum | pushInt2 | Interrupt variable 2 |
| intnum | pushInt3 | Interrupt variable 3 |
| pos | pos1 | Actual position variable |
| num | place | Place id |

*Table 2. Global variables*