Apéndice A

Cableado de las E/S digitales del robot

En este apéndice se incluye el cableado físico existente entre el robot FANUC y la isla de E/S digitales «Advantys» en el momento de publicar esta memoria. Para más información sobre cómo funcionan las entradas descritas, y la forma de conexión, véase la bibliografía contenida en: [9] y [11].

Además de una serie de E/S binarias de propósito general, se va a hacer uso de una serie de E/S de proceso y de seguridad con el fin de facilitar cierta comunicación entre el robot y el autómata encargado de la coordinación de actividades de la célula. Concretamente:

Salidas robot. Además de 8 DO, se señalizarán hacia el coordinador los siguientes estados:

- Parada de emergencia (provocada por las setas de emergencia, Jaula abierta)
- Estado de fallo (independientemente de su causa)
- Indicación de modo de operación local o remoto (estado de la llave del SOP)
- Indicación de robot preparado (encendido y con la secuencia de arranque finalizada)
- Indicación de programa en ejecución
- Indicación de programa temporalmente detenido
- Reconocimiento de la señal de ejecución remota de un programa asociado a RSR1 (únicamente ACK1)

ENTRADAS ADVANTYS																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SALIDAS ROBOT	DO1	DO2	DO3	DO4	DO5	DO6	DO7	DO8	EO1	EO3	FLT	CMDENBL	PROGRUN	HELD	ACK1	SYSRDY
PIN CRM2B FANUC	33	34	35	36	38	39	40	41								
PIN CRM2A FANUC											39	33	35	38	45	34
PANEL IOBPS									5	7						

Entradas robot. Además de 9 DI, se utilizarán 7 señales para permitir al coordinador cierto manejo remoto del robot. Concretamente:

- Detención temporal (y su correspondiente reanudación) de una aplicación en ejecución.
- Detención definitiva de una aplicación en ejecución.
- El lanzamiento de una aplicación preestablecida vía RSR1.
- El lanzamiento de cualquier aplicación seleccionada previamente desde el Teach Pendant.
- El reseteo del estado de fallo.
- La propia habilitación del manejo remoto (requiere además que la llave correspondiente esté en la posición adecuada)

SALIDAS ADVANTYS																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ENTRADAS ROBOT	DI1	DI2	DI3	DI4	DI5	DI6	DI7	DI8	DI9	HOLD	CSTOPI	FAULT RESET	START	ENBL	RSR1	PROD START
PIN CRM2B FANUC			1	2	3	4	5	6	7							
PIN CRM2A FANUC	31	32								2	4	5	6	8	9	30

Apéndice B

Programas del robot en lenguaje KAREL

Se incluye a continuación el código en lenguaje Karel del programa principal del robot, que a su vez está compuesto de varios sub-programas a modo de rutinas.
El programa «Integra», que es el que a continuación se adjunta, contiene los siguientes programas o sub-rutinas:

«**Trasl_cinta**» Se encarga de recoger una pieza de la cinta transportadora, y depositarla en otra cinta transportadora.

«**Paletiza**» Se ocupa de apilar piezas que recoge de la cinta transportadora en tres columnas.

«**Desecha**» Coge una pieza de la cinta transportadora, y procede a su desechado.

«**Mueve_tr**» Coge una pieza del tren, indicada mediante una variable, y la deposita en la posición que se le diga mediante otra variable.

«**Desm_tr**» Desmonta el trenecito mediante llamadas sucesivas a la rutina «Mueve_tr».

«**Monta_tr**» Monta el trenecito mediante llamadas sucesivas a la rutina «Mueve_tr».

«**Cargar**» Rutina que carga tres piezas procedentes de la cinta transportadora en uno de los carritos.

«**Descargar**» Sub-programa que descarga las tres piezas de un carrito y las deja en la cinta transportadora.

```
PROGRAM integra

%ENVIRONMENT IOSETUP            -- Directiva necesaria para definir GIN[1]
%INCLUDE KLIOTYPS          -- Incluye un archivo que iguala parametros a números
-----------------------------------------------------------------------------------------
--      AUTOR: JOSE IGNACIO ABAD MARTINEZ.
--      BASADO: EN UN PROGRAMA DE RICARDO BELTRAN ROYO.
--      DIRECTORES: JOSECHU GUERRERO Y GONZALO LOPEZ
--      FECHA DE REALIZACIÓN: Diciembre de 2013
--      FECHA ÚLTIMO MODIFICADO: Enero de 2013
--      DESCRIPCIÓN: Este programa sirve para enseñar la integracion de un robot y un automata;
-----------------------------------------------------------------------------------------

CONST
        distancia = 100         -- utilizada en los movimiento: move away y move near
        max_vel = 2000          -- velocidad de los movimientos sin peligro
        aprox_vel = 100         -- velocidad de los movimientos para aproximaciones

VAR
        herramienta: POSITION -- (0,0,101.34,-180,0,0)
        mundo: POSITION
        P_casa: JOINTPOS6               --(0,0,90,0,0,0) articulares
        aproxima: INTEGER


-- LOCALIZACIONES NECESARIAS PARA EL PROGRAMA (Declaración)

        P_rec_pi: XYZWPR
        P_aprox_pi: XYZWPR
        P_aprox_plt: XYZWPR
        P_palet_1: XYZWPR
        P_palet_2: XYZWPR
        P_palet_3: XYZWPR
        P_palet_1_1: XYZWPR
        P_palet_2_1: XYZWPR
        P_palet_3_1: XYZWPR
        P_dej_cin: XYZWPR
        P_aprox_dej: XYZWPR

        pal_1_1: XYZWPR
        pal_1_2: XYZWPR
        pal_1_3: XYZWPR
        pal_2_1: XYZWPR
        pal_2_2: XYZWPR
        pal_2_3: XYZWPR
        P_vio: XYZWPR
        P_dej_pi: XYZWPR

        P_reposo: JOINTPOS6
        P_paso_1: JOINTPOS6
        P_desecha: XYZWPR               -- posición donde desecha la pieza

-- LOCALIZACIONES PARA MONTAR Y DESMONTAR LAS PIEZAS EN EL TREN
        Dist_segu: XYZWPR               -- distancia de 2mm en z para en las aproximaciones
                                        -- a dejar no chocar con la mesa y causarle esfuerzos
        Almacen: XYZWPR
        CogeRectan: XYZWPR      -- "Almacen:CogeRectan" lugar donde se recoge el rectangulo que se
                                --  coloca el 2º
        RectanArriba: XYZWPR -- "Almacen:CogeRectan:RectanArriba" lugar donde se recoge el
                                --  rectangulo que se coloca el 1º
```



```
        CogeCilin: XYZWPR       -- "Almacen:CogeCilin" lugar donde se recoge el cilindro que se
                                --  coloca el 2º
        CilinArriba: XYZWPR     -- "Almacen:CogeCilin:CilinArriba" lugar donde se recoge el cilindro
                                --  que se coloca el 1º

        Vagon2: XYZWPR
        ColocaRectan: XYZWPR-- "Vagon2:ColocaRectan" lugar donde se coloca el rectangulo que se
                                --  coloca el 1º
                                -- "Vagon2:ColocaRectan:RectanArriba" lugar donde se coloca el
                                --  rectangulo que se coloca el 2º
        Vagon1: XYZWPR
        ColocaCilin1: XYZWPR -- "Vagon1:ColocaCilin1" lugar donde se coloca el cilindro que se
                                --  coloca el 1º
        ColocaCilin2: XYZWPR -- "Vagon1:ColocaCilin2" lugar donde se coloca el cilindro que se
                                --  coloca el 2º


-- GENERALES PARA EL FUNCIONAMIENTO DEL PROGRAMA

        r1 : integer            -- utilizada para realizar el SELECT en el programa principal

        paletizador: integer

        config_habit: CONFIG    -- configuracion utilizada para todas las posiciones (0,0,0,flip,codo
                                -- arriba,frente)

        clock:integer           -- contador para la subrutina expedir
        clk_var: integer        -- contador para la subrutina pieza_x
        clk_indi:integer        -- contador para el encendido intermitente de DOUT[8]

        terminar:boolean        -- variable que será usada para indicar la finalización del programa si
                                -- fuera necesario
        puerta_ab:boolean       -- variable que permite la parada si se abre la puerta


-----------------------------------------------------------------------------------------
--                              PROCEDIMIENTOS
-----------------------------------------------------------------------------------------

-- Rutina que pone en una variable de tipo JOINTPOS6 "lug_j" ;
-- las coordenadas articulares "A,B,C,D,E,F";

ROUTINE create_jpos(lug_j:JOINTPOS6;A,B,C,D,E,F: REAL)
VAR
        vec_pos: ARRAY[6] OF REAL
        exito2: INTEGER
BEGIN
        vec_pos[1]=A;
        vec_pos[2]=B;
        vec_pos[3]=C;
        vec_pos[4]=D;
        vec_pos[5]=E;
        vec_pos[6]=F;
        CNV_REL_JPOS(vec_pos,lug_j,exito2);
END create_jpos
```

```
-- Rutina que sirve para iniciar una variable de tipo CONFIG, dandole los tres integers "a,b,c" y ;
-- su configuración con booleanos: flip/no flip "fl",codo arriba/abajo "u" y frente/espalda "fr";

ROUTINE create_confi(confi: CONFIG;a,b,c:INTEGER;fl,u,fr:BOOLEAN)
BEGIN
        confi.cfg_turn_no1=a;
        confi.cfg_turn_no2=b;
        confi.cfg_turn_no3=c;
        confi.cfg_flip=fl;
        confi.cfg_up=u;
        confi.cfg_front=fr;
END create_confi

-- Rutina que inicializa las localizaciones utilizadas en el programa

ROUTINE ini_pos
BEGIN
        create_jpos(P_casa,0,0,90,0,0,0);
        create_jpos(P_reposo,-101, 15, -36, 0, -57, 31);
        create_jpos(P_paso_1,25.025, 10.064, -16.23, 0, -76.4, 3.5);
        P_rec_pi=POS(170.0,-593.0,-300.0,0,0,-90,config_habit);
        P_aprox_pi=POS(170.0,-593.0,-150.0,0,0,-90,config_habit);
        P_aprox_plt=POS(-555, -270, -150, 0, 0, -179,config_habit);
        P_palet_1=POS(-505, -270, -345, 0, 0, -179,config_habit);
        P_palet_2=POS(-555, -270, -345, 0, 0, -179,config_habit);
        P_palet_3=POS(-605, -270, -345, 0, 0, -179,config_habit);
        P_palet_1_1=POS(-505, -270, -313, 0, 0, -179,config_habit);
        P_palet_2_1=POS(-555, -270, -313, 0, 0, -179,config_habit);
        P_palet_3_1=POS(-605, -270, -313, 0, 0, -179,config_habit);
        P_dej_pi=POS(162.4, -794.1, -300, 0, 0, -90, config_habit);
        P_dej_cin=POS(355, 273.75, -300, 0, 0, 0,config_habit);
        P_aprox_dej=POS(355, 310, -150, 0, 0, 0,config_habit);
        P_desecha= POS(-500.000,-370.605,-200.000,0,0,-90,config_habit);

        Dist_segu= POS(0,0,2,0,0,0,config_habit);

        Almacen= POS(-124.000,-528.000,-355.000,0,0,0,config_habit);
        CogeRectan= POS(24.7,50,2,0,0,0,config_habit);
        RectanArriba= POS(0,0,13.05,0,0,0,config_habit);
        CogeCilin= POS(24.7,50,39.1,0,0,0,config_habit);
        CilinArriba= POS(0,0,26,0,0,0,config_habit);
        Vagon2= POS(-106.000,-700.000,-333.000,0,0,0,config_habit);
        ColocaRectan= POS(-57.5,27.2,6.525,0,0,0,config_habit);
        Vagon1= POS(-228.000,-700.000,-333.000,0,0,0,config_habit);
        ColocaCilin1= POS(-83.1,27.2,17,0,0,0,config_habit);
        ColocaCilin2= POS(-32.9,27.2,17,0,0,0,config_habit);

        pal_1_1=POS(369, -663.272, -259.84, 0, 0, -90, config_habit);
        pal_1_2=POS(369, -731.3, -259.84, 0, 0, -90, config_habit);
        pal_1_3=POS(342.2, -698, -259.84, 0, 0, -90, config_habit);
        pal_2_1=POS(372.27, 730.358, -259.84, 0, 0, 90 ,config_habit);
        pal_2_2=POS(372.27, 661, -259.84,0, 0, 90, config_habit);
        pal_2_3=POS(344.6, 695.8,-259.84, 0, 0, 90, config_habit);

        P_vio=POS(762, 0, 2, 0, 0, 0, config_habit);

END ini_pos
```



```
-- Rutina que inicializa las variables de sistema

ROUTINE init_system
VAR
        exito: INTEGER
BEGIN
-- parametrizado de la herramienta
        herramienta= POS(0,0,101.34,-180,0,0,config_habit);
        $UTOOL = herramienta;
        WRITE (CR,'GARRA HABILITADA',CR);

-- parametrizado de el mundo
        mundo= POS(0,0,0,0,0,0,config_habit);
        $UFRAME = mundo;

-- hacer complementarias las salidas del robot que controlan la muñeca;
        SET_PORT_MOD(io_rdo,1,2,exito);
        WRITE('RDO1 Y RDO2 complementarias',exito,CR);

-- hacer un grupo de señales digitales
        SET_PORT_ASG(io_gpin,1,0,1,io_din,19,5,exito);
        WRITE('ASIGNADAS LAS ENTRADAS',exito,CR);

-- parametrizado de las velocidades de los movimientos
        $GROUP[1].$SPEED=2000;
        $MCR_grp[1].$PRGOVERRIDE= 100;

        $PARAM_GROUP[1].$JNTVELLIM[1]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[2]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[3]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[4]= 400;
        $PARAM_GROUP[1].$JNTVELLIM[5]= 330;
        $PARAM_GROUP[1].$JNTVELLIM[6]= 480;
        $PARAM_GROUP[1].$JNTVELLIM[7]= 0;
        $PARAM_GROUP[1].$JNTVELLIM[8]= 0;
        $PARAM_GROUP[1].$JNTVELLIM[9]= 0;

-- parametrizado de la interpolación de la orientación
        $ORIENT_TYPE= RSWORLD                -- orientado mediante el metodo de los dos angulos:

-- parametrizado de la aceleración
        $GROUP[1].$usemaxaccel= TRUE
        $PARAM_GROUP[1].$cart_accel1= 224    -- aceleración1 = 2 * aceleración2
        $PARAM_GROUP[1].$cart_accel2= 112    -- metodo de 2º orden, reduce vibraciones de la
estructura
        $PARAM_GROUP[1].$accel_time1[1]= 224
        $PARAM_GROUP[1].$accel_time1[2]= 224
        $PARAM_GROUP[1].$accel_time1[3]= 224
        $PARAM_GROUP[1].$accel_time1[4]= 224
        $PARAM_GROUP[1].$accel_time1[5]= 224
        $PARAM_GROUP[1].$accel_time1[6]= 224
        $PARAM_GROUP[1].$accel_time2[1]= 112
        $PARAM_GROUP[1].$accel_time2[2]= 112
        $PARAM_GROUP[1].$accel_time2[3]= 112
        $PARAM_GROUP[1].$accel_time2[4]= 112
        $PARAM_GROUP[1].$accel_time2[5]= 112
        $PARAM_GROUP[1].$accel_time2[6]= 112

--Incicializacion de variables:
```

```
        terminar = false;
        paletizador=0;
        aproxima=0;

END init_system

--Rutina que traslada las piezas de una cinta a otra

ROUTINE trasl_cinta

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_pi;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_aprox_pi;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_paso_1;

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_dej;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_dej_cin;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_dej;
```



```
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;
        DOUT[1]=OFF;

END trasl_cinta

-- Rutina que apila las piezas entres montones indefininamente

ROUTINE paletiza

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        If paletizador >2 then
                paletizador=0;
                aproxima=aproxima+30;
        ENDIF

        paletizador=paletizador+1;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_pi;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = 200;
        MOVE TO P_aprox_pi;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_plt;

        $MOTYPE = LINEAR;
```

```
        $TERMTYPE = FINE;
        $SPEED = 200;

        SELECT paletizador OF
                CASE (1): MOVE NEAR P_palet_1 BY -aproxima
                CASE (2): MOVE NEAR P_palet_2 BY -aproxima
                CASE (3): MOVE NEAR P_palet_3 BY -aproxima
                CASE (4): terminar=TRUE; GO TO acabando
        ENDSELECT

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_aprox_plt;

        CLOSE HAND 1;

acabando::
        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;
        DOUT[1]=OFF;

END paletiza
```

Rutina que desecha una pieza procedente de la cinta transportadora

```
ROUTINE desecha

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = 200;
        MOVE TO P_aprox_pi;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
```



```
        MOVE TO P_reposo;

        $TERMTYPE = FINE;
        MOVE TO P_desecha;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;
        DELAY 100;
        CLOSE HAND 1;

        MOVE TO P_reposo;
        DOUT[2]=ON;
        DOUT[1]=OFF;
END desecha
```

-- Rutina que se encarga de coger una pieza del tren y depositarla en otro lado;

```
ROUTINE mueve_tr(Tren_coge,Tren_deja: XYZWPR)
BEGIN
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_coge BY -distancia
        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_coge BY -(distancia+2);      -- asegurarnos despues de la interrupcion de
                                                    --llegar al sitio
        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = aprox_vel;
        MOVE TO Tren_coge;
        CLOSE HAND 1;
        DELAY 200;
        WRITE(CR,'PIEZA COGIDA',CR);
        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = aprox_vel;
        MOVE AWAY -distancia;
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_deja BY -distancia;
        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = aprox_vel;
        MOVE TO Tren_deja;
        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;
        DELAY 200;
        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = aprox_vel;
        MOVE AWAY -distancia;
END mueve_tr
```

```
-- Rutina que desmonta el tren mediante el uso de la instrucción "mueve_tr"

ROUTINE desm_tr

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        mueve_tr(Vagon2:ColocaRectan:RectanArriba,Almacen:CogeRectan);
        mueve_tr(Vagon2:ColocaRectan,Almacen:CogeRectan:RectanArriba);
        mueve_tr(Vagon1:ColocaCilin2,Almacen:CogeCilin:Dist_segu);
        mueve_tr(Vagon1:ColocaCilin1,Almacen:CogeCilin:CilinArriba:Dist_segu);
        DOUT[2]=ON;
        DOUT[1]=OFF;
END desm_tr

--Rutina que monta el tren mediante el uso de la instrucci´´on "mueve_tr"

ROUTINE monta_tr

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        mueve_tr(Almacen:CogeCilin:CilinArriba,Vagon1:ColocaCilin1:Dist_segu);
        mueve_tr(Almacen:CogeCilin,Vagon1:ColocaCilin2:Dist_segu);
        mueve_tr(Almacen:CogeRectan:RectanArriba,Vagon2:ColocaRectan:Dist_segu);
        mueve_tr(Almacen:CogeRectan,Vagon2:ColocaRectan:RectanArriba:Dist_segu);
        DOUT[2]=ON;
        DOUT[1]=OFF;
END monta_tr

-- Rutina que carga tres piezas en uno de las plataformas destinadas a ello sobre un carrito

ROUTINE cargar

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_aprox_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        MOVE TO P_aprox_pi;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR pal_1_1 BY -100;
```



```
        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO Pal_1_1;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_aprox_pi;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        MOVE TO P_aprox_pi;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR pal_1_2 BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO Pal_1_2;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_aprox_pi;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        MOVE TO P_aprox_pi;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
```

```
        $SPEED=max_vel;
        MOVE NEAR pal_1_3 BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO Pal_1_3;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=FINE;
        $SPEED=max_vel;
        MOVE TO P_reposo;

        CLOSE HAND 1;
        DOUT[2]=ON;
        DOUT[1]=OFF;
END cargar
```

-Rutina que descarga las piezas de la plataforma situada sobre un carrito, y las deja en la cinta

```
ROUTINE descargar

BEGIN
        DOUT[1]=ON;
        DOUT[3]=OFF;
        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR pal_2_1 BY -100;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO pal_2_1;

        CLOSE HAND 1;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR P_dej_pi BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
```



```
        MOVE TO P_dej_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        MOVE NEAR pal_2_2 BY -100;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO pal_2_2;

        CLOSE HAND 1;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        MOVE NEAR P_dej_pi BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_dej_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR pal_2_3 BY -100;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO pal_2_3;
```

```
        CLOSE HAND 1;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR P_dej_pi BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_dej_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;
        DOUT[1]=OFF;
END descargar

-- Inicio del programa principal. Con pocas instrucciones y utilizando las retinas creadas, se maneja el
--  robot.

BEGIN
        create_confi(config_habit,0,0,0,false,true,true);        --ponemos el valor de la configuración
                                                                 --  habitual

        -- Condition handlers

        CONDITION[3]:
                WHEN DIN[6] DO                  --nos indican que acabemos el programa
                terminar=true
        ENDCONDITION;

        CONDITION[4]:
                WHEN DIN[7] DO
                STOP
                ENABLE CONDITION[5]
                DISABLE CONDITION[4]
        ENDCONDITION;

        CONDITION[5]:
                WHEN NOT DIN[7] DO
                RESUME
                DISABLE CONDITION[5]
        ENABLE CONDITION[4]
        ENDCONDITION;
```



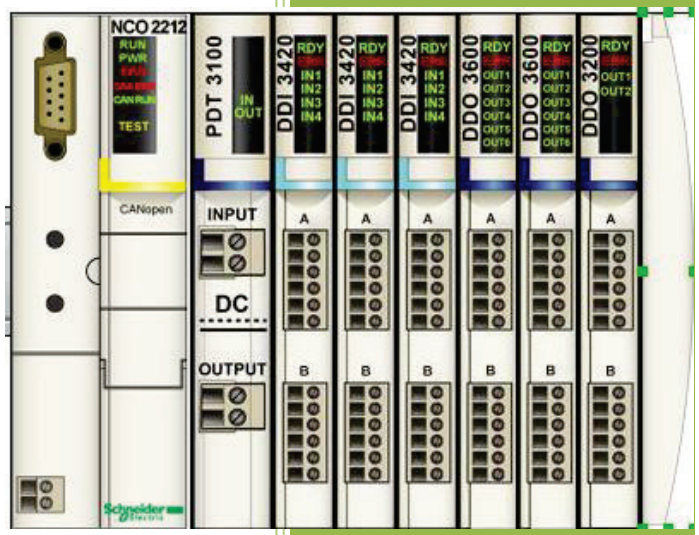
```
---------------------------------------------- Inicio del programa;

        init_system;
        ini_pos;
        ENABLE CONDITION[3];        --habilita la posibilidad de apagar el robot
        ENABLE CONDITION[4];        --habilita la parada de emergencia si se abre la puerta (señal
                                    --numero 12)

        -- Ejecucion del programa

        WHILE NOT terminar DO
                DOUT[3]=ON;
                IF din[1] THEN
                        trasl_cinta;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF;

                IF din[2] THEN
                        paletiza;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF

                IF din[3] THEN
                        desecha;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF

                IF din[4] THEN
                        monta_tr;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF

                IF din[5] THEN
                        desm_tr;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF

                IF din[9] THEN
                        cargar;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF

                IF din[10] THEN
                        descargar;
                        DELAY 100;
                        DOUT[2]=OFF;
                ENDIF
        ENDWHILE

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_casa;
```

Apéndice C

Manual de configuración de un módulo Advantys

En este apéndice se incluye el manual de configuración realizado para una isla «Advantys STB» como las utilizadas durante este proyecto.
En dicho manual se detalla todo el proceso necesario para la configuración y su posterior utilización mediante el programa «Unity», desde su instalación hasta su introducción al programa.

Manual de configuración de un módulo ADVANTYS mediante software UNITY



José Ignacio Abad Martínez

EINA Universidad de Zaragoza

Índice

0. Versiones, correcciones y compatibilidad.
1. Introducción.
   1.1. Isla Advantys.
   1.2. Autómata Modicon M340.
   1.3. Bus CANopen.
2. Instalación de la isla Advantys.
   2.1. Instalación física de la Isla.
   2.2. Características particulares de la instalación.
   2.3. Errores habituales y código de parpadeos LED.
3. Configuración de la isla Advantys mediante el programa “ADVANTYS”.
   3.1. Inicialización de ADVANTYS y nueva isla.
   3.2. Selección de los módulos y colocación en orden.
   3.3. Nombramiento de variables.
   3.4. Primera exportación (*DCF).
4. Configuración básica del autómata mediante el programa UNITY.
   4.1. Inicialización de UNITY.
   4.2. Configuración del bus CANopen.
   4.3. Reserva de espacio para variables.
5. Coordinación de ambas configuraciones y puesta a punto de señales.
   5.1. Exportación desde ADVANTYS del archivo (*XSY).
   5.2. Importación en UNITY del archivo (*XSY) y adecuación de variables.
6. Prueba de señales y finalización de la configuración.
   6.1. Creación de una tabla de animación.
   6.2. Conexión con el autómata, carga del programa, ejecución y comprobación.
   6.3. Errores habituales

0. Versiones, correcciones y compatibilidad.

Versión 1: Agosto de 2013.

Versión de Advantys utilizada: 24.11.2010

Versión de Unity utilizada: Unity Pro XL V6.0 – 110610

Última revisión: 28/8/2013.

1. Introducción.

1.1. Isla Advantys.

Una isla Advantys STB es un montaje de módulos de entradas y/o salidas distribuidas y alimentación, entre otros, que funcionan juntos como un nodo de isla en una red de bus de campo abierta.

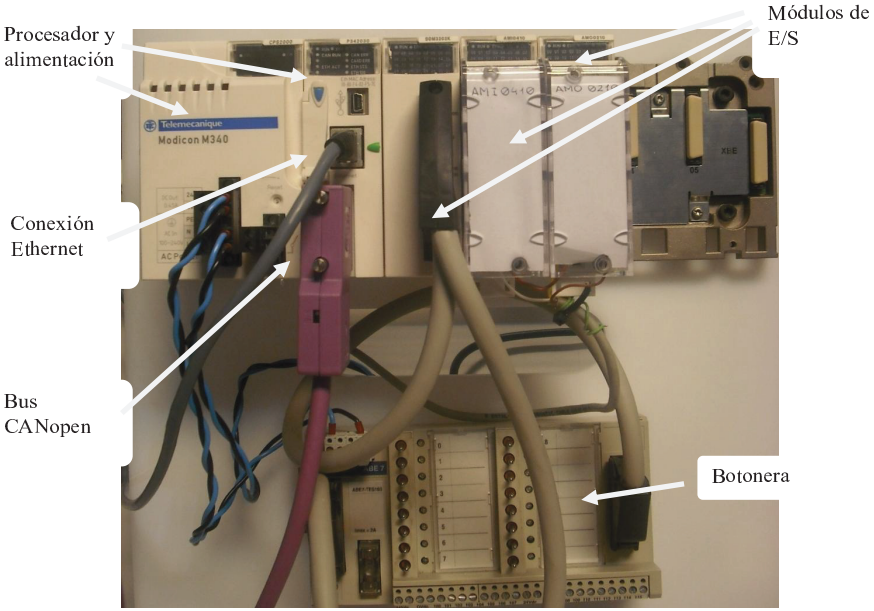
Su función principal es proveer a un autómata, o más comúnmente a una red de ellos, de un buen número de entradas y salidas, generalmente de carácter digital.

Su conexión con los autómatas se puede realizar mediante una conexión de tipo BUS CANopen, o mediante otros tipos de conexión como podría ser Ethernet…

1.2. Autómata Modicon M340.

El PLC Modicon M340 es un autómata modular, de alto rendimiento y ampliable. Está especialmente diseñado para maquinaria compleja e infraestructuras de tamaño medio, o como en nuestro caso, para el uso docente.

Tiene diferentes versiones diferenciadas por su conectividad. En este caso la conectividad se realizará mediante el bus CANopen y la red Ethernet.



Vista de un autómata Modicon M340.

1.3. Bus CANopen.

CANopen es un protocolo de comunicaciones de alto nivel, para uso industrial, basado en el bus CAN. Dispone de diferentes modelos de comunicación, de los cuales el que nos interesa en nuestro caso es el modelo Maestro/Esclavo.

Este modelo de comunicación implica que en todo momento hay un único maestro que dirige las comunicaciones para una determinada funcionalidad; el resto de dispositivos se consideran esclavos. El maestro hace una petición y el esclavo direccionado responde (si es necesario).

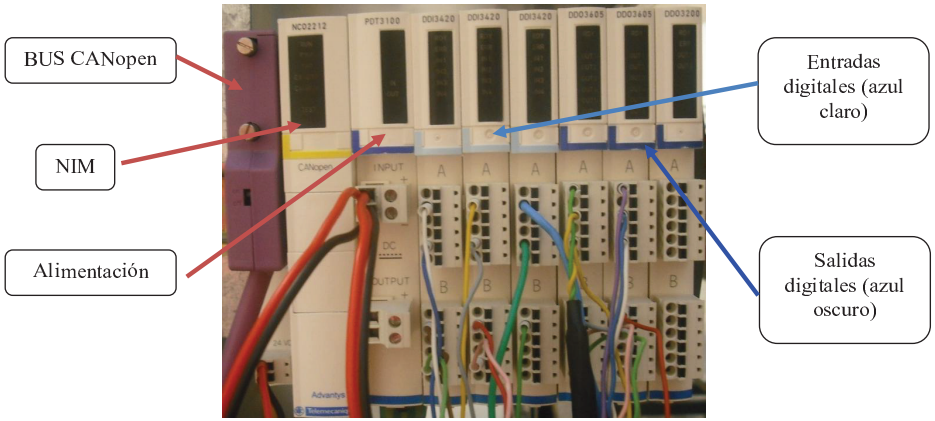
2. Instalación de la isla Advantys.

2.1. Instalación física de la isla.

- Componentes de una isla Advantys:

La isla Advantys se compone fundamentalmente de tres tipos de módulos.

En primer lugar tenemos el módulo NIM, que es el segmento básico y el primer módulo que se coloca ordenado físicamente en una isla Advantys. Sus funciones son básicamente dos: Por una parte es el módulo que actúa de interfaz con el bus de campo que se utiliza. Y por otra parte es el encargado de asignar las direcciones físicas a los módulos que componen el resto de la isla, lo cual implicará que cualquier cambio físico de un módulo nos obligará a reprogramarlo.



Vista general de una isla Advantys.

Otro de sus componentes, y el segundo a colocar en el orden físico, es la fuente de alimentación. Sin ella, o con ella mal conectada al raíl, no funcionará correctamente la isla Advantys.

Por último están los módulos de entradas y salidas. Estos módulos se diferencian por su denominación (DDO o DDI) y una vez determinado si es de entradas o de salidas, por su numeración; Que va en función el número de entradas o salidas que tienen y de algunas características especiales como el que estén protegidas contra sobretensiones…

Es muy importante el orden de los módulos E/S, ya que siempre deben colocarse físicamente en primer lugar los módulos de entradas, y después los de salidas. Dentro de cada tipo, lo convencional es comenzar situando los de mayor número de pines en primer lugar.

Hay que mencionar la existencia de algún otro tipo de módulos para isla Advantys tales como módulos de relés o módulos de distribución de alimentación para aumentar con más módulos una isla.

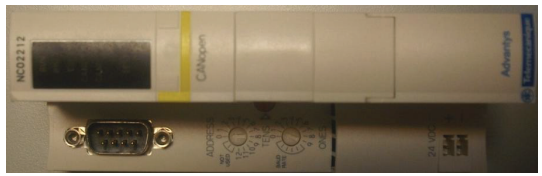
- Soporte físico y conexión con el maestro:

La isla Advantys se instala sobre un raíl tipo DIN, que a la postre resulta fundamental para el funcionamiento de la misma. Por una parte, el módulo NIM se instala directamente sobre el raíl, mientras que el resto de módulos se instalan sobre unas plataformas especiales.

Es importante asegurarse de que las plataformas están bien conectadas entre sí, con el raíl, y con el módulo NIM. Si no, la isla no funcionará de manera adecuada.

Vista de la isla con el rail DIN y las placas de sujeción.

La comunicación con el maestro se realiza por parte del NIM, a través del bus CAN. Los parámetros que lo configuran se pueden modificar mediante las ruletas que existen en el NIM.



Módulo NIM.

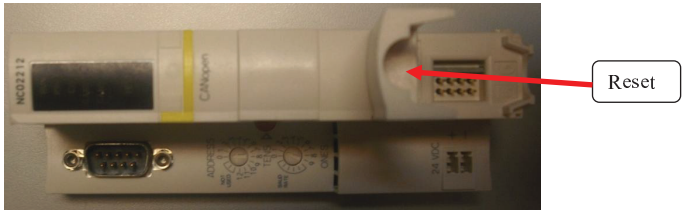
2.2 Características particulares de la instalación.

A la hora de instalar una nueva isla Advantys, o de modificar una existente añadiendo, quitando o simplemente cambiando módulos, hay que tener en cuenta las siguientes consideraciones:

Siempre ha de existir el mismo número de bases que de módulos. Si no, el comportamiento puede ser extraño, tal como no reconocer algún módulo intermedio.

Siempre se debe terminar la isla con una placa de terminación. En caso contrario el NIM dará un error.

Después de cada modificación, en el caso de que se haya cambiado un módulo por otro que no es igual, se deberá resetear el NIM. En caso contrario aparecerá un error. Aparte de ello, será necesario reprogramar la isla Advantys que incluimos en el programa del autómata, como se explicará en los capítulos 3 y sucesivos. Es posible que tras resetear el NIM, deba hacerse lo mismo con el autómata, pero no siempre es necesario.



Detalle del Reset del NIM

Por último, siempre es importante al realizar la primera instalación, el comprobar que la alimentación se realiza a la tensión adecuada, y que el raíl no está en tensión.

2.3. Errores habituales y código de parpadeos LED.

Se mencionan aquí algunos de los errores más habituales que se pueden cometer, y se adjunta el código de los Led de los módulos, que indican mediante sus parpadeos o su ausencia, el tipo de error cometido.

- El NIM tiene encendida la luz de CAN ERROR: Puede tratarse o bien de que el autómata no ha sido todavía configurado, o de que la configuración que el NIM recibe del autómata no se corresponde con la real. Primero se tiene que resetear el NIM, para que vuelva a direccionar todos sus módulos, si el error persiste, entonces es problema del autómata. Cuando el problema es del autómata, lo más probable es que se trate de que o bien se ha cargado una configuración de la isla que no se corresponde, o que no se ha cargado ninguna. Hay que volver a cargarla, como se explicará en los capítulos posteriores.

- Algunos módulos de forma aislada (o incluso aleatoria) señalan que están en error: Hay varias opciones, la más habitual es que alguno de ellos no esté bien anclado a su base, o que su base no esté bien conectada con las demás. otra opción es que se hayan puesto en desorden. Y por último, que la alimentación esté fallando, lo cual puede deberse a que la fuente está mal, o algún problema con el raíl o las placas de conexión.
- Cuando activo señales que en la isla Advantys son de entrada, no se enciende ninguna luz en los módulos: Suponiendo que no hay luz alguna de error en dicho módulo, puede ser que el pin no está conectado en su lugar (ver lateral del módulo para ubicarlo bien). Otra opción es que el pin no esté bien conectado o haga mal contacto. Y otra más podría ser que no se esté alimentando a los requerimientos de tensión e intensidad necesarios (ver lateral del módulo).

Se adjunta a continuación y a modo de ejemplo, una tabla de errores según sus indicadores luminosos para el módulo DDI 3420. Todas las tablas están disponibles en el manual "STB modules".

RDY	ERR	IN1	IN2	IN3	IN4	Significado	Qué hacer
apagado	apagado					El módulo no recibe alimentación lógica o ha fallado.	Comprobar la alimentación
parpadeo*	apagado					El autodireccionamiento está en progreso.	
encendido	apagado					El módulo ha conseguido todo lo que sigue: • tiene alimentación • ha pasado sus pruebas de confianza • está operativo	
		encendido				Hay tensión en el canal de entrada 1.	
		apagado				No existe tensión en el canal de entrada 1.	
			encendido			Hay tensión en el canal de entrada 2.	
			apagado			No existe tensión en el canal de entrada 2.	
				encendido		Hay tensión en el canal de entrada 3.	
				apagado		No existe tensión en el canal de entrada 3.	
					encendido	Hay tensión en el canal de entrada 4.	
					apagado	No existe tensión en el canal de entrada 4.	

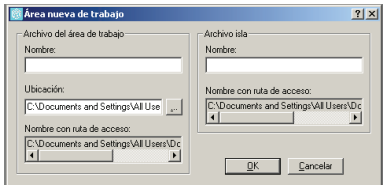
Tabla de errores del módulo Advantys DDI 3420

3. Configuración de la isla Advantys mediante el programa "ADVANTYS".

3.1. Inicialización de ADVANTYS y nueva isla.

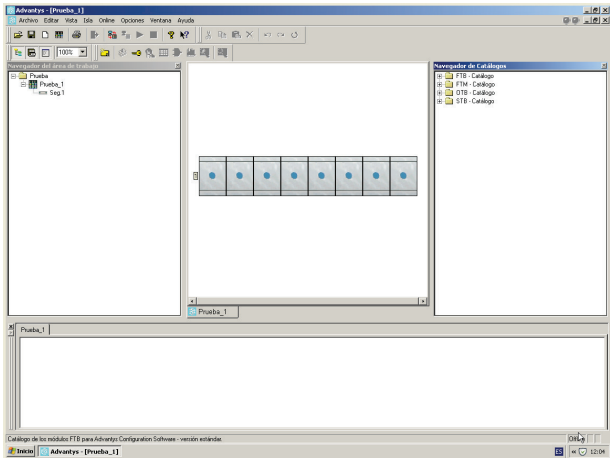
Los primeros pasos para realizar la configuración de la isla, se van a dar mediante el programa Advantys, que es proporcionado por el propio fabricante.

Para comenzar, una vez abierto el programa, seleccionamos "Nueva isla". Entonces aparecerá el siguiente menú, en el cual damos nombre al área de trabajo que contendrá los diferentes archivos que vamos a generar, y nombramos también la isla propiamente dicha.



Detalle de nueva isla

Cuando hayamos ejecutado estos pasos, nos aparecerá un raíl totalmente vacío. Es el momento de situar los componentes.



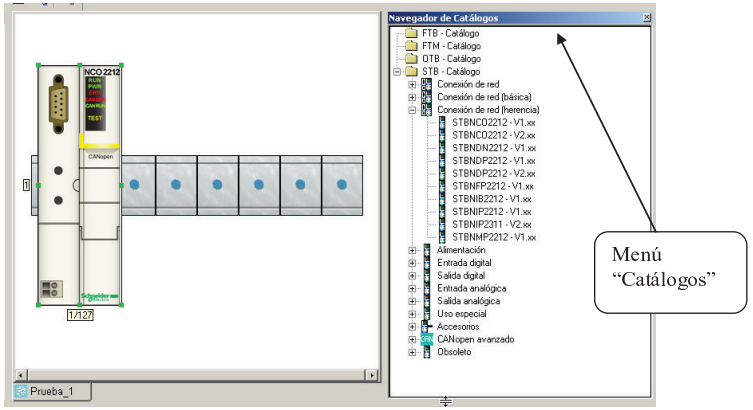
Detalle del rail vacío.

3.2. Selección de los módulos y colocación en orden.

Ahora es el momento de ir colocando, en riguroso orden según estén en la isla real, los diferentes componentes de la misma.

Para ello, hay que localizarlos en el menú "catálogo", que se puede desplazar al punto de la pantalla donde moleste menos.

Primero como es lógico, se colocará el NIM. Podría ser cualquiera de los que existen, pero en nuestro caso, los disponibles en el laboratorio son todos el modelo NCO 2212 versión 1. Está un poco escondido, por lo que habrá que ir a "conexión de red/herencia".



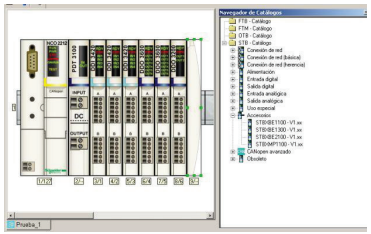
Detalle de la inserción del NIM

Para colocarlo, simplemente basta con arrastrarlo hasta su posición.

Después tendremos que colocar la fuente de alimentación, que en nuestro caso, y nuevamente por cuestión de disponibilidad se trata de la PTD 3100.

A continuación iremos añadiendo los diferentes módulos que tenga nuestra isla.

Por último, y muy importante, hay que añadir la placa de terminación de la isla. En el laboratorio se encuentran disponibles las XMP 1100, pero hay que asegurarse antes de que sea la correcta



Detalle de placa terminada.

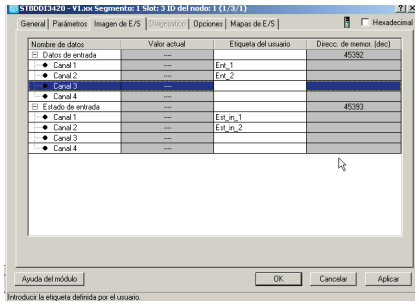
Una vez tenemos la isla montada como la real, toca pasar a nombrar las variables.

3.3. Nombramiento de variables.

Ahora toca un paso importante, que es el asignar nombres a las diferentes variables que se crearán para almacenar los estados de las entradas y salidas. Nuestro único papel en este proceso es el de nombrar las variables, ya que su creación será cosa de los procedimientos internos del programa.

Para comenzar, seleccionamos el primer módulo que tengamos, y hacemos doble clic sobre él. Nos aparecerá una pantalla con los diferentes canales que tiene, su valor actual, y una "etiqueta de usuario". Este último es el apartado que tenemos que rellenar.

Cuando se trata de entradas, hay dos apartados, "Datos de entrada" y "Estado de entrada". En principio es recomendable darle etiquetas a ambos, pero dependiendo de lo que se vaya a hacer con ellas, y de la versión de Advantys, puede bastar con nombrar solo los datos de entrada.



Detalle de nombramiento de las variables.

Para el caso de las variables de salida solo tenemos una opción. Por lo que la rellenamos.

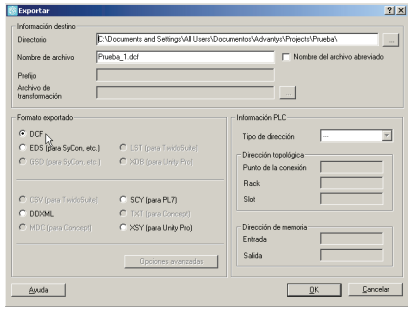
Una vez nombradas todas las variables es momento de exportar el archivo.

3.4. Primera exportación (*DCF).

Vamos a hablar de la primera exportación desde Advantys a Unity, porque en realidad va a haber dos exportaciones diferentes.

En este caso toca exportar los datos intrínsecos a la isla, y se realiza mediante un archivo con formato *DCF.

Para ello, archivo/exportar, y seleccionamos archivo DCF. Le damos el nombre deseado y aquí termina la primera parte del trabajo con Advantys. Luego tocará volver brevemente a trabajar con él.



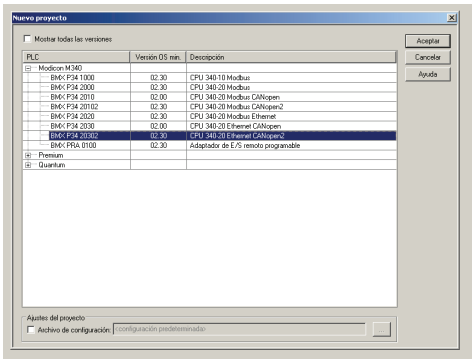
Detalle de exportación *DCF

4. Configuración básica del autómata mediante el programa UNITY.

4.1. Inicialización de UNITY.

Para comenzar a trabajar con UNITY, lo primero de todo es abrir una consola virtual. En este caso se trabajará con la consola "Schneider" dentro de la aplicación virtual-box de Sun Microsystems.

Una vez abierto el programa, pasaremos a abrir un fichero nuevo. Como nos pedirá el tipo de autómata que vamos a utilizar, lo seleccionamos. En este caso se trata de un Modicon M340, con Ethernet y CANopen. Seleccionamos el 20302.

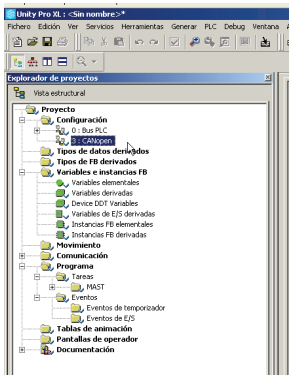


Inicialización de Unity

Superado este paso, vamos a configurar el bus CAN.

4.2. Configuración del bus CANopen.

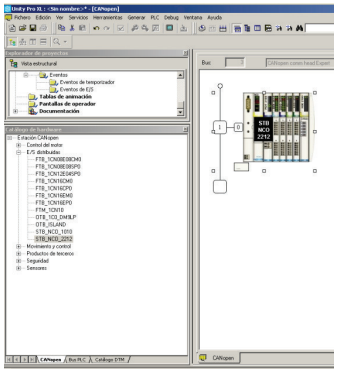
En el explorador de proyectos, vamos a la pestaña "Proyecto/Configuración" y doble clic en "CANopen". Se abrirá una nueva pantalla en la cual tenemos que seleccionar del catálogo de hardware nuestro terminal del bus CAN. En este caso, está en la pestaña "E/S distribuidas", y es el modelo STB_NCO_2212.



Detalle Proyecto/Configuracion/CANopen.

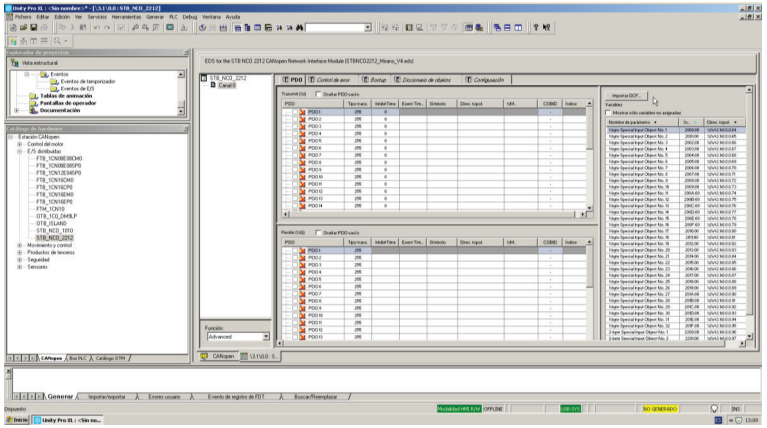
Lo arrastramos hasta situarlo en el primer nodo del bus, y seleccionaremos la dirección topológica 1. Si hay más islas conectadas al autómata: la 2,3...

Hecho esto, aparece una imagen genérica de una isla Advantys. Haciendo doble clic sobre ella se abre una nueva ventana. Abajo a la izquierda de ella aparece un desplegable, en el cual hay que seleccionar la opción "simple". Ya seleccionado "simple", se hace doble clic sobre STB_NCO_2212 y nos preguntará si deseamos validar. Decimos sí.



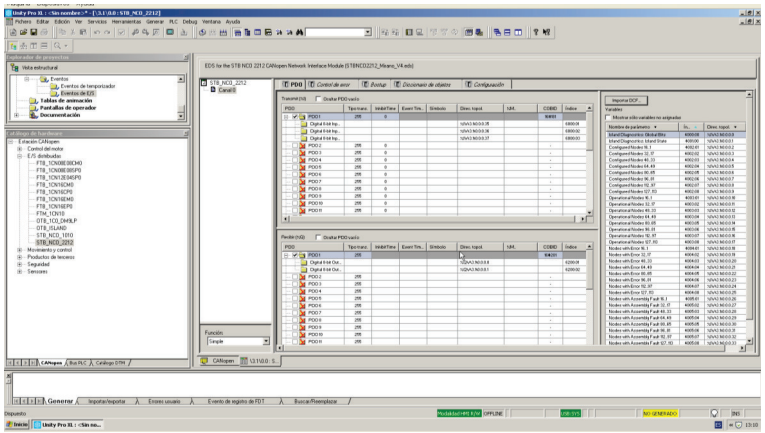
Detalle isla genérica

A continuación, se hace doble clic sobre el “canal 0”, y accedemos a la pestaña PDO. Clicamos en “Importar DCF”, y seleccionaremos el archivo que antes hemos exportado desde Advantys.



Detalle “canal 0/pestaña PDO”

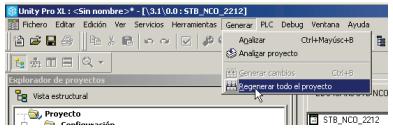
Transcurrido un tiempo, se cargarán las variables. En este momento el programa ya sabe cuántas variables hay, pero no tienen asignada dirección. Para ello tenemos que seguir unos pasos más.



Detalle de las variables cargadas

4.3. Reserva de espacio para variables.

Después de guardar (hacerlo de vez en cuando por si acaso…), pulsamos el botón “regenerar todo el proyecto”, situado en el menú “Generar”.



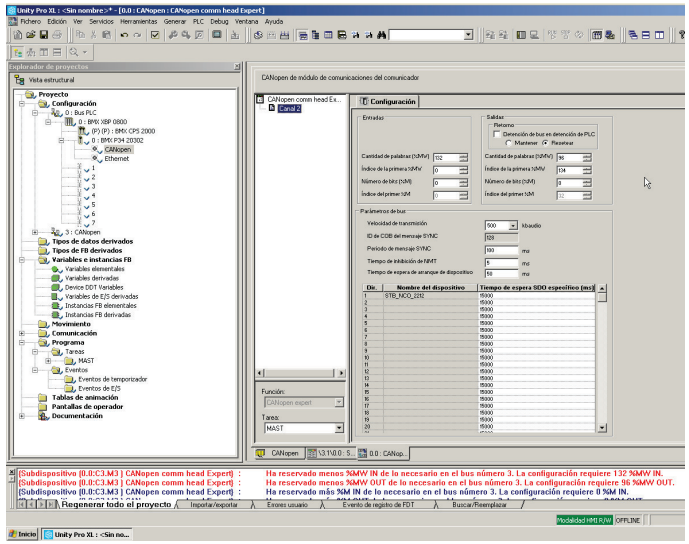
Detalle de la regeneración del proyecto.

Nos dirá que se genera con errores. De ellos se puede extraer valiosa información.

Estos errores aparecen debido a que no hemos indicado el espacio a reservar para las variables, pero a su vez en ellos se nos indica cuánto hay que reservar. Esto nos lo dirá de la forma: “se requiere… %MW IN”, que nos está indicando la memoria que necesitamos reservar para las entradas.

Para corregir esta situación, vamos a “Estación/Configuración/BUS PLC/BMX XBP 0800/BMX P34 2030/CANopen” y dentro de este al canal 2.

Aparece el cuadro de configuración del mapa de memoria de las variables. Tenemos que reservar el espacio necesario, sabiendo que las entradas van siempre antes que las salidas. Además, entre entradas y salidas hay que dejar 2 bytes de diferencia. Por ejemplo en nuestro caso, reservamos 132 posiciones para entradas, comenzando en el byte 0, y 96 posiciones para las salidas, comenzando en el byte 134. Reservamos 0 bits (%M), como nos indican los errores.

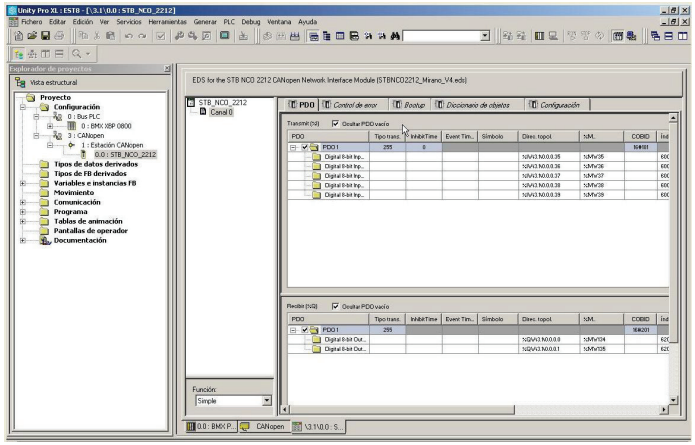


Ventana de configuración del mapa de memoria y detalle de los errores.

También tendremos que fijar la velocidad de conexión en 500 kilo Baudios, que es la que viene predeterminada por el módulo NIM.

Realizado esto, hacemos doble clic en “CANopen com head…” y confirmamos los cambios. Ahora generamos los cambios en el menú “Generar”. No deberían aparecer errores.

Si volvemos al paso 4.2, y volvemos a mirar las variables que hay, comprobaremos que ahora cada grupo de entradas ya tiene una dirección. Por comodidad se pueden ocultar las no utilizadas mediante la casilla “Ocultar PDO vacíos.



Detalle de los grupos de variables, con las no utilizadas ocultas.

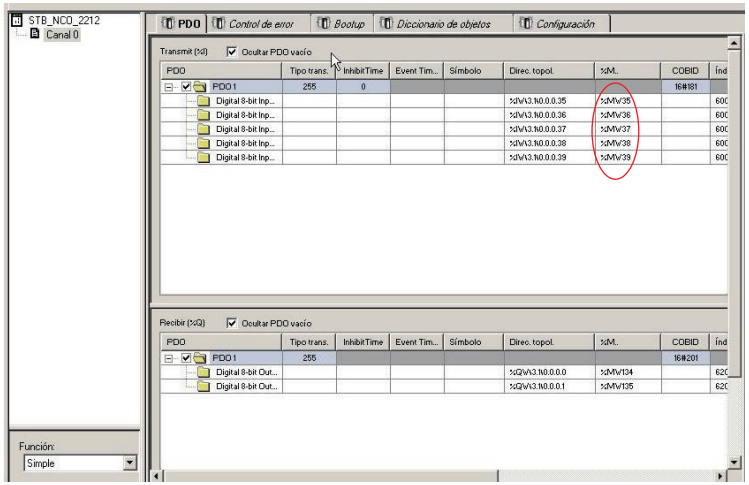
Es el momento de volver al programa Advantys.

5. Coordinación de ambas configuraciones y puesta a punto de señales.

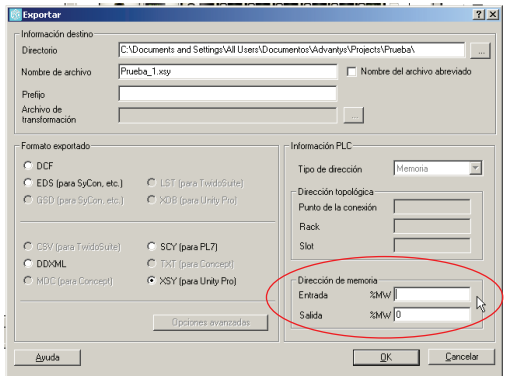
5.1. Exportación desde ADVANTYS del archivo (*XSY).

Retomando el proyecto que teníamos abierto en Advantys, ahora toca exportar el archivo *XSY, que va a contener el nombre de las variables y las relacionará con sus direcciones topológicas dadas en Unity.

Para ello debemos saber en qué posiciones comienzan las entradas y las salidas. Tal y como lo hemos definido, las entradas empezarán en el byte 35, y las salidas en el 134. Para asegurarnos, o mirarlo de nuevo en caso de otras configuraciones hay que ir al programa Unity, y mirar en la configuración del bus CANopen, en la ventana que se muestra a continuación.



Detalle de las posiciones de memoria ocupadas por las variables.

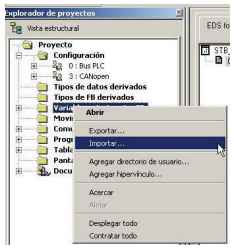


Detalle de la ventana de exportación

Una vez exportado, volveremos a Unity para terminar el proceso.

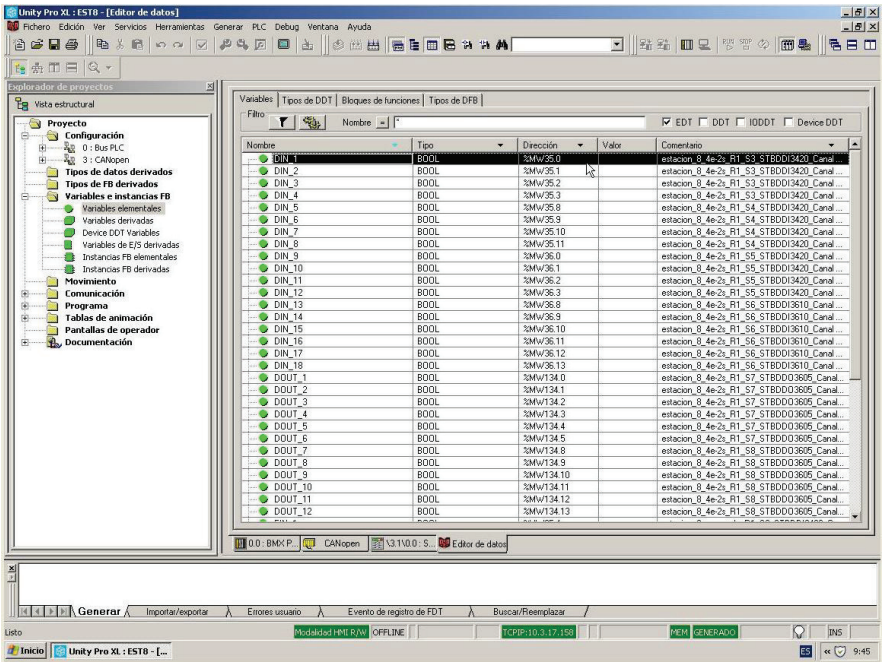
5.2. Importación en UNITY del archivo (*XSY) y adecuación de variables.

Para importar el nuevo archivo, hay que ir a la pestaña de “Estación/variables e instancias FB”, y haciendo clic con el botón alternativo, “Importar”.



Detalle de importar.

Una vez lo hayamos importado, tendremos ya una lista con las variables, sus nombres, y la dirección que les corresponde. El proceso puede tardar un rato.



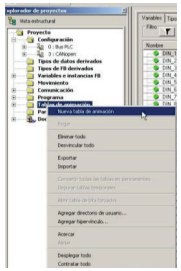
Aspecto del programa con las variables cargadas.

Cuando ya lo tengamos, estará casi acabado nuestro trabajo con Unity. Damos al botón generar cambios en el menú “PLC”.

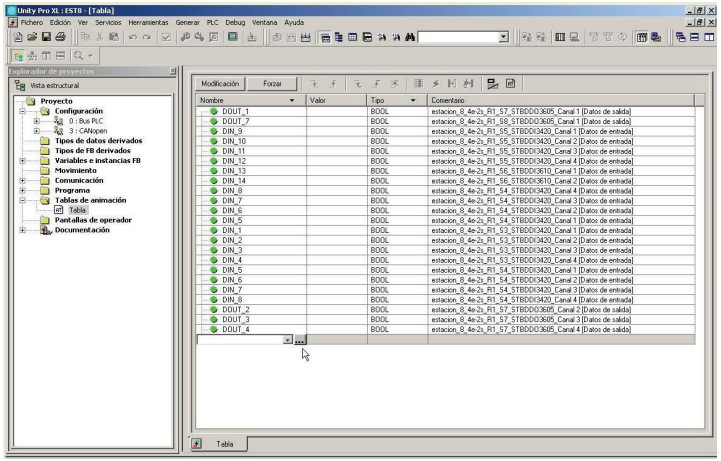
6. Prueba de señales y finalización de la configuración.

6.1. Creación de una tabla de animación.

Para probar las señales, vamos a requerir de una tabla de animación. Para ello vamos a la pestaña “Tablas de animación” y con el botón alternativo “Nueva tabla”.



Detalle “Nueva tabla”



Detalle de una tabla de animación.

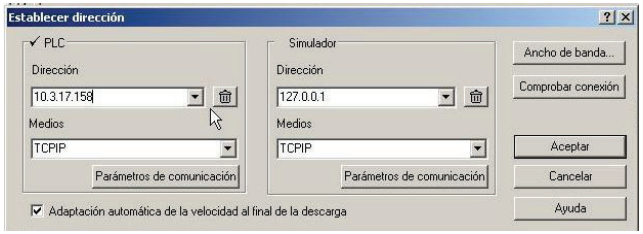
Una vez dado un nombre, nos disponemos a añadir las variables que deseemos probar mediante el desplegable que se nos ofrece.

Cuando terminemos, volvemos a generar cambios, si así nos lo permite Unity.

6.2. Conexión con el autómata, carga del programa, ejecución y comprobación.

Ya solo nos queda conectar con el autómata y probar nuestra tabla de animación.

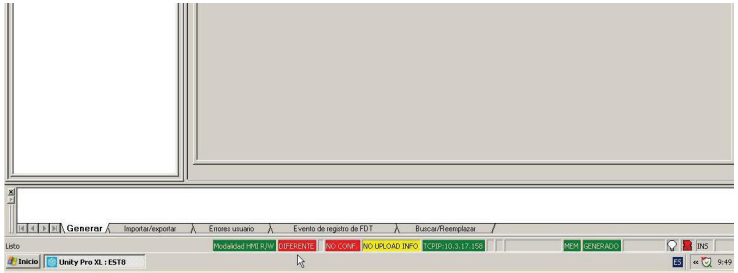
Para conectar, primero hemos de definir la dirección IP del autómata. Ello implica ir al menú "PLC" y seleccionar la pestaña correspondiente.



Detalle de direccionamiento.

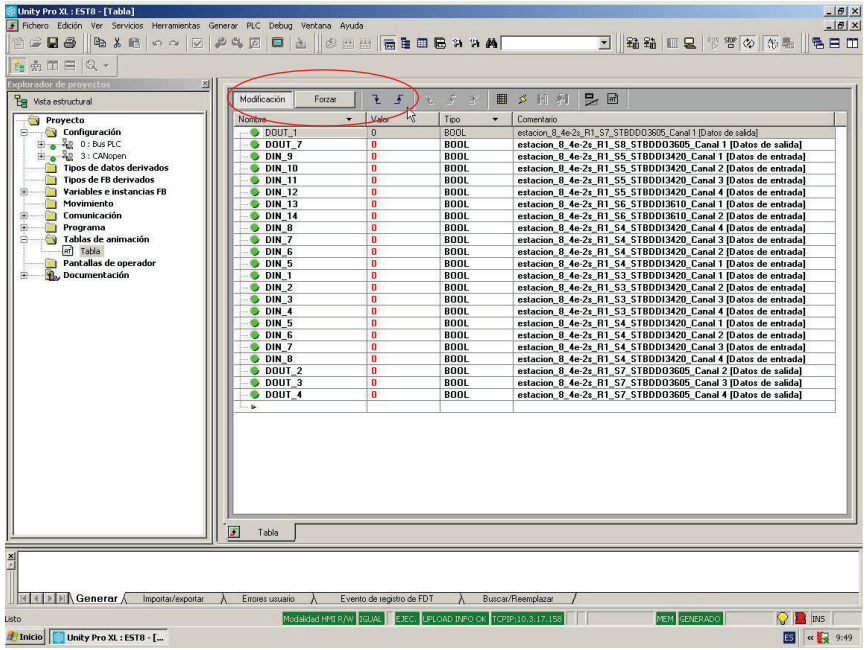
Una vez tengamos direccionado, clicaremos en "Conectar". Y si nadie está conectado al mismo, obtendremos la conexión.

Entonces lo más lógico es que abajo, nos aparezca un cartel que diga en rojo: "Diferente", esto quiere decir que el programa que tiene cargado el autómata es diferente de que hemos creado. Para cargar nuestro programa, "PLC/transferir".



Detalle de diferencia de programas.

Y una vez transferido, y si hemos hecho todo bien, podremos ejecutar el programa. Y mientras lo ejecutamos, en la tabla de animación podremos ver si cambian las variables correspondientes a los sensores, y nosotros podremos cambiar el valor de las salidas mediante la pestaña modificación y los botones de flanco de subida o de bajada.



Detalle de modificación de variables.

6.3 Errores habituales.

- Probando con una tabla de animación, algunas entradas no aparecen, y otras aparecen en desorden. (Y además las salidas tampoco funcionan bien todas).
  Es posible que exista una pequeña incompatibilidad entre Advantys y Unity. Lo primero a realizar es comprobar que el primer bloque de entradas y el primero de salidas (los correspondientes al primer módulo DDI y al primero DDO) funcionan correctamente.
  En caso de ser así, es muy probable que el problema radique en que las direcciones topológicas del resto de variables están mal asignadas. Nos toca entonces asignarlas "a mano". Para ello sirva el siguiente ejemplo. En él se considera una isla Advantys con 3 módulos DDI 3420, dos DDO 3600 y un DDO 3200.

	Mal asignadas		"Reasignación manual"	Otra opción
DDI 1	Entrada 1	%MW 35.0	%MW 35.0	%MW 35.0
	Entrada 2	%MW 35.1	%MW 35.1	%MW 35.1
	Entrada 3	%MW 35.2	%MW 35.2	%MW 35.2
	Entrada 4	%MW 35.3	%MW 35.3	%MW 35.3
DDI 2	Entrada 5	%MW 35.8	%MW 36.0	%MW 37.0
	Entrada 6	%MW 35.9	%MW 36.1	%MW 37.1
	Entrada 7	%MW 35.10	%MW 36.2	%MW 37.2
	Entrada 8	%MW 35.11	%MW 36.3	%MW 37.3
DDI 3	Entrada 9	%MW 36.0	%MW 37.0	%MW 39.0
	Entrada 10	%MW 36.1	%MW 37.1	%MW 39.1
	Entrada 11	%MW 36.2	%MW 37.2	%MW 39.2
	Entrada 12	%MW 36.3	%MW 37.3	%MW 39.3
DDO 1	Salida 1	%MW 134.0	%MW 134.0	%MW 134.0
	Salida 2	%MW 134.1	%MW 134.1	%MW 134.1
	Salida 3	%MW 134.2	%MW 134.2	%MW 134.2
	Salida 4	%MW 134.3	%MW 134.3	%MW 134.3
	Salida 5	%MW 134.4	%MW 134.4	%MW 134.4
	Salida 6	%MW 134.5	%MW 134.5	%MW 134.5
DDO 2	Salida 7	%MW 134.9	%MW 135.0	%MW 135.0
	Salida 8	%MW 134.10	%MW 135.1	%MW 135.1
	Salida 9	%MW 134.11	%MW 135.2	%MW 135.2
	Salida 10	%MW 134.12	%MW 135.3	%MW 135.3
	Salida 11	%MW 134.13	%MW 135.4	%MW 135.4
	Salida 12	%MW 134.14	%MW 135.5	%MW 135.5
DDO 3	Salida 13	%MW 134.16	%MW 135.6	%MW 135.6
	Salida 14	%MW 134.17	%MW 135.7	%MW 135.7

Tabla de ejemplo de reconfiguración manual de las variables.

En el ejemplo dado, lo más probable es que las señales pertenecientes al DDI 1 coincidieran con las que hemos probado, y las correspondientes al DDI 2 "parecieran" corresponder con las del DDI 3. Daría la sensación de que el programa se "comiera" un módulo. Por otra parte, solo debería funcionar el primero módulo de salidas, inutilizando los otros dos.

Se presentan dos opciones diferentes de direccionamiento, porque se ha comprobado que en diferentes ocasiones ha sido necesario emplear una de las dos. No obstante para un único caso, solo funcionará una de ellas.

En caso de que siga sin funcionar correctamente, será necesario probar más direccionamientos, o comprobar que no se trate de otro error.

Apéndice D

Manual de uso de E/S por red Ethernet

Se incluye en este apéndice el manual realizado para la configuración de las entradas y salidas digitales pertenecientes a diferentes elementos enclavados en una red Ethernet, a través de un módulo de comunicaciones «NOE 0100».
El interés de este manual radica fundamentalmente en la gran sencillez que conlleva utilizar el módulo Ethernet «NOE 0100» en lugar de las complicadas funciones internas de los autómatas «M340».

Guía de utilización de entradas y salidas mediante red Ethernet para su uso con un autómata M340, mediante software UNITY.

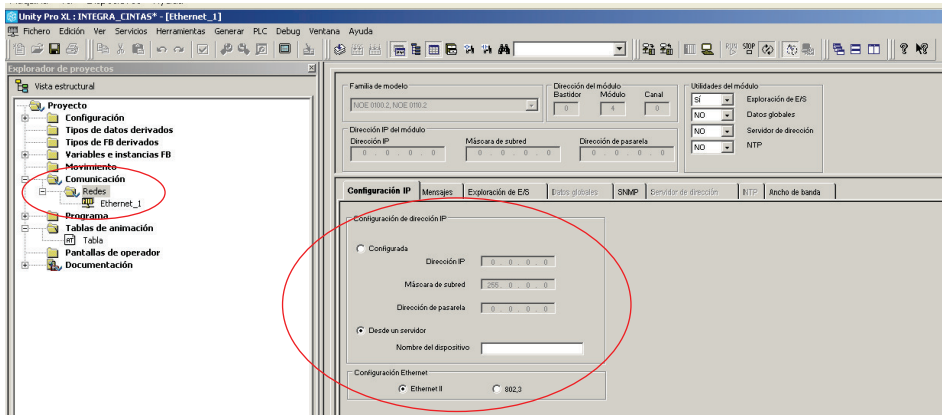
José Ignacio Abad Martínez.

A) Módulos TSX Momentum.

En primer lugar, para poder utilizar estos módulos, es necesario disponer de un módulo NOE de comunicaciones instalado en el autómata.

Después, aparte de las obligatorias configuraciones del propio autómata de cara a poder utilizar todos sus módulos, habrá que hacer lo siguiente con el programa Unity:

1. Configurar una red Ethernet. En el apartado de "Comunicaciones", añadimos red Ethernet. Con doble clic sobre ella, la configuraremos introduciendo la dirección IP correspondiente al módulo NOE. (Lo normal es que sea una IP fija, dependiente de la dirección MAC del módulo). Si está conectado a un servidor, entonces se selecciona dicha opción y se selecciona también "Ethernet".

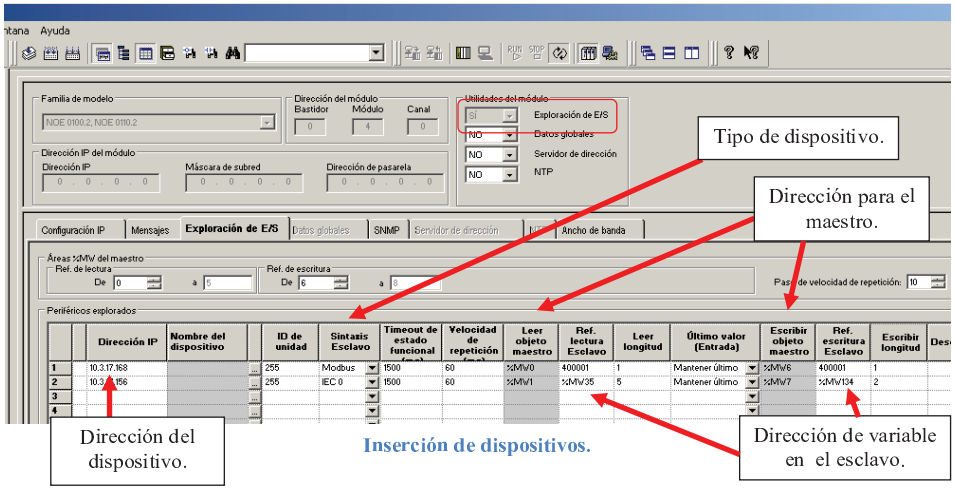


Configuración de la red Ethernet.

2. Una vez configurada la red Ethernet (y siempre en modo offline), cerramos la ventana de Ethernet y decimos que sí que queremos aplicar cambios. Ahora, clic en "generar/generar cambios".

3. Volvemos a entrar en la red Ethernet, y en la ventana de configuración, clicamos en que sí que deseamos permitir exploración de entradas y salidas. Una vez hecho esto y validados los cambios, nos permitirá el acceso a la pestaña "Exploración de E/S" en la misma ventana de configuración de Ethernet.

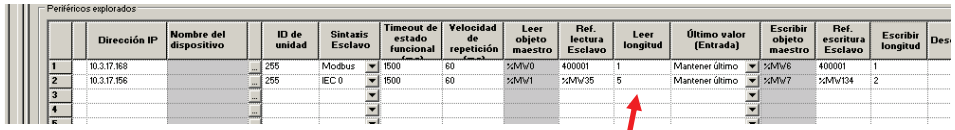
4. En la pestaña "Exploración de E/S" hay que añadir los diferentes esclavos que van a existir. En este caso, añadimos la IP del TSX Momentum, e indicamos que es del tipo Modbus. Indicamos a su vez que se han de leer longitud de entradas 1 y de salidas 1 también.



Inserción de dispositivos.

5. En la misma pestaña nos indicará qué direcciones del autómata van a ser las correspondientes con las del TSX Momentum. Por ejemplo: si indica que el registro de entradas va a ser el %MW1, y a su vez tenemos que la longitud de lectura es 1, tendremos entradas (si están cableadas) entre %MW1.0 y %MW1.15, todas correspondientes a entradas al TSX Momentum. De igual modo con las salidas.

Estas direcciones se pueden usar libremente como otras cualesquiera que hubiéramos introducido en Unity.



Longitud de palabra a leer. En el caso de un módulo TSX, cada palabra son hasta 16 variables (0 a 15).

B) Autómatas Modicon M340.

Los pasos a seguir en este caso son muy parecidos al anterior. En concreto, los tres primeros son los mismos.

Después, aparte de las obligatorias configuraciones del propio autómata de cara a poder utilizar todos sus módulos, habrá que hacer lo siguiente con el programa Unity:

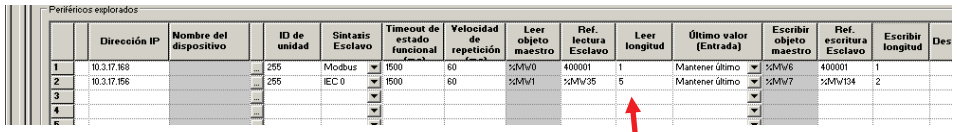
1. Configurar una red Ethernet. En el apartado de "Comunicaciones", añadimos red Ethernet. Con doble clic sobre ella, la configuraremos introduciendo la dirección IP correspondiente al módulo NOE. (Lo normal es que sea una IP fija, dependiente de la dirección MAC del módulo). Si está conectado a un servidor, entonces se selecciona dicha opción y se selecciona también "Ethernet".

2. Una vez configurada la red Ethernet (y siempre en modo offline), cerramos la ventana de Ethernet y decimo que sí que queremos aplicar cambios. Ahora, clic en "generar/generar cambios".

3. Volvemos a entrar en la red Ethernet, y en la ventana de configuración, clicamos en que sí que deseamos permitir exploración de entradas y salidas. Una vez hecho esto y validados los cambios, nos permitirá el acceso a la pestaña "Exploración de E/S" en la misma ventana de configuración de Ethernet.

4. En la pestaña "Exploración de E/S" hay que añadir los diferentes esclavos que van a existir. En este caso, añadimos la IP del M340 al que queramos acceder, e indicamos que es del tipo IEC0. Ahora habrá que indicar la cantidad de palabras que se desea leer (una por cada %MW que se desee consultar. Por ejemplo, si queremos leer desde %MW1 hasta %MW3, harán falta 3 palabras).

5. Ahora, en esta misma pestaña, Unity indica la equivalencia entre las variables del maestro y las del esclavo. Hay que tener mucho cuidado en no confundirlas, ya que en este caso ambas empiezan por "%MW". Pero las que podremos manejar en el programa maestro solo serán las del maestro.



Longitud de palabra a leer. En el caso de un autómata M340, la longitud es el mismo número de palabras %MW que queremos leer. Hay que recordar que unas palabras serán las del maestro y otras las del esclavo, no confundirlas.

Apéndice E

Programas de los autómatas

En este apéndice se detallan los diferentes programas que se han creado para la utilización de los autómatas empleados en este proyecto.

Están contenidos los resúmenes de los programas correspondientes al autómata que controla toda la célula cuando esta funciona de manera integrada, y al autómata que controla la estación de expedición de palets.
En ambos casos se incluyen las variables utilizadas, con su tipología y dirección de memoria; Las transiciones y acciones; Y los diagramas «SFC» que controlan su funcionamiento.

DOCUMENTACIÓN TÉCNICA

Proyecto

Proyecto	Estacion robot y cintas
Diseñador	José Ignacio Abad Martínez
Aplicación	DEMO_ESTACION_ROBOT_Y_CINTAS.stu
Versión del software	Unity Pro XL V6.0
Fecha de creación	10/02/2014 17:35:27
Fecha de la última modificación	10/02/2014 17:35:27
PLC de destino	BMX P34 20302 02.30CPU 340-20 Ethernet CANopen2

Variables e instancias FB

BOOL

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
avanzar_1_pos	NO				4	NO
Avanzar_puesto	NO	%MW37.0			6	NO
Carga_ready	NO				2	NO
cargador	NO	%MW37.1			4	NO
Cintas_1	NO	%MW7.11			3	NO
Cintas_2	NO	%MW7.12			3	NO
Descarga_ready	NO				2	NO
descargador	NO	%MW37.2			3	NO
descartador	NO	%MW37.3			2	NO
desmontador	NO	%MW37.4			3	NO
Desmontatren	NO				2	NO
Desvio_intermedio	NO	%MW7.4			1	NO
Detector_2	NO	%MW0.12			2	NO
Detector_3	NO	%MW0.13			2	NO
Detector_4	NO	%MW0.14			2	NO
Detenedor	NO	%MW37.10			3	NO
Elevador_1	NO	%MW7.0			2	NO
Elevador_2	NO	%MW7.1			2	NO
Elevador_3	NO	%MW7.2			2	NO
Elevador_4	NO	%MW7.3			2	NO
emergen	NO				2	NO
Ent_1	NO	%MW67.0	ADVANTYS_ROBOT_R1_S3_STBDDI3420_Canal 1 [Datos de entrada]		12	NO
Ent_2	NO	%MW67.1	ADVANTYS_ROBOT_R1_S3_STBDDI3420_Canal 2 [Datos de entrada]		8	NO
Ent_3	NO	%MW67.2	ADVANTYS_ROBOT_R1_S3_STBDDI3420_Canal 3 [Datos de entrada]		2	NO
Ent_4	NO	%MW67.3	ADVANTYS_ROBOT_R1_S3_STBDDI3420_Canal 4 [Datos de entrada]		1	NO
Ent_5	NO	%MW67.4	ADVANTYS_ROBOT_R1_S4_STBDDI3420_Canal 1 [Datos de entrada]		1	NO
Ent_6	NO	%MW67.5	ADVANTYS_ROBOT_R1_S4_STBDDI3420_Canal 2 [Datos de entrada]		1	NO
Ent_7	NO	%MW67.6	ADVANTYS_ROBOT_R1_S4_STBDDI3420_Canal 3 [Datos de entrada]		1	NO
Ent_8	NO	%MW67.7	ADVANTYS_ROBOT_R1_S4_STBDDI3420_Canal 4 [Datos de entrada]		2	NO
Ent_9	NO	%MW67.8	ADVANTYS_ROBOT_R1_S5_STBDDI3420_Canal 1 [Datos de entrada]		4	NO
Ent_10	NO	%MW67.9	ADVANTYS_ROBOT_R1_S5_STBDDI3420_Canal 2 [Datos de entrada]		5	NO
Ent_11	NO	%MW67.10	ADVANTYS_ROBOT_R1_S5_STBDDI3420_Canal 3 [Datos de entrada]		1	NO
Ent_12	NO	%MW67.11	ADVANTYS_ROBOT_R1_S5_STBDDI3420_Canal 4 [Datos de entrada]		1	NO
espera_1	NO				2	NO
expedir	NO	%MW10.1			2	NO
extrae_palet	NO	%MW41.1			2	NO
fin_cargado	NO				0	NO
fin_cargado1	NO				2	NO
fin_cargado_1	NO				2	NO
fin_paletizado	NO				3	NO
Finalizador	NO	%MW37.9			2	NO
Han_salido	NO				2	NO
IND_cintas	NO	%MW0.9			2	NO
Init	NO				2	NO
llegada	NO				2	NO
MAN_cintas	NO	%MW0.11			2	NO

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
Marcha_cintas	NO	%MW0.8			3	NO
marcha_soft	NO	%MW1.3			0	NO
Marcha_virtual	NO	%MW42.2			2	NO
Metal_1	NO	%MW0.0			2	NO
Metal_2	NO	%MW0.1			2	NO
Metal_3	NO	%MW0.2			2	NO
Metal_4	NO	%MW0.3			2	NO
Metal_int	NO	%MW0.5			1	NO
modo_auto	NO				4	NO
momentum_4	NO	%MW0.4			1	NO
momentum_6	NO	%MW0.6			1	NO
momentum_sal_10	NO	%MW7.10			1	NO
montador	NO	%MW37.5			2	NO
Montatren	NO				2	NO
palet_completo	NO				6	NO
paletizador	NO	%MW37.6			2	NO
Paletizar	NO				2	NO
permiso	NO	%MW41.0			2	NO
poner_palet	NO	%MW10.1			4	NO
Puente_din_0	NO	%MW38.0			1	NO
Puente_din_1	NO	%MW38.1			1	NO
Puente_din_2	NO	%MW38.2			0	NO
Puente_din_3	NO	%MW38.3			0	NO
Puente_din_4	NO	%MW38.4			0	NO
Puente_din_5	NO	%MW38.5			0	NO
Puente_din_6	NO	%MW38.6			0	NO
Puente_din_7	NO	%MW38.7			0	NO
Puente_din_8	NO	%MW38.8			0	NO
Puente_din_9	NO	%MW38.9			0	NO
Puente_din_10	NO	%MW38.10			0	NO
Puente_din_11	NO	%MW38.11			0	NO
Puente_din_12	NO	%MW38.12			0	NO
Puente_din_13	NO	%MW38.13			0	NO
Puente_din_14	NO	%MW38.14			0	NO
Puente_din_15	NO	%MW38.15			0	NO
Reset_cintas	NO	%MW0.10			3	NO
Reset_virtual	NO	%MW42.1			1	NO
Retencion_1	NO	%MW7.5			3	NO
Retencion_2	NO	%MW7.6			3	NO
Retencion_3	NO	%MW7.7			3	NO
Retencion_4	NO	%MW7.8			3	NO
Retencion_intermedia	NO	%MW7.9			1	NO
Robot_activo	NO				2	NO
Sal_1	NO	%MW166.0	ADVANTYS_ROBOT_R1_S6_STBDDO3605_Canal 1 [Datos de salida]		3	NO
Sal_2	NO	%MW166.1	ADVANTYS_ROBOT_R1_S6_STBDDO3605_Canal 2 [Datos de salida]		3	NO
Sal_3	NO	%MW166.2	ADVANTYS_ROBOT_R1_S6_STBDDO3605_Canal 3 [Datos de salida]		3	NO
Sal_4	NO	%MW166.3	ADVANTYS_ROBOT_R1_S6_STBDDO3605_Canal 4 [Datos de salida]		3	NO
Sal_5	NO	%MW166.4	ADVANTYS_ROBOT_R1_S6_STBDDO3605_Canal 5 [Datos de salida]		3	NO
Sal_6	NO	%MW166.5	ADVANTYS_ROBOT_R1_S6_STBDDO3605_Canal 6 [Datos de salida]		3	NO
Sal_7	NO	%MW166.6	ADVANTYS_ROBOT_R1_S7_STBDDO3605_Canal 1 [Datos de salida]		3	NO
Sal_8	NO	%MW166.7	ADVANTYS_ROBOT_R1_S7_STBDDO3605_Canal 2 [Datos de salida]		2	NO

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
Sal_9	NO	%MW166.8	ADVANTYS_ROBOT_R1_S7_STBDDO3605_Canal 3 [Datos de salida]		3	NO
Sal_10	NO	%MW166.9	ADVANTYS_ROBOT_R1_S7_STBDDO3605_Canal 4 [Datos de salida]		3	NO
Sal_11	NO	%MW166.10	ADVANTYS_ROBOT_R1_S7_STBDDO3605_Canal 5 [Datos de salida]		1	NO
Sal_12	NO	%MW166.11	ADVANTYS_ROBOT_R1_S7_STBDDO3605_Canal 6 [Datos de salida]		1	NO
Sal_13	NO	%MW166.12	ADVANTYS_ROBOT_R1_S8_STBDDO3200_Canal 1 [Datos de salida]		1	NO
Sal_14	NO	%MW166.13	ADVANTYS_ROBOT_R1_S8_STBDDO3200_Canal 2 [Datos de salida]		1	NO
selec_palet	NO	%MW10.2			2	NO
Selec_palet_hard	NO	%MW3.4			2	NO
Seta_cintas	NO	%MW0.7			2	NO
Seta_virtual	NO	%MW42.0			4	NO
T_desechar	NO				2	NO
trasladador	NO	%MW37.7			2	NO
Trasladar	NO				2	NO
Tren_montado	NO	%MW37.8			5	NO
verdadera	NO				1	NO

EBOOL

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
DIN_0	NO	%I0.1.0			2	NO
DIN_1	NO	%I0.1.1			2	NO
DIN_2	NO	%I0.1.2			2	NO
DIN_3	NO	%I0.1.3			2	NO
DIN_4	NO	%I0.1.4			2	NO
DIN_5	NO	%I0.1.5			2	NO
DIN_6	NO	%I0.1.6			3	NO
DIN_7	NO	%I0.1.7			3	NO
DIN_8	NO	%I0.1.8			0	NO
DIN_9	NO	%I0.1.9			1	NO
DIN_10	NO	%I0.1.10			2	NO
DIN_11	NO	%I0.1.11			0	NO
DIN_12	NO	%I0.1.12			0	NO
DIN_13	NO	%I0.1.13			0	NO
DIN_14	NO	%I0.1.14			2	NO
DIN_15	NO	%I0.1.15			2	NO

INT

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
carros_llenos	NO	%MW6			2	NO
hueco	NO	%MW39			8	NO
repeticiones	NO				0	NO

SFCCHART_STATE

Nombre	Const	Dirección	Comentario	Utilizado
Cintas	NO			1
Modo_automatico	NO			0
Red_Petri	NO			1

SFCSTEP_STATE

Nombre	Const	Dirección	Comentario	Utilizado
Apagado	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Utilizado
avance	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Avanzar	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
cargado_palet	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Cargando_palet_2	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Cargar	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Cargar_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Comienza_avance	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Descargar	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Descargar_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Desechar	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Desechar_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Desmontar_tren	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Utilizado
Desmontar_tren_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Emergencia	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Inicio	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Montar_tren	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Montar_tren_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Mover_cintas	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Paletiza	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Paletiza_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Primer_cargado	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Reposo	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Reposo_1	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Reposo_auto	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Utilizado
Robot_encendido	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Robot_preparado	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
S_3_8	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
solicitud_carga_del_pale t	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
terminando	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Trasl_cinta	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Trasl_cinta_2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			

Estructura de la aplicación

VISTA ESTRUCTURAL

SECCIÓN	CONDICIÓN DE VALIDACIÓN	COMENTARIO DE SECCIÓN	MÓDULO	LENGUAJE
Codigo				ST
Red_Petri				SFC
Chart				SFC
huecos_menos_3				ST
hueco_mas_3				ST
hueco_mas_1				ST
Robot_activo				ST
Montatren				ST
Desmontatren				ST
Paletizar				ST
Trasladar				ST
T_desechar				ST
emergen				ST
Carga_ready				ST
Descarga_ready				ST
Cintas				SFC
Chart				SFC
verdadera				ST
llegada				ST
Han_salido				ST
Modo_automatico				SFC
Chart				SFC
fin_cargado1				ST
fin_cargado_1				ST
espera_1				ST

CALL TREE

```
Programa
└─ Tareas
    └─ MAST
        └─ Secciones
            ├─ Codigo
            ├─ Red_Petri
            │   ├─ Chart
            │   ├─ Acciones
            │   │   ├─ huecos_menos_3
            │   │   ├─ hueco_mas_3
            │   │   └─ hueco_mas_1
            │   └─ Transiciones
            │       ├─ Robot_activo
            │       ├─ Montatren
            │       ├─ Desmontatren
            │       ├─ Paletizar
            │       ├─ Trasladar
            │       ├─ T_desechar
            │       ├─ emergen
            │       ├─ Carga_ready
            │       └─ Descarga_ready
            ├─ Cintas
            │   ├─ Chart
            │   └─ Transiciones
            │       ├─ verdadera
            │       ├─ llegada
            │       └─ Han_salido
            └─ Modo_automatico
                ├─ Chart
                └─ Transiciones
```

Estructura de la aplicación

```
├─ fin_cargado1
├─ fin_cargado_1
└─ espera_1
```

Tipo de red: Ethernet	Familia: Ethernet_Micro_Basic_NOE_V2	Nombre: Ethernet_1
Comentario:		
Red conectada: SÍ	Módulo de dirección: \0.0\0.4.0	

Configuración IP

Configuración de dirección IP	Desde un servidor

Configuración Ethernet	Ethernet II

Mensajes

Configuración de conexión

Control de acceso:	Bloquear

Exploración de E/S

Leer ref.	De	0
	a	6
Escribir ref.	De	7
	a	10

E/S explorados

Dirección IP	Estado del dispositivo	Nombre del dispositivo	Tipo de dispositivo	Longitud de ENTRADA necesaria	Longitud de SALIDA necesaria	ID de unidad
10.3.17.168	No hay dispositivos			0	0	255
10.3.17.156	No hay dispositivos			0	0	255

Dirección IP	Sintaxis de esclavo	Timeout de perturbación (ms)	Velocidad de repetición (ms)	Leer objeto maestro	Leer Ref. de esclavo	Leer longitud
10.3.17.168	Modbus	1500	60	%MW0	400001	1
10.3.17.156	IEC 0	1500	60	%MW1	%MW35	6

Dirección IP	Último valor (entrada)	Escribir objeto maestro	Escribir Ref. de esclavo	Escribir longitud	Descripción
10.3.17.168	Mantener último	%MW7	400001	1	

10.3.17.156	Mantener último	%MW8	%MW134	3

SNMP Ethernet

Administradores de dirección IP

Administrador 1 de dirección IP: 0.0.0.0 Administrador 2 de dirección IP: 0.0.0.0

Agente

Localización (SysLocation):
Contacto (SysContact):
Administrador SNMP : Bloquear

Nombres de comunidad	Ajustar:	public
	Obtener:	public
	Captura:	public
Seguridad	Habilitar captura de errores de autenticación :	Bloquear

Ancho de banda

Datos globales Información 0 Datos globales estimados (/s)

Información de mensajes 0 Mensajes estimados (/s)

Entorno Ethernet: 0

MAST

Propiedades específicas

Configuración	Cíclica
Configuración del periodo de tareas	0
Configuración del tiempo de watchdog	250

Codigo : [MAST]

```
1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      113|

Sal_8:=Ent_8;
puente_din_0:=din_0;
seta_virtual:= seta_virtual OR seta_cintas;

If MAN_cintas=false then
Sal_6:=DIN_6;
sal_7:=DIN_7;
Finalizador:=Din_6;
Detenedor:=Din_7;
trasladador:=Din_1;
paletizador:=Din_2;
descartador:=Din_3;
montador:=Din_4;
desmontador:=Din_5;
cargador:=Din_14;
Descargador:=Din_15;
selec_palet:=selec_palet_hard;
avanzar_1_pos:=DIN_10 OR avanzar_puesto;
reset_virtual:=Reset_cintas;
avanzar_puesto:=marcha_cintas;

else
marcha_virtual:=marcha_cintas;
Sal_6:=finalizador;
sal_7:=detenedor;
expedir:= extrae_palet OR poner_palet;
avanzar_1_pos:=avanzar_puesto;

END_IF;

If (IND_cintas) then
        if (marcha_virtual)then
        modo_auto:=true;
        end_if;
(*modo_auto:=false;*)
end_if;

if carros_llenos <4 then
permiso:=true;
else
permiso:=false;
end_if;

If (seta_virtual) then
detenedor:=true;
Init:=initchart(Red_Petri, true);
Init:=initchart(Cintas, true);
end_if;

If (reset_cintas) then
seta_virtual:=0;
end_if;
```

Red_Petri : [MAST]

Comentario

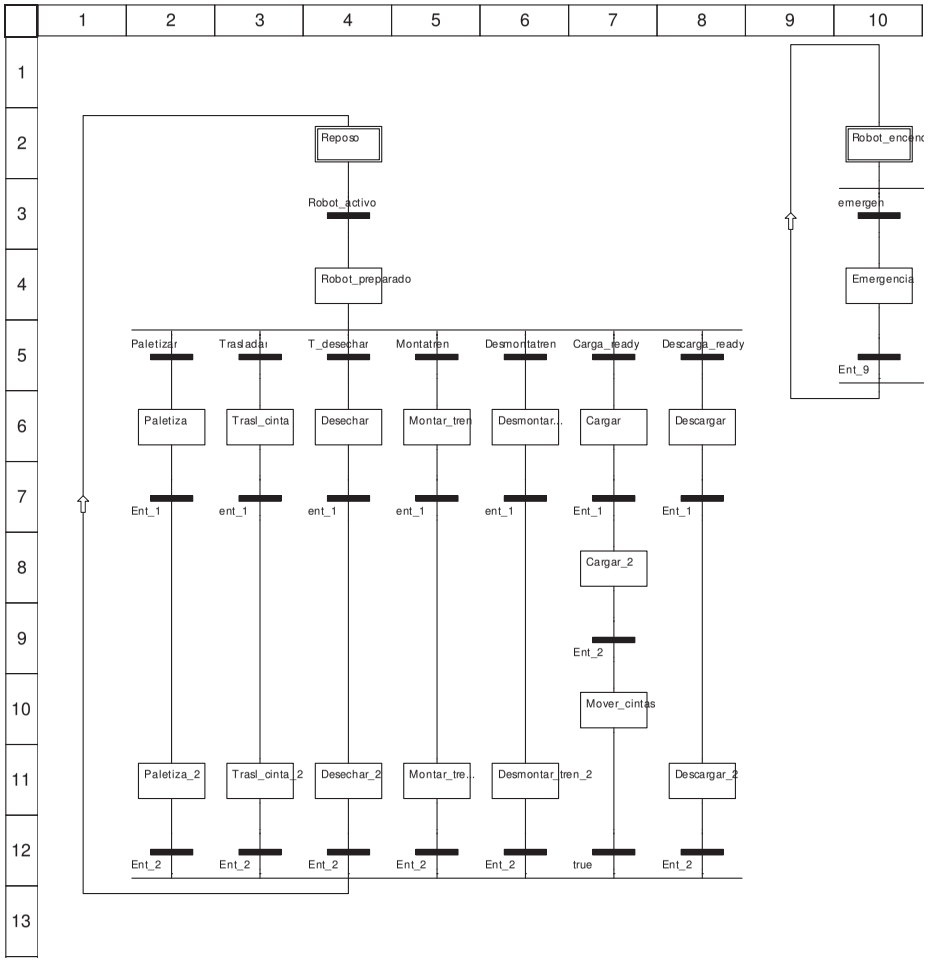
Propiedades comunes

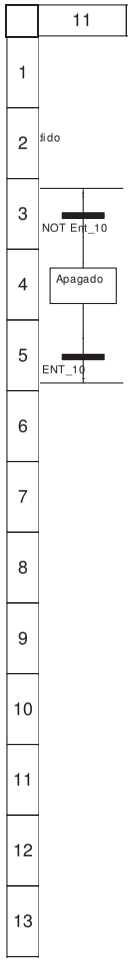
Módulo funcional	
Nombre de la condición	

Propiedades específicas

Control de operador	No
Número de área	0

Chart : [MAST - Red_Petri]





Descripción de objeto

Pasos:

Apagado	(11, 4)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Cargar			(7, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_9	
Descriptor: P1	Tiempo:	Sección: ST :: hueco_mas_3	
Descriptor: S	Tiempo:	Variable: palet_completo	

Cargar_2			(7, 8)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: sal_9	

Descargar			(8, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_10	
Descriptor: P1	Tiempo:	Sección: ST :: huecos_menos_3	

Descargar_2			(8, 11)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_10	

Desechar			(4, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_3	
Descriptor: P1	Tiempo:	Sección: ST :: hueco_mas_1	

Desechar_2			(4, 11)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_3	

Desmontar_tren			(6, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_5	
Descriptor: R	Tiempo:	Variable: Tren_montado	

Desmontar_tren_2			(6, 11)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_5	

Emergencia			(10, 4)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			

Montar_tren			(5, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_4	
Descriptor: S	Tiempo:	Variable: Tren_montado	

Montar_tren_2			(5, 11)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_4	

Mover_cintas			(7, 10)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: P	Tiempo:	Variable: avanzar_puesto	
Descriptor: R	Tiempo:	Variable: palet_completo	

Paletiza			(2, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_2	
Descriptor: P1	Tiempo:	Sección: ST :: hueco_mas_1	

Paletiza_2			(2, 11)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_2	

Reposo (paso inicial)			(4, 2)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			

Robot_encendido (paso inicial)			(10, 2)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			

Robot_preparado			(4, 4)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			

Trasl_cinta			(3, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_1	
Descriptor: P1	Tiempo:	Sección: ST :: hueco_mas_1	

Trasl_cinta_2			(3, 11)

Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Sal_1	

Transiciones:

Nombre	Tipo de condición	Posición	Comentario
ST :: Carga_ready	Sección	(7, 5)	
ST :: Descarga_ready	Sección	(8, 5)	
ST :: Desmontatren	Sección	(6, 5)	
ENT_10	Variable	(11, 5)	
Ent_1	Variable	(2, 7)	
Ent_1	Variable	(7, 7)	
Ent_1	Variable	(8, 7)	
Ent_2	Variable	(2, 12)	
Ent_2	Variable	(3, 12)	
Ent_2	Variable	(4, 12)	
Ent_2	Variable	(5, 12)	
Ent_2	Variable	(6, 12)	
Ent_2	Variable	(7, 9)	
Ent_2	Variable	(8, 12)	
Ent_9	Variable	(10, 5)	
ST :: Montatren	Sección	(5, 5)	
NOT Ent_10	Variable	(11, 3)	
ST :: Paletizar	Sección	(2, 5)	
ST :: Robot_activo	Sección	(4, 3)	
ST :: T_desechar	Sección	(4, 5)	
ST :: Trasladar	Sección	(3, 5)	
ST :: emergen	Sección	(10, 3)	
ent_1	Variable	(3, 7)	
ent_1	Variable	(4, 7)	
ent_1	Variable	(5, 7)	
ent_1	Variable	(6, 7)	
true	Constante	(7, 12)	

Cintas : [MAST]

Comentario

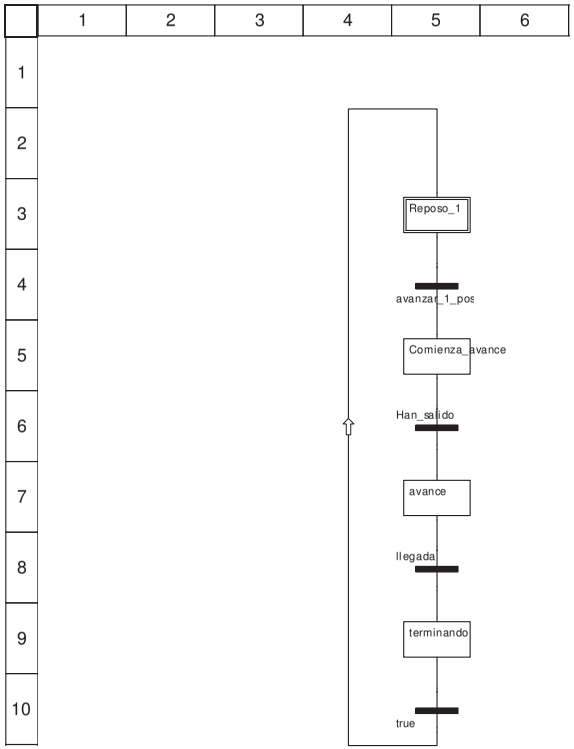
Propiedades comunes

Módulo funcional	
Nombre de la condición	

Propiedades específicas

Control de operador	No
Número de área	0

Chart : [MAST - Cintas]



Descripción de objeto

Pasos:

Comienza_avance		(5, 5)
Tiempo de supervisión mín./máx.:		Tiempo de retardo: t#5s
Comentario:		
Acciones:		
Descriptor: S	Tiempo:	Variable: Retencion_1
Descriptor: S	Tiempo:	Variable: Retencion_2
Descriptor: S	Tiempo:	Variable: Retencion_3
Descriptor: S	Tiempo:	Variable: Retencion_4
Descriptor: S	Tiempo:	Variable: Cintas_2
Descriptor: S	Tiempo:	Variable: Cintas_1

Reposo_1 (paso inicial)	(5, 3)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Acciones:		
Descriptor: None	Tiempo:	Variable: Elevador_1
Descriptor: None	Tiempo:	Variable: Elevador_2
Descriptor: None	Tiempo:	Variable: Elevador_3
Descriptor: None	Tiempo:	Variable: Elevador_4
Descriptor: R	Tiempo:	Variable: cintas_1
Descriptor: R	Tiempo:	Variable: cintas_2

avance		(5, 7)
Tiempo de supervisión mín./máx.:		Tiempo de retardo: t#1s
Comentario:		
Acciones:		
Descriptor: R	Tiempo:	Variable: retencion_4
Descriptor: R	Tiempo:	Variable: retencion_3
Descriptor: R	Tiempo:	Variable: Retencion_2
Descriptor: R	Tiempo:	Variable: Retencion_1
Descriptor: R	Tiempo:	Variable: palet_completo

terminando		(5, 9)
Tiempo de supervisión mín./máx.:		Tiempo de retardo: t#1s
Comentario:		
Acciones:		
Descriptor: R	Tiempo:	Variable: palet_completo
Descriptor: N	Tiempo:	Variable: fin_paletizado

Transiciones:

Nombre	Tipo de condición	Posición	Comentario
ST :: Han_salido	Sección	(5, 6)	
avanzar_1_pos	Variable	(5, 4)	
ST :: llegada	Sección	(5, 8)	
true	Constante	(5, 10)	

Modo_automatico : [MAST]

Comentario

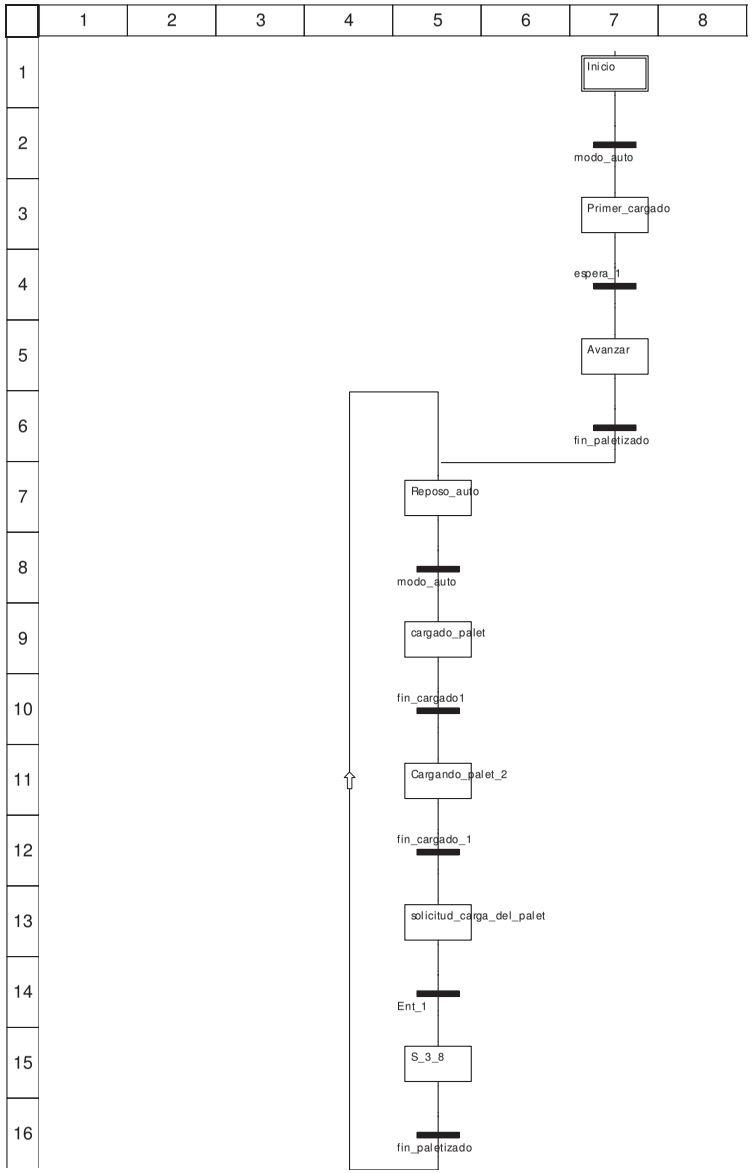
Propiedades comunes

Módulo funcional	
Nombre de la condición	

Propiedades específicas

Control de operador	No
Número de área	0

Chart : [MAST - Modo_automatico]



	1	2	3	4	5	6	7	8

Descripción de objeto

Pasos:

Avanzar		(7, 5)
Tiempo de supervisión mín./máx.:		Tiempo de retardo: t#3s
Comentario:		
Acciones:		
Descriptor: P	Tiempo:	Variable: Avanzar_puesto

Cargando_palet_2	(5, 11)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Inicio (paso inicial)	(7, 1)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Primer_cargado		(7, 3)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: N	Tiempo:	Variable: poner_palet

Reposo_auto	(5, 7)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

S_3_8	(5, 15)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

cargado_palet		(5, 9)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: poner_palet

solicitud_carga_del_palet		(5, 13)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: N	Tiempo:	Variable: cargador

Transiciones:

Nombre	Tipo de condición	Posición	Comentario
Ent_1	Variable	(5, 14)	
ST :: espera_1	Sección	(7, 4)	
ST :: fin_cargado1	Sección	(5, 10)	
ST :: fin_cargado_1	Sección	(5, 12)	
fin_paletizado	Variable	(5, 16)	
fin_paletizado	Variable	(7, 6)	
modo_auto	Variable	(5, 8)	
modo_auto	Variable	(7, 2)	

DOCUMENTACIÓN TÉCNICA

Estación

Proyecto	Estacion expendedora de palets
Diseñador	José Ignacio Abad Martínez
Aplicación	PRACTICA_SCADA_ESTACION_6.stu
Versión del software	Unity Pro XL V6.0
Fecha de creación	05/08/2013 14:37:31
Fecha de la última modificación	29/11/2013 18:22:12
PLC de destino	BMX P34 20302 02.30CPU 340-20 Ethernet CANopen2

Variables e instancias FB

BOOL

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
adelanteydcha	NO				2	NO
Bajar_brazo	NO	%MW134.3	celula_6_R1_S6_STBDDO3600_Canal 4 [Datos de salida]		4	NO
baliza	NO	%MW135.2	celula_6_R1_S7_STBDDO3600_Canal 3 [Datos de salida]		2	NO
Blanco	NO				2	NO
Brazo_abajo	NO	%MW35.5	celula_6_R1_S3_STBDDI3610_Canal 6 [Datos de entrada]		3	NO
Brazo_adelante	NO	%MW39.2	celula_6_R1_S5_STBDDI3420_Canal 3 [Datos de entrada]		2	NO
Brazo_arriba	NO	%MW35.4	celula_6_R1_S3_STBDDI3610_Canal 5 [Datos de entrada]		3	NO
Brazo_atras	NO	%MW39.1	celula_6_R1_S5_STBDDI3420_Canal 2 [Datos de entrada]		2	NO
Brazo_dcha	NO	%MW37.0	celula_6_R1_S4_STBDDI3610_Canal 1 [Datos de entrada]		3	NO
Brazo_izda	NO	%MW37.1	celula_6_R1_S4_STBDDI3610_Canal 2 [Datos de entrada]		2	NO
Coger_placa	NO	%MW134.2	celula_6_R1_S6_STBDDO3600_Canal 3 [Datos de salida]		5	NO
Dcho_atras	NO	%MW35.2	celula_6_R1_S3_STBDDI3610_Canal 3 [Datos de entrada]		1	NO
Expulsar_dcho	NO	%MW134.1	celula_6_R1_S6_STBDDO3600_Canal 2 [Datos de salida]		2	NO
Expulsar_izdo	NO	%MW134.0	celula_6_R1_S6_STBDDO3600_Canal 1 [Datos de salida]		2	NO
ind_int_soft	NO	%MW136.2			3	NO
Initstate	NO				1	NO
INT_IND	NO	%MW39.0	celula_6_R1_S5_STBDDI3420_Canal 1 [Datos de entrada]		2	NO
Izdo_atras	NO	%MW35.0	celula_6_R1_S3_STBDDI3610_Canal 1 [Datos de entrada]		1	NO
Man_Aut	NO	%MW37.4	celula_6_R1_S4_STBDDI3610_Canal 5 [Datos de entrada]		2	NO
man_soft	NO	%MW136.0			1	NO
Marcha	NO	%MW37.3	celula_6_R1_S4_STBDDI3610_Canal 4 [Datos de entrada]		2	NO
marcha_soft	NO	%MW136.1			3	NO
Mover_adelante	NO	%MW135.0	celula_6_R1_S7_STBDDO3600_Canal 1 [Datos de salida]		2	NO
Mover_atras	NO	%MW135.1	celula_6_R1_S7_STBDDO3600_Canal 2 [Datos de salida]		2	NO
Mover_dcha	NO	%MW134.5	celula_6_R1_S6_STBDDO3600_Canal 6 [Datos de salida]		3	NO
Mover_izda	NO	%MW134.4	celula_6_R1_S6_STBDDO3600_Canal 5 [Datos de salida]		2	NO
Negro	NO				2	NO
Optico_dcho	NO	%MW35.3	celula_6_R1_S3_STBDDI3610_Canal 4 [Datos de entrada]		2	NO
Optico_izdo	NO	%MW35.1	celula_6_R1_S3_STBDDI3610_Canal 2 [Datos de entrada]		2	NO
reposar	NO				2	NO
Reset	NO	%MW37.5	celula_6_R1_S4_STBDDI3610_Canal 6 [Datos de entrada]		3	NO
reset_soft	NO	%MW136.3			1	NO
Seta_emergencia	NO	%MW37.2	celula_6_R1_S4_STBDDI3610_Canal 3 [Datos de entrada]		7	NO
seta_soft	NO	%MW136.4			10	NO

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
Vacio	NO	%MW39.3	celula_6_R1_S5_STBDDI3420_Canal 4 [Datos de entrada]		2	NO

INT

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
carros_llenos	NO	%MW40			3	NO
Estado_estacion	NO	%MW137			3	NO

SFCCHART_STATE

Nombre	Const	Dirección	Comentario	Utilizado
Practica2_Est6_Inicial	NO			2

SFCSTEP_STATE

Nombre	Const	Dirección	Comentario	Utilizado
Adelante_y_dcha	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Bajar	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Bajar_1	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Coger	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Expulsar_negro	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Ir_dcha	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Ir_izda	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Pallet_blanco	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Pallet_negro	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Reposo	NO			3

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Utilizado
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Sacar_blanco	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Subir	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Volver	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			

TON

Nombre	Comentario	Valor	Utilizado	DG
TON_0			0	
<entradas>				
IN	Start delay			
PT	Preset delay time			
<salidas>				
Q	Delayed output			
ET	Internal time			

Estructura de la aplicación

VISTA ESTRUCTURAL

SECCIÓN	CONDICIÓN DE VALIDACIÓN	COMENTARIO DE SECCIÓN	MÓDULO	LENGUAJE
Practica2_Est6_Inicial				SFC
Chart				SFC
llenar_carro				ST
Blanco				ST
Negro				ST
reposar				ST
adelanteydcha				ST
Conexiones				ST
Reseteo				ST

CALL TREE

```
Programa
└─ Tareas
   └─ MAST
      └─ Secciones
         ├─ Practica2_Est6_Inicial
         │  ├─ Chart
         │  ├─ Acciones
         │  │  └─ llenar_carro
         │  └─ Transiciones
         │     ├─ Blanco
         │     ├─ Negro
         │     ├─ reposar
         │     └─ adelanteydcha
         ├─ Conexiones
         └─ Reseteo
```

Tipo de red: Ethernet Familia: Ethernet_Micro_Basic_NOE Nombre: Ethernet_1
Comentario:
Red conectada: NO Módulo de dirección:

Configuración IP

Configuración de dirección IP	Configurado
Dirección IP:	10.3.17.156
Máscara de subred:	255.255.255.0
Dirección de pasarela:	10.3.17.254
Configuración Ethernet	Ethernet II

Mensajes

Configuración de conexión

Control de acceso: Bloquear

SNMP Ethernet

Administradores de dirección IP

Administrador 1 de dirección IP: 0.0.0.0 Administrador 2 de dirección IP: 0.0.0.0

Agente

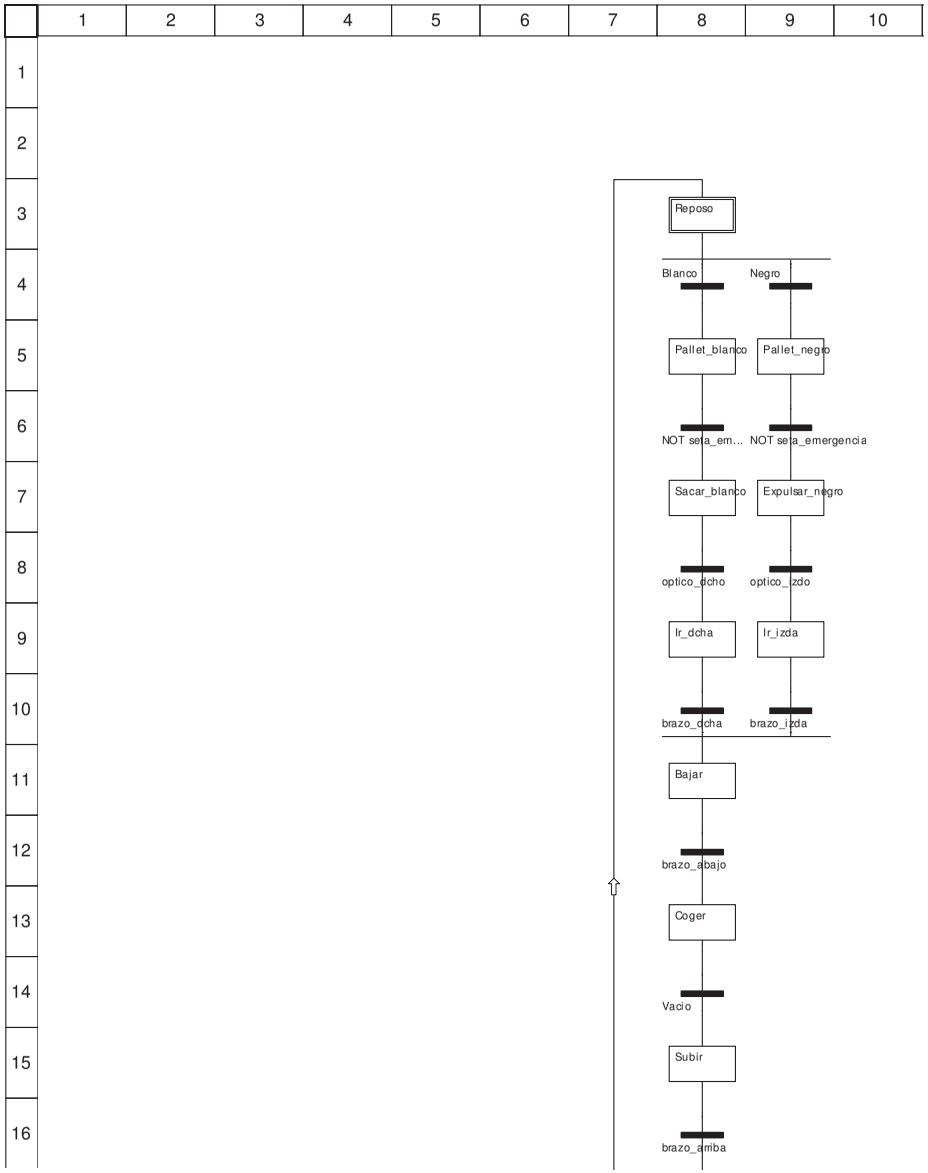
Localización (SysLocation):
Contacto (SysContact):
Administrador SNMP : Bloquear

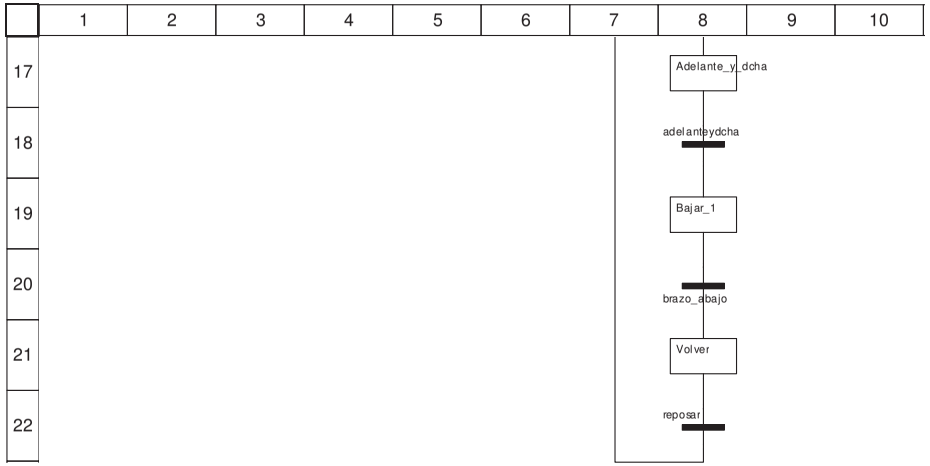
Nombres de comunidad	Ajustar:	public
	Obtener:	public
	Captura:	public
Seguridad	Habilitar captura de errores de autenticación :	Bloquear

Ancho de banda

Datos globales Información	0	Datos globales estimados (/s)
Información de mensajes	0	Mensajes estimados (/s)
Entorno Ethernet:	0	

Chart : [MAST - Practica2_Est6_Inicial]





Descripción de objeto

Pasos:

Adelante_y_dcha			(8, 17)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: Mover_adelante	
Descriptor: None	Tiempo:	Variable: coger_placa	
Descriptor: None	Tiempo:	Variable: mover_dcha	

Bajar			(8, 11)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: bajar_brazo	

Bajar_1			(8, 19)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: bajar_brazo	
Descriptor: None	Tiempo:	Variable: coger_placa	

Coger			(8, 13)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Variable: coger_placa	

Descriptor: None	Tiempo:	Variable: bajar_brazo

Expulsar_negro	(9, 7)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: None	Tiempo:	Variable: Expulsar_izdo

Ir_dcha	(8, 9)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: None	Tiempo:	Variable: Mover_dcha

Ir_izda	(9, 9)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: None	Tiempo:	Variable: Mover_izda

Pallet_blanco	(8, 5)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Pallet_negro	(9, 5)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Reposo (paso inicial)	(8, 3)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Sacar_blanco	(8, 7)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: None	Tiempo:	Variable: Expulsar_dcho

Subir	(8, 15)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: None	Tiempo:	Variable: coger_placa

Volver	(8, 21)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: None	Tiempo:	Variable: mover_atras
Descriptor: P1	Tiempo:	Sección: ST :: llenar_carro

Transiciones:

Nombre	Tipo de condición	Posición	Comentario
ST :: Blanco	Sección	(8, 4)	
NOT seta_emergencia	Variable	(8, 6)	
NOT seta_emergencia	Variable	(9, 6)	
ST :: Negro	Sección	(9, 4)	
Vacio	Variable	(8, 14)	
ST :: adelanteydcha	Sección	(8, 18)	
brazo_abajo	Variable	(8, 12)	
brazo_abajo	Variable	(8, 20)	
brazo_arriba	Variable	(8, 16)	
brazo_dcha	Variable	(8, 10)	
brazo_izda	Variable	(9, 10)	
optico_dcho	Variable	(8, 8)	
optico_izdo	Variable	(9, 8)	
ST :: reposar	Sección	(8, 22)	

Conexiones : [MAST]

```
if NOT Man_Aut then
        marcha_soft:=marcha;
        reset_soft:=reset;
        ind_int_soft:=INT_IND;
        seta_soft:=seta_emergencia OR seta_soft;
else
        seta_soft:=seta_emergencia OR seta_soft;
end_if;

if reset then
seta_soft:=0;
end_if;

if seta_soft then
baliza:=%S6;
end_if;

if (Reposo.x = 1) then
Estado_estacion:=0;
end_if;

if (Seta_emergencia) then
Estado_estacion:=2;
end_if;

if (Reposo.x = 0)and not Seta_emergencia then
Estado_estacion:=1;
end_if;
```

Reseteo : [MAST]

```
if seta_soft then
Initstate := INITCHART (Practica2_Est6_Inicial, seta_soft);
end_if;
```

Apéndice F

Manual de uso de Robot-Studio

Se detalla a continuación el manual de uso que se realizó para uso interno de la Universidad de Zaragoza sobre el software Robot-Studio.
En este manual se incluyen detalladamente todos los pasos necesarios para la utilización básica del programa. Desde la instalación y primera ejecución, hasta la grabación de vídeos de la programación realizada.

Tutorial de Robot Studio

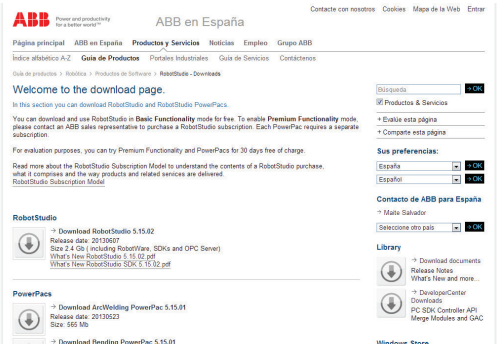
Simulador de robots Industriales

José Ignacio Abad Martínez

0. Instalación del programa.

Desde la página de descarga de software de ABB, descargamos el programa Robot Studio, que ocupa aproximadamente 2.4 Gb.

Una vez descargado, realizamos una instalación completa, incluyendo necesariamente el programa Robot Ware, sin el cual no funcionará Robot Studio.



La página es:

http://www.abb.es/product/ap/seitp327/a1078d9e5145d9a9c1257991004f3e69.aspx

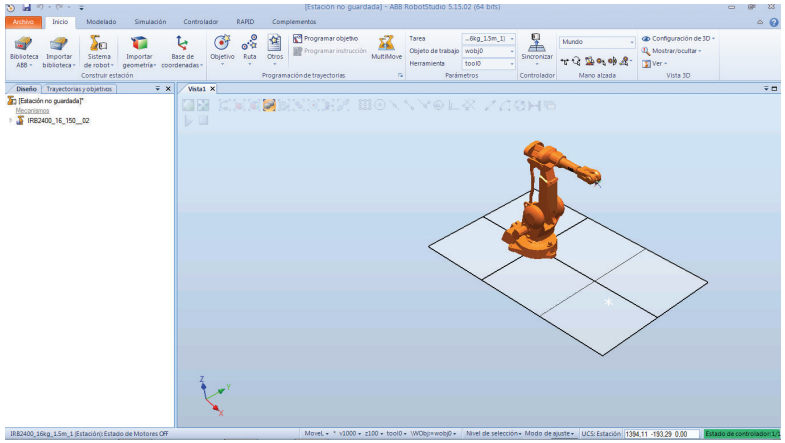
1. Crear nuevo escenario.

El primer paso a realizar cuando queramos trabajar con este simulador es o bien abrir un escenario básico, o crearlo.

En el caso de tener que crearlo, en la pantalla de inicio, seleccionamos "Estación con controlador de robot". Y elegimos el robot que más se adecúe a nuestras necesidades de todos los que aparecerán a la derecha de la pantalla. Sus principales características son el tamaño y la capacidad de carga.

Una vez hayamos seleccionado de la biblioteca uno que nos interese, nos aparecerá una pantalla como la siguiente:

Pudiera ser que nos deje elegir controlador, elegimos uno cualquiera, ya que es una aplicación virtual. Nos importaría si tuviéramos que comprar el robot.



Mediante la pestaña "Importar biblioteca", podremos añadir nuevos componentes a nuestro escenario, tales como un soporte para nuestro robot o una herramienta.

Vamos a añadir una herramienta:

> Clic en "Importar biblioteca" / "Equipamiento" / "Tools", y seleccionamos por ejemplo la pistola "Binzel Air 22".

Nos aparecerá en el origen de coordenadas del robot, lo cual la hace un poco inaccesible. Ahora vamos a colocársela al robot en el extremo de la muñeca.

> En la pestaña "Diseño" (A la izquierda de la pantalla), clic alternativo sobre la herramienta (Siempre aparecen con un icono de llave inglesa), y seleccionamos "conectar a". Seleccionamos nuestro robot, y le decimos que sí que deseamos actualizar la posición.

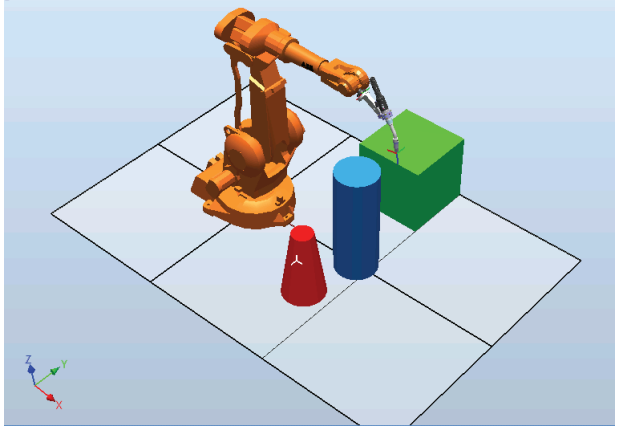
Ahora la herramienta está situada en su sitio, y el TCP (Tool Center Point) se ha actualizado solo.

Para completar un poco más nuestro escenario, vamos a añadir algún objeto. Para ello, en la pestaña de modelado hacemos clic en "Sólido", y seleccionamos por ejemplo un hexaedro (aunque el programa lo llama tetraedro).

Por hacerlo sencillo, creamos un hexaedro de 500 mm de lado, en la posición (500, 500, 0), de la referencia mundo. Una vez pre-visualizado, le damos a "Crear".

Para los sólidos, hay diversas opciones como colorearlos, unirlos con otros, hacer intersecciones… El sistema con el que se trabajan los sólidos 3D es bastante intuitivo, por lo que se anima a probar diferentes posibilidades.

Podemos probar a realizar un escenario simple del estilo que se puede observar a continuación:



El tronco de cono es algo más complicado de realizar, pero se anima a intentarlo mediante un cono y una resta con cualquier otro objeto o superficie.

1.1 Cambio de vista.

Para los cambios de vista hay fundamentalmente 3 posibilidades:

> Desplazamiento lateral. Pulsando "Control" y clic izquierdo del ratón, arrastramos la vista mediante el movimiento del ratón.
>
> Zoom. Pulsando "Control" y arrastrando mientras se pulsa el botón alternativo del ratón. O mediante la ruleta del ratón (si la tiene).
>
> Cambio de vista 3D. Pulsando el botón central del ratón y el alternativo, mediante el movimiento del ratón podemos cambiar la vista.

2. Movimiento.

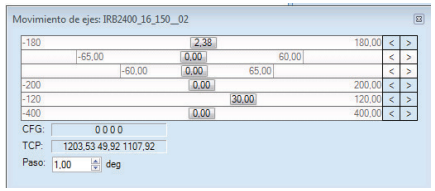
Ahora hay que pasar a la parte más importante de un simulador de robots, el movimiento del robot.

Fundamentalmente robot Studio permite dos movimientos diferentes, el movimiento de un eje (De tipo articular) y el movimiento lineal del TCP.

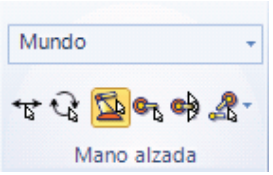
A su vez, cada uno de ellos nos ofrece dos opciones para realizarlo.

Si hacemos clic en la pestaña "Trayectorias y objetivos", y luego clic derecho sobre nuestro robot (T_ROB…), nos da dos opciones de movimiento, ambas mediante una

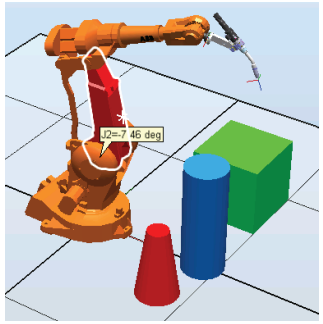
interfaz. Una es “Movimiento de ejes de mecanismo” y la otra “Movimiento lineal del mecanismo”.



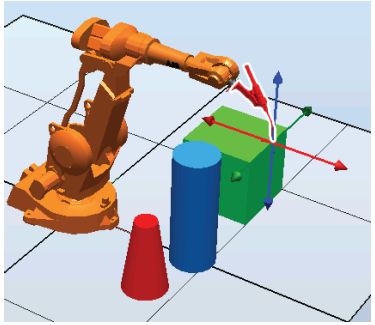
La otra opción está en la barra de herramientas principal. En el menú “Inicio”.



Nuevamente está la opción de realizar el movimiento lineal o articular, solo que esta vez es “a mano”. Es decir, en el caso articular hay que hacer clic en la articulación que deseamos mover, y manteniendo clicado, desplazar el ratón.



En el caso lineal aparecen unas flechas al clicar sobre la herramienta, que nos indican los posibles movimientos lineales de la misma.



Lógicamente ofrece mayor precisión trabajar con la interfaz. Y de hecho, si conocemos las coordenadas del punto al que deseamos ir, lo mejor es introducirlas numéricamente.

NOTA: Cuando utilicemos movimientos lineales, en el simulador existen las mismas restricciones que tendríamos con el robot real. Es decir, para llegar a ciertas posiciones es muy probable que tengamos que realizar primero un movimiento articular y después otro lineal.

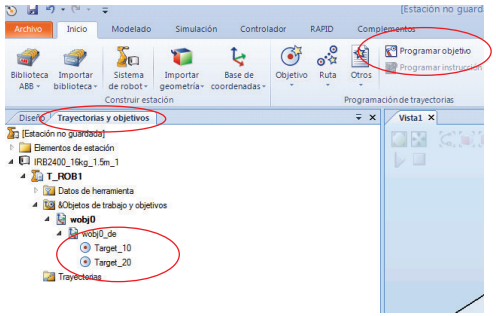
3. Guardado de posiciones.

Para guardar una posición, hay dos opciones diferentes. Primero se explicará la más intuitiva.

Supongamos que hemos llevado el robot hasta la posición deseada. Ahora hacemos lo siguiente:

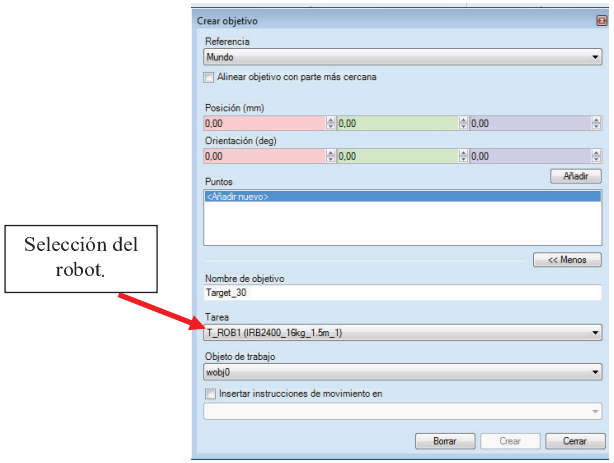
Hacemos clic en “Programar objetivo”. Nos dará un aviso sobre que utilizamos cierto sistema de referencia.

El objetivo se habrá quedado guardado en la pestaña “Trayectorias y objetivos”.



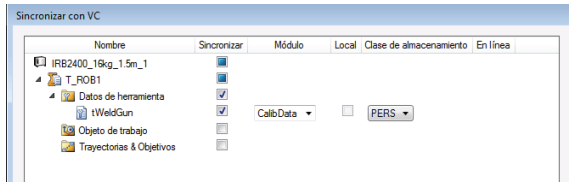
En el caso de que conozcamos ya las coordenadas del objetivo, y la orientación con la que hay que llegar a él, hacemos lo siguiente:

Clic en “Objetivo / Crear objetivo”. Elegimos la referencia que deseemos, e introducimos las coordenadas en las que se situará el objetivo.



Cuando tenemos dos robots, para seleccionar de qué robot queremos crear el objetivo, hay que hacer un objetivo con esta última opción, y ya se queda establecido el robot destino del objetivo. Luego ya podemos utilizar cualquiera de las dos opciones.

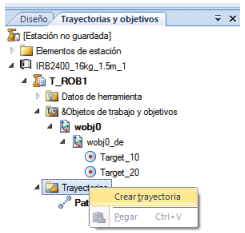
Cuando queramos pasar los objetivos a código RAPID, haremos clic en el botón “Sincronizar” del menú “RAPID”, que los transformará en código legible por un robot, y por el simulador.



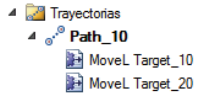
En dicha ventana, podremos seleccionar qué partes de código deseamos que se actualicen, ya que podría no interesarnos actualizar todas.

4. Creación de trayectorias.

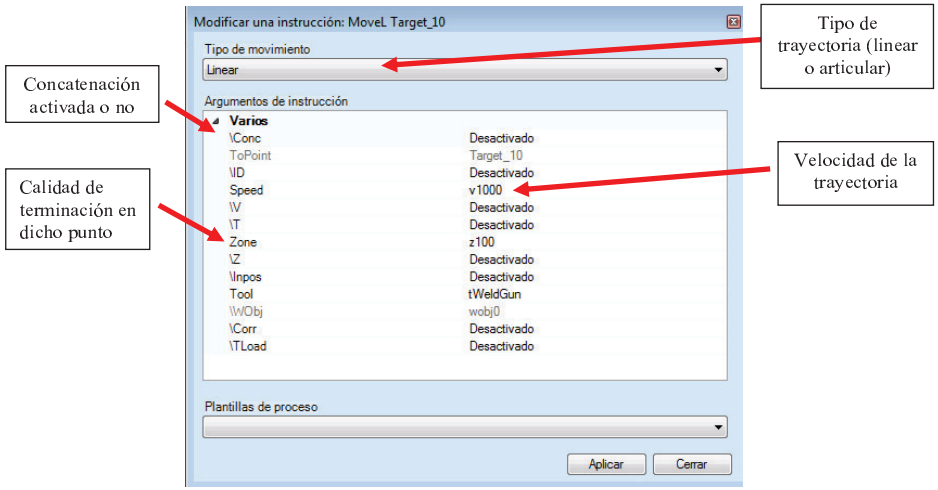
Para crear una trayectoria, Robot Studio nos ofrece básicamente dos alternativas. En caso de tener los puntos entre los cuales deseamos realizar la trayectoria, haremos clic en “Trayectoria / crear trayectoria”.



Una vez tenemos una trayectoria creada (está vacía), tenemos que “arrastrar” los puntos que compondrán dicha trayectoria.

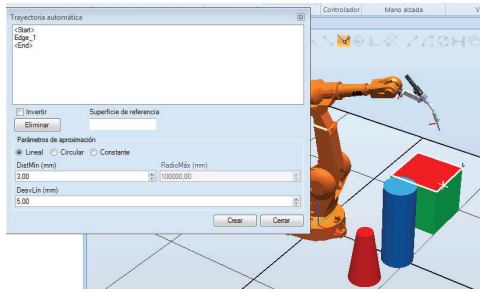


Haciendo clic alternativo sobre las instrucciones de movimiento y seleccionando “Modificar una instrucción” nos permitirá modificar los diferentes parámetros de la misma, tal y como se detalla a continuación.



La otra opción que se nos proporciona para crear trayectorias es crear una trayectoria automática a partir de una superficie, arista, curva…

Para ello hacemos clic en “Ruta / trayectoria automática”. Y una vez dentro, seleccionamos por dónde se ha de realizar la trayectoria.



Una vez tengamos una trayectoria definida, y estemos satisfechos con ella, es momento de probarla.

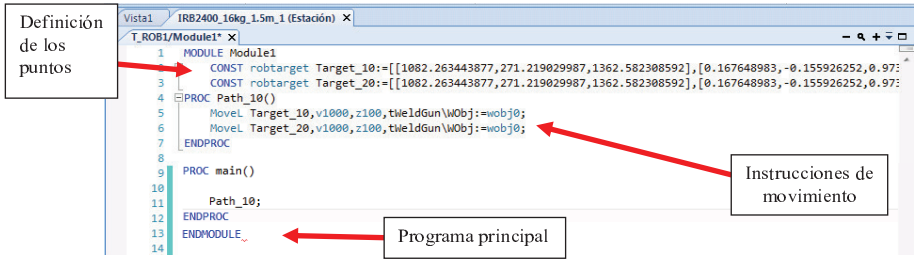
Para probar una trayectoria, y estar seguros de que es realizable, hay que realizar las siguientes operaciones:

> Para comprobar la alcanzabilidad de los diferentes puntos, clic derecho sobre la trayectoria a comprobar (En la pestaña trayectorias y objetivos) y seleccionamos “Alcanzabilidad”. El programa nos indicará si los diferentes puntos son alcanzables o no.
>
> Comprobación del movimiento. Clic alternativo sobre la trayectoria, y seleccionamos “Moverse a lo largo de la trayectoria”. Con esto comprobaremos visualmente que el robot hace la trayectoria que deseamos. Además nos dará información útil del estilo de “trayectoria en esquina no realizable” u otras situaciones similares.
>
> En caso de que algún punto no sea alcanzable o la trayectoria tenga problemas, podemos reconfigurar la muñeca del robot (si se puede). Para ello clic derecho en la trayectoria, y seleccionamos “Configuraciones”. También está disponible para cada instrucción de movimiento.

Una vez que quedemos satisfechos con la trayectoria, la tenemos que transferir a código. Para ello hacemos clic en “Sincronizar”, al igual que hicimos con los objetivos.

Para ver el código que se ha generado, vamos a la pestaña “RAPID”, y en el controlador del robot que estemos manejando, hacemos clic en “RAPID”/T_ROB/Module…”.

Podemos observar que hay un proceso creado, con el nombre de la trayectoria que hemos hecho. Para que el robot pueda ejecutar el programa, tendremos que añadir un módulo principal. Como este:



Una vez hecho esto, hay que aplicar los cambios, que equivaldría a algo así como compilar el código. Para ello clic en el botón “Aplicar” de la pestaña “RAPID”.

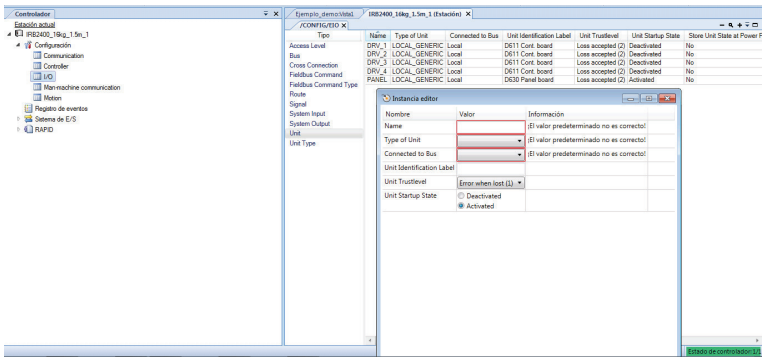
Ahora, si se desea (y no ha habido errores), se puede pasar a simular el comportamiento del robot. Simplemente hay que ir a la pestaña “Simulación” y clic en “Reproducir”. Cuando haya terminado la simulación, el botón “Restablecer” nos permitirá volver a la situación de partida.

5. Manejo de señales.

Para poder utilizar señales de entrada y salida al robot, muy útiles para que el robot se pueda comunicar con otros equipos o con otros robots, hay que seguir los siguientes pasos:

En la pestaña “Controlador”, en el apartado “Configuración / IO”, hacemos doble clic sobre “IO”. Se abrirá una nueva pestaña.

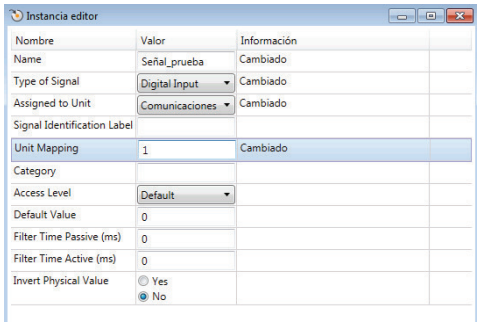
En la nueva pestaña abierta, hacemos clic sobre “Unit”, y después clic derecho y seleccionamos “Nuevo Unit”. Nos aparecerá lo siguiente:



Le ponemos un nombre cualquiera a la nueva unidad, por ejemplo: Comunicaciones. El tipo de unidad deberá ser virtual, y la conectaremos al bus “Virtual1”. El resto de opciones podemos dejarlas tal y como viene por defecto.

Nos aparecerá un aviso diciendo que los cambios no hará efecto hasta el próximo reinicio en caliente, por lo que procedemos a ello. Para eso vamos a la pestaña controlador, y sobre el controlador de nuestra estación (IRB….) hacemos clic derecho y seleccionamos “Reiniciar/Arranque en caliente”.

Cuando termine de reiniciarse podemos continuar. Pasamos a crear las nuevas señales, haciendo clic derecho sobre “Signal” y “Nuevo Signal”. Ahora le pondremos el nombre que necesitemos (Se recomienda poner nombres intuitivos). El tipo de señal será el que le queramos asignar, fundamentalmente vamos a trabajar con entradas y salidas digitales. La asignaremos a la unidad que hayamos creado antes (Comunicaciones), y en “Unit Mapping” le asignaremos un valor correlativo según el número de señal que hayamos creado.



Ahora ya tenemos una señal lista para usarse. Solo queda volver a hacer un arranque en caliente.

Para crear un salida digital, solo tendremos que repetir los mismos pasos, pero seleccionando “Digital Output”. Será interesante que en el nivel de acceso seleccionemos “ALL” para poder acceder desde el simulador de entradas y salidas.

La manera de activar y desactivar funciones puede ser de dos tipos. Ahora explicamos la manera de hacerlo mediante código, y más adelante (en el apartado gestor de eventos) se explicará la otra manera.

Para manear señales digitales por medio del código RAPID, tenemos, entre otras, las instrucciones siguientes:

```
11  PROC main()
12
13      SetDO señal_pueba_2,1;
14      waitDI señal_prueba,1;
15
16      Path_10;
17
18  ENDPROC
```

->Pone la señal a 1.

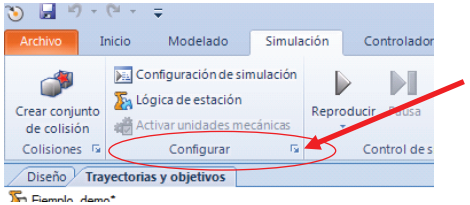
-> Espera a que la señal valga 1

6. Gestor de eventos.

El gestor de eventos es la herramienta que nos va a permitir por ejemplo coger o dejar objetos mediante una herramienta adosada al robot. También permitirá otras acciones como el cambio automático de señales por consecuencia de una acción o incluso reacciones ante una colisión.

En este caso, vamos a hacer todo lo necesario para que nuestro robot sea capaz de coger un objeto. Para ello, creamos un escenario con el robot que queramos. Después le añadimos la herramienta “ISO PLATE SMALL”. Más tarde, creamos la señal (de salida), que nos permitirá indicar cuándo queremos que se coja el objeto. Por último, vamos al gestor de eventos y realizamos lo que a continuación se indica.

Para abrir el gestor de eventos, vamos a la pestaña “Simulación” y hacemos clic en el botón pequeño que hay a la derecha de “configurar”.

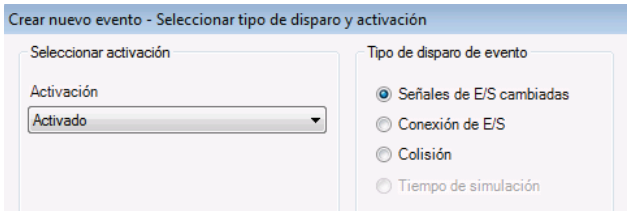


Se nos abrirá una nueva pestaña, en la cual seleccionaríamos la opción “Añadir, en el momento en que tuviéramos listos todos los requisitos.

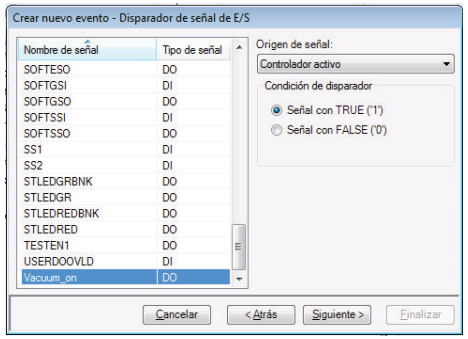
Los requisitos previos son haber creado la señal virtual que queremos utilizar para activar la ventosa, y tener preparados los puntos donde cogeremos y dejaremos el objeto. Una vez listos y recordando haber reiniciado en caliente el controlador, pasaremos a la siguiente parte.

Es interesante que al crear la señal, seleccionemos el nivel de permisos como “All” ya que así permitiremos modificarla desde el simulador de entradas y salidas.

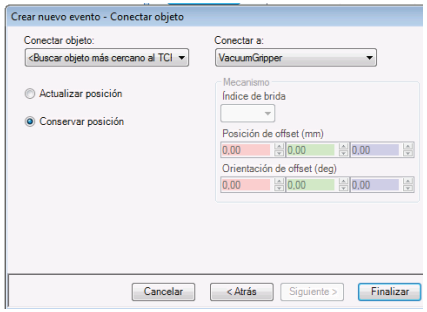
Es ahora momento de crear los eventos que permitirán coger y dejar objetos con la herramienta. Para ello vamos al gestor de eventos y añadimos evento. Nos interesa que se active cuando cambie una señal de entrada. Seleccionamos lo dicho.



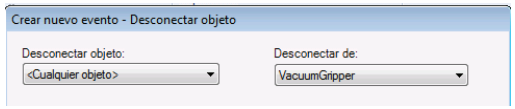
En el siguiente paso, seleccionamos la señal de entrada que queramos que active la ventosa (Será señal de entrada a la ventosa, y por lo tanto de salida del robot). Nos interesa que la condición sea señal en "TRUE".



El siguiente paso es seleccionar "Conectar objeto" y dentro de las opciones que nos aparecerán, lo más conveniente será "Buscar objeto más cercano a TCP" y conectar a "Iso_plate". Seleccionamos "Conservar posición".



El siguiente paso a dar es crear el evento que permita desconectar el objeto de una manera segura. Para ello realizamos los mismos pasos que antes, excepto que ahora seleccionaremos "Señal en FALSE (0)", y "Desconectar objeto". Luego elegiremos "Cualquier objeto" y desconectar de "Iso_plate".

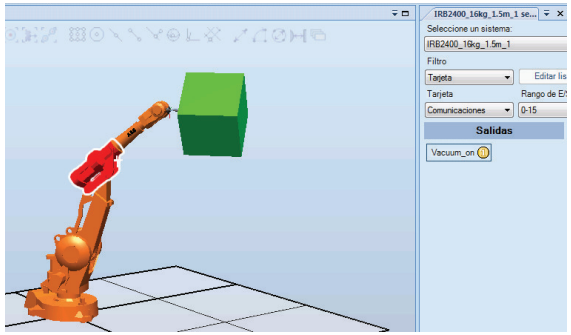


Ahora para comprobar que funciona, vamos a la pestaña "Simulación" y seleccionamos el "Simulador de E/S". Con este simulador podremos cambiar las entradas y salidas que necesitemos de una forma manual, como si tuviéramos una botonera.

Seleccionamos la tarjeta que hayamos creado nosotros (Comunicaciones) y cuando hagamos clic sobre la señal, observaremos que se activa o desactiva según el caso.

Para evitar un pequeño fallo del programa, pulsamos el botón "Reproducir" del menú "Simulación", como no hemos escrito código, no hará nada, pero así lograremos poder coger objetos y dejarlos mientras trasteamos con el robot.

Para comprobar su funcionamiento, llevamos el robot a la posición de coger el objeto, activamos la señal de coger objeto, y movemos el robot libremente. El objeto debería moverse con él.



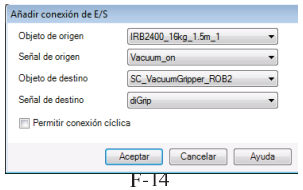
Si todo es correcto, ya podemos pasar al siguiente apartado. En caso de que no funcione como quisiéramos, hay que comprobar que hayamos seleccionado adecuadamente las opciones del gestor de eventos.

Además de eventos para coger objetos, se pueden utilizar otros para activar señales en función de diversos factores, mover objetos sin tocarlos (Por ejemplo una cinta transportadora…), y otras acciones…

7. Lógica de estación.

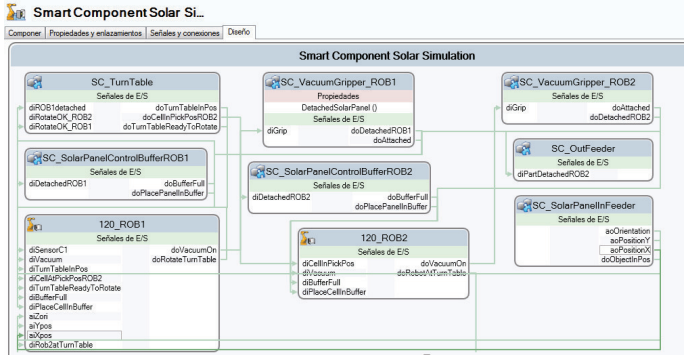
El gestor de lógica de estación, que se abre desde la pestaña "Simulación / Lógica de estación", permite muchas opciones a la hora de manejar señales de un modo robusto y fiable.

Permite desde "cablear virtualmente" señales de un robot con sus herramientas u otras máquinas (Pestaña "Señales y conexiones"), a componer nuevas señales en base a



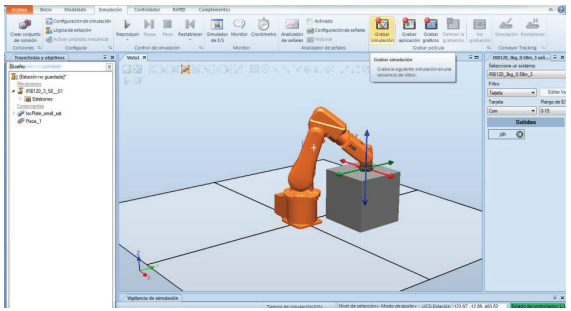
puertas lógicas.

Si bien es un método un poco más complejo que hacerlo mediante código RAPID, ofrece la ventaja de que aunque el controlador se pueda haber metido en un bucle (infinito o no…), mediante la lógica cableada se puede hacer que el robot no represente una amenaza. Ya sea desconectándose o apagando alguna de sus funciones…



8. Crear un vídeo.

Para crear un vídeo, lo único que hay que hacer una vez hayamos probado el funcionamiento del robot mediante simulación, es en la pestaña "Simulación", hacer clic sobre el botón "Grabar simulación", y luego clic en reproducir.



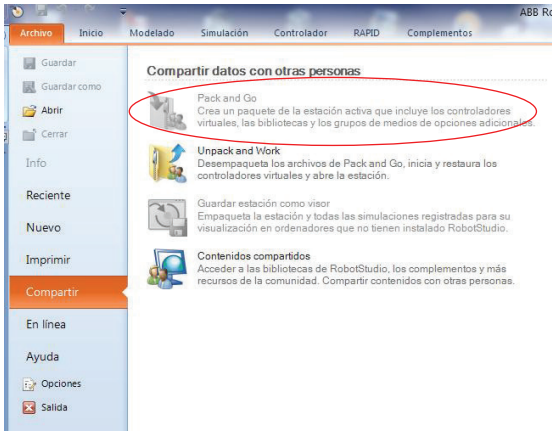
El programa grabará los movimientos del robot, y además nos permitirá modificar los puntos de vista a nuestro antojo y en tiempo real.

9. Empaquetar y desempaquetar escenarios:

En algunas ocasiones, a la hora de guardar un escenario, puede interesarnos incluir en él ciertos elementos que han podido ser creados por nosotros, o estar extraídos de otros escenarios, y que por tanto no tendrá todo el mundo que reciba nuestro escenario.

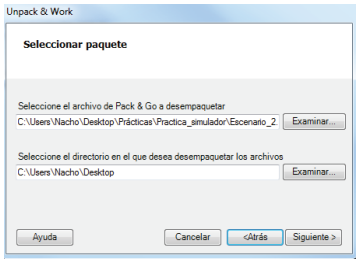
Para ello, existe la opción de empaquetar todo un escenario, incluyendo los diferentes controladores y herramientas que pudieran estar incluidos.

Dicha opción está en el menú archivo, en la pestaña de "Compartir". Cuando tengamos el escenario listo, seleccionamos "Pack and Go", y un asistente nos guiará sencillamente para poder crear el escenario empaquetado.



En el caso contrario, que deseemos desempaquetar un archivo que nos hayan pasado, seleccionamos "Unpack and Work" y hacemos lo siguiente:

Primero seleccionamos el archivo a desempaquetar, y el directorio donde deseemos que sea desempaquetado. Después seleccionamos la versión de Robot Ware adecuada (la que viene por defecto). Por último le damos a "Finalizar", tras lo cual hará una serie de operaciones y terminará por cargar el escenario.



Apéndice G

Guión de la práctica de Guiado

A continuación se adjunta la propuesta de guión para prácticas de laboratorio elaborada a partir de parte del material obtenido en esta práctica.
En esta propuesta de práctica, se procede a la programación por medio de guiado de un robot industrial modelo «FANUC ArcMate 50 IL».
El guión incluye plantillas para la programación, y la resolución de los casos propuestos.

PRÁCTICA R

PROGRAMACIÓN Y FUNCIONAMIENTO DE UN ROBOT INDUSTRIAL.

1. Trabajo preparatorio.

Para el correcto seguimiento y aprovechamiento de esta sesión práctica, se hace necesario leer detenidamente el guión completo de la misma.

Será asimismo necesario pensar cómo se va a hacer el programa y traerlo en un estado avanzado de programación. Es lógico que aparezcan dudas durante la preparación del mismo, pero vendrán muy bien para mejorar el aprovechamiento de la práctica y dinamizarla.

Toda función extra que realice el robot y que no esté entre las obligatorias, será tenida en cuenta.

Se recomienda además echar un vistazo al manual de Karel que se adjunta externo al guión de esta práctica.

2. Descripción de la práctica.

La práctica va a consistir en la programación de un robot industrial para realizar una tarea sencilla, en este caso se simulará el soldado de una caja. Los detalles de la misma se darán más tarde.

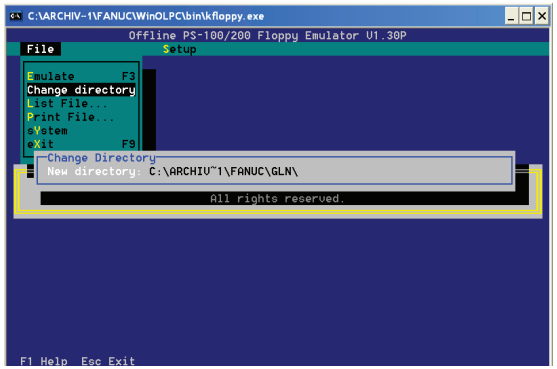
3. Desarrollo de la práctica.

1) Inicialización de los programas necesarios para trabajar con el robot.

En primer lugar, y sin encender el robot hasta que se indique, hay que abrir el programa WinOLPC. Este programa es el que nos permitirá traducir las instrucciones desde lenguaje Karel, al lenguaje del robot.

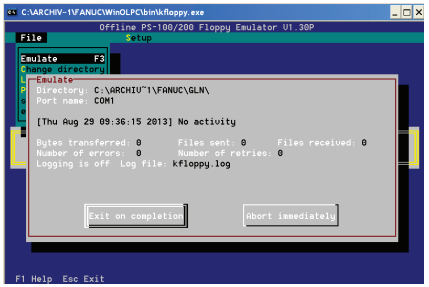
El siguiente paso va a ser ejecutar el programa KFLOPPY. Este programa ejercerá de disquetera virtual, permitiendo pasar programas del PC al robot y viceversa.

Para su correcto uso, hay que definirle la carpeta desde donde debe trabajar. Para ello existen dos métodos. El primero consiste en abrir desde WinOLPC un archivo cualquiera que esté contenido en la carpeta donde deseamos trabajar, y el programa KFLOPPY la tomará automáticamente como referencia. El otro método consiste en ejecutar el programa KFLOPPY, y en la pestaña "File/Change directory", escribir la dirección completa del directorio de referencia.



Detalle de cambio de directorio desde KFLOPPY.

Una vez nos aseguramos de que el programa KFLOPPY está direccionado al directorio correcto, pasamos a emular la disquetera virtual. Para ello, clic en "File/Emulate".



Detalle de la emulación con KFLOPPY.

Un último detalle cuando se trabaja con KFLOPPY es que para salir del programa, hay que hacer “File/Exit”, ya que la cruz de la ventana de Windows no funcionará en este programa.

Por último ya solo queda abrir un Hyper-terminal, que nos permitirá dar diferentes instrucciones al robot. Para ello, vamos a la carpeta de referencia en esta práctica, y clicamos sobre el archivo de terminal del robot. Se abrirá automáticamente.

2) Manejo del robot mediante el terminal de enseñanza.

El terminal de enseñanza del robot permite muchas funciones, de las cuales solo vamos a utilizar alguna de las más simples.

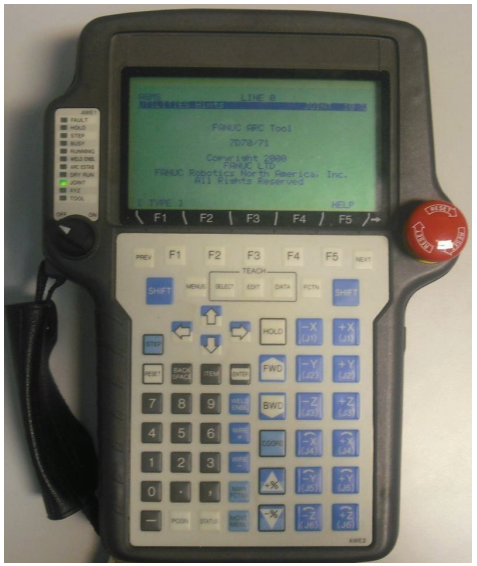
Para mover el robot, una vez puesto el mando en posición de “ON”, hay que pulsar el botón “Shift” a la vez que se selecciona la articulación a mover, o la dirección coordenada en la que se desea el movimiento.

Para cambiar entre movimientos articulares y coordenados, solo hay que pulsar el botón “COORD” hasta que aparezca en pantalla el tipo de movimiento deseado.

La velocidad puede cambiarse mediante los botones +% y -%.

Por último, queda conocer cómo se abre o cierra la pinza. Para ello habrá que acceder al menú de las variables mediante la tecla “Menus” y seleccionando “Signals”. Una vez dentro, pulsamos el botón correspondiente a “Type” y seleccionamos “Robot”. Ahora ya es visible la señal que activa y desactiva la pinza.

Ya solo queda saber cómo se graban los puntos que vamos a enseñarle al robot, para poder reaprovecharlos más adelante y hacer un programa más complejo.



Vista general del terminal de enseñanza.

Lo primero es crear un nuevo programa de tipo TP. Esto lo vamos a hacer desde el botón “Menus” en el apartado “Files”. Una vez dentro, con el botón “select” seleccionamos el tipo de archivos TP. Después, con las opciones F1 a F5, creamos uno nuevo y le damos nombre. Una vez dado el nombre, para crear un punto, pulsamos el botón que la pantalla indica que es “Point”, y después seleccionamos el tipo de movimiento que debe hacer hasta ese punto.

Luego ya podemos dirigir el robot a la siguiente posición y pulsar otra vez el botón “Point”.

3) Copiado de archivos al ordenador.

Cuando tengamos todos los puntos ya casi habrá acabado nuestra labor con el robot. Nos queda tan solo pasarlos al ordenador. Para lograrlo lo único que hay que hacer es ir al apartado “Files”, pulsando previamente “Menus”. Seleccionamos los archivos de tipo TP, y le damos al botón que indica “BackUp”. Decimos que sí a copiar el nuestro, y pulsamos no para los demás.

Ahora el programa en lenguaje TP ya está en nuestro ordenador. Solo queda traducirlo a algo legible.

Lo vamos a hacer mediante un programa en DOS, llamado PrintTp. Dicho programa está contenido en el directorio C\Archivos de programa\FANUC\WinOLPC\bin

Una vez lo tengamos localizado, abrimos una ventana de símbolo del sistema, y direccionamos mediante la instrucción “CHDIR” hasta la carpeta contenedora de nuestros archivos (La misma donde hemos direccionado el KFLOPPY).

Ahora “arrastramos” el icono con el programa PrintTp a dentro del símbolo del sistema, y le escribimos a continuación el nombre de nuestro archivo creado en el robot (ejemplo.tp), y a continuación también el nombre del archivo a generar (ejemplo.txt). Y si no hay ningún problema al presionar intro, tendremos ya el programa traducido a código ASCII y disponible para copiar y pegar las posiciones al programa Karel.

A continuación se muestra una imagen acerca de lo que aparece en DOS, y un ejemplo de la traducción que realiza el ordenador.

```
C:\Archivos de programa\FANUC\gln>"C:\Archivos de programa\FANUC\WinOLPC\bin\Pri
nttp.exe" AGMS.tp
All Rights Reserved, Copyright (C) 1994, FANUC Robotics North America.

/PROG  AGMS
/ATTR
OWNER           = MNEDITOR;
COMMENT         = "";
PROG_SIZE       = 501;
CREATE          = DATE 13-08-22  TIME 13:16:23;
MODIFIED        = DATE 13-08-22  TIME 13:17:23;
FILE_NAME       = ;
VERSION         = 0;
LINE_COUNT      = 2;
MEMORY_SIZE     = 829;
PROTECT         = READ_WRITE;
TCD:  STACK_SIZE        = 0,
      TASK_PRIORITY     = 50,
      TIME_SLICE        = 0,
      BUSY_LAMP_OFF     = 0,
      ABORT_REQUEST     = 0,
      PAUSE_REQUEST     = 0;
DEFAULT_GROUP   = 1,*,*,*,*;
CONTROL_CODE    = 00000000 00000000;
/APPL
  ARC Welding Equipment Number : 1 ;
/MN
   1:J P[1] 100% FINE  ;
   2:J P[2] 100% FINE  ;
/POS
P[1]{
   GP1:
        UF : 1, UT : 2,         CONFIG : 'N U T, 0, 0, 0',
        X =   480.96  mm,       Y =    80.10  mm,       Z =   693.36  mm,
        W =    32.56 deg,       P =   -27.14 deg,       R =   135.00 deg
};
P[2]{
   GP1:
        UF : 1, UT : 2,         CONFIG : 'N U T, 0, 0, 0',
        X =   235.10  mm,       Y =   427.16  mm,       Z =   693.36  mm,
        W =    32.56 deg,       P =   -27.14 deg,       R =  -173.28 deg
};
/END
```

MS-DOS con traducción del programa AGMS.tp

```
C:\Archivos de programa\FANUC\gln>"C:\Archivos de programa\FANUC\WinOLPC\bin\Pri
nttp.exe" AGMS.tp kk.txt
All Rights Reserved, Copyright (C) 1994, FANUC Robotics North America.
```

MS-DOS con traducción del programa AGMS.tp y guardado en el archivo kk.txt.

4) Creación del programa en Karel.

En este momento comienza la labor propiamente dicha de programación. Mediante la plantilla que se adjunta más adelante y el pequeño manual de instrucciones de Karel también adjunto, se debe crear un programa que simule estar soldando los bordes de la caja. Debe constar de al menos dos puntos de soldadura (cierre de pinza en el borde), cada uno en un borde diferente, y al menos dos cordones de soldadura (seguimiento del borde con la pinza abierta).

Una vez se tenga el programa, hay que guardarlo con el editor de texto (wordpad o bloc de notas), en formato .kl.

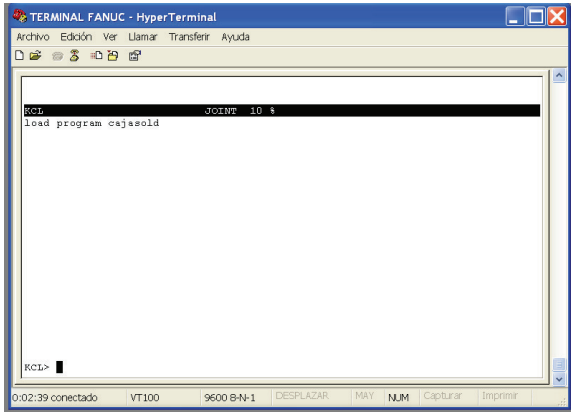
Hay que recordar poner el nombre tanto en la cabecera como en el final, y se recomienda poner el mismo nombre al programa y al archivo que lo contiene.

5) Compilación del programa y copiado al robot.

Como ya se ha comentado, la compilación del programa se va a realizar mediante uno de los accesorios de WinOLPC. En concreto vamos a la pestaña "Tools" y "Translate karel program". Seleccionamos el nuestro, y si no hay errores aparecerá en la barra inferior como traducción con éxito. En caso de que haya errores, tendremos que subsanarlos. Para ello el programa nos dará cierta información al respecto.

Una vez compilado con éxito, para copiarlo al robot, tenemos que ir al Hyper-terminal que hemos abierto anteriormente. Mediante la instrucción "load program *ejemplo*" cargaremos el programa "ejemplo" donde ejemplo corresponde con el nombre que hemos puesto en la cabecera del mismo, que no tiene por qué ser el mismo que hemos puesto en el archivo.

Cuando esté terminado de copiar, aparecerá otra vez el puntero de programa en blanco.



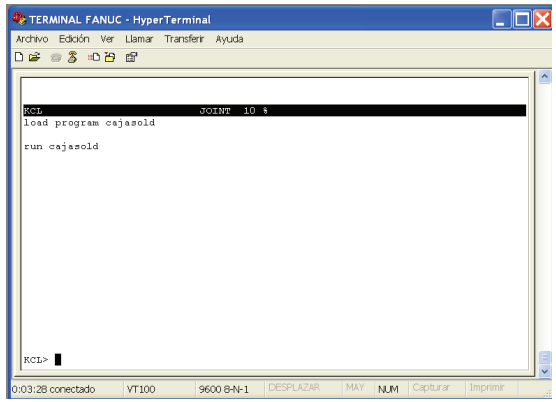
Hyper-terminal con programa ya cargado.

6) Ejecución del programa.

Ya solo queda ejecutar el programa que hemos creado. Para ello nos servimos de la instrucción "run *ejemplo*", que ejecuta el programa "ejemplo". Desde el momento en el que se empieza a ejecutar el programa, es importante el seguimiento del mismo teniendo siempre a mano una de las setas de seguridad.

Ante un comportamiento extraño o no esperado, es más que recomendable apretar la seta de seguridad, ya que así prevendremos daños al robot.

Hay que recordar que está absolutamente prohibido acceder al robot durante su funcionamiento.



Hyper-terminal con un programa en curso.

Durante la ejecución del programa, si hemos hecho uso de ellas, se pueden utilizar las señales simuladas mediante el comando: SIM DIN[x]=true/false.

4. Plantilla del programa.

Sustituir el nombre del programa "plantilla" por el nombre deseado. ¡Al inicio y al final del programa!

```
PROGRAM plantilla

%ENVIRONMENT IOSETUP        -- Directiva necesaria para definir GIN[1]
%INCLUDE KLIOTYPS           -- Incluye un archivo que iguala parámetros a números
-----------------------------------------------------------------------------------
--      AUTOR: JOSE IGNACIO ABAD MARTINEZ.
--      BASADO: EN UN PROGRAMA DE RICARDO BELTRAN ROYO.
--      TUTORES: JOSECHU GUERRERO Y GONZALO LOPEZ
--      FECHA DE REALIZACIÓN: 18/7/2013
--      MODIFICADO:
--      DESCRIPCIÓN: Programa plantilla para el lenguaje Karel;
-----------------------------------------------------------------------------------

CONST
        distancia = 100         -- utilizada en los movimiento: move away y move near
        max_vel = 2000          -- velocidad de los movimientos sin peligro
        aprox_vel = 100         -- velocidad de los movimientos para aproximaciones

VAR
        herramienta: POSITION       -- (0,0,101.34,-180,0,0)
        mundo: POSITION
        P_casa: JOINTPOS6           --(0,0,90,0,0,0) articulares

-- GENERALES PARA EL FUNCIONAMIENTO DEL PROGRAMA

        r1 : integer    -- utilizada para realizar el SELECT en el programa principal

        config_habit: CONFIG        -- configuración utilizada para todas las posiciones
(0,0,0,flip,codo arriba,frente)

        terminar:boolean            -- variable que será usada para indicar la
finalización del programa si fuera necesario

        puerta_ab:boolean   -- variable que permite la parada si se abre la puerta

-----------------------------------------------------------------------------------
--                      PROCEDIMIENTOS
-----------------------------------------------------------------------------------

ROUTINE create_jpos(lug_j:JOINTPOS6;A,B,C,D,E,F: REAL)
VAR
        vec_pos: ARRAY[6] OF REAL
        exito2: INTEGER
BEGIN
        vec_pos[1]=A;
        vec_pos[2]=B;
        vec_pos[3]=C;
```

Definición de las diferentes variables que serán utilizadas.

Rutina que pone en una variable de tipo JOINTPOS6 "lug_j" las coordenadas articulares "A, B, C, D, E, F";

```
        vec_pos[4]=D;
        vec_pos[5]=E;
        vec_pos[6]=F;
        CNV_REL_JPOS(vec_pos,lug_j,exito2);
END create_jpos


ROUTINE create_confi(confi: CONFIG;a,b,c:INTEGER;fl,u,fr:BOOLEAN)
BEGIN
        confi.cfg_turn_no1=a;
        confi.cfg_turn_no2=b;
        confi.cfg_turn_no3=c;
        confi.cfg_flip=fl;
        confi.cfg_up=u;
        confi.cfg_front=fr;
END create_confi
```

Rutina que sirve para iniciar una variable de tipo CONFIG, dándole los tres integers "a,b,c" y su configuración con booleanos: flip/no flip "fl", codo arriba/abajo "u" y frente/espalda "fr";

```
ROUTINE ini_pos
BEGIN
        create_jpos(P_casa,0,0,90,0,0,0);
```

Rutina que inicializa las localizaciones utilizadas en el programa

Incluir aqui todas las inicializaciones o definiciones de posiciones necesarias.
Tanto las de tipo cartesiana, mediante la función POS(), como las de tipo articular mediante la función create_jpos().

```
END ini_pos


ROUTINE init_system
VAR
        exito: INTEGER
BEGIN
-- parametrizado de la herramienta
        herramienta= POS(0,0,101.34,-180,0,0,config_habit);
        $UTOOL = herramienta;
        WRITE (CR,'GARRA HABILITADA',CR);

-- parametrizado del mundo
        mundo= POS(0,0,0,0,0,0,config_habit);
        $UFRAME = mundo;

-- hacer complementarias las salidas del robot que controlan la muñeca;
        SET_PORT_MOD(io_rdo,1,2,exito);
        WRITE('RDO1 Y RDO2 complementarias',exito,CR);
```

Rutina que inicializa las variables de sistema


```
-- hacer un grupo de señales digitales
        SET_PORT_ASG(io_gpin,1,0,1,io_din,19,5,exito);
        WRITE('ASIGNADAS LAS ENTRADAS',exito,CR);

-- parametrizado de las velocidades de los movimientos
        $GROUP[1].$SPEED=2000;
        $MCR_grp[1].$PRGOVERRIDE= 100;

        $PARAM_GROUP[1].$JNTVELLIM[1]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[2]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[3]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[4]= 400;
        $PARAM_GROUP[1].$JNTVELLIM[5]= 330;
        $PARAM_GROUP[1].$JNTVELLIM[6]= 480;
        $PARAM_GROUP[1].$JNTVELLIM[7]= 0;
        $PARAM_GROUP[1].$JNTVELLIM[8]= 0;
        $PARAM_GROUP[1].$JNTVELLIM[9]= 0;

--      $PARAM_GROUP[1].$SPEEDLIM= 2000;        -- mensaje de error write
protected;
--      $PARAM_GROUP[1].$SPEEDLIMJNT= 2000;     -- mensaje de error write
protected;

-- parametrizado de la interpolación de la orientación
        $ORIENT_TYPE= RSWORLD       -- orientado mediante el método de los dos
angulos:

-- parametrizado de la aceleración
        $GROUP[1].$usemaxaccel= TRUE
        $PARAM_GROUP[1].$cart_accel1= 224    -- aceleración1 = 2 * aceleración2
        $PARAM_GROUP[1].$cart_accel2= 112    -- método de 2º orden, reduce
vibraciones de la estructura
        $PARAM_GROUP[1].$accel_time1[1]= 224
        $PARAM_GROUP[1].$accel_time1[2]= 224
        $PARAM_GROUP[1].$accel_time1[3]= 224
        $PARAM_GROUP[1].$accel_time1[4]= 224
        $PARAM_GROUP[1].$accel_time1[5]= 224
        $PARAM_GROUP[1].$accel_time1[6]= 224
        $PARAM_GROUP[1].$accel_time2[1]= 112
        $PARAM_GROUP[1].$accel_time2[2]= 112
        $PARAM_GROUP[1].$accel_time2[3]= 112
        $PARAM_GROUP[1].$accel_time2[4]= 112
        $PARAM_GROUP[1].$accel_time2[5]= 112
        $PARAM_GROUP[1].$accel_time2[6]= 112
END init_system
```

```
BEGIN                                   Inicio del programa

        create_confi(config_habit,0,0,0,false,true,true);      --ponemos el valor de la
configuración habitual

        -- Condition handlers

        CONDITION[3]:
                WHEN DIN[7] DO          --nos indican que acabemos el programa
                terminar=true
        ENDCONDITION;

        CONDITION[4]:
                WHEN DIN[1] DO
                STOP
                ENABLE CONDITION[5]
                DISABLE CONDITION[4]
        ENDCONDITION;

        CONDITION[5]:
                WHEN NOT DIN[12] DO
                RESUME
                DISABLE CONDITION[5]
        ENABLE CONDITION[4]
        ENDCONDITION;

        init_system;
        ini_pos;
        ENABLE CONDITION[3];  --habilita la posibilidad de apagar el robot
        ENABLE CONDITION[4];  --habilita la parada de emergencia si se abre la
puerta (señal 1)
```

A rellenar con el código del programa a ejecutar.

END plantilla

5. Pequeño manual de lenguaje Karel:

Formas de declarar posiciones:

Coordenadas articulares: **JOINTPOS**

Coordenadas del usuario: **XYZWPR**

Tipos de movimientos:

MOVE TO *posición* Mueve el extremo de la herramienta a "posición".

MOVE NEAR *posición* **BY** *distancia* Se desplaza hasta la posición definida, con un offset en la dirección Z del TCP. Se expresa en mm.

MOVE AWAY *distancia* Mueve la herramienta la distancia indicada en la dirección Z del TCP.

Control de trayectoria:

Trayectoria articular coordinada: **$MOTYPE = JOINT**

Trayectoria rectilínea: **$MOTYPE = LINEAR**

Control de terminación

Deceleración y parada en el punto indicado: **$TERMTYPE = FINE**

Movimiento continuo: **$TERMTYPE = NODECEL**

Control de la garra:

Abrir garra: **OPEN HAND 1**

Cerrar garra: **CLOSE HAND 1**

Control de velocidad:

$SPEED= *velocidad*

Esperar tiempo:

DELAY *tiempo* (en milisegundos)

Hay que recordar que las instrucciones de modificación del movimiento se mantienen activas hasta que otra las cambia.

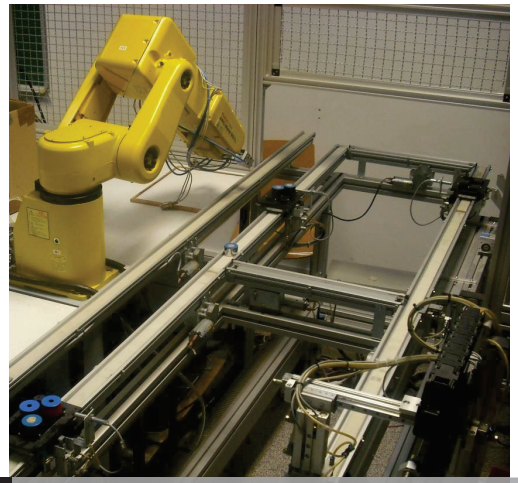
Pequeño ejemplo de programación:

```
$MOTYPE=JOINT;

$TERMTYPE=FINE

$SPEED=max_vel;

MOVE NEAR P_agarra BY -distancia;

OPEN HAND 1;
```

Apéndice H

Guión de la práctica de Integración

Se adjunta en este apéndice una propuesta de guión de prácticas sobre la integración de varios componentes de la célula de fabricación del laboratorio L0.6 del edificio *Ada Byron*, resultante de parte del material obtenido durante la realización de este proyecto.



EINA – UNIVERSIDAD DE ZARAGOZA

PRÁCTICA DE INTEGRACIÓN DE UN ROBOT CON UN AUTÓMATA EN UNA CÉLULA DE TRANSPORTE Y ALMACENAJE.

José Ignacio Abad Martínez.

Introducción.

Parte A. Programación del robot.

A.1, trabajo preparatorio.

A.2, descripción de la práctica.

A.3, organización de la práctica y evaluación.

A.4, desarrollo de la práctica.

A.4.1, plantilla común a todos los programas a realizar.

A.4.2, programas a realizar.

A.5, apartados opcionales.

A.6, soluciones del programa.

Parte B. Programación del autómata.

B.1, trabajo preparatorio.

B.2, descripción de la práctica.

B.3, organización de la práctica y evaluación.

B.4, desarrollo de la práctica.

B.4.1, abriendo Unity.

B.4.2, configurando las variables.

B.4.3, creación de la red de Petri.

B.4.4, conexión, carga y ejecución.

B.5, plantillas de la práctica

B.5.1, red de Petri.

B.5.2, variables.

B.5.3, transiciones.

B.5.4, acciones.

B.5, apartados opcionales.

B.6, soluciones del programa.

B.6.1, red de Petri.

B.6.2, variables.

B.6.2.1, transiciones.

B.6.2.2, acciones.

B.6.3, conexiones entre variables y código.

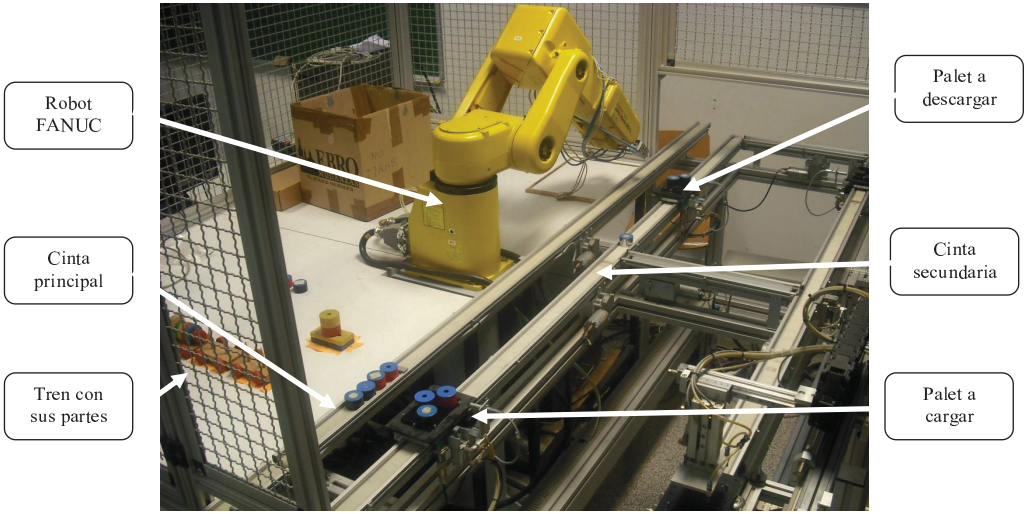
Introducción.

La práctica que aquí se presenta se pretende que sirva como ejercicio didáctico de cómo en la industria real se pueden integrar en una sola célula: robots, módulos de entradas y salidas, buses de campo, autómatas e interfaces hombre-máquina.

Para ello se contará con un robot "Fanuc Arc mate 50 il", un autómata "Modicon M340", un bus de campo "CANopen", un módulo de E/S "Advantys STB" y un terminal de interfaz gráfica táctil "Magelis". Además de PCs.

La finalidad última de esta práctica será que el alumno consiga controlar de manera segura el robot, mediante una serie de programas creados por él en base a unas plantillas, que se referirán tanto al robot, como al autómata y al terminal de interfaz.

Se adjunta a continuación una fotografía de la célula de transporte y almacenaje, con los principales componentes señalados.



Vista de la célula de almacenamiento y transporte

Parte A. Programación del robot.

A 1. Trabajo preparatorio.

Antes de la realización de la práctica, es absolutamente imprescindible haber leído y comprendido el guión de la misma y realizar el estudio previo según el caso que se haya asignado.

La práctica es especialmente laboriosa, y en caso de que no se logre terminar durante la clase, las partes relativas al uso del robot no se podrán probar más adelante, ya que el uso del robot requiere la presencia en todo momento de un profesor.

A 2. Descripción de la práctica.

La programación del robot consiste en la implementación de un programa escrito en lenguaje de alto nivel Karel, que permita al autómata controlar al robot, y gestione tanto el intercambio de información entre ambos, como el almacenamiento de las instrucciones de las tareas que ha de realizar el robot en función de las indicaciones del autómata.

El programa a su vez viene estructurado en tres partes bien diferenciadas.

En primer lugar se tienen las rutinas de almacenamiento de posiciones, inicialización de variables…

En segundo lugar aparecen los subprogramas que realizan las tareas que el autómata ordena, y que son hasta siete diferentes.

Por último, tenemos el programa principal que contiene los bucles de control, comunicación y selección de subprogramas que permiten funcionar al conjunto.

A 3. Organización de la práctica y evaluación.

La parte consta de una parte obligatoria que todos los alumnos deberán completar, y posteriormente enseñar al profesor, y de una serie de apartados opcionales que permitirán alcanzar la máxima calificación.

Parte obligatoria (Hasta 7 puntos de 10):

- Lectura detenida del guión, y resolución de dudas sobre las plantillas del programa que más adelante se adjuntan.
- Preparar y llevar al laboratorio de prácticas el trozo de programa que haya sido designado.

Parte voluntaria (Hasta 3 puntos de 10):

- Realizar los apartados opcionales que se adjuntan al final de este guión.

NOTA: los programas a realizar se asignarán en la clase de teoría. La puntuación a la que se opta en función del programa asignado no es igual, tal y como detallará el profesor.

A 4. Desarrollo de la práctica.

Para el desarrollo de la práctica se adjuntan una serie de plantillas correspondientes a los diferentes programas que se puede pedir realizar, así como de las instrucciones sobre qué se tiene que hacer con cada uno de los programas. En cualquier caso, todas las plantillas contienen todos los datos necesarios, y que serán iguales para todos los grupos. No obstante es lógico que se pueden realizar diferentes versiones de un miso programa funcionando todas ellas y además de una manera correcta

A 4.1. Plantilla común a todos los programas a realizar.

La siguiente plantilla da la información sobre las variables que se utilizarán, el tipo que son y alguna otra información. No es necesario incluirla en el programa a entregar, puesto que el que tiene el profesor para integrar todos los apartados ya la incluye.

```
PROGRAM integra

%ENVIRONMENT IOSETUP            -- Directiva necesaria para definir GIN[1]
%INCLUDE KLIOTYPS           -- Incluye un archivo que iguala parámetros a números
-----------------------------------------------------------------------------------
--      AUTOR: JOSE IGNACIO ABAD MARTINEZ.
--       BASADO: EN UN PROGRAMA DE RICARDO BELTRAN ROYO.
--      TUTORES: JOSECHU GUERRERO Y GONZALO LOPEZ
--      FECHA DE REALIZACIÓN: 30/8/2013
--      MODIFICADO: (última vez 6/9/2013).
--      DESCRIPCIÓN: Este programa sirve para enseñar la integración de un robot y un autómata;
-----------------------------------------------------------------------------------

CONST
        distancia = 100         -- utilizada en los movimiento: move away y move near
        max_vel = 2000          -- velocidad de los movimientos sin peligro
        aprox_vel = 100         -- velocidad de los movimientos para aproximaciones

VAR
        herramienta: POSITION           -- (0,0,101.34,-180,0,0)
        mundo: POSITION
        P_casa: JOINTPOS6               --(0, 0, 90, 0, 0, 0) articulares
        aproxima: INTEGER

-- LOCALIZACIONES NECESARIAS PARA EL PROGRAMA

        P_rec_pi: XYZWPR                -- Posición donde se recogen las piezas de la cinta principal.
        P_aprox_pi: XYZWPR              -- Posición de aproximación a la cinta principal.
        P_aprox_plt: XYZWPR             -- Posición de aproximación al palet.
        P_palet_1: XYZWPR               -- Posiciones de los lugares en los que hacer los montones de
        P_palet_2: XYZWPR               -- piezas.
        P_palet_3: XYZWPR
        P_palet_1_1: XYZWPR
        P_palet_2_1: XYZWPR
        P_palet_3_1: XYZWPR

        P_dej_cin: XYZWPR               -- Posición de dejar piezas en la cinta secundaria.
        P_aprox_dej: XYZWPR             -- Aproximación a la cinta secundaria.

        pal_1_1: XYZWPR                 -- Posiciones de los huecos en los dos palets alcanzables.
        pal_1_2: XYZWPR
        pal_1_3: XYZWPR
        pal_2_1: XYZWPR
        pal_2_2: XYZWPR
```



```
        pal_2_3: XYZWPR
        P_vio: XYZWPR                   -- Punto intermedio para acceder a los palets.
        P_dej_pi: XYZWPR                -- Punto de la cinta donde se deja la pieza al descargar.

        P_reposo: JOINTPOS6             -- Posición de reposo entre operación y operación.
        P_paso_1: JOINTPOS6             -- Posición auxiliar usada como punto vía.
        P_desecha: XYZWPR               -- Posición donde desecha la pieza.

-- LOCALIZACIONES PARA MONTAR Y DESMONTAR LAS PIEZAS EN EL TREN

        Dist_segu: XYZWPR               -- distancia de 2mm en z para en las aproximaciones
                                        -- a dejar no chocar con la mesa y causarle esfuerzos
        Almacen: XYZWPR
        CogeRectan: XYZWPR      --  Lugar donde se recoge el rectángulo que se coloca el 2º.
        RectanArriba: XYZWPR    --  Lugar donde se recoge el rectángulo que se coloca el 1º.
        CogeCilin: XYZWPR       --  Lugar donde se recoge el cilindro que se coloca el 2º.
        CilinArriba: XYZWPR     --  Lugar donde se recoge el cilindro que se coloca el 1º.

        Vagon2: XYZWPR
        ColocaRectan: XYZWPR -- "Vagon2:ColocaRectan" lugar donde se coloca el rectángulo que se
                             -- coloca el 1º.
                             -- "Vagon2:ColocaRectan:RectanArriba" lugar donde se coloca el
                             -- rectángulo que se coloca el 2º.
        Vagon1: XYZWPR
        ColocaCilin1: XYZWPR -- "Vagon1:ColocaCilin1" lugar donde se coloca el cilindro que se
                             -- coloca el 1º.
        ColocaCilin2: XYZWPR -- "Vagon1:ColocaCilin2" lugar donde se coloca el cilindro que se
                             -- coloca el 2º.

-- GENERALES PARA EL FUNCIONAMIENTO DEL PROGRAMA

        r1 : integer

        paletizador: integer    -- Variable que permitirá apilar indefinidamente piezas.

        config_habit: CONFIG    -- Configuración utilizada para todas las posiciones (0,0,0,flip,codo
                                -- arriba,frente).

        terminar:boolean        -- Variable que será usada para indicar la finalización del programa si
                                --fuera necesario.
        puerta_ab:boolean       -- Variable que permite la parada si se abre la puerta.
```

A 4.2. Programas a realizar:

- **Traslado de una cinta a otra:**

El robot deberá recoger una pieza de la cinta principal, y trasladarla a la cinta secundaria. Se recomienda hacer el movimiento más sencillo posible, y tratar de abrir y cerrar la pinza en momentos adecuados y que no retrasen el movimiento.

```
ROUTINE trasl_cinta

BEGIN
        DOUT[1]=ON;   -- Comunicación con autómata.
        DOUT[3]=OFF;  -- Comunicación con autómata.

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
```

```
        MOVE TO P_reposo;
        DOUT[2]=ON;                 -- Comunicación con autómata.
        DOUT[1]=OFF;                -- Comunicación con autómata.
END trasl_cinta
```

- **Apilamiento de piezas en tres montículos:**

El robot cogerá piezas de la cinta principal, y las irá colocando en orden en las tres posiciones indicadas. Además, una vez completa una fila, deberá elevar la altura a la que las deja para poder completar la siguiente.

```
ROUTINE paletiza

BEGIN
        DOUT[1]=ON;   -- Comunicación con autómata.
        DOUT[3]=OFF;  -- Comunicación con autómata.

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;                 -- Comunicación con autómata.
        DOUT[1]=OFF;                -- Comunicación con autómata.

END paletiza
```

- **Desechado de una pieza:**

El programa consistirá en que el robot debe de coger una pieza de la cinta principal y desecharla en la posición indicada. La rapidez del proceso es importante, ya que al desechar una pieza no importa la delicadeza con la que se trate. No obstante ha de ser totalmente segura para el robot.

```
ROUTINE desecha

BEGIN
        DOUT[1]=ON;                 -- Comunicación con autómata.
        DOUT[3]=OFF;                -- Comunicación con autómata.
……
        MOVE TO P_reposo;
        DOUT[2]=ON;                 -- Comunicación con el autómata.
        DOUT[1]=OFF;                -- Comunicación con el autómata.

END desecha
```

- **Montado y desmontado del tren:**

El robot cogerá las piezas de un almacén previamente establecido, y las depositará en el sitio adecuado del tren de juguete. Este programa conlleva una mayor complejidad, ya que se realiza mediante una subrutina llamada "mueve tren".

```
ROUTINE desm_tr

BEGIN
        DOUT[1]=ON;             -- Comunicación con el autómata.
        DOUT[3]=OFF;            -- Comunicación con el autómata.
        mueve_tr(Vagon2:ColocaRectan:RectanArriba,Almacen:CogeRectan);
        mueve_tr(Vagon2:ColocaRectan,Almacen:CogeRectan:RectanArriba);
```



```
        mueve_tr(Vagon1:ColocaCilin2,Almacen:CogeCilin:Dist_segu);
        mueve_tr(Vagon1:ColocaCilin1,Almacen:CogeCilin:CilinArriba:Dist_segu);
        DOUT[2]=ON;             -- Comunicación con el autómata.
        DOUT[1]=OFF;            -- Comunicación con el autómata.

END desm_tr

ROUTINE monta_tr

BEGIN
        DOUT[1]=ON;             -- Comunicación con el autómata.
        DOUT[3]=OFF;            -- Comunicación con el autómata.
        mueve_tr(Almacen:CogeCilin:CilinArriba,Vagon1:ColocaCilin1:Dist_segu);
        mueve_tr(Almacen:CogeCilin,Vagon1:ColocaCilin2:Dist_segu);
        mueve_tr(Almacen:CogeRectan:RectanArriba,Vagon2:ColocaRectan:Dist_segu);
        mueve_tr(Almacen:CogeRectan,Vagon2:ColocaRectan:RectanArriba:Dist_segu);
        DOUT[2]=ON;             -- Comunicación con el autómata.
        DOUT[1]=OFF;            -- Comunicación con el autómata.

END monta_tr

ROUTINE mueve_tr(Tren_coge,Tren_deja: XYZWPR)

BEGIN
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_coge BY –distancia

……
        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = aprox_vel;
        MOVE AWAY -distancia;

END mueve_tr
```

- **Carga de un palet:**

El robot debe de cargar uno de los palets (en concreto el de la derecha), tomando para ello las piezas de la cinta principal. Además hay que activar alguna señal que indique que el palet ya está cargado y no es posible cargarlo de nuevo.

```
ROUTINE cargar

BEGIN
        DOUT[1]=ON;                 -- Comunicación con el autómata.
        DOUT[3]=OFF;                -- Comunicación con el autómata.
        $SPEED=400;
        MOVE TO P_rec_pi;
……..
        $MOTYPE=JOINT;
        $TERMTYPE=FINE;
        $SPEED=max_vel;
        MOVE TO P_reposo;

        DOUT[2]=ON;                 -- Comunicación con el autómata.
        DOUT[1]=OFF;                -- Comunicación con el autómata.

END cargar
```

- **Descargado de un palet:**

El robot se encargará de descargar las tres piezas situadas en el palet de la izquierda. Las piezas se dejarán en la posición indicada de la cinta principal.

```
ROUTINE descargar

BEGIN
        DOUT[1]=ON;                     -- Comunicación con el autómata.
        DOUT[3]=OFF;                    -- Comunicación con el autómata.
……..
        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;                     -- Comunicación con autómata.
        DOUT[1]=OFF;                    -- Comunicación con autómata.

END descargar
```

A 5. Apartados opcionales.

Se proponen a continuación los siguientes apartados opcionales:

- Realizar el programa que haya sido designado, mediante la utilización de diferentes velocidades y tipos de movimiento y precisión.
  Se valorará el uso de movimientos relativos mediante las instrucciones "MOVE NEAR___ BY" y "MOVE AWAY", concatenación de movimientos mediante la instrucción "NODECEL", y la rapidez y seguridad con la que se realicen dichos movimientos.

- Realizar otros programas que no hayan sido asignados.

- Programar las instrucciones necesarias para el uso de los programas "monta tren" y "desmonta tren", mediante la subrutina "mueve tren".

A 6. Soluciones del programa.

- Programa implementado (integra.kl).

```
PROGRAM integra

%ENVIRONMENT IOSETUP            -- Directiva necesaria para definir GIN[1]
%INCLUDE KLIOTYPS           -- Incluye un archivo que iguala parámetros a números
-----------------------------------------------------------------------------------
--      AUTOR: JOSE IGNACIO ABAD MARTINEZ.
--       BASADO: EN UN PROGRAMA DE RICARDO BELTRAN ROYO.
--      TUTORES: JOSECHU GUERRERO Y GONZALO LOPEZ
--      FECHA DE REALIZACIÓN: 30/8/2013
--      MODIFICADO: (última vez 6/9/2013).
--      DESCRIPCIÓN: Este programa sirve para enseñar la integración de un robot y un autómata;
-----------------------------------------------------------------------------------

CONST
        distancia = 100         -- utilizada en los movimiento: move away y move near
        max_vel = 2000          -- velocidad de los movimientos sin peligro
        aprox_vel = 100         -- velocidad de los movimientos para aproximaciones

VAR
        herramienta: POSITION           -- (0,0,101.34,-180,0,0)
        mundo: POSITION
        P_casa: JOINTPOS6               --(0, 0, 90, 0, 0, 0) articulares
        aproxima: INTEGER

-- LOCALIZACIONES NECESARIAS PARA EL PROGRAMA

        P_rec_pi: XYZWPR                -- Posición donde se recogen las piezas de la cinta principal.
        P_aprox_pi: XYZWPR              -- Posición de aproximación a la cinta principal.
        P_aprox_plt: XYZWPR             -- Posición de aproximación al palet.
        P_palet_1: XYZWPR               -- Posiciones de los lugares en los que hacer los montones de
        P_palet_2: XYZWPR               -- piezas.
        P_palet_3: XYZWPR
        P_palet_1_1: XYZWPR
        P_palet_2_1: XYZWPR
        P_palet_3_1: XYZWPR

        P_dej_cin: XYZWPR               -- Posición de dejar piezas en la cinta secundaria.
        P_aprox_dej: XYZWPR             -- Aproximación a la cinta secundaria.

        pal_1_1: XYZWPR                 -- Posiciones de los huecos en los dos palets alcanzables.
        pal_1_2: XYZWPR
        pal_1_3: XYZWPR
        pal_2_1: XYZWPR
        pal_2_2: XYZWPR
        pal_2_3: XYZWPR
        P_vio: XYZWPR                   -- Punto intermedio para acceder a los palets.
        P_dej_pi: XYZWPR                -- Punto de la cinta donde se deja la pieza al descargar.

        P_reposo: JOINTPOS6             -- Posición de reposo entre operación y operación.
        P_paso_1: JOINTPOS6             -- Posición auxiliar usada como punto vía.
        P_desecha: XYZWPR               -- Posición donde desecha la pieza.

-- LOCALIZACIONES PARA MONTAR Y DESMONTAR LAS PIEZAS EN EL TREN

        Dist_segu: XYZWPR               -- distancia de 2mm en z para en las aproximaciones
                                        -- a dejar no chocar con la mesa y causarle esfuerzos
```

```
        Almacen: XYZWPR
        CogeRectan: XYZWPR    --  Lugar donde se recoge el rectángulo que se coloca el 2º.
        RectanArriba: XYZWPR  --  Lugar donde se recoge el rectángulo que se coloca el 1º.
        CogeCilin: XYZWPR     --  Lugar donde se recoge el cilindro que se coloca el 2º.
        CilinArriba: XYZWPR   --  Lugar donde se recoge el cilindro que se coloca el 1º.

        Vagon2: XYZWPR
        ColocaRectan: XYZWPR -- "Vagon2:ColocaRectan" lugar donde se coloca el rectángulo que se
                             -- coloca el 1º.
                             -- "Vagon2:ColocaRectan:RectanArriba" lugar donde se coloca el
                             -- rectángulo que se coloca el 2º.
        Vagon1: XYZWPR
        ColocaCilin1: XYZWPR -- "Vagon1:ColocaCilin1" lugar donde se coloca el cilindro que se
                             -- coloca el 1º.
        ColocaCilin2: XYZWPR -- "Vagon1:ColocaCilin2" lugar donde se coloca el cilindro que se
                             -- coloca el 2º.


-- GENERALES PARA EL FUNCIONAMIENTO DEL PROGRAMA

        r1 : integer

        paletizador: integer     -- Variable que permitirá apilar indefinidamente piezas.

        config_habit: CONFIG     -- Configuración utilizada para todas las posiciones (0,0,0,flip,codo
                                 -- arriba,frente).

        terminar:boolean         -- Variable que será usada para indicar la finalización del programa si
                                 --fuera necesario.
        puerta_ab:boolean        -- Variable que permite la parada si se abre la puerta.

-----------------------------------------------------------------------------------------
--                              PROCEDIMIENTOS
-----------------------------------------------------------------------------------------

ROUTINE create_jpos(lug_j:JOINTPOS6;A,B,C,D,E,F: REAL)

VAR
        vec_pos: ARRAY[6] OF REAL
        exito2: INTEGER
BEGIN
        vec_pos[1]=A;
        vec_pos[2]=B;
        vec_pos[3]=C;
        vec_pos[4]=D;
        vec_pos[5]=E;
        vec_pos[6]=F;
        CNV_REL_JPOS(vec_pos,lug_j,exito2);
END create_jpos

ROUTINE create_confi(confi: CONFIG;a,b,c:INTEGER;fl,u,fr:BOOLEAN)
BEGIN
        confi.cfg_turn_no1=a;
        confi.cfg_turn_no2=b;
        confi.cfg_turn_no3=c;
        confi.cfg_flip=fl;
        confi.cfg_up=u;
        confi.cfg_front=fr;
END create_confi
```

Rutina que pone en una variable de tipo JOINTPOS6 "lug_j" las coordenadas articulares "A,B,C,D,E,F".

Rutina que sirve para iniciar una variable de tipo CONFIG, dándole los tres enteros "a,b,c" y su configuración con booleanos: flip/no flip "fl",codo arriba/abajo "u" y frente/espalda "fr";


```
ROUTINE ini_pos

BEGIN
        create_jpos(P_casa,0,0,90,0,0,0);
        create_jpos(P_reposo,-101, 15, -36, 0, -57, 31);
        create_jpos(P_paso_1,25.025, 10.064, -16.23, 0, -76.4, 3.5);
        P_rec_pi=POS(170.0,-593.0,-300.0,0,0,-90,config_habit);
        P_aprox_pi=POS(170.0,-593.0,-150.0,0,0,-90,config_habit);
        P_aprox_plt=POS(-555, -270, -150, 0, 0, -179,config_habit);
        P_palet_1=POS(-505, -270, -345, 0, 0, -179,config_habit);
        P_palet_2=POS(-555, -270, -345, 0, 0, -179,config_habit);
        P_palet_3=POS(-605, -270, -345, 0, 0, -179,config_habit);
        P_palet_1_1=POS(-505, -270, -313, 0, 0, -179,config_habit);
        P_palet_2_1=POS(-555, -270, -313, 0, 0, -179,config_habit);
        P_palet_3_1=POS(-605, -270, -313, 0, 0, -179,config_habit);
        P_dej_pi=POS(162.4, -794.1, -300, 0, 0, -90, config_habit);
        P_dej_cin=POS(355, 273.75, -300, 0, 0, 0,config_habit);
        P_aprox_dej=POS(355, 310, -150, 0, 0, 0,config_habit);
        P_desecha= POS(-500.000,-370.605,-200.000,0,0,-90,config_habit);

        Dist_segu= POS(0,0,2,0,0,0,config_habit);

        Almacen= POS(-124.000,-528.000,-355.000,0,0,0,config_habit);
        CogeRectan= POS(24.7,50,2,0,0,0,config_habit);
        RectanArriba= POS(0,0,13.05,0,0,0,config_habit);
        CogeCilin= POS(24.7,50,39.1,0,0,0,config_habit);
        CilinArriba= POS(0,0,26,0,0,0,config_habit);
        Vagon2= POS(-106.000,-700.000,-333.000,0,0,0,config_habit);
        ColocaRectan= POS(-57.5,27.2,6.525,0,0,0,config_habit);
        Vagon1= POS(-228.000,-700.000,-333.000,0,0,0,config_habit);
        ColocaCilin1= POS(-83.1,27.2,17,0,0,0,config_habit);
        ColocaCilin2= POS(-32.9,27.2,17,0,0,0,config_habit);

        pal_1_1=POS(369, -663.272, -259.84, 0, 0, -90, config_habit);
        pal_1_2=POS(369, -731.3, -259.84, 0, 0, -90, config_habit);
        pal_1_3=POS(342.2, -698, -259.84, 0, 0, -90, config_habit);
        pal_2_1=POS(372.27, 730.358, -259.84, 0, 0, 90 ,config_habit);
        pal_2_2=POS(372.27, 661, -259.84,0, 0, 90, config_habit);
        pal_2_3=POS(344.6, 695.8,-259.84, 0, 0, 90, config_habit);

        P_vio=POS(762, 0, 2, 0, 0, 0, config_habit);

END ini_pos

ROUTINE init_system
VAR
        exito: INTEGER
BEGIN
-- Parametrizado de la herramienta
        herramienta= POS(0,0,101.34,-180,0,0,config_habit);
        $UTOOL = herramienta;
        WRITE (CR,'GARRA HABILITADA',CR);

-- Parametrizado del mundo
        mundo= POS(0,0,0,0,0,0,config_habit);
        $UFRAME = mundo;

-- Hacer complementarias las salidas del robot que controlan la muñeca;
        SET_PORT_MOD(io_rdo,1,2,exito);
```

Rutina que inicializa las localizaciones utilizadas en el programa.

Rutina que inicializa las variables de sistema

```
        WRITE('RDO1 Y RDO2 complementarias',exito,CR);

-- Hacer un grupo de señales digitales
        SET_PORT_ASG(io_gpin,1,0,1,io_din,19,5,exito);
        WRITE('ASIGNADAS LAS ENTRADAS',exito,CR);

-- Parametrizado de las velocidades de los movimientos
        $GROUP[1].$SPEED=2000;
        $MCR_grp[1].$PRGOVERRIDE= 100;

        $PARAM_GROUP[1].$JNTVELLIM[1]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[2]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[3]= 90;
        $PARAM_GROUP[1].$JNTVELLIM[4]= 400;
        $PARAM_GROUP[1].$JNTVELLIM[5]= 330;
        $PARAM_GROUP[1].$JNTVELLIM[6]= 480;
        $PARAM_GROUP[1].$JNTVELLIM[7]= 0;
        $PARAM_GROUP[1].$JNTVELLIM[8]= 0;
        $PARAM_GROUP[1].$JNTVELLIM[9]= 0;

-- Parametrizado de la interpolación de la orientación
        $ORIENT_TYPE= RSWORLD              -- Orientado mediante el método de los dos ángulos:

-- Parametrizado de la aceleración
        $GROUP[1].$usemaxaccel= TRUE
        $PARAM_GROUP[1].$cart_accel1= 224   -- Aceleración1 = 2 * aceleración2.
        $PARAM_GROUP[1].$cart_accel2= 112   -- Método de 2º orden, reduce vibraciones.
        $PARAM_GROUP[1].$accel_time1[1]= 224
        $PARAM_GROUP[1].$accel_time1[2]= 224
        $PARAM_GROUP[1].$accel_time1[3]= 224
        $PARAM_GROUP[1].$accel_time1[4]= 224
        $PARAM_GROUP[1].$accel_time1[5]= 224
        $PARAM_GROUP[1].$accel_time1[6]= 224
        $PARAM_GROUP[1].$accel_time2[1]= 112
        $PARAM_GROUP[1].$accel_time2[2]= 112
        $PARAM_GROUP[1].$accel_time2[3]= 112
        $PARAM_GROUP[1].$accel_time2[4]= 112
        $PARAM_GROUP[1].$accel_time2[5]= 112
        $PARAM_GROUP[1].$accel_time2[6]= 112

-- Inicialización de variables:

        terminar = false;
        paletizador=0;
        aproxima=0;

END init_system
```


Subprograma que se encarga de trasladar piezas desde la cinta principal a la cinta secundaria.

Traslada una pieza cada vez que se invoca.

```
ROUTINE trasl_cinta

BEGIN
        DOUT[1]=ON;   -- Comunicación con autómata.
        DOUT[3]=OFF;  -- Comunicación con autómata.

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_pi;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;
        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_aprox_pi;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_paso_1;

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_dej;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_dej_cin;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_dej;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;                   -- Comunicación con autómata.
        DOUT[1]=OFF;                  -- Comunicación con autómata.
END trasl_cinta
```

Subprograma que se encarga de ir amontonando las piezas en tres montones. Hace siempre una fila completa antes de subir a la siguiente altura.

Pone una pieza cada vez que es invocado y guarda la posición de la última pieza entre invocación e invocación.

```
ROUTINE paletiza

BEGIN
        DOUT[1]=ON;    -- Comunicación con autómata.
        DOUT[3]=OFF;   -- Comunicación con autómata.

        If paletizador >2 then
                paletizador=0;
                aproxima=aproxima+30;
        ENDIF

        paletizador=paletizador+1;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_pi;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = 200;
        MOVE TO P_aprox_pi;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_plt;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;

        SELECT paletizador OF
                CASE (1): MOVE NEAR P_palet_1 BY -aproxima
                CASE (2): MOVE NEAR P_palet_2 BY -aproxima
                CASE (3): MOVE NEAR P_palet_3 BY -aproxima
                CASE (4): terminar=TRUE; GO TO acabando
        ENDSELECT

        OPEN HAND 1;
```

Bucle que permite alternar entre las diferentes posiciones de los tres montones, y a su vez aumenta una altura cuando se ha completado una fila.


```
        WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_aprox_plt;

        CLOSE HAND 1;

acabando::
        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;                   -- Comunicación con autómata.
        DOUT[1]=OFF;                  -- Comunicación con autómata.

END paletiza
```

Subprograma que coge una pieza de la cinta principal y la desecha.

```
ROUTINE desecha

BEGIN
        DOUT[1]=ON;                 -- Comunicación con autómata.
        DOUT[3]=OFF;                -- Comunicación con autómata.
        $MOTYPE = JOINT;
        $TERMTYPE = FINE;
        $SPEED = max_vel;
        MOVE TO P_aprox_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;        -- Comprobación de garra abierta.

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = 200;
        MOVE TO P_rec_pi;

        CLOSE HAND 1;

        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = 200;
        MOVE TO P_aprox_pi;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE TO P_reposo;

        $TERMTYPE = FINE;
        MOVE TO P_desecha;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;        -- Comprobación de garra abierta.
        DELAY 100;                  -- Tiempo de espera para asegurar la caída de la pieza.
        CLOSE HAND 1;

        MOVE TO P_reposo;
        DOUT[2]=ON;                 -- Comunicación con el autómata.
        DOUT[1]=OFF;                -- Comunicación con el autómata.

END desecha
```


Rutina que se encarga de coger una pieza del tren y depositarla en un almacén. O de coger una pieza del almacén y dejarla en el tren.

```
ROUTINE mueve_tr(Tren_coge,Tren_deja: XYZWPR)

BEGIN
        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_coge BY –distancia

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;        -- Comprobación de garra abierta.

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_coge BY -(distancia+2);

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = aprox_vel;
        MOVE TO Tren_coge;

        CLOSE HAND 1;
        DELAY 200;

        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = aprox_vel;
        MOVE AWAY -distancia;

        $MOTYPE = JOINT;
        $TERMTYPE = NODECEL;
        $SPEED = max_vel;
        MOVE NEAR Tren_deja BY -distancia;

        $MOTYPE = LINEAR;
        $TERMTYPE = FINE;
        $SPEED = aprox_vel;
        MOVE TO Tren_deja;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;        -- Comprobación de garra abierta.
        DELAY 200;                  -- Tiempo de espera para asegurarse de haber dejado la pieza.

        $MOTYPE = LINEAR;
        $TERMTYPE = NODECEL;
        $SPEED = aprox_vel;
        MOVE AWAY -distancia;

END mueve_tr
```

Subprograma que invoca en el orden adecuado a la rutina de mover las piezas del tren, con el objeto de desmontarlo.

```
ROUTINE desm_tr

BEGIN
    DOUT[1]=ON;          -- Comunicación con el autómata.
    DOUT[3]=OFF;         -- Comunicación con el autómata.
    mueve_tr(Vagon2:ColocaRectan:RectanArriba,Almacen:CogeRectan);
    mueve_tr(Vagon2:ColocaRectan,Almacen:CogeRectan:RectanArriba);
    mueve_tr(Vagon1:ColocaCilin2,Almacen:CogeCilin:Dist_segu);
    mueve_tr(Vagon1:ColocaCilin1,Almacen:CogeCilin:CilinArriba:Dist_segu);
    DOUT[2]=ON;          -- Comunicación con el autómata.
    DOUT[1]=OFF;         -- Comunicación con el autómata.

END desm_tr
```

Subprograma que invoca en el orden adecuado a la rutina de mover las piezas del tren, con el objeto de montarlo.

```
ROUTINE monta_tr

BEGIN
    DOUT[1]=ON;          -- Comunicación con el autómata.
    DOUT[3]=OFF;         -- Comunicación con el autómata.
    mueve_tr(Almacen:CogeCilin:CilinArriba,Vagon1:ColocaCilin1:Dist_segu);
    mueve_tr(Almacen:CogeCilin,Vagon1:ColocaCilin2:Dist_segu);
    mueve_tr(Almacen:CogeRectan:RectanArriba,Vagon2:ColocaRectan:Dist_segu);
    mueve_tr(Almacen:CogeRectan,Vagon2:ColocaRectan:RectanArriba:Dist_segu);
    DOUT[2]=ON;          -- Comunicación con el autómata.
    DOUT[1]=OFF;         -- Comunicación con el autómata.

END monta_tr
```


Programa que se encarga de cargar en un palet (el de la derecha) tres piezas cogidas de la cinta principal.

```
ROUTINE cargar

BEGIN
    DOUT[1]=ON;          -- Comunicación con el autómata.
    DOUT[3]=OFF;         -- Comunicación con el autómata.
    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE TO P_aprox_pi;

    OPEN HAND 1;
    WAIT FOR DIN[8]=OFF;     -- Comprobación de garra abierta.

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    $SPEED=400;
    MOVE TO P_rec_pi;

    CLOSE HAND 1;

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    MOVE TO P_aprox_pi;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE NEAR pal_1_1 BY -100;

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    $SPEED=400;
    MOVE TO Pal_1_1;

    OPEN HAND 1;
    WAIT FOR DIN[8]=OFF;     -- Comprobación de garra abierta.

    MOVE AWAY -100;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE TO P_aprox_pi;

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    $SPEED=400;
    MOVE TO P_rec_pi;

    CLOSE HAND 1;

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    MOVE TO P_aprox_pi;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE NEAR pal_1_2 BY -100;
```

```
$MOTYPE=LINEAR;
$TERMTYPE=FINE;
$SPEED=400;
MOVE TO Pal_1_2;

OPEN HAND 1;
WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.

MOVE AWAY -100;

$MOTYPE=JOINT;
$TERMTYPE=NODECEL;
$SPEED=max_vel;
MOVE TO P_aprox_pi;

$MOTYPE=LINEAR;
$TERMTYPE=FINE;
$SPEED=400;
MOVE TO P_rec_pi;

CLOSE HAND 1;

$MOTYPE=LINEAR;
$TERMTYPE=FINE;
MOVE TO P_aprox_pi;

$MOTYPE=JOINT;
$TERMTYPE=NODECEL;
$SPEED=max_vel;
MOVE NEAR pal_1_3 BY -100;

$MOTYPE=LINEAR;
$TERMTYPE=FINE;
$SPEED=400;
MOVE TO Pal_1_3;

OPEN HAND 1;
WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.

MOVE AWAY -100;

$MOTYPE=JOINT;
$TERMTYPE=FINE;
$SPEED=max_vel;
MOVE TO P_reposo;

CLOSE HAND 1;
DOUT[2]=ON;                   -- Comunicación con el autómata.
DOUT[1]=OFF;                  -- Comunicación con el autómata.

END cargar
```



```
ROUTINE descargar
```

Programa que se encarga de descargar en un palet (el de la izquierda) tres piezas, y dejarlas en la cinta principal.

```
BEGIN
    DOUT[1]=ON;               -- Comunicación con el autómata.
    DOUT[3]=OFF;              -- Comunicación con el autómata.
    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE NEAR pal_2_1 BY -100;

    OPEN HAND 1;
    WAIT FOR DIN[8]=OFF;      -- Comprobación de garra abierta.

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    $SPEED=400;
    MOVE TO pal_2_1;

    CLOSE HAND 1;

    MOVE AWAY -100;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE TO P_vio;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE NEAR P_dej_pi BY -100;

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    $SPEED=400;
    MOVE TO P_dej_pi;

    OPEN HAND 1;
    WAIT FOR DIN[8]=OFF;      -- Comprobación de garra abierta.
    MOVE AWAY -100;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
    MOVE TO P_vio;

    MOVE NEAR pal_2_2 BY -100;

    $MOTYPE=LINEAR;
    $TERMTYPE=FINE;
    $SPEED=400;
    MOVE TO pal_2_2;

    CLOSE HAND 1;

    MOVE AWAY -100;

    $MOTYPE=JOINT;
    $TERMTYPE=NODECEL;
    $SPEED=max_vel;
```

```
        MOVE TO P_vio;

        MOVE NEAR P_dej_pi BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_dej_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.
        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR pal_2_3 BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO pal_2_3;

        CLOSE HAND 1;

        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_vio;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE NEAR P_dej_pi BY -100;

        $MOTYPE=LINEAR;
        $TERMTYPE=FINE;
        $SPEED=400;
        MOVE TO P_dej_pi;

        OPEN HAND 1;
        WAIT FOR DIN[8]=OFF;          -- Comprobación de garra abierta.
        MOVE AWAY -100;

        $MOTYPE=JOINT;
        $TERMTYPE=NODECEL;
        $SPEED=max_vel;
        MOVE TO P_reposo;
        DOUT[2]=ON;                   -- Comunicación con autómata.
        DOUT[1]=OFF;                  -- Comunicación con autómata.

END descargar
```


Inicio del programa principal.

```
BEGIN
        create_confi(config_habit,0,0,0,false,true,true);     --ponemos el valor de la conf. habitual

        ---------------------------
        -- Condition handlers
        ---------------------------

        CONDITION[3]:
                WHEN DIN[6] DO                -- Nos indican que acabemos el programa
                terminar=true
        ENDCONDITION;

        CONDITION[4]:
                WHEN DIN[7] DO                -- Parada momentánea del programa.
                STOP
                ENABLE CONDITION[5]
                DISABLE CONDITION[4]
        ENDCONDITION;

        CONDITION[5]:                         -- Reanudación después de la parada momentánea.
                WHEN NOT DIN[7] DO
                RESUME
                DISABLE CONDITION[5]
                ENABLE CONDITION[4]
        ENDCONDITION;

        -------------------------
        -- Inicio del programa
        -------------------------

        init_system;
        ini_pos;
        ENABLE CONDITION[3];          -- Habilita la posibilidad de apagar el robot.
        ENABLE CONDITION[4];          -- Habilita la parada si se abre la puerta (señal 7).

        ------------------------------
        -- Ejecución del programa.
        ------------------------------

        WHILE NOT terminar DO
                DOUT[3]=ON;           -- Comunicación con el autómata.

                IF din[1] THEN
                        trasl_cinta;
                        DELAY 100;
                        DOUT[2]=OFF;  -- Comunicación con el autómata.
                ENDIF;

                IF din[2] THEN
                        paletiza;
                        DELAY 100;
                        DOUT[2]=OFF;  -- Comunicación con el autómata.
                ENDIF

                IF din[3] THEN
                        desecha;
                        DELAY 100;
                        DOUT[2]=OFF;  -- Comunicación con el autómata.
                ENDIF
```

```
        IF din[4] THEN
            monta_tr;
            DELAY 100;
            DOUT[2]=OFF;  -- Comunicación con el autómata.
        ENDIF

        IF din[5] THEN
            desm_tr;
            DELAY 100;
            DOUT[2]=OFF;  -- Comunicación con el autómata.
        ENDIF

        IF din[9] THEN
            cargar;
            DELAY 100;
            DOUT[2]=OFF;  -- Comunicación con el autómata.
        ENDIF

        IF din[10] THEN
            descargar;
            DELAY 100;
            DOUT[2]=OFF;  -- Comunicación con el autómata.
        ENDIF

    ENDWHILE

    $MOTYPE = JOINT;
    $TERMTYPE = NODECEL;
    $SPEED = max_vel;
    MOVE TO P_casa;

END integra
```

Fin del programa principal.

Parte B. Programación del autómata.

B 1. Trabajo preparatorio

Para la realización de esta práctica, es indispensable haber leído el guión, y estar seguro de cómo se han de dar los pasos que en él se describen.

Además, resultará imprescindible llevar preparada al menos en un esquema, la red de Petri que se ha de utilizar en la programación del programa del autómata.

Si nunca se ha trabajado con el software Unity, se recomienda familiarizarse con el de alguna forma.

B 2. Descripción de la práctica.

Para detallar cómo se ha de programar el autómata, lo primero que hay que presentar es el método elegido para ello, en este caso una red de Petri, convertida en grafcet para funcionar con el software Unity.

Lo ideal a la hora de plantear la red de Petri, es que sea de la morfología más simple posible, pero que a su vez cumpla todos los requisitos pedidos. Deberá evitarse siempre que se pueda la proliferación de diferentes ramas de la red, ya que la complica en exceso y puede llevar a problemas de seguridad.

De cara a los aspectos de comunicación, debido a que hay suficientes entradas y salidas para que cada programa a ejecutar tenga la suya, se optará por esta opción. Hay que tener en cuenta que ante más tareas, lo normal sería pasarle al robot la instrucción a ejecutar por medio de una cadena de bits, que ofrece muchas más combinaciones posibles, y es mucho más eficiente si se tienen muchas opciones.

Por otra parte, la comunicación entre el robot y el autómata de cara al estado del programa, disponibilidad… Se realiza de la siguiente manera:

> El robot siempre que esté encendido, envía la señal de “Encendido”. Y activa otra señal siempre que no esté en parada de emergencia. (“Emergencia”).
>
> Además, cuando el robot está parado porque no hay ningún subprograma en ejecución, activa la señal de “Reposo”.
>
> Por el contrario, si está dentro de un subprograma, pero aún no están finalizadas las operaciones, activa la señal de “Ocupado” y desactiva la de “Reposo”.
>
> Por último, cuando termina un programa, activa la señal de “Finalizado” y la de “Reposo”, y desactiva la de “Ocupado”.

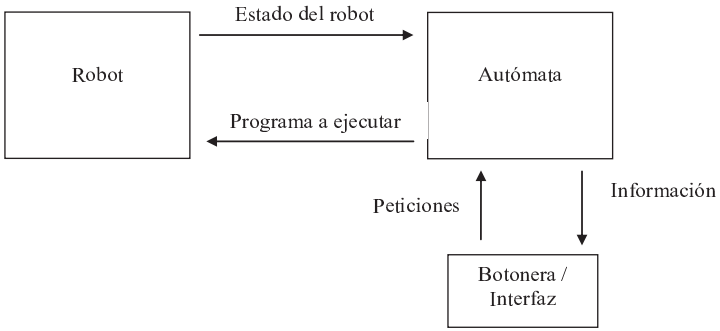
De cara al autómata, la señal de “Reposo” es necesaria para poder pasar al estado “Reposo” que a su vez es el previo a poder solicitar la ejecución de un programa. Por otra parte, la señal de “Ocupado” permite que la red de Petri avance al estado de ejecución del programa, una vez que se ha solicitado uno. Y la señal de finalización es la que activa la transición que envía al programa otra vez al estado de reposo.

La selección de los programas a ejecutar por parte del robot corre a cargo del usuario, y se permite realizarla de dos maneras diferentes. Por una parte, se puede hacer desde la

botonera del autómata, y por otra, desde un terminal táctil "Magelis" en el cual hay cargado un programa de supervisión "Scada".

Para evitar conflictos entre las dos posibilidades de interacción por parte del usuario, se deberá configurar de tal manera que la botonera del autómata pueda habilitar e inhabilitar al terminal gráfico "Magelis". Si bien cuando el terminal esté habilitado, la botonera quedará inhabilitada y viceversa.

Un esquema rápido de cómo deberían quedar las conexiones es el siguiente:



Las variables que el usuario podrá utilizar mediante la botonera son las siguientes:

Botonera	Nombre de la variable	Dirección interna Unity
DIN 0	IN_panel	%I0.1.0
DIN 1	IN_trasladar	%I0.1.1
DIN 2	IN_paletizar	%I0.1.2
DIN 3	IN_desechar	%I0.1.3
DIN 4	IN_montar_tren	%I0.1.4
DIN 5	IN_desmontar_tren	%I0.1.5
DIN 6	IN_finalizar	%I0.1.6
DIN 7	IN_detener	%I0.1.7
DIN 9	IN_cargar	%I0.1.9
DIN 10	IN_descargar	%I0.1.10

Variables de la botonera, nombres y direccionamientos.

Por parte del robot, las entradas y salidas que se deben configurar son las siguientes:

Variable (respecto Unity)	Dirección interna Unity	Comentario / Acción
Ent_1	%MW67.0	Dentro del subprograma.
Ent_2	%MW67.1	Terminado subprograma.
Ent_3	%MW67.2	Dentro del bucle principal (en ejecución).
Ent_8	%MW67.7	Estado de la garra del robot.
Ent_9	%MW67.8	Robot encendido.
Ent_10	%MW67.9	Emergencia.
Sal_1	%MW166.0	Ejecutar programa "Trasladar".
Sal_2	%MW166.1	Ejecutar programa "Paletizar".
Sal_3	%MW166.2	Ejecutar programa "Desechar".
Sal_4	%MW166.3	Ejecutar programa "Montar tren".
Sal_5	%MW166.4	Ejecutar programa "Desmontar tren".
Sal_6	%MW166.5	Solicitar fin de programa.
Sal_7	%MW166.6	Ordenar parada temporal inmediata.
Sal_8	%MW166.7	Indicar al robot estado de la garra.
Sal_9	%MW166.8	Ejecutar programa "Cargar palet".
Sal_10	%MW166.9	Ejecutar programa "Descargar Palet".

B 3. Organización de la práctica y evaluación.

La práctica nuevamente comprende dos partes, una obligatoria, y otra opcional. En este caso será mucho más importante la obligatoria.

Parte obligatoria (Hasta 8.5 puntos de 10):

- Lectura detenida del guión de la práctica y resolución de dudas.
- Preparar la red de Petri, y las posibles partes de código. Esto debe estar ya preparado antes de la práctica, y en la práctica solo debería programarse y probarse.
- Desarrollar durante la clase el programa que permita la integración del robot con el autómata, y enseñarle el correcto funcionamiento al profesor.

Parte opcional (Hasta 1.5 puntos de 10):

- Cualquiera de los apartados opcionales propuestos al final de este guión.
- Hay que recordar que se deberán traer también pensados, ya que el tiempo podría ser ajustado.

NOTA: La puntuación a recibir en ambas partes no dependerá exclusivamente de que se haya logrado realizar la práctica, sino que también se valorará la sencillez, corrección y elegancia. (Eficacia no es lo mismo que eficiencia).

B 4. Desarrollo de la práctica.

El desarrollo físico de esta práctica se va a realizar mediante el software de programación de autómatas Unity, y el autómata "Telemecanique Modicon M340".

A continuación se detallan los pasos para crear el programa, configurar las variables y hacer una red de Petri.

B 4.1 Abriendo Unity.

En el PC que nos haya sido asignado, abrimos la máquina virtual "Virtual Box" de la compañía "Sun Microsystems", y seleccionamos la máquina "Schneider".

Tras esperar pacientemente a que se cargue, desde el menú: inicio/programas/automática/Schneider/Unity, abrimos el programa Unity.

Tras esperar nuevamente a que se cargue, abrimos la plantilla de la práctica, que se encontrará en nuestra carpeta de prácticas.

Una vez abierta, lo primero que vamos a hacer será introducir la dirección del PLC al que deberemos conectarnos. Es la siguiente: 10.3.17.158. Para introducirla, vamos al menú "PLC" y "Establecer dirección". Lógicamente será de tipo TCP/IP.

Detalle de la conexión Ethernet.

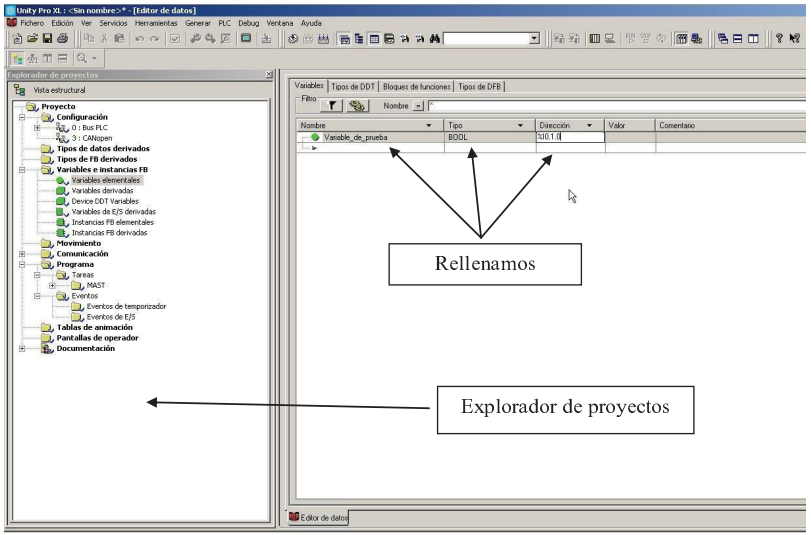
B 4.2 Configurando las variables.

Ahora es el turno de configurar las diferentes variables que vamos a necesitar como obligatorias. No obstante, más adelante se podrán añadir las que se consideren oportunas para programar la red de Petri.

Para ello vamos al explorador del proyecto (suele ser una ventanita a la izquierda de la pantalla), y abrimos el apartado variables elementales. Ahora toca añadirlas.

Añadimos las variables de la siguiente forma: Con clic en la casilla en blanco de la izquierda, añadimos el nombre de la variable. Nos aparecerá un tipo de variable por defecto (el último utilizado), cambiamos el tipo de variable a la que corresponda. Después solo habrá que rellenar la casilla de dirección. Dicha dirección es exactamente la que se ha adjuntado en la tabla en el apartado anterior. Hay que tener cuidado en escribirlas bien para ahorrar problemas futuros.

Respecto a los tipos de variables existentes, las variables que vengan de la botonera serán "Ebool", mientras que las variables de entrada y salida al robot deberán ser "bool", así como las variables internas del autómata. Por su parte, si tenemos algún contador, deberá ser lógicamente un "Integer".



Explorador de proyectos y ventana de variables.

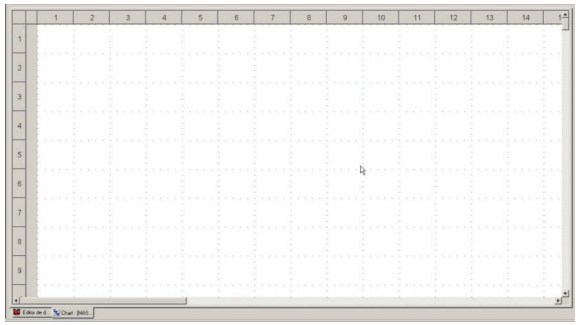
Cuando ya tengamos las variables, comprobaremos que de momento todo vaya bien dándole al botón generar cambios, que está presente en el menú "generar". Si todo va bien y no muestra errores, pasamos a guardar. (Se recomienda guardar de vez en cuando…).

B 4.3 Creación de la red de Petri.

Ahora toca introducir la red de Petri que ya viene pensada de casa en el programa. El modo de operar es bastante sencillo. En el apartado "programa/tareas/mast/secciones" del explorador del proyecto, hacemos clic derecho sobre "secciones" y añadir nuevo. Seleccionamos el tipo SFC.

Creación de la ventana de red de Petri.

Nos aparecerá una ventana como la que se muestra a continuación, en la cual podremos dibujar con facilidad la malla de la red de Petri.



Ventana de la red de Petri.

Sobre esta pantalla en blanco es hora de empezar a dibujar nuestra red. Para ello disponemos de los botones de estados y transiciones. Así como de desdoblamiento y reagrupación de caminos.

Detalle de los botones para la red de Petri.

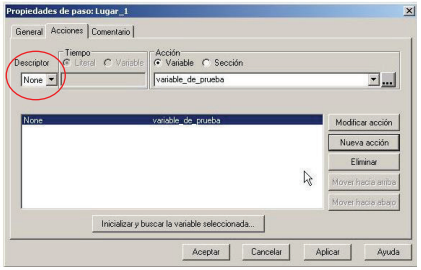
Literalmente hay que dibujar la red mediante la inserción de los bloques que toque poner. A cada bloque se le puede dar nombre haciendo doble clic sobre él.

Transiciones: Para configurar las transiciones hay dos opciones posibles. Hacemos doble clic sobre la transición a configurar. Ahora, la condición de la transición puede ser de dos tipos. Una señal simple (negada o no), o bien una condición más compleja. Si queremos seleccionar la condición compleja, habrá que hacer clic en "Sección", con lo cual se creará una nueva sección a la que deberemos dar nombre y programar. Hay que seleccionar el lenguaje ST.

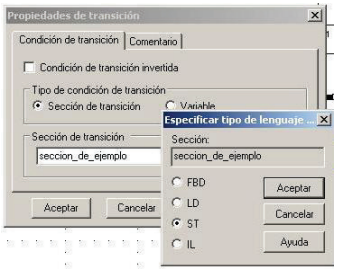
Lugares: Los lugares también tienen la opción de activar señales simples o hacer acciones más complejas. Ambas se pueden ejecutar una sola vez por cada vez que

Ejemplo de red simple.

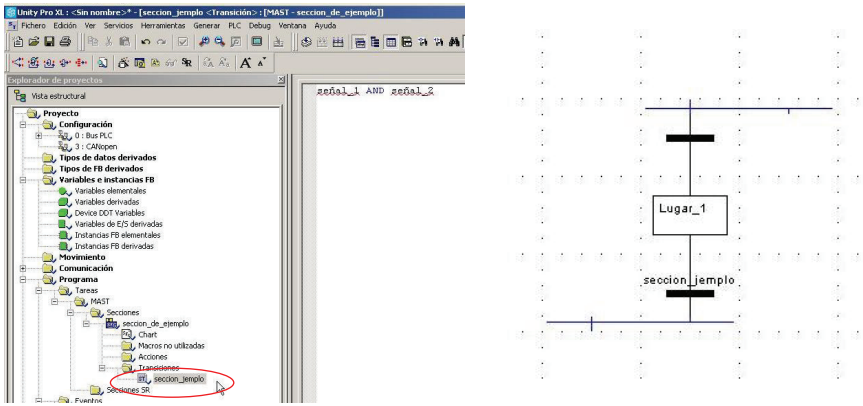
se pase por ese lugar, activarlo mientras esté en ese lugar o incluso desactivar una señal mientras se mantenga en ese lugar. Ello se cambia mediante los "descriptores" que aparecen a la izquierda de la señal y que se comentarán más adelante.



Creación de un lugar.

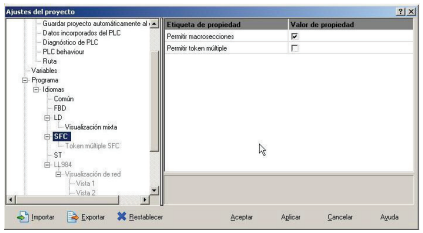


Creación de una transición.



Modificación de una condición de transición.

Como se observa, en el ejemplo dado hay más de una rama posible, por lo que podría darse el caso de que dos ramas avanzaran a la vez… El programa nos daría un error de proliferación de marcas (llamadas tokens), que soluciona yendo a "herramientas/ajustes del proyecto", y en la opción SFC, permitimos proliferar los tokens.



Proliferación de tokens.

Para saber concluir una red de Petri, no quedaría más que cerrar el bucle, conectando el final de la red con el principio mediante la herramienta del conector.

B 4.4. Conexión, carga y ejecución.

Ahora que ya está terminada la programación de la red, solo queda validar los cambios, para ello hay que generar cambios (generar/generar cambios). Si no hay ningún error, pasamos a transferirlo al autómata mediante "PLC/transferir proyecto". Cuando acabe, clic en ejecutar, y si todo está bien hecho, el programa se ejecutará sin problemas, y se podrá ver la evolución del mismo en la pantalla de la red de Petri.

B 5. Plantillas para la práctica.

Se dan a continuación algunas pistas e información útil para realizar la práctica.

B 5.1. Red de Petri.

Se recomienda hacer una red de Petri disjunta. Una de las sub-redes manejará los estados principales del robot y se ocupará de las emergencias, y la otra red se ocupará de atender los requerimientos del usuario.

Para la red de emergencias, se recomienda usar al menos los estados "Robot encendido", "emergencia" y "apagado".

Para la red de manejo de los programas se recomienda al menos un estado por cada programa de los siete que se pueden ejecutar, así como un estado ed reposo o robot preparado.

Se recomienda estudiar bien las variables que se indican en el sub-apartado B2, ya que en base a ellas se pueden obtener muchas pistas para hacer una red de Petri sencilla.

B 5.2. Variables.

Las variables que provoquen las transiciones a los distintos estados deben ser accesibles tanto por el autómata como por algún otro dispositivo externo, por lo que se recomienda pensar cómo (quizá con alguna variable auxiliar) se puede hacer para que unas veces opere la botonera y otras no.

Pista:

Mediante una variable auxiliar y un condicional, se puede hacer que unas veces se pueda acceder a una variable desde la botonera, y otras veces desde otro dispositivo.

Por ejemplo:

```
If "condición 1" then
        variable_principal := variable_fuente_1
else
        variable_principal := variable_fuente_2
end
```


B 5.3. Transiciones.

Sería bastante interesante que el autómata no cambie nunca de estado hasta recibir confirmación de que el robot ha terminado, o ha entrado en otro estado…

También es importante que la red de Petri no pueda cambiar de estados mientras el robot se encuentre en estado de emergencia o apagado…

B 5.4. Acciones.

Puede que algunas de las acciones deban tener las variables activadas durante todo el tiempo que la red está en ese lugar, y puede que otras (como el manejo de los huecos) solo deba producirse una vez cada vez que se pase por un lugar…

El descriptor "P1" del cuadro de diálogo de la creación de acciones, permite que sólo se realice esa acción una vez cada vez que se pasa por un lugar. El descriptor "N" mantiene encendida o apagada la acción mientras se esté en ese lugar.

B 6. Apartados opcionales.

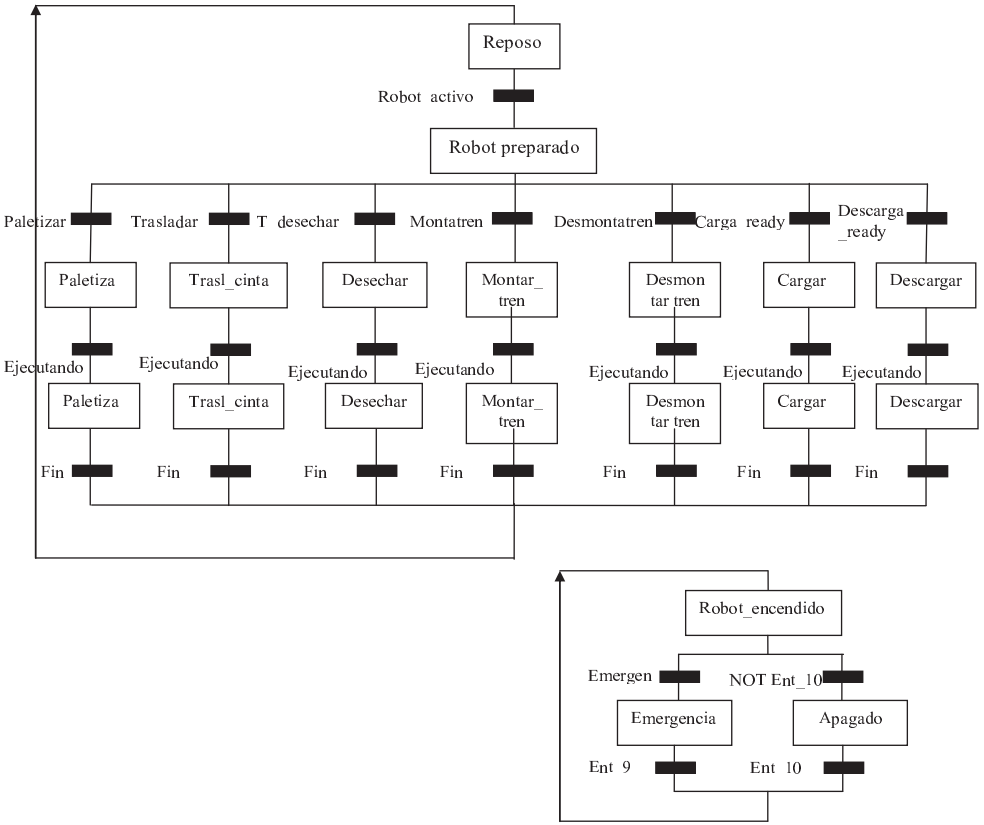
Se proponen a continuación para su resolución voluntaria y mejor calificación, los siguientes apartados opcionales.

- Inclusión de una serie de temporizadores en la red de Petri, que permitan pasar al estado de incidencia en caso de que ocurra un bloqueo dentro de un programa, o de que uno de los programas se quede parado sin ejecutar ninguna acción. Hay que tener en cuenta que dichos bloqueos podrían proceder tanto de un fallo del programa, como de que el usuario no indique ninguna acción durante un periodo largo de tiempo.
  Se valorará tanto la forma de implementarlos como la función que cumplan.
- Mejora de la subred que gestiona las incidencias.

B 7. Soluciones del programa.

B 7.1 Red de Petri.

Se propone la siguiente red de Petri disjunta. La primera red maneja los estados del programa que ejecuta el robot (Cargar, desechar…), y la segunda red maneja el estado del propio robot (Encendido, emergencia…).



B 7.2 Variables.

De cara a la utilización de variables, la recomendación es que las peticiones que se realicen tanto desde la botonera como desde el futuro terminal SCADA, sean tenidas en cuenta a través de unas variables "puente". De tal forma que por cada entrada de la botonera, exista un clon de dicha variable, que en el caso de estar habilitada la botonera, esté identificada con dicha variable, y en caso de estar habilitado el terminal SCADA, se puentee con el terminal.

En el caso propuesto, son las variables que se ha dado en llamar "puente_DIN".

Por otra parte, son necesarias otra serie de variables como el contador de huecos, algún booleano para saber si el tren está montado o el palet lleno…

Todas las variables utilizadas, se adjuntan en la siguiente tabla.

Variable	Tipo	Dirección	Comentarios
DIN_0	%I0.1.0	EBOOL	Entradas de la botonera.
DIN_1	%I0.1.1	EBOOL	
DIN_2	%I0.1.2	EBOOL	
DIN_3	%I0.1.3	EBOOL	
DIN_4	%I0.1.4	EBOOL	
DIN_5	%I0.1.5	EBOOL	
DIN_6	%I0.1.6	EBOOL	
DIN_7	%I0.1.7	EBOOL	
DIN_8	%I0.1.8	EBOOL	
DIN_9	%I0.1.9	EBOOL	
DIN_10	%I0.1.10	EBOOL	
DIN_11	%I0.1.11	EBOOL	
DIN_12	%I0.1.12	EBOOL	
DIN_13	%I0.1.13	EBOOL	
DIN_14	%I0.1.14	EBOOL	
DIN_15	%I0.1.15	EBOOL	
Ent_1	%MW67.0	BOOL	Entradas del módulo Advantys (del robot)
Ent_2	%MW67.1	BOOL	
Ent_3	%MW67.2	BOOL	
Ent_4	%MW67.3	BOOL	
Ent_5	%MW67.4	BOOL	
Ent_6	%MW67.5	BOOL	
Ent_7	%MW67.6	BOOL	
Ent_8	%MW67.7	BOOL	
Ent_9	%MW67.8	BOOL	
Ent_10	%MW67.9	BOOL	
Ent_11	%MW67.10	BOOL	
Ent_12	%MW67.11	BOOL	
Ent_13	%MW67.12	BOOL	
Ent_14	%MW67.13	BOOL	
Ent_15	%MW67.14	BOOL	
Ent_16	%MW67.15	BOOL	
Sal_1	%MW166.0	BOOL	Salidas del modulo Advantys (al robot)
Sal_2	%MW166.1	BOOL	
Sal_3	%MW166.2	BOOL	

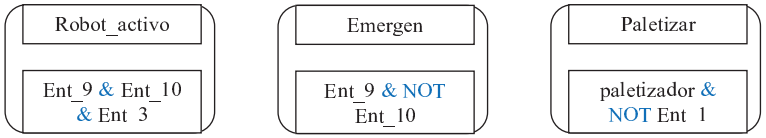
Sal_4	%MW166.3	BOOL	
Sal_5	%MW166.4	BOOL	
Sal_6	%MW166.5	BOOL	
Sal_7	%MW166.6	BOOL	
Sal_8	%MW166.7	BOOL	
Sal_9	%MW166.8	BOOL	
Sal_10	%MW166.9	BOOL	
Sal_11	%MW166.10	BOOL	
Sal_12	%MW166.11	BOOL	
Sal_13	%MW166.12	BOOL	
Sal_14	%MW166.13	BOOL	
Sal_15	%MW166.14	BOOL	
Sal_16	%MW166.15	BOOL	
Puente_din_0	%MW38.0	BOOL	Variables puente
Puente_din_1	%MW38.1	BOOL	
Puente_din_2	%MW38.2	BOOL	
Puente_din_3	%MW38.3	BOOL	
Puente_din_4	%MW38.4	BOOL	
Puente_din_5	%MW38.5	BOOL	
Puente_din_6	%MW38.6	BOOL	
Puente_din_7	%MW38.7	BOOL	
Puente_din_8	%MW38.8	BOOL	
Puente_din_9	%MW38.9	BOOL	
Puente_din_10	%MW38.10	BOOL	
Puente_din_11	%MW38.11	BOOL	
Puente_din_12	%MW38.12	BOOL	
Puente_din_13	%MW38.13	BOOL	
Tren_montado		BOOL	Indicador de si el tren está montado.
Palet_completo		BOOL	Indicador sobre si el palet ya está lleno.
Paletizador		BOOL	Petición de paletizar.
Descartador		BOOL	Petición de descartar.
Montador		BOOL	Petición de montar el tren.
Desmontador		BOOL	Petición de desmontar el tren.
Cargador		BOOL	Petición de cargar el palet.
Descargador		BOOL	Petición de descargar el palet.
Trasladador		BOOL	Petición de trasladar de cinta.
Hueco		INTEGER	Número de huecos disponibles en la cinta principal.

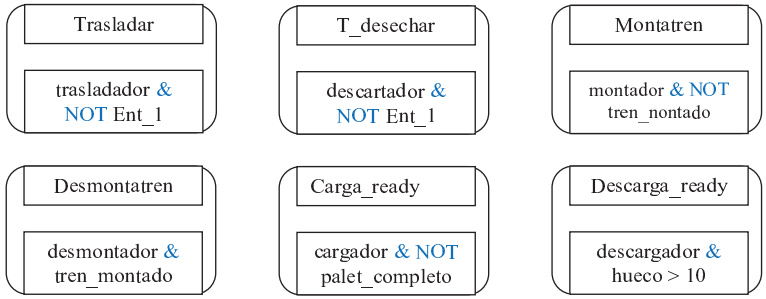
Tabla de todas las variables implementadas.

B 7.2.1 Transiciones

Para las transiciones, se sugiere que se implementen mediante código, ya que lo normal en este caso será que en cada transición se tengan en cuenta al menos dos variables diferentes.

A continuación se detallan todas las transiciones y sus variables activadoras.





B 7.2.2 Acciones.

Algunas acciones corresponden simplemente con la activación de la señal que corresponde a la petición de ejecutar un subprograma. Estas no merecen mayor consideración, porque consisten simplemente en la activación de dicha señal utilizando el descriptor N (ningún descriptor).

Sin embargo, sí que merecen la atención las que añaden o restan huecos a los existentes en la cinta. Dichas acciones deben ir acompañadas del descriptor "P1", ya que solo las activa una vez por ciclo e impide que cuenten varios huecos cuando debería contar solo uno.

Las acciones que se realizan mediante código son las siguientes:

huecos_menos_tres → hueco:=hueco-3;

huecos_menos_uno → hueco:=hueco-1;

huecos_mas_tres → hueco:=hueco+3;

Evidentemente, la acción "huecos menos tres" se ejecuta cuando el robot descarga un palet. La acción "huecos más tres" cuando el robot carga un palet. Y la acción "huecos menos uno", cuando el robot apila piezas, las traslada o las desecha.

B 7.3 Conexiones entre variables y código.

A continuación se adjuntan las conexiones internas que se realizan entre variables, algunas son de forma directa, como ya se explicará, y otras de forma indirecta a través de "puentes", cuya utilidad ya se ha explicado antes.

```
Sal_6:=Din_6;
Sal_7:=Din_7;
Sal_8:=Ent_8;
```

Variables conectadas permanentemente, fundamentalmente por motivos de seguridad. La entrada 6 indica el fin del programa, y la 7 la detención temporal. Mientras que la 8 es una "trampa" para que el robot lea su propia pinza.

```
IF din_0=false THEN
Puente_din_0:=Din_0;
Puente_din_1:=Din_1;
Puente_din_2:=Din_2;
Puente_din_3:=Din_3;
Puente_din_4:=Din_4;
Puente_din_5:=Din_5;
```

Condicional que permite la habilitación (o no) del terminal de control SCADA. Las variables "puente" sirven para que el terminal pueda leerlas, aunque no pueda modificar las variables que activan las transiciones.

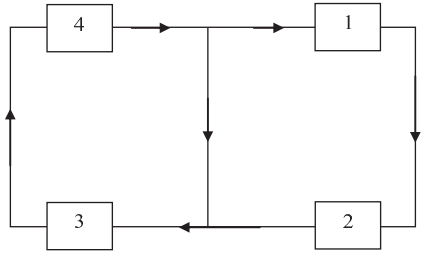
```
Puente_din_6:=Din_6;
Puente_din_7:=Din_7;
Puente_din_8:=DIN_8;
Puente_din_9:=Din_9;
Puente_din_10:=Din_10;
Puente_din_11:=Din_11;
Puente_din_12:=Din_12;
Puente_din_13:=Din_13;
Puente_din_14:=Din_14;
Puente_din_15:=Din_15;
trasladador:=Din_1;
paletizador:=Din_2;
descartador:=Din_3;
montador:=Din_4;
desmontador:=Din_5;
cargador:=Din_14;
Descargador:=Din_15;
ELSE
trasladador:=puente_din_1;
paletizador:=puente_din_2;
descartador:=puente_din_3;
montador:=puente_din_4;
desmontador:=puente_din_5;
cargador:=puente_din_14;
Descargador:=puente_din_15;
END_IF;
```

Variables que controlan las transiciones. Cuando el terminal está deshabilitado, son inaccesibles para él. Pero cuando se habilita, se puentean con las que el terminal puede escribir.

Aquí termina la programación del autómata. En estos momentos ya es capaz de comunicarse con el robot, y mediante la red de Petri creada puede manejar diferentes situaciones, y darnos información sobre qué está haciendo, o si ha sufrido algún percance.

El siguiente paso será hacer una interfaz de comunicación humano – máquina. Pero eso ya es otra práctica.

Anexo Entradas y Salidas de la célula de transporte del Robot:



Cada puesto de la célula tiene un tetón que actúa a modo de retención, y un elevador que hace que la cinta no roce con los transportadores.

Además, todos los puestos tienen un detector de metales para los trasportadores, y uno de presencia para las piezas. Excepto el puesto 1 que solo tiene detector de metal.

Puesto	Retención	Elevador	Detector	Det. Metales
1	O 0.5	O 0.0		I 0.0
2	O 0.6	O 0.1	I 1.5	I 0.1
3	O 0.7	O 0.2	I 1.6	I 0.2
4	O 1.0	O 0.3	I 1.7	I 0.3

Variable	Acción
O 0.4	Desvío intermedio
O 1.1	Retención intermedia
O 1.3	Cintas
O 1.4	Cintas
I 0.5	Detector de metal intermedio
I 0.7	Seta de emergencia
I 1.0	Marcha
I 1.1	IND / INT
I 1.2	Reset
I 1.3	MAN / AUTO

El funcionamiento de las cintas se produce de tal forma que los carros se muevan una posición después de cada vez que se haya ejecutado el programa de carga.

También existe la posibilidad de hacerlo manualmente mediante la botonera.

El control de las cintas se realiza mediante una red de Petri secuencial, que primero libera los carros, y después pone en marcha las cintas hasta que se vuelven a detectar todos los carros de nuevo.

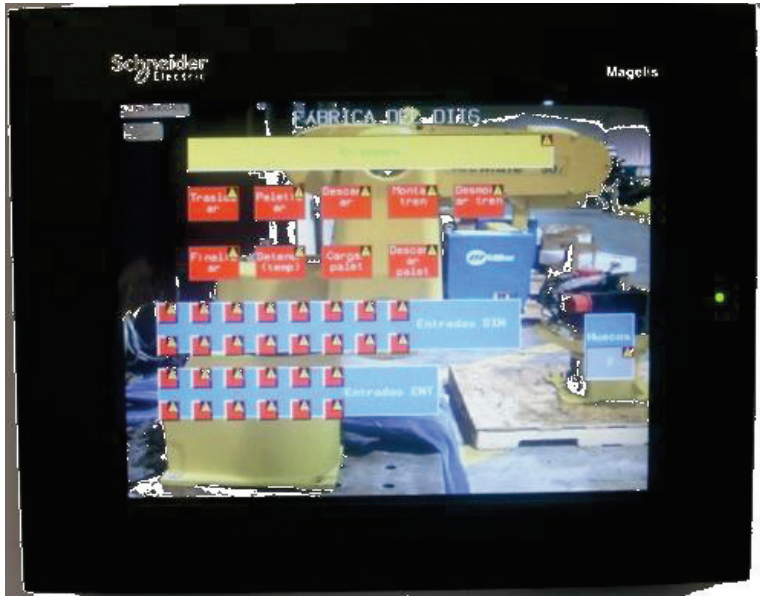
Apéndice I

Guión de la práctica de Integración (II)

Segunda parte de la práctica de integración del robot industrial con algunos componentes de la célula de fabricación del laboratorio L0.6.

EINA – UNIVERSIDAD DE ZARAGOZA

PROGRAMACIÓN DE UNA INTERFAZ DE TIPO SCADA PARA UNA CÉLULA DE ALMACENAJE Y TRANSPORTE.



| José Ignacio Abad Martínez

Programación de un sistema de supervisión SCADA.

1, trabajo preparatorio.

2, descripción de la práctica.

3, organización de la práctica y evaluación.

4, desarrollo de la práctica (Autómata del robot).

- 4.1, piloto.
- 4.2, interruptor con piloto.
- 4.3, panel de mensajes.
- 4.4, diseño propuesto de la ventana gráfica.

5, plantillas y ayuda para la resolución.

6, desarrollo de la práctica (Autómata de la estación 6).

7, apartados opcionales.

1. Trabajo preparatorio.

Para el correcto seguimiento y aprovechamiento de esta sesión práctica, se hace necesario leer detenidamente el guión completo de la misma.

Además de leer el guión, es necesario traer un boceto de cómo se piensan organizar los diferentes indicadores y botones dentro del espacio de la pantalla. Y también un esquema de cómo se van a organizar las conexiones, y qué señal o señales piensan usarse para dar la información necesaria.

Hay que traer también el guión y la resolución de la práctica anterior.

2. Descripción de la práctica.

Los sistemas de interfaz entre humano y máquina (HMI por sus siglas en inglés), están muy extendidos y son muy útiles en tanto en cuanto ayudan mucho a facilitar la comunicación entre dispositivos automáticos de control y las personas que se encargan de supervisarlos, revisarlos o trabajar con ellos.

En este caso se va a trabajar con el programa "Vijeo Designer", del fabricante de PLC Schneider. Además, también se utilizará la plataforma "Magelis", que es un dispositivo de pantalla táctil sobre el que se cargarán los programas SCADA.

Terminal "Magelis".

La supervisión que va a realizar el terminal una vez tenga listo su programa SCADA, va a consistir en dar al usuario información sobre las tareas que se están realizando, el estado de la célula y recibir peticiones por parte del usuario.

Para ello, el terminal dispondrá de varios botones táctiles, así como de diversos indicadores que permitirán recibir y visualizar la información de la célula.

A un nivel más bajo, el terminal va a acceder en modo de lectura, y en algunas ocasiones de escritura, mediante la red Ethernet, a las variables que se encuentran dentro del autómata. Con ello se logrará saber en qué estado se encuentra, qué salidas está activando, qué entradas le llegan…

En concreto, la práctica va a tener dos partes diferentes, aunque con una visión global. En primer lugar se programará la supervisión aislada de la célula que corresponde al robot. Más adelante se reprogramará para poder integrar con dicha célula, la estación 6 (servidor de pallets).

Los puntos imprescindibles a mostrar y a poder manejar mediante la pantalla del terminal "Magelis" en el caso del robot son los siguientes:

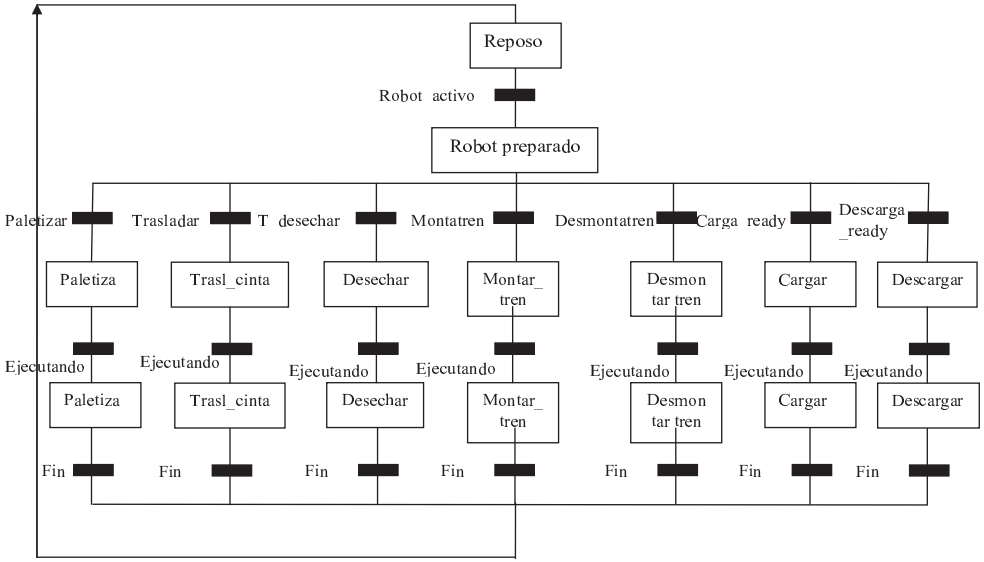
- Las entradas y salidas físicas (cableadas) al autómata.
- El estado global en el que se encuentra en cada momento el robot (Encendido, activado, programa en ejecución…).
- Los programas que se pueden seleccionar y si se están ejecutando o no.
- Si el terminal "Magelis" está activado o no.

Para el caso de la célula 6, como estará integrada con el robot habrá que tener en cuenta algún aspecto diferente. Habrá que añadir a lo anterior:

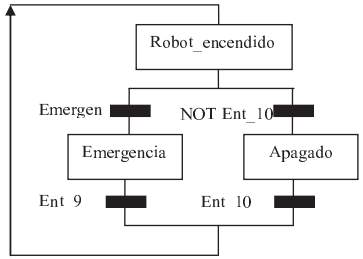
- Un seleccionador sobre qué pallet se desea expedir (negro o blanco).
- Un indicador de si el robot está en disposición de utilizar un nuevo pallet. (Debe estar impedido expedir sin que se permita…)
- La orden de expedir un pallet nuevo.
- El número de pallets expedidos, diferenciados por tipos.

2.1 Aspectos a conocer del programa de Unity sobre la estación del robot.

La estación del robot se encontrará preparada con un programa que utiliza la siguiente red de Petri:



Se trata de una red disjunta, de tal forma que hay otra en paralelo que controla las emergencias.



Además de la red, también es importante conocer las variables existentes. Se detallan a continuación.

Variable	Tipo	Dirección	Comentarios
DIN_0	%I0.1.0	EBOOL	Habilitador del panel.
DIN_1	%I0.1.1	EBOOL	Petición trasladar botonera.
DIN_2	%I0.1.2	EBOOL	Petición paletizar botonera.
DIN_3	%I0.1.3	EBOOL	Petición desechar botonera.
DIN_4	%I0.1.4	EBOOL	Petición montar tren botonera.
DIN_5	%I0.1.5	EBOOL	Petición desmontar tren botonera.
DIN_6	%I0.1.6	EBOOL	Petición finalizar botonera.
DIN_7	%I0.1.7	EBOOL	Petición detener botonera.
DIN_9	%I0.1.9	EBOOL	Petición cargar botonera.
DIN_10	%I0.1.10	EBOOL	Petición descargar botonera.
Ent_1	%MW67.0	BOOL	Dentro del subprograma.
Ent_2	%MW67.1	BOOL	Terminado subprograma.
Ent_3	%MW67.2	BOOL	Dentro del bucle principal (en ejecución).
Ent_8	%MW67.7	BOOL	Estado de la garra del robot.
Ent_9	%MW67.8	BOOL	Robot encendido.
Ent_10	%MW67.9	BOOL	Emergencia.
Sal_1	%MW166.0	BOOL	Ejecutar programa “Trasladar”.
Sal_2	%MW166.1	BOOL	Ejecutar programa “Paletizar”.
Sal_3	%MW166.2	BOOL	Ejecutar programa “Desechar”.
Sal_4	%MW166.3	BOOL	Ejecutar programa “Montar tren”.
Sal_5	%MW166.4	BOOL	Ejecutar programa “Desmontar tren”.
Sal_6	%MW166.5	BOOL	Solicitar fin de programa.
Sal_7	%MW166.6	BOOL	Ordenar parada temporal inmediata.
Sal_8	%MW166.7	BOOL	Indicar al robot estado de la garra.
Sal_9	%MW166.8	BOOL	Ejecutar programa “Cargar palet”.
Sal_10	%MW166.9	BOOL	Ejecutar programa “Descargar Palet”.
Tren_montado		BOOL	Indicador de si el tren está montado.
Palet_completo		BOOL	Indicador sobre si el palet ya está lleno.
Paletizador	%MW38.1	BOOL	Petición de paletizar.
Descartador	%MW38.2	BOOL	Petición de descartar.
Montador	%MW38.3	BOOL	Petición de montar el tren.
Desmontador	%MW38.4	BOOL	Petición de desmontar el tren.
Cargador	%MW38.5	BOOL	Petición de cargar el palet.
Descargador	%MW38.14	BOOL	Petición de descargar el palet.
Trasladador	%MW38.15	BOOL	Petición de trasladar de cinta.
Hueco		INTEGER	Número de huecos disponibles en la cinta principal.

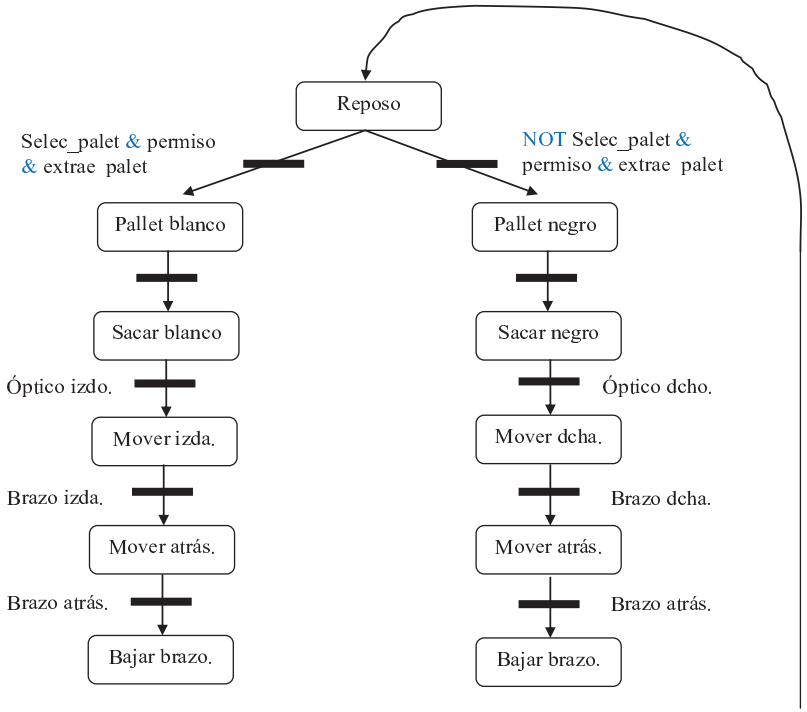
Tabla de todas las variables implementadas.

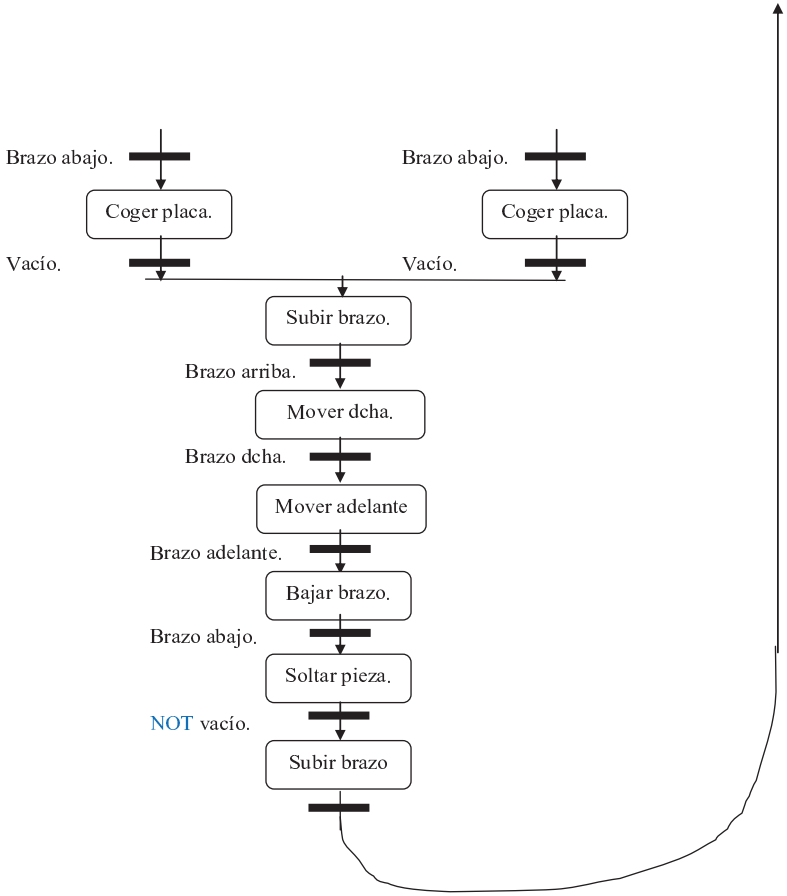
De ellas solo nos interesan aquellas que van a permitirnos solicitar acciones, o darnos información. Dichas variables se presentan marcadas con un fondo sombreado.

Para saber qué programa está ejecutando en cada momento, podemos recurrir a un sencillo truco consistente en mirar el estado de las salidas del autómata. Una salida activa nos indicará que se está ejecutando dicho programa. (Concretamente las variables “Sal” se encienden para ejecutar el programa que indican).

2.2. Aspectos a conocer del programa de Unity sobre la estación 6.

La estación 6 se encontrará preparada con un programa que utiliza la siguiente red de Petri:





Las señales, y direcciones que hay en el programa son las siguientes:

Variable	Dirección	Acción / comentario.
Expulsar izdo.	%MW134.0	Extiende el expulsor izdo.
Expulsar dcho.	%MW134.1	Extiende el expulsor dcho.
Coger placa.	%MW134.2	Genera vacío en las ventosas.
Bajar brazo.	%MW134.3	Baja el brazo.
Mover izda.	%MW134.4	Mueve el brazo a la izda.
Mover dcha.	%MW134.5	Mueve el brazo a la dcha.
Mover adelante.	%MW135.0	Mueve el brazo adelante.
Mover atrás.	%MW135.1	Mueve el brazo atrás.
Baliza.	%MW135.2	Enciende la baliza.
Izdo. Atrás.	%MW35.0	Indica que el impulsor izdo. Está atrás.
Óptico izdo.	%MW35.1	Señala la existencia de placa en la posición izda.
Dcho. Atrás.	%MW35.2	Indica que el impulsor dcho. Está atrás.
Óptico dcho.	%MW35.3	Señala la existencia de placa en la posición dcha.
Brazo arriba	%MW35.4	Indica que el brazo está arriba.
Brazo abajo	%MW35.5	Indica que el brazo está abajo.
Brazo dcha.	%MW37.0	Indica que el brazo está a la dcha.
Brazo izda.	%MW37.1	Indica que el brazo está a la izda.
Seta emergencia.	%MW37.2	Muestra si la seta de emergencia se ha pulsado.
Marcha	%MW37.3	Indica que el pulsador marcha está apretado.
Man/aut.	%MW37.4	Indica las dos posiciones de MAN. (Selección de la placa).
Reset.	%MW37.5	Indica que el pulsador “Reset” está apretado.
Ind/Int.	%MW39.0	Indica las posiciones de IND/INT.
Brazo atrás.	%MW39.1	Indica que el brazo está atrás.
Brazo adelante.	%MW39.2	Indica que el brazo está adelante.
Vacío.	%MW39.3	Indica que se ha efectuado el agarre de la pieza.

NOTAS:

- La selección de tipo de pallet en la botonera se realizará mediante el conmutador “MAN”.
- La seta de emergencia no viene implementada en la red porque a nivel hardware ya bloquea todo el proceso. No obstante, podría ser útil implementarla a nivel software y que permitiera alcanzar un estado seguro pulsando luego reset.

3. Organización de la práctica y evaluación.

La práctica consta de dos partes obligatorias, más algunos apartados opcionales.

Por supuesto se da por hecho que el guión es de obligada lectura antes de la práctica, así como su preparación. La preparación de esta práctica no requiere de escribir código, sino que fundamentalmente consiste en pensar cómo se han de conectar las variables, el diseño que se desea darle a la ventana del programa SCADA y cómo se van a cumplir los diversos requisitos planteados.

Célula del robot (OBLIGATORIO. Hasta 5 puntos de 10).

- Realizar el apartado 4. Simplemente consiste en seguir el guión y desarrollar las ideas sobre la interfaz gráfica que se hayan tenido. Esta parte es imprescindible terminarla durante la práctica.

Célula de expedición de pallets (Célula 6). (OBLIGATORIO. Hasta 3 puntos de 10).

- Realizar el apartado 6. Hay que seguir el guión, pero es muy importante haber comprendido los datos que se dan como previo de esta parte de la práctica. No es necesario ni implementar ni cambiar el programa de Unity sobre esta parte, en caso de que se desee probar alguna mejora en el apartado de los extras puede que sea necesario.

Extras y opcionales. (Opcional. Hasta 2 puntos de 10).

- Hacer cualquiera de los opcionales que se indican. Se valorará tanto cuántos se hagan como la calidad de los hechos.
  Es importante la calidad de lo hecho, por lo que más extras no implican necesariamente mejor nota…

NOTA: En todos los apartados se valorará la calidad de lo hecho, así como su grado de corrección. Por tanto terminar toda la práctica no implica obtener toda la nota.

4. Desarrollo de la práctica.

El primer paso lógicamente es abrir el programa "Vijeo Designer". Está dentro de "Automática/Schneider".

Hay que crear un nuevo proyecto y darle nombre. Luego se le debe dar nombre al destino (un único destino) (destino es el nombre que se le da al terminal Magelis), que es del tipo XBTGT 4000 Series. Concretamente el 4330, (640 x 480).

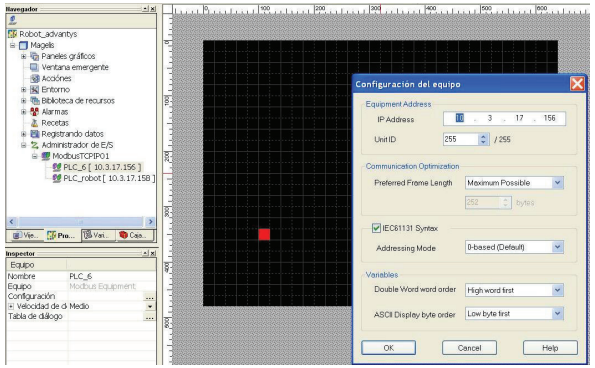
Se configura la dirección IP del destino (10.3.17.xxx). La máscara de subred es la 255.255.255.0, y la puerta de enlace 10.3.17.254.

Ahora hay que añadir un controlador (el autómata), del fabricante Schneider, modbus TCP, modbus equipment, y "Finalizar".

A la izquierda de la pantalla (del ordenador), está el navegador de la aplicación. Ahora toca configurar los dos dispositivos que se van a utilizar. Para ello, clic en "Modbus equipment01", con el botón derecho, y le ponemos un nombre más adecuado (PLC, PLC_cliente…). Haciendo doble clic sobre él, introducimos la dirección IP del

autómata (que lógicamente será 10.3.17.xxx), y se marca la casilla "IEC 61131 Syntax" con la que se indica que utilizaremos el mismo direccionamiento que en Unity.

Se aceptan los dos cuadros que aparecen.



Configuración de la IP del autómata.

Este último paso, se ha de repetir tantas veces como autómatas se vayan a supervisar a un mismo tiempo.

Ahora que ya está configurado el esqueleto del programa se pasa a una parte aburrida, y luego a otra más entretenida.

La aburrida es tener que dar de alta las variables. Para ello, vamos al cuadro de variables y vamos introduciendo las que necesitemos. Casi todas (todas menos una) serán de tipo discreto, y su dirección será del tipo %MWiXj. Esto coincide con la forma de direccionarlas en Unity (%MW35.2 por ejemplo). Obtendríamos que es: (%MW35X2).

Para el caso de los huecos, que era un entero, lo que hay que hacer es seleccionar el tipo entero, y luego introducir su dirección que es del tipo (%MW38).

NOTA: Hay que acordarse de seleccionar la fuente correcta de las variables, porque en caso contrario se tiene el riesgo de estar leyendo variables de otro autómata.

Una vez tengamos las variables necesarias introducidas, pasaremos a "dibujar" nuestra interfaz. Para ello se explica a continuación cómo hacer un piloto, un botón con piloto y un panel de mensajes. Poco más es necesario, además de que es bastante intuitivo.

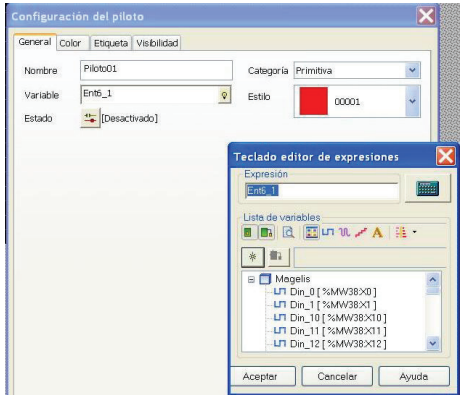
Una vez que hayamos terminado el diseño, lo validaremos con "Generar / generar". Y después toca pasarlo al terminal Magelis, para lo cual solo hay que darle a "descargar a (TCP…).

4.1. Piloto indicador.

Para introducir un piloto indicador de una señal, lo que hay que hacer es sencillamente clicar en el botón de piloto (una bombillita), y luego arrastrar el área que se desea que ocupe. Después hay que configurarlo. La pantalla puede salir sola al poner el piloto o ser necesario hacer doble clic sobre él.

En cualquier caso aparecerá una pantalla igual a la de la derecha.

En ella hay que seleccionar la variable que se mostrará a través del piloto, y en la pestaña color podemos cambiar los colores con los cuales indica si está activada o desactivada.

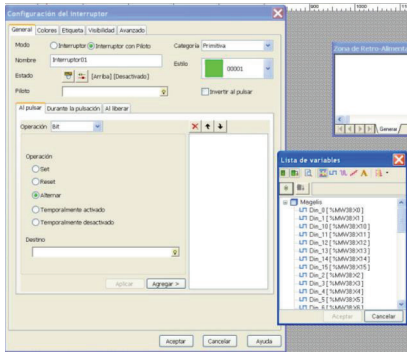


Configuración de un piloto.

Por defecto aparece rojo para activado y verde para inactivo, lo que es conveniente cambiar si queremos entenderlo fácilmente.

Se recomienda hacer alguna prueba utilizando la botonera.

4.2 Botón con piloto.



Configuración de botón con piloto.

El botón con piloto se comienza haciendo igual que el piloto, seleccionando su tecla, y arrastrando sobre el área deseada. Después, cuando abramos la ventana de configuración, aparecerá la que tenemos a la izquierda de estas líneas.

Se selecciona "interruptor con piloto" y hay que seleccionar dos variables. El piloto será la variable que queremos saber su estado, y el destino aquella que queremos modificar. Puede tratarse de la misma en ambos casos.

Si se desea que sea un interruptor o solo un pulsador, puede seleccionarse con los botones de "operación".

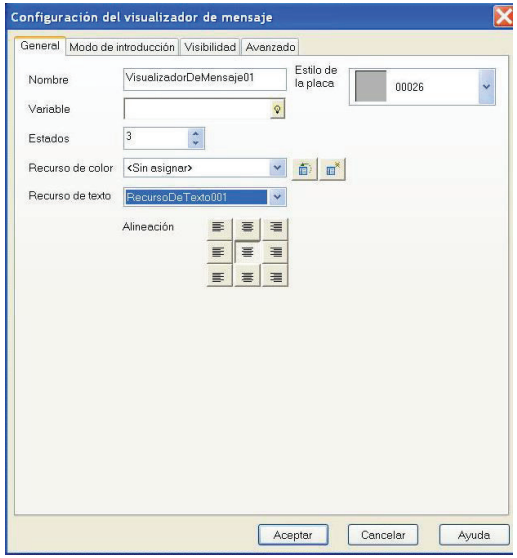
4.3. Panel de mensajes.

El último ejemplo que se configura es un panel de mensajes. Esto es: un panel en el que pueden aparecer diferentes mensajes dependiendo del estado o el valor de la variable que se le indique.

Para ello clic en el botón de "visualizador de mensaje". Y después de seleccionar el área que queremos que ocupe, lo configuramos.

Aquí hay que indicar en primer lugar la variable a controlar. Puede ser un bit, o un entero, dependiendo del número de mensajes a mostrar. Hay que seleccionar el número de estados que queremos que tenga.

Después habrá que configurar el recurso de texto. En el caso de un bit es muy sencillo porque solo hay dos opciones.



Configuración del visualizador de mensajes.

Pero en el caso de una cadena de bits (guardada como un entero), habrá que indicar en decimal el valor que va a tener, es decir: habrá que pasar de binario a decimal. Por ejemplo, si queremos que al encenderse la entrada 3 de una cadena de 8 bits, se active un mensaje, dicho mensaje irá asociado a: 00100000, lo cual en decimal equivaldrá a un 4.

4.4. Diseño propuesto de la ventana gráfica:

Aunque como es lógico, se permitirá cierta flexibilidad a la hora de situar los elementos necesarios dentro de la ventana que se va a mostrar en el terminal Magelis, se detallan a continuación los elementos que van a ser imprescindibles, y una propuesta de solución.

Evidentemente, se valorará adecuadamente el hacer un diseño mejor y/o más atractivo.

<u>Elementos obligatorios:</u>

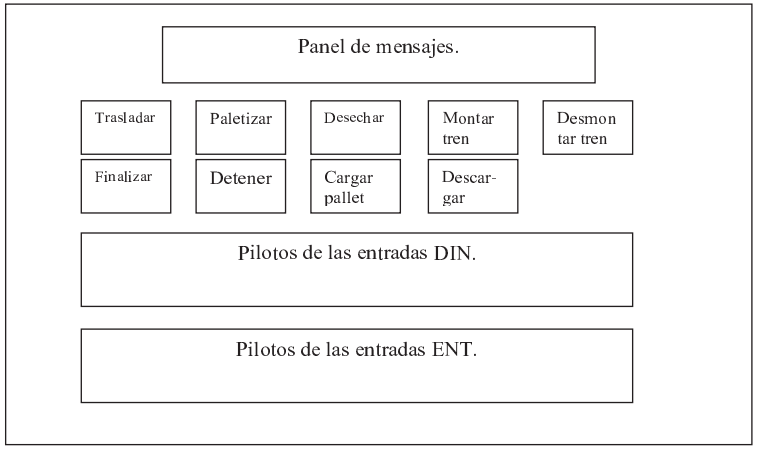
Un piloto por cada entrada DIN de la botonera.

Un piloto por cada entrada "ENT". (Salidas del robot).

Un botón con piloto por cada acción que es posible solicitar. (Trasladar, paletizar, desechar, cargar, descargar, montar tren y desmontar tren).

Un panel de mensajes que indique si el robot está en reposo, o qué programa está ejecutando.

Propuesta de cómo puede quedar la ventana:



Ejemplo de ventana para la interfaz SCADA.

5. Plantillas y ayuda para la resolución.

Se adjunta la siguiente imagen de cómo puede quedar la pantalla del terminal Magelis.

Ejemplo de ventana del terminal SCADA.

Es importante recordar que para que el terminal funcione correctamente, hay que utilizar las variables "puente" que se comentaron en la práctica anterior. Esto se debe a que quedan activadas o desactivadas en función de la botonera, lo cual es importante para la seguridad.

Pistas:

- No confundir las variables de tipo "ENT", con las variables de tipo "DIN". En este caso las variables de tipo "DIN" se recomienda que sean las variables puente del Unity. Hay que tener cuidado porque el SCADA puede leer cualquier variable del autómata, pero solo puede escribir en las que están dentro de la memoria del mismo.
- Ya se ha dicho antes, pero se recuerda que hay que seleccionar las variables de la fuente correcta.
- Para el visualizador de mensajes, lo mejor es crear un entero que contenga las variables que nos interesan, y "traducir" sus 1 y 0 a decimal.

6. Desarrollo de la práctica (Estación 6).

Ahora se pasa a realizar una segunda parte de la práctica, en la cual, desde el mismo autómata que se utilizaba antes, se pasará a controlar tanto el robot como la estación 6 de la célula de fabricación.

Al actuar el autómata de la estación 6 como esclavo del que estamos accediendo, no se necesitará más que acceder a las variables adecuadas en dicho autómata.

Otra opción diferente sería acceder a los dos autómatas por separado mediante el programa SCADA que estamos haciendo, pero esta opción presenta mayores problemas de robustez y fiabilidad.

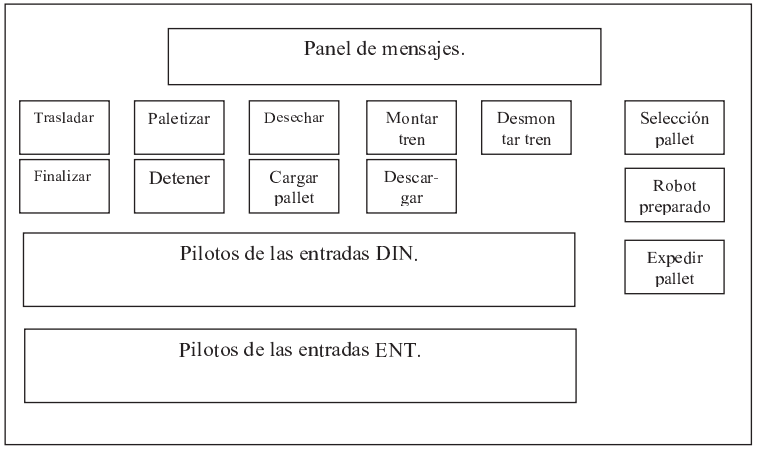
Variable	Dirección	Comentarios/Acciones
Selec_palet	%MW8.9	Selecciona el tipo de pallet a sacar.
Permiso	%MW40.0	Indica si el robot permite sacar otro pallet.
Extrae_palet	%MW40.1	Inicia la extracción de un pallet.
Carros_llenos	%MW10	Indica el numero de carros llenos (<=4)

Variables que controlan la estación 6.

Ahora solo es necesario añadir al terminal que anteriormente habíamos diseñado los nuevos botones y visualizadores necesarios. En concreto:

- Un seleccionador sobre qué pallet se desea expedir (negro o blanco).
- Un indicador de si el robot está en disposición de utilizar un nuevo pallet. (Debe estar impedido expedir sin que se permita…)
- La orden de expedir un pallet nuevo.

Una buena opción de panel sería la siguiente:



Panel SCADA del conjutno integrado

Una buena idea sería que el botón de selección de pallet se coloree en blanco o negro según la selección.

También puede ser buena idea camuflar el botón de expedir pallet cuando no esté habilitado, ponerlo en rojo, o hacerlo menos visible.

7. Apartados opcionales.

Se proponen los siguientes apartados opcionales, los cuales se podrán realizar durante la clase. Se recomienda traerlos preparados lo máximo posible de casa, y de esta forma se podrán hacer todos.

NOTA: Se recomienda hacer primero las dos partes de las prácticas, y luego los extras. Aunque algunos se pueden hacer directamente sobre la primera parte, es mejor no apurar el tiempo.

- Situar un visualizador de datos que indique cuantos huecos libres existen, mediante mostrar la variable “huecos”.
- Inclusión de un panel de aviso que indique cuándo está disponible la opción de descargar un pallet.
- Añadir un panel que indique el estado en el que se encuentra el robot. Tanto operativo, emergencia…, como en reposo, ejecutando programa…
- Introducir un nuevo panel (o modificar el existente) en el que se muestre el estado conjunto de la célula del robot y la de los pallets.
- Añadir a nivel software, tanto en la parte del autómata como en la parte de la interfaz SCADA, la opción de desbloquear una vez utilizada la parada de emergencia.
- Por supuesto, se valorará cualquier añadido o modificación que se haga, siempre y cuando quede demostrada y justificada su utilidad.

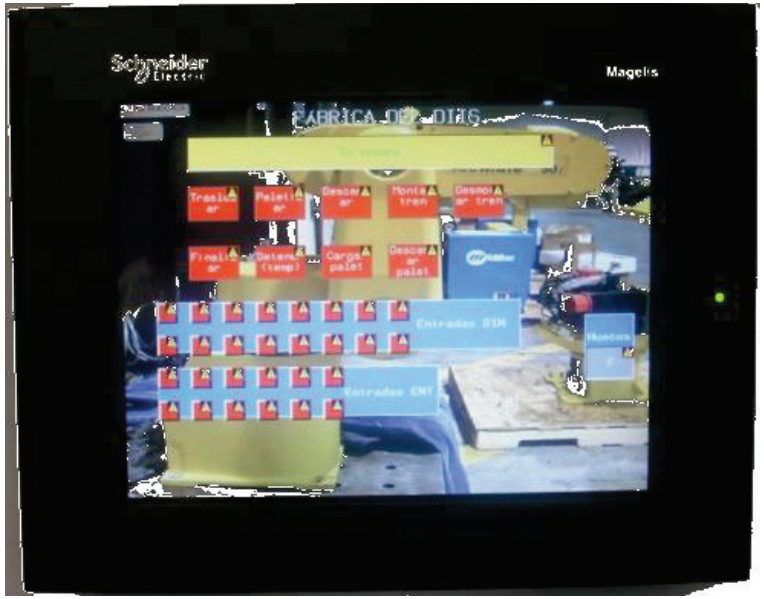
Apéndice J

Guión de la práctica de terminales de diálogo

Se adjunta a continuación la propuesta de guión para una práctica sobre la programación de terminales de diálogo. Dicha propuesta ha sido realizada en base al material obtenido durante el desarrollo de este proyecto.

EINA – UNIVERSIDAD DE ZARAGOZA

PROGRAMACIÓN DE UNA INTERFAZ DE TIPO SCADA PARA UNA CÉLULA DE ALMACENAJE Y TRANSPORTE.



| José Ignacio Abad Martínez

Programación de un sistema de supervisión SCADA.

1, trabajo preparatorio.

2, descripción de la práctica.

2.1, aspectos a conocer del programa de Unity sobre las estaciones.

3, organización de la práctica y evaluación.

4, control de la estación asignada.

4.1, piloto.

4.2, interruptor con piloto.

4.3, panel de mensajes.

5, apartados opcionales.

Anexos. (Imprime solo el que te corresponda).

1. Trabajo preparatorio.

Para el correcto seguimiento y aprovechamiento de esta sesión práctica, se hace necesario leer detenidamente el guión completo de la misma.

Además de leer el guión, es necesario traer un boceto de cómo se piensan organizar los diferentes indicadores y botones dentro del espacio de la pantalla. Y también un esquema de cómo se van a organizar las conexiones, y qué señal o señales piensan usarse para dar la información necesaria.

2. Descripción de la práctica.

Los sistemas de interfaz entre humano y máquina (HMI por sus siglas en inglés), están muy extendidos y son muy útiles en tanto en cuanto ayudan mucho a facilitar la comunicación entre dispositivos automáticos de control y las personas que se encargan de supervisarlos, revisarlos o trabajar con ellos.

En este caso se va a trabajar con el programa "Vijeo Designer", del fabricante de PLC Schneider. Además, también se utilizará la plataforma "Magelis", que es un dispositivo de pantalla táctil sobre el que se cargarán los programas.

Terminal "Magelis".

La supervisión que va a realizar el terminal una vez tenga listo el programa, va a consistir en dar al usuario información sobre las tareas que se están realizando, el estado de la célula y recibir peticiones por parte del usuario.

Para ello, el terminal dispondrá de varios botones táctiles, así como de diversos indicadores que permitirán recibir y visualizar la información de la célula.

A un nivel más bajo, el terminal va a acceder en modo de lectura, y en algunas ocasiones de escritura, mediante la red Ethernet, a las variables que se encuentran dentro del autómata. Con ello se logrará saber en qué estado se encuentra, qué salidas está activando, qué entradas le llegan…

Como se puede ver en las diferentes estaciones, existe una botonera industrial en cada una de ellas. La estación podrá ser controlada mediante esta botonera o mediante el terminal que hemos de diseñar. Para seleccionar quién tiene el control (los dos a la vez podría ser peligroso…), se utilizará la selección MAN/AUTO, presente en la botonera real.

Habrá tres alternativas para la realización de esta práctica. Podrá realizarse el control de la estación 1 (Expendedor de piezas), de la estación 3 (Enroscado de tapas) o de la estación 6 (Expendedora de pallets).

Los elementos a mostrar y controlar en los terminales en función de las diferentes estaciones dependerán de cada una de ellas. No obstante, todos los terminales tendrán los siguientes puntos en común.

- Visualización del estado de la botonera física del autómata.
- Visualización clara de la autorización de uso del terminal (Quién controla al autómata).
- Visualización clara del estado de la estación: En ejecución, parada, en emergencia...
- Botonera virtual. (Si está habilitada, permitirá controlar al autómata del mismo modo que la botonera física).

Botonera física de las estaciones.

2.1. Aspectos a conocer del programa de Unity sobre las estaciones.

→ Ahora es el momento de leer el anexo de la estación que te haya tocado.

Si bien el programa de control implementado en Unity, se dará ya cargado en las estaciones, es importante comprender su funcionamiento básico, y las variables más importantes que hay en cada uno. Así como las direcciones de memoria del autómata donde se colocan dichas variables.

Por lo tanto: Hay que tener localizadas las variables y sus direcciones.

Además, si se desea se podrá modificar el programa para mejorar su funcionalidad y su controlabilidad desde un terminal externo (Esto requiere conocimientos de Unity).

NOTA: En todas las estaciones el botón "MAN/AUT" de la botonera física decidirá quién controla la estación: La botonera real o el terminal Magelis. En cualquier caso, las setas de emergencia deberán funcionar en ambos controladores, y para desbloquear la estación será necesario desbloquear ambas setas, o la de la estación y pulsar "rearme".

3. Organización de la práctica y evaluación.

La práctica consta de una parte obligatoria, más algunos apartados opcionales.

Por supuesto se da por hecho que el guión es de obligada lectura antes de la práctica, así como su preparación. La preparación de esta práctica no requiere de escribir código, sino que fundamentalmente consiste en pensar cómo se han de conectar las variables, el

diseño que se desea darle a la ventana del programa SCADA y cómo se van a cumplir los diversos requisitos planteados.

Control básico de la célula asignada (OBLIGATORIO. Hasta 8 puntos de 10).

- Realizar el apartado 4. Simplemente consiste en seguir el guión y desarrollar las ideas sobre la interfaz gráfica que se hayan tenido. Esta parte es imprescindible terminarla durante la práctica.

Extras y opcionales. (Opcional. Hasta 2 puntos de 10).

- Hacer cualquiera de los opcionales que se indican más adelante. Se valorará tanto cuántos se hagan como la calidad de los hechos.

NOTA: En todos los apartados se valorará la calidad de lo hecho, así como su grado de corrección.

4. Control básico de la estación asignada.

El primer paso lógicamente es abrir el programa "Vijeo Designer". Está dentro de "Automática/Schneider".

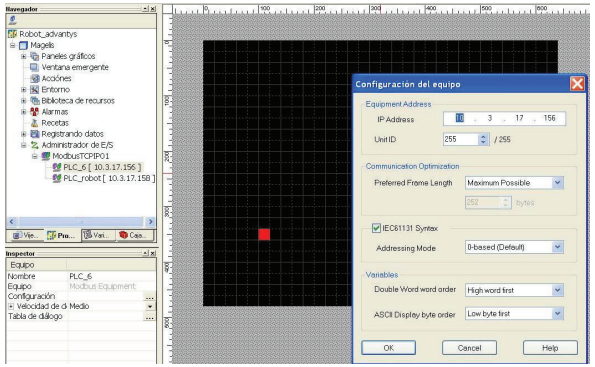
Hay que crear un nuevo proyecto y darle nombre. Luego se le debe dar nombre al destino (un único destino) (destino es el nombre que se le da al terminal Magelis), que es del tipo XBTGT 4000 Series. Concretamente el 4330, (640 x 480).

Se configura la dirección IP del destino (10.3.17.xxx). La máscara de subred es la 255.255.255.0, y la puerta de enlace 10.3.17.254.

Ahora hay que añadir un controlador (el autómata), del fabricante Schneider, modbus TCP, modbus equipment, y "Finalizar".

A la izquierda de la pantalla (del ordenador), está el navegador de la aplicación. Ahora toca configurar los dos dispositivos que se van a utilizar. Para ello, clic en "Modbus equipment01", con el botón derecho, y le ponemos un nombre más adecuado (PLC, PLC_cliente…). Haciendo doble clic sobre él, introducimos la dirección IP del autómata (que lógicamente será 10.3.17.xxx), y se marca la casilla "IEC 61131 Syntax" con la que se indica que utilizaremos el mismo direccionamiento que en Unity.

Se aceptan los dos cuadros que aparecen.



Configuración de la IP del autómata.

Este último paso, se ha de repetir tantas veces como autómatas se vayan a supervisar a un mismo tiempo.

Ahora que ya está configurado el esqueleto del programa se pasa a una parte aburrida, y luego a otra más entretenida.

La aburrida es tener que dar de alta las variables. Para ello, vamos al cuadro de variables y vamos introduciendo las que necesitemos. Casi todas (todas menos una) serán de tipo discreto, y su dirección será del tipo %MWiXj. Esto coincide con la forma de direccionarlas en Unity (%MW35.2 por ejemplo). Obtendríamos que es: (%MW35X2).

Para el caso de los huecos, que era un entero, lo que hay que hacer es seleccionar el tipo entero, y luego introducir su dirección que es del tipo (%MW38).

NOTA: Hay que acordarse de seleccionar la fuente correcta de las variables, porque en caso contrario se tiene el riesgo de estar leyendo variables de otro autómata.

Una vez tengamos las variables necesarias introducidas, pasaremos a "dibujar" nuestra interfaz. Para ello se explica a continuación cómo hacer un piloto, un botón con piloto y un panel de mensajes. Poco más es necesario, además de que es bastante intuitivo.

Los apartados obligatorios para esta parte son:

- Visualización del estado de la botonera física del autómata. (Pilotos).
- Visualización clara de la autorización de uso del terminal (Quién controla al autómata).
- Botonera virtual. (Permitirá controlar al autómata del mismo modo que la botonera física). (Lo lógico sería que los botones del terminal "Magelis" hicieran las dos funciones: Piloto y botón).
- Visualización del estado de la estación.

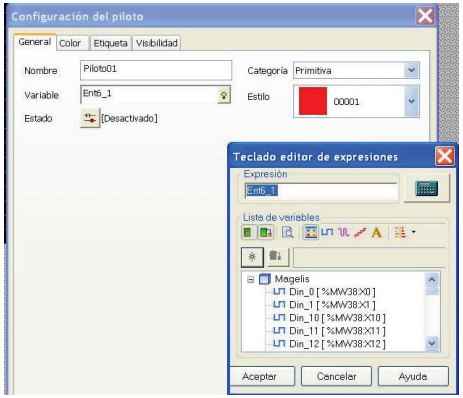
Una vez que hayamos terminado el diseño, lo validaremos con "Generar / generar". Y después toca pasarlo al terminal Magelis, para lo cual solo hay que darle a "descargar a (TCP…).

4.1. Piloto indicador.

Para introducir un piloto indicador de una señal, lo que hay que hacer es sencillamente clicar en el botón de piloto (una bombillita), y luego arrastrar el área que se desea que ocupe. Después hay que configurarlo. La pantalla puede salir sola al poner el piloto o ser necesario hacer doble clic sobre él.

En cualquier caso aparecerá una pantalla igual a la de la derecha.

En ella hay que seleccionar la variable que se mostrará a través del piloto, y en la pestaña color podemos cambiar los colores con los cuales indica si está activada o desactivada.

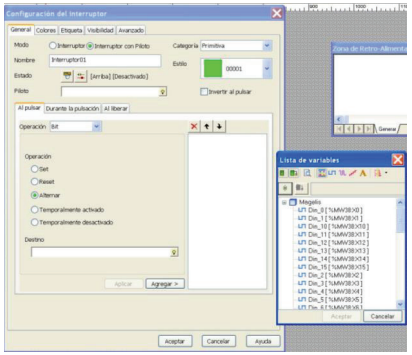


Configuración de un piloto.

Por defecto aparece rojo para activado y verde para inactivo, lo que es conveniente cambiar si queremos entenderlo fácilmente.

Se recomienda hacer alguna prueba utilizando la botonera.

4.2 Botón con piloto.



Configuración de botón con piloto.

El botón con piloto se comienza haciendo igual que el piloto, seleccionando su tecla, y arrastrando sobre el área deseada. Después, cuando abramos la ventana de configuración, aparecerá la que tenemos a la izquierda de estas líneas.

Se selecciona "interruptor con piloto" y hay que seleccionar dos variables. El piloto será la variable que queremos saber su estado, y el destino aquella que queremos modificar. Puede tratarse de la misma en ambos casos.

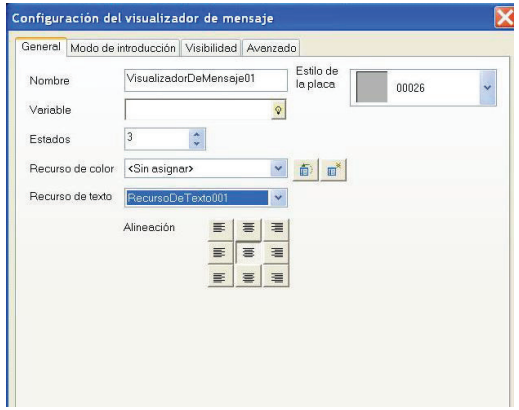
Si se desea que sea un interruptor o solo un pulsador, puede seleccionarse con los botones de "operación".

4.3. Panel de mensajes.

El último ejemplo que se configura es un panel de mensajes. Esto es: un panel en el que pueden aparecer diferentes mensajes dependiendo del estado o el valor de la variable que se le indique.

Para ello clic en el botón de "visualizador de mensaje". Y después de seleccionar el área que queremos que ocupe, lo configuramos.

Aquí hay que indicar en primer lugar la variable a controlar. Puede ser un bit, o un entero, dependiendo del número de mensajes a mostrar. Hay que seleccionar el número de estados que queremos que tenga.



Configuración del visualizador de mensajes.

Después habrá que configurar el recurso de texto. En el caso de un bit es muy sencillo porque solo hay dos opciones.

Pero en el caso de una cadena de bits (guardada como un entero), habrá que indicar en decimal el valor que va a tener, es decir: habrá que pasar de binario a decimal. Por ejemplo, si queremos que al encenderse la entrada 3 de una cadena de 8 bits, se active un mensaje, dicho mensaje irá asociado a: 00100000, lo cual en decimal equivaldrá a un 4.

5. Apartados opcionales.

Se proponen los siguientes apartados opcionales, los cuales se podrán realizar durante la clase. Se recomienda traerlos preparados lo máximo posible de casa, y de esta forma se podrán hacer todos. Hay que indicar que no todos los apartados se pueden hacer en todas las estaciones.

NOTA: Se recomienda hacer primero las dos partes de la práctica, y luego los extras. Aunque algunos se pueden hacer directamente sobre la primera parte, es mejor no apurar el tiempo.

- Inclusión de un panel de aviso que indique cuándo está disponible la opción de volver a ejecutar (implica otro botón que para indicar que se ha descargado).
- Añadir a nivel software, tanto en la parte del autómata como en la parte de la interfaz SCADA, una opción que permita la restauración segura.
- Por supuesto, se valorará cualquier añadido o modificación que se haga, siempre y cuando quede demostrada y justificada su utilidad.

Es posible que alguna de estas modificaciones implique cambiar la red de Petri de la estación, añadirle alguna variable o acceder a más variables de las que se han utilizado hasta el momento.

Anexo 1: descripción de la estación 1

Esta estación es la encargada de poner en el carro la primera parte de un cilindro o pieza sólida: su camisa. En un cargador por gravedad se almacenan camisas de cilindro y piezas sólidas apiladas en orden aleatorio.
La estación se encargará de ir sacando camisas de dicho almacén hasta que se encuentre una pieza con las características solicitadas. Para ello consta de un empujador que extrae una camisa, una zona de detección, un cabezal manipulador de 3 grados de libertad con una pinza en su extremo, y un cabezal con un sensor óptico para verificar si la camisa depositada en el carro de salida lleva tapa o no.
La identificación de las características de la pieza tratada en cada momento se realiza por medio de una tripleta de detectores: uno capacitivo, uno inductivo y uno óptico. Con la combinación de señales provenientes de dichos detectores se ha de ser capaz de diferenciar las diferentes camisas. A continuación se presenta una tabla resumen:

	Pieza negra	Pieza roja	Pieza metálica
Sensor capacitivo	Activado	Activado	Activado
Sensor óptico	Desactivado	Activado	Activado
Sensor inductivo	Desactivado	Desactivado	Activado

Funcionamiento básico

El inicio del proceso ha de lanzarse cuando el operario pulse el botón `Marcha` de la botonera de la estación. Eso permite trabajar con las camisas de una en una, teniendo el tiempo suficiente para retirar manualmente las camisas del carro, y –lo que es más importante para la seguridad del operario– hacerlo con la máquina parada.
En una aplicación real, algún ente jerárquicamente superior al controlador de esta estación le informaría de cuándo ha llegado un nuevo carro, y de qué tipo de camisa se necesita para comenzar a fabricar el cilindro (o si lo que se necesita es una pieza sólida de un color determinado). Como vas a trabajar de forma aislada, esa misma información hay que suministrársela al autómata.
Si abstraemos la selección de la carcasa a llevar al carro, el funcionamiento automático básico es el siguiente:

1) activar el empujador para sacar una camisa.

2) usando la información proporcionada por la tripleta de sensores, comparar la camisa extraída con la solicitada; si no es la correcta, hay que tirarla al depósito azul de camisas rechazadas; si es correcta, hay que llevarla al carro.

3) en cualquiera de los casos anteriores, hay que bajar el cabezal manipulador, cerrar la pinza, subir el cabezal, y desplazarlo (bien sea hacia el carro, o bien hacia el depósito de camisas rechazadas), bajar el cabezal1, y abrir la pinza.

4) aún si la camisa es correcta, todavía puede ocurrir que lo que la máquina haya sacado al carro sea una pieza sólida en lugar de una camisa para cilindro (o viceversa), en cuyo caso también hay que desecharla llevándola al contenedor de camisas rechazadas; para ello se dispone de un sensor fotoeléctrico montado sobre una pieza móvil, la cual hay que extender (con un cilindro neumático) para acceder a la parte superior de la pieza y comprobar si lleva tapa o no.

Descripción de las entradas y salidas

A continuación se tabulan las entradas y salidas de este automatismo. Puesto que se utiliza una periferia descentralizada (basada en bus CAN), las entradas y las salidas están *mapeadas* sobre bits de determinadas palabras de memoria (por lo que tendrás que tener especial cuidado cuando accedas a ellas en modo escritura. De hecho no se podrá acceder a estas variables en modo de escritura, sino que lo que hay que hacer es una "copia" de estas variables, que tendrá el valor de las leídas o de las que se les dé por el terminal Magelis en función de si el switch "MAN/AUT" está en una posición u otra.
Nótese que el símbolo `Reset` no puede usarse, al coincidir con una palabra reservada del lenguaje de programación, y en su lugar se ha usado `Rearme`. Tampoco existen acentos en los símbolos, evidentemente.

ENTRADAS		
Objeto	**Símbolo**	**Condición de puesta a 1**
%MW35.0	Cabezal_atras	Cabezal sobre la zona de detección de camisas.
%MW35.1	Cabezal_adelante	Cabezal sobre el carro.
%MW35.2	Cabezal_dentro	Cabezal sobre la zona de detección de camisas.
%MW35.3	Cabezal_fuera	Cabezal sobre el depósito de camisas rechazadas.
%MW35.4	Cabezal_arriba	Cabezal arriba.
%MW35.5	Cabezal_abajo	Cabezal abajo.
%MW37.0	Empujador_extendido	El empujador está extendido (saca una camisa).
%MW37.1	Empujador_contraido	El empujador está contraído.
%MW37.2	Seta_emergencia	Seta de emergencia enclavada.
%MW37.3	Marcha	Pulsador MARCHA apretado.
%MW37.4	Manual_automatico	En posición AUTO.
%MW37.5	Ind_int	En posición IND.
%MW39.0	Rearme	Pulsador RESET apretado.
%MW39.1	Capacitivo_camisa	El sensor capacitivo detecta camisa.

%MW39.2	Optico_camisa	El sensor óptico detecta camisa.
%MW39.3	Inductivo_camisa	El sensor inductivo detecta camisa.
%MW39.4	Lector_adelante	Cabezal del lector óptico extendido.
%MW39.5	Lector_atras	Cabezal del lector óptico recogido.
%MW41.0	Optico_lector	El sensor óptico del cabezal lector indica que existe tapa.

Respecto a las salidas, las números %MW134.4 a 7 corresponden a cilindros neumáticos de simple efecto. Las salidas %MW134.0 a 3 pueden (deben) manejarse de forma impulsional, ya que corresponden a cilindros de doble efecto.

SALIDAS		
Objeto	**Símbolo**	**Acción ejecutada si su valor es 1**
%MW134.0	Avanzar_cabezal	Lleva el cabezal hacia el carro.
%MW134.1	Retroceder_cabezal	Lleva el cabezal hacia la zona de detección de camisas.
%MW134.2	Sacar_cabezal	Saca el cabezal hacia el depósito de camisas rechazadas.
%MW134.3	Meter_cabezal	Mete el cabezal hacia la zona de detección de camisas.
%MW134.4	Bajar_cabezal	Baja el cabezal.
%MW134.5	Extender_cargador	Extiende el cargador que saca una camisa del alimentador por gravedad.
%MW134.6	Cerrar_pinza	Cierra la pinza del cabezal, cogiendo una camisa.
%MW134.7	Sacar_lector	Saca el lector óptico de verificación de tapa.

El cierre de la pinza no lleva un detector asociado, por lo que es necesario realizar una breve temporización antes de continuar con el proceso.

Botonera virtual: (Ind_Int permite la selección de pieza con o sin tapa).

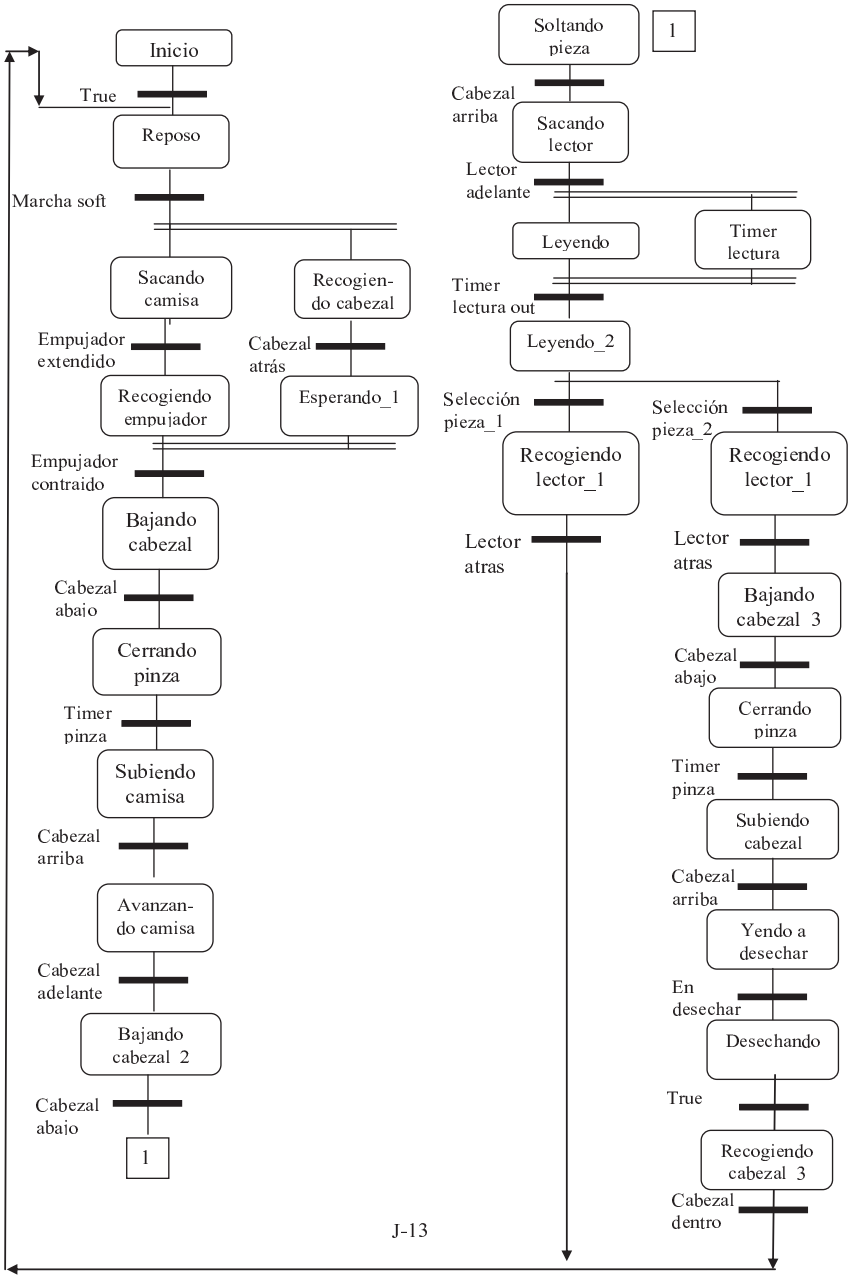
%MW 34.0	Seta_soft
%MW 34.1	Marcha_soft
%MW 34.2	Ind_Int_soft
%MW 34.3	Rearme_soft
%MW 33	Estado_estacion

Estado: 0 si reposo, 1 si produciendo, 2 si emergencia.

```
if not (man_auto) then
        marcha_soft:=marcha;
        rearme_soft:=rearme;
        ind_int_soft:=ind_int;
end_if
```

NOTA: Esta estación permite la implementación de numerosos controles adicionales, tales como la selección de las piezas. No hay necesidad de complicarse en exceso con ellos, de hecho se recomienda hacer primero los opcionales propuestos en el guión principal, y si queda tiempo intentar alguna mejora más.

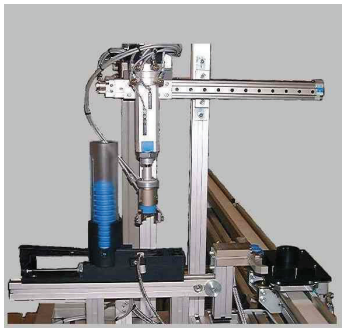
Red de Petri implementada en el autómata (SFC).



Anexo 2: descripción de la estación 3

Esta estación es la encargada de poner una tapa a los cilindros provenientes de la estación anterior. Consta fundamentalmente de un alimentador por gravedad de tapas (que supondremos que siempre tiene tapas), un empujador que las saca, una pinza que agarra las piezas a las que hay que colocar la tapa, y un brazo con movimiento en varios ejes que coge la tapa, la sitúa encima de la pieza y la enrosca.

Funcionamiento básico



El inicio del proceso ha de lanzarse cuando el operario pulse el botón `Marcha` de la botonera de la estación. Eso permite sacar las piezas de una en una, teniendo el tiempo suficiente para retirar manualmente la que se acaba de dejar, y –lo que es más importante para la seguridad del operario– hacerlo con la máquina parada. El funcionamiento automático básico es el siguiente:

1) sacar una tapa del alimentador por gravedad, para lo cual debe extenderse un cilindro de simple efecto existente en su base (salida `Sacar_tapa`); se entenderá que siempre hay al menos una tapa disponible en el alimentador; a pesar de existir un detector óptico (`Tapa_detectada`) que indica la presencia de tapa, ha de mantenerse la acción durante 1 segundo más para llevar la tapa hasta su posición correcta).

2) bajar el cabezal con la pinza (salida `Bajar_pinza`).

3) cerrar la pinza (salida `Cerrar_pinza`); esperar 1 segundo para asegurar la correcta sujeción de la tapa.

4) subir el cabezal con la pinza y llevarlo hacia la pieza (salidas `Bajar_pinza=0` y `Avanzar_brazo`); estos movimientos se pueden hacer simultáneamente, ahorrando tiempo respecto a una versión puramente secuencial).

5) sujetar la pieza con la mordaza (salida `Cerrar_mordaza`); realmente esto puede hacerse en cualquier momento desde que se haya pulsado `Marcha`.

6) bajar el cabezal.

7) roscar la tapa sobre la pieza (salida `Roscar`), lo que realiza un giro en el sentido de las agujas del reloj (*clockwise*).

8) abrir pinza (salida `Cerrar_pinza=0`); esperar 0.1 segundos para asegurar que está completamente abierta (no existe detector para ello).

9) subir el cabezal, girar la pinza en sentido contrario a las agujas del reloj (*counter-clockwise*) y llevar el cabezal hacia el alimentador por gravedad; tras esto, la máquina estará preparada para un nuevo ciclo.

En la figura adjunta se muestra la máquina en su posición de reposo, con el empujador de tapas sin extender, y el brazo con el cabezal atrás (es decir, listo para coger la tapa que ha de montar) y arriba. En la parte derecha, abajo, se aprecia un palet negro con una pieza (también negra) sobre la que hay que montar la tapa.

Descripción de las entradas y salidas

A continuación se tabulan las entradas y salidas de este automatismo. Puesto que se utiliza una periferia descentralizada (basada en bus CAN), las entradas y las salidas están *mapeadas* sobre bits de determinadas palabras de memoria (por lo que tendrás que tener especial cuidado cuando accedas a ellas en modo escritura. De hecho no se podrá acceder a estas variables en modo de escritura, sino que lo que hay que hacer es una "copia" de estas variables, que tendrá el valor de las leídas o de las que se les dé por el terminal Magelis en función de si el switch "MAN/AUT" está en una posición u otra.
Nótese que el símbolo `Reset` no puede usarse, al coincidir con una palabra reservada del lenguaje de programación, y en su lugar se ha usado `Rearme`. Tampoco existen acentos en los símbolos, evidentemente.

ENTRADAS		
Objeto	**Etiqueta**	**Condición de puesta a 1**
%MW35.0	Brazo_atras	El brazo está sobre el alimentador de tapas.
%MW35.1	Brazo_adelante	El brazo está sobre el palet con la pieza para montar la tapa.
%MW35.2	Cabezal_counter_cw	El cabezal está girado a *counter-clockwise*.
%MW35.3	Cabezal_cw	El cabezal está girado a *clockwise*.
%MW35.4	Cabezal_arriba	El cabezal con la pinza está arriba.
%MW35.5	Cabezal_abajo	El cabezal con la pinza está abajo.
%MW37.0	Tapa_detectada	El detector óptico detecta la existencia de tapa.
%MW37.1	Seta_emergencia	Seta de emergencia enclavada.
%MW37.2	Marcha	Pulsador MARCHA apretado.
%MW37.3	Ind_int	En posición IND.
%MW37.4	Rearme	Pulsador RESET apretado.
%MW37.5	Man_aut	En posición AUTO.

Respecto a las salidas, sólo las números 0 y 1 corresponden a cilindros neumáticos de doble efecto; esto hace que si ambas están activas, el cilindro neumático no se moverá. Para el resto, cuando la salida se pone a 0 se realiza el movimiento opuesto (por la acción del muelle interno).

SALIDAS		
Objeto	**Símbolo**	**Acción ejecutada si su valor es 1**
%MW134.0	Avanzar_brazo	Llevar el brazo hacia el palet.
%MW134.1	Retroceder_brazo	Llevar el brazo hacia el alimentador de tapas.
%MW134.2	Roscar	Girar el cabezal *clockwise*.
%MW134.3	Bajar_pinza	Bajar la pinza.
%MW134.4	Sacar_tapa	Extender el empujador del alimentador de tapas.
%MW134.5	Cerrar_mordaza	Cierra la mordaza de sujeción de la pieza.
%MW135.0	Cerrar_pinza	Cierra la pinza que sujeta la tapa.

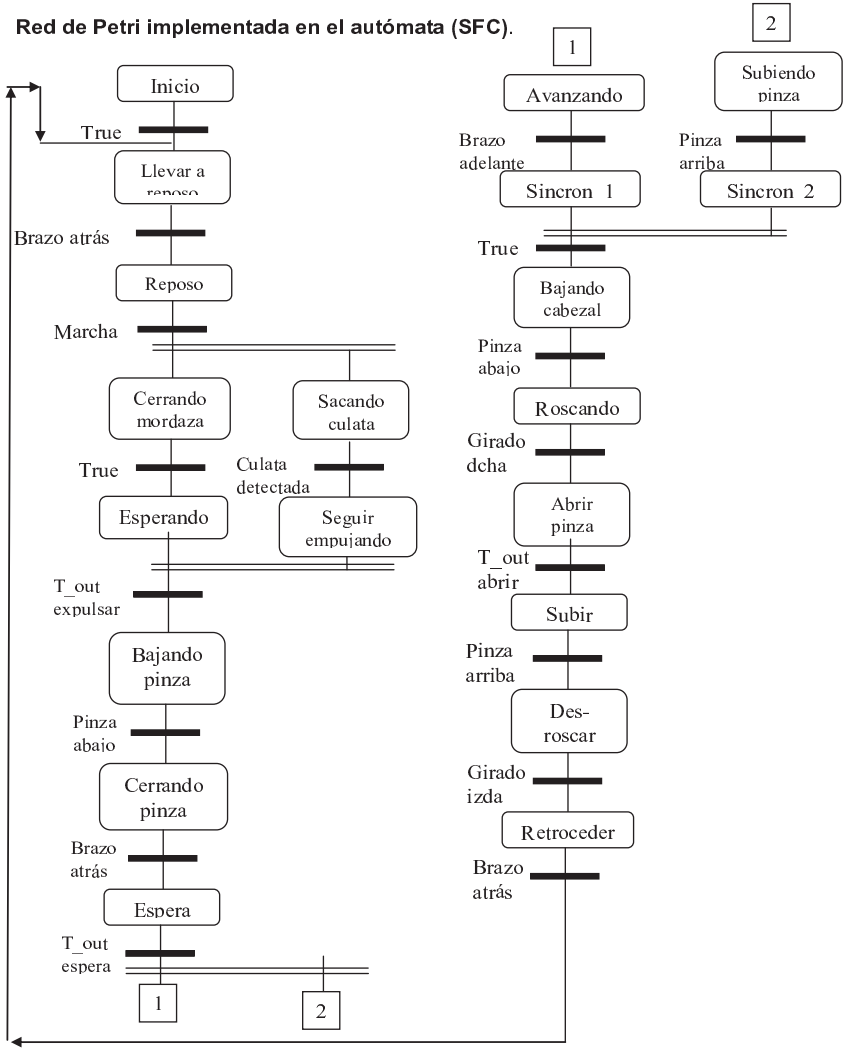
Botonera virtual:

%MW 30.0	Ind_Int_soft
%MW 30.1	Marcha_soft
%MW 30.2	Reset_soft
%MW 30.3	Seta_soft
%MW 30.4	Man_auto_soft
%MW 29	estado_estacion

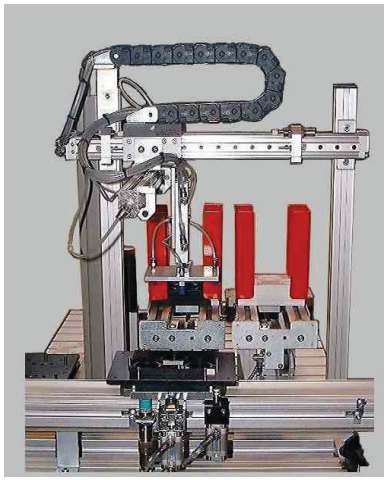
```
if not (man_auto) then
        marcha_soft:=marcha;
        reset_soft:=reset;
        ind_int_soft:=ind_int;
end_if
```

Estado: 0 si reposo, 1 si produciendo, 2 si emergencia.

Red de Petri implementada en el autómata (SFC).



Anexo 2: descripción de la estación 6



La estación 6 suministra las placas base sobre las que se montan los pedidos que luego se almacenarían en la estación 7. Existen dos tipos diferentes de placas, cuya única diferencia es el color: negro y blanco. Por medio de una serie de cilindros neumáticos y unas ventosas de vacío tendremos la posibilidad de trasladar las piezas de un punto a otro hasta colocarlas sobre el palet. La estación posee dos alimentadores por gravedad sobre los cuales introduciremos cada uno de los dos tipos de base posibles, uno en cada uno de ellos. La situación de cada uno de los tipos de base debe ser la mostrada en la figura adjunta. También hay que colocar las placas correctamente orientadas: si se meten de forma errónea las ventosas del brazo que debe manipularlas coincidirán con los huecos de la placa y no se podrán levantar (al perderse el efecto de succión sobre la placa). Así que es necesario mirar la disposición de las ventosas en el brazo manipulador antes de introducir placas en los almacenes.

Funcionamiento básico

El inicio del proceso ha de lanzarse cuando el operario pulse el botón `Marcha` de la botonera de la estación. Eso permite sacar las piezas de una en una, teniendo el tiempo suficiente para retirar manualmente la que se acaba de dejar, y –lo que es más importante para la seguridad del operario– hacerlo con la máquina parada. El funcionamiento automático básico es el siguiente:

1) sacar una placa del alimentador por gravedad, para lo cual debe extenderse un cilindro de simple efecto existente en su base

2) con el cilindro completamente extendido (no existe sensor para detectar dicha extensión): si existe placa será detectada por un sensor óptico (y ya puede recogerse el cilindro que lo sacó); si no existe placa habrá que avisar al operador de que debe recargar el alimentador que se ha quedado vacío (lo que puede hacerse mediante una baliza existente en la propia estación)

3) posicionar el brazo sobre la placa recién extraída, bajarlo, hacer vacío para sujetar la placa: cuando el sensor detecte que se ha producido el vacío, significará que la placa está sujeta, y podremos subir el brazo

4) llevar la placa hacia el carro, bajarla, y desactivar la succión; cuando el sensor de vacío nos indique que se ha dejado de succionar, subir el brazo y llevarlo donde corresponda

Descripción de las entradas y salidas

A continuación se tabulan las entradas y salidas de este automatismo. Puesto que se utiliza una periferia descentralizada (basada en bus CAN), las entradas y las salidas están *mapeadas* sobre bits de determinadas palabras de memoria (por lo que tendrás que tener especial cuidado cuando accedas a ellas en modo escritura. De hecho no se podrá acceder a estas variables en modo de escritura, sino que lo que hay que hacer es una "copia" de estas variables, que tendrá el valor de las leídas o de las que se les dé por el terminal Magelis en función de si el switch "MAN/AUT" está en una posición u otra. Nótese que el símbolo `Reset` no puede usarse, al coincidir con una palabra reservada del lenguaje de programación, y en su lugar se ha usado `Rearme`. Tampoco existen acentos en los símbolos, evidentemente.

ENTRADAS		
Objeto	**Símbolo**	**Condición de puesta a 1**
%MW35.0	Izdo_atras	El expulsor izquierdo está atrás.
%MW35.1	Optico_izdo	Hay placa sobre el sensor óptico izquierdo.
%MW35.2	Dcho_atrás	El expulsor derecho está atrás.
%MW35.3	Optico_dcho	Hay placa sobre el sensor óptico derecho.
%MW35.4	Brazo_arriba	El brazo que manipula la placa está arriba.
%MW35.5	Brazo_abajo	El brazo que manipula la placa está abajo.
%MW37.0	Brazo_dcha	El brazo que manipula la placa está a la derecha.
%MW37.1	Brazo_izda	El brazo que manipula la placa está a la izquierda.
%MW37.2	Seta_emergencia	Seta de emergencia enclavada.
%MW37.3	Marcha	Pulsador MARCHA apretado.
%MW37.4	Man_aut	En posición AUT.
%MW37.5	Rearme	Pulsador RESET apretado.
%MW39.0	Ind_int	En posición INT.
%MW39.1	Brazo_atras	El brazo que manipula la placa está a la atrás.
%MW39.2	Brazo_adelante	El brazo que manipula la placa está a la adelante.
%MW39.3	Vacio	El vacuostato detecta vacío.

Respecto a las salidas, las %MW134.0, .1 y .3 corresponden a cilindros neumáticos de simple efecto. Cuando la salida %MW134.2 vale 0, se deja de generar vacío. Las salidas %MW134.4 a %MW135.1 pueden (deben) manejarse de forma impulsional, ya que corresponden a cilindros de doble efecto.

SALIDAS		
Objeto	**Símbolo**	**Acción ejecutada si su valor es 1**
%MW134.0	Expulsar_izdo	Extender expulsor izquierdo.
%MW134.1	Expulsar_dcho	Extender expulsor derecho.
%MW134.2	Coger_placa	Generar vacío en las ventosas.
%MW134.3	Bajar_brazo	Bajar el brazo.
%MW134.4	Mover_izda	Mover el brazo hacia la izquierda.
%MW134.5	Mover_dcha	Mover el brazo hacia la derecha.
%MW135.0	Mover_adelante	Mover el brazo hacia la adelante.
%MW135.1	Mover_atras	Mover el brazo hacia la atrás.
%MW135.2	Baliza	Encender la luz de la baliza luminosa.

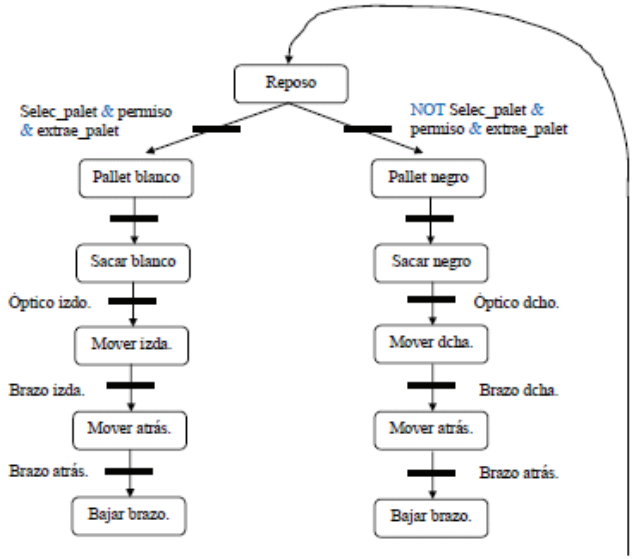
Botonera virtual: (Ind_int permite la selección de base blanca o negra)

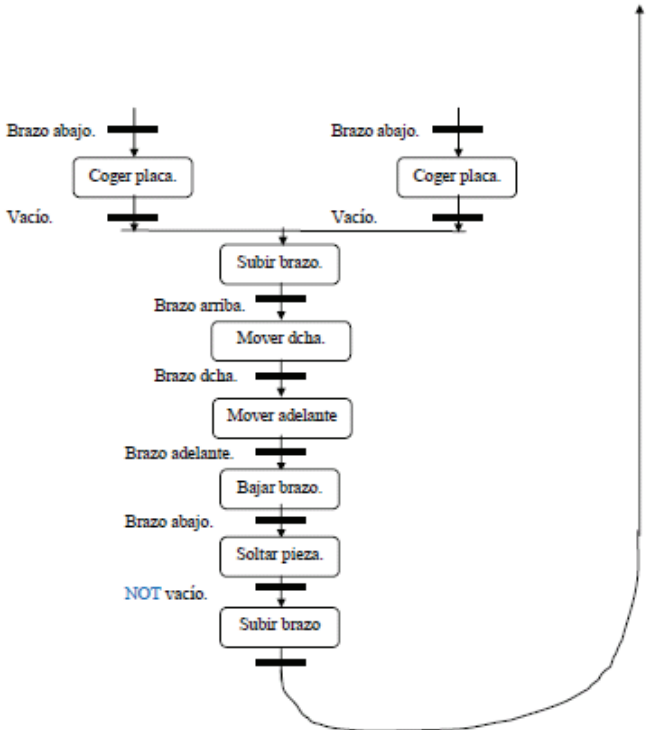
%MW 136.0	Man_soft
%MW 136.1	Marcha_soft
%MW 136.2	Ind_int_soft
%MW 136.3	reset_soft
%MW 136.4	seta_soft
%MW 137	estado_estacion

```
if not (man_auto) then
        marcha_soft:=marcha;
        reset_soft:=reset;
        ind_int_soft:=ind_int;
end_if
```

Estado: 0 si reposo, 1 si produciendo, 2 si emergencia.

Red de Petri implementada en el autómata (SFC).




Brazo abajo.
Coger placa.
Vacío.
Brazo abajo.
Coger placa.
Vacío.
Subir brazo.
Brazo arriba.
Mover dcha.
Brazo dcha.
Mover adelante
Brazo adelante.
Bajar brazo.
Brazo abajo.
Soltar pieza.
NOT vacío.
Subir brazo

Apéndice K

Guión de la práctica del simulador Robot-Studio

Se incluye en este apéndice la propuesta de guión para una práctica de laboratorio o un trabajo de asignatura basado en el empleo del simulador de robots *Robot-Studio*. Todo el material incluido en esta propuesta ha sido obtenido a partir del desarrollo de este proyecto.

Práctica S.

Programación de robots sobre un simulador.

1.- OBJETIVOS.

Conocer y valorar la utilidad de un simulador gráfico de robots manipuladores.

Aprender y practicar las distintas formas de programación de robots.

Implementar sobre el simulador una tarea de ensamblado utilizando guiado y lenguaje textual explícito. Documentar la aplicación incluyendo explicaciones sobre la asignación de referencias y la planificación de movimientos realizada (evitación de obstáculos, tipos de movimientos, etc.).

Resolver problemas que se presentan habitualmente al programar una tarea robotizada como son: evitar obstáculos, localizar objetos, evitar singularidades, seleccionar configuraciones del robot, etc.

2.- TRABAJO PREPARATORIO.

Leer detenidamente el enunciado completo de la práctica.

Leer la introducción a RobotStudio que se adjunta al final del guión.

Mirar en el manual del simulador aquellos aspectos que se consideren más difíciles. El lenguaje de robot que utiliza el simulador es el RAPID.

Planificar la tarea a programar, pensando en los sistemas de referencia necesarios y los tipos de movimientos y trayectorias a utilizar.

La práctica puede realizarse o bien seleccionando un escenario y realizando las tareas que en él se indican, o bien con un trabajo alternativo indicado más adelante. Hay que pensar cuál de las dos opciones se desea hacer y si se elige la opción de un escenario ya hecho, cuál se desea hacer.

Lógicamente, las diferentes dificultades de los escenarios, propiciarán una mejor o peor calificación.

3.- DESCRIPCIÓN DE LA TAREA A ROBOTIZAR.

Se indican a continuación los diferentes escenarios. En los anexos se adjunta toda la información necesaria para la realización de la programación.

Las principales tareas a programar son:

- Guiado del robot a diferentes posiciones.
- Localización mediante coordenadas de algunas posiciones.
- Recogida de objetos.
- Seguimiento de trayectorias.
- Ensamblado de objetos.
- Manejo de señales.
- Evitación de obstáculos.
- Optimización del ciclo.

No todos los escenarios tienen todas las tareas, si bien en todos ellos habrá que llevar el robot a diferentes posiciones, ya sea por guiado o calculándolas, y habrá que evitar obstáculos…

4.- TRABAJO PRÁCTICO.

El trabajo consistirá en desarrollar todas las tareas que se indiquen para cada escenario, de la manera más sencilla posible y evitando colisiones entre los diferentes objetos que se manejen y tratando de no provocar situaciones descontroladas al robot.

5.- TRABAJO OPCIONAL.

Como trabajo opcional para subir nota se proponen las siguientes cuestiones:

a) Realizar las trayectorias a una velocidad doble de la predeterminada, o incluso superior. Utilizar diferentes tipos de terminaciones y precisiones. Utilizar diferentes velocidades dependiendo de la tarea que se esté realizando.

b) Tratar de optimizar al máximo el tiempo del robot para la realización de la tarea. En una primera fase sin cambiar las piezas de sitio. En una segunda fase se puede tratar de mejorar la posición de partida de las piezas.

c) Plantear otras posibles tareas a robotizar y realizar el código. Se puede pensar en tareas de soldadura (puntos y arco), ensamblado, manipulación, paletizado, etc. En la documentación se debería especificar la tarea a realizar. Aparte de la corrección de la definición y solución de la tarea, se valorará su complejidad y originalidad.

6.- TRABAJO ALTERNATIVO.

Se propone asimismo un trabajo alternativo a realizar consistente en la creación de un escenario completo, de un modo totalmente libre y creativo. En dicho escenario, deberán a su vez realizarse las siguientes tareas:

- Creación de no menos de dos objetos que se puedan transportar, y al menos un objeto compuesto por la unión de otros dos objetos, su intersección o su resta.
- Traslado de un lugar a otro de al menos un objeto.
- Generación de no menos de dos trayectorias por los bordes de alguno de los objetos creados, simulando un soldado, un sellado, una limpieza…
- Gestión de señales y tiempos de espera.

En este trabajo alternativo, todo lo que no esté descrito anteriormente, se considerará como opcional. No obstante se proponen los siguientes a modo de ejemplo:

- Utilización de trayectorias complejas (Circulares, concatenaciones…).
- Diferentes velocidades, terminaciones y precisiones.
- Similitud del escenario y las tareas con una situación real.

7.- PRESENTACIÓN Y CRITERIOS DE VALORACIÓN.

Se realizará una exposición oral, explicando la asignación de referencias y la planificación de movimientos realizados y los problemas aparecidos y su solución. Además se realizará una breve demostración práctica que permita valorar el trabajo realizado.

Material a entregar:

- Documentación breve que incluya:

    Esquema con la asignación de referencias utilizada.

    Planificación de los movimientos y trayectorias realizados.

    Listados de los programas de aplicación.

- Archivo que contenga el o los programas fuente.

Criterios de valoración:

- Corrección de la asignación de referencias y de la planificación (evitando colisiones, singularidades, etc.).

- Corrección y claridad de las implementaciones y su funcionamiento.

- Legibilidad del código utilizado.

- Corrección de la documentación.

- Se valorará tal y como ya se ha comentado la complejidad del escenario seleccionado.

- En el caso del trabajo alternativo:

    - Originalidad y sentido práctico.

    - Complejidad del escenario creado.

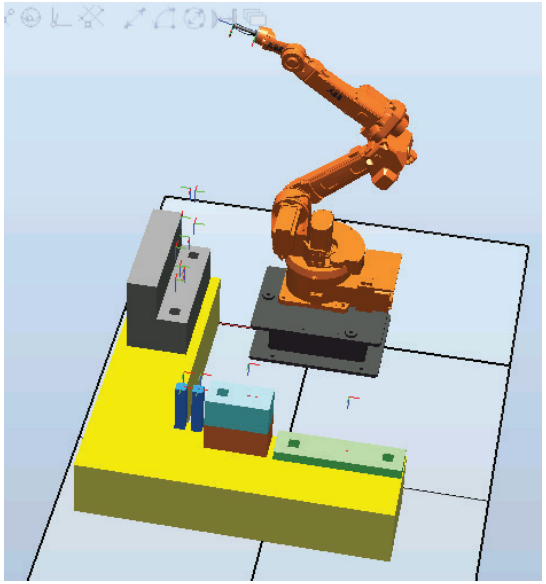
    - Complejidad de las tareas realizadas.

    - Cantidad de robots utilizados.

ANEXO A: ESCENARIOS DE LAS PRÁCTICAS.

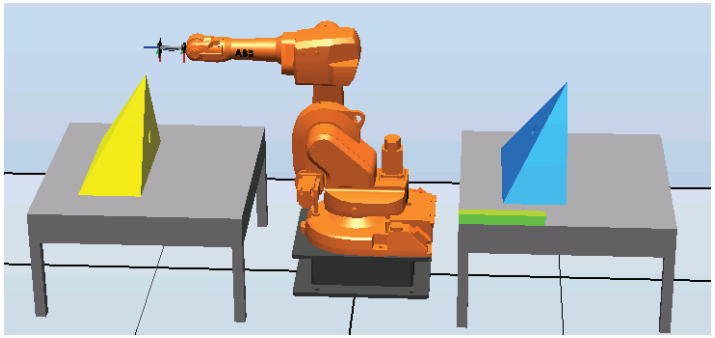
Opción 1: Ensamblado de dos bloques mediante dos pernos y una junta.

El escenario está compuesto de dos bloques que deben ser situados sobre la pieza más grande, pero previamente debe ser colocada la junta de color verde, y todas las piezas deberán quedar sujetas mediante los pernos de color azul.



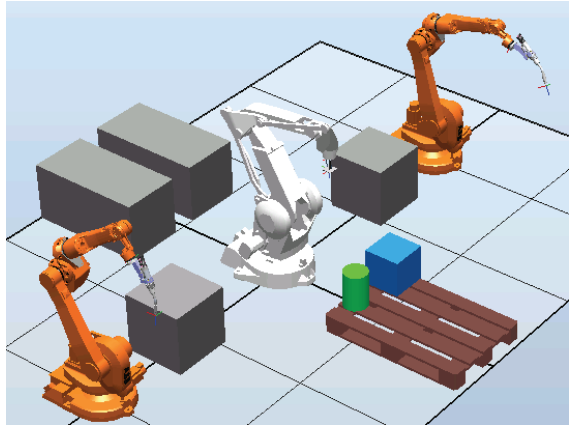
Opción 2: Ensamblado de un cuerpo piramidal con un perno en posición oblicua.

Se trata de volver a componer la pirámide que se observa partida, introduciendo además el perno que se encuentra situado sobre la mesa.



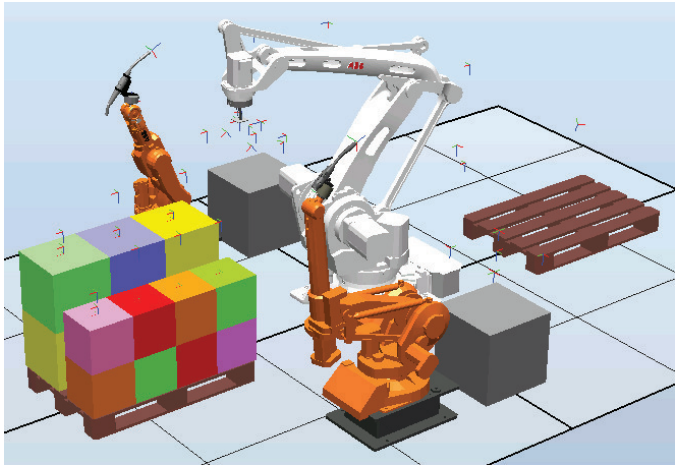
Opción 3: Traslado y limpieza de dos piezas diferentes.

Consiste en trasladar desde el pallet donde se encuentran inicialmente, las dos piezas que se observan. Cada una a uno de los dos puestos existentes, y una vez en ellos efectuar la limpieza de al menos una de sus caras. Después hay que trasladarlos hasta los dos puestos que hay en el otro lado del escenario.



Opción 4: Traslado y sellado de múltiples paquetes.

Se trata de trasladar de un pallet a otro al menos dos cajas de cada tipo, y entre tanto realizar el sellado de al menos todas las aristas de una cara.



ANEXO B: BREVE INTRODUCCIÓN A ROBOT-STUDIO.

1) Creación de objetivos:

1.1) Mediante movimiento de las articulaciones del robot:

Es posible indicar cuál ha de ser una posición objetivo del robot mediante varias formas. La primera de ellas es moviendo libremente las articulaciones del robot mediante el uso del ratón.

Para ello lo único que hay que hacer es clicar en el botón "movimiento de eje", en el menú inicio, pestaña de "mano alzada".

Cuando se haya logrado la posición deseada, se hace clic en el botón de "programar objetivo", en el menú inicio, submenú "programación de trayectorias".

Existe otra opción para realizar esta operación, consistente en ir a la pestaña "diseño" y hacer clic derecho sobre el mecanismo que se desea modificar su posición e indicar "movimiento de ejes de mecanismo".

Los objetivos creados pueden ser referidos a diferentes sistemas de referencia y ser modificados del modo que se desee.

1.2) Mediante movimientos lineales de la herramienta:

Otra opción para programar objetivos consiste en mover la herramienta sobre sus propios ejes, o sobre otra referencia que se pueda desear. Para ello vuelve a haber dos opciones. La primera de ellas es seleccionar el eslabón de la herramienta y en el menú "mano alzada", hacer clic en el botón de movimiento lineal. Después ya intuitivamente se puede mover en los ejes que se hayan seleccionado.

La otra opción es nuevamente hacer clic derecho sobre el mecanismo que se desea modificar, en la pestaña "diseño". Ahora se introducen las coordenadas deseadas, o se hace un ajuste fino variando las unidades de poco en poco.

1.3) Mediante introducción de las coordenadas deseadas:

En el menú "programación de trayectorias", se clica en el botón objetivo, y se selecciona crear objetivo. Se abrirá un panel en el que se permite introducir todos los datos de un punto, así como seleccionar la referencia en la que se desea, y otras opciones más.

Es importante comprobar la alcanzabilidad del robot para los puntos introducidos de esta manera. Para ello se seleccionan los puntos que se desea comprobar y que están disponibles en la pestaña "trayectorias y objetivos", en el mecanismo para el que se haya creado, y en la referencia que tenga asignada. Encontrado el objetivo, clic derecho sobre él, y botón "alcanzabilidad".

ATENCIÓN: Es necesario hacer este método al menos una vez cada vez que se cambie de robot, para que guarde los objetivos en dicho robot y no en los anteriores.

AYUDA: No existe opción automática para medir en el programa, por lo que lo más práctico es clicar con el cursor en las esquinas del objeto a medir y hacerlo con lápiz y

papel. También se puede llevar el extremo del robot a las diferentes esquinas y anotar sus coordenadas.

2) Creación de trayectorias:

2.1) A partir de objetivos previamente creados:

En el menú inicio, clic en "ruta", y después clic en "trayectoria vacía". Una vez hecho esto, aparece una nueva trayectoria en la pestaña "trayectorias y objetivos", en el mecanismo que estuviera seleccionado.

Para darle contenido a la trayectoria, arrastrar los puntos que se desean recorrer a la trayectoria. Después es necesario ordenarlos según interese. Por último, es posible seleccionar si se desea un movimiento lineal o articular. Para ello clic derecho sobre la instrucción de movimiento a modificar, y clic en "modificar instrucción".

Aparte, también existen otras opciones en este apartado, como ver al robot en posición, recorrer trayectoria, o verificar la alcanzabilidad o las configuraciones.

2.2) Trayectoria automática:

Esta es otra opción, que permite crear una trayectoria a partir de algún elemento ya existente. Para ello seleccionamos el elemento en cuestión, por ejemplo una de las caras de un cubo, y clicando ene las diferentes aristas, o en los vértices, podemos crear una trayectoria según nuestras necesidades.

También es posible realizarlo mediante curvas, líneas y otros elementos.

3) Modelado de componentes:

Es posible modelar tanto componentes pasivos como activos para el sistema del robot. Para ello, se utilizará el menú "modelado".

3.1) Modelado de un objeto pasivo:

Para el modelado de uno de estos objetos, como por ejemplo un soporte o una caja a transportar, está el botón "sólido", que permite crear sólidos sencillos tales como tetraedros, cilindros, cubos o conos. Se realiza de forma muy intuitiva.

Para obtener sólidos más complejos es necesario unir varios sólidos simples. Esto solo requiere de crear los dos sólidos, ponerlos en la posición deseada y en la pestaña de "modelado", clic derecho sobre el objeto a unir y se selecciona a qué otro objeto se quiere unir.

3.2) Curvas, superficies, bordes:

Otra opción que se puede realizar es crear cualquiera de estas tres geometrías, útiles por ejemplo para definir un área de pintado, o una zona que el robot no debe atravesar.

3.3) Operaciones de CAD:

RobotStudio permite la importación de archivos de CAD, de tal forma que se pueden usar objetos creados externamente al simulador. Además permite realizar diferentes operaciones tanto sobre estos objetos importados como sobre los creados en el simulador.

3.4) Creación de mecanismos:

Al hacer clic en "crear mecanismo" se abre un submenú en el que se indican todos los pasos a seguir para crear un mecanismo desde cero, Este mecanismo puede ser un nuevo robot, una herramienta, un transportador un eje externo, o cualquier clase de dispositivo.

Para crear un mecanismo es necesario tener previamente todos sus eslabones, y ejes de referencia, sistemas de coordenadas… Y en el caso de una herramienta habrá que considerar señales, movimientos automáticos….

4) Simulación:

En este menú se permite la realización de simulaciones, tal y como se ejecutarían en un controlador real. Aparte de poder grabarlas, y de otros aspectos visuales, hay una serie de menús importantes.

ATENCIÓN: Es necesario realizar al menos una simulación (aunque no se simule nada), para poder visualizar los movimientos del robot, y poder coger o dejar objetos.

AYUDA: Puede ser muy útil cuando ya se hayan creado varias trayectorias, crear un estado de restablecimiento. Para ello clic en "Restablecer" y luego en "Guardar estado actual".

4.1) Simulador de entradas y salidas:

Este botón abre una nueva pestaña, en la que se detallan todas las entradas y salidas que tiene el sistema, y el estado en el que se encuentran. Además también permite cambiarlas de estado.

A la hora de utilizar señales, hay que advertir que las que se dan ya como predefinidas son las que vendrían de serie en un robot de la casa ABB, y por tano no se pueden utilizar más que para lectura. Son por ejemplo el indicador de sobrecarga, y otras señales internas del robot.

Para poder utilizar señales para fines tales como sincronización, activación de mecanismos, etc., Es necesario crear nuevas señales tal y como se indicará a continuación.

4.2) Creación de señales:

Las señales disponibles tienen diferentes estatus. Unas son internas al robot, o están contenidas en módulos inaccesibles. Otras son virtuales y son plenamente accesibles. Estas últimas son las interesantes.

Para crear una señal, hay que tener en cuenta cuál va a ser su cometido, y desde donde es necesario acceder a ella. Para el caso de señales que solo sean necesarias en el código RAPID, como por ejemplo señales de sincronización, no es necesario hacer otra cosa que definirlas en el propio código.

Para utilizar señales que activen mecanismos (herramientas, transportadores…) es necesario crearlas desde un menú superior al código. El procedimiento es el siguiente:

> **I)** Clicando en el menú "controlador", aparece la pestaña "controlador". Dentro de esa pestaña se selecciona el mecanismo en el que queremos la nueva señal, y

en el desplegable “configuración”, se selecciona “I/O” y se abre mediante doble clic.

II) Una vez dentro de “I/O”, se pueden observar tanto las variables disponibles, como los buses de conexiones… En concreto ahora hay que crear una nueva unidad. Para ello, en “unit”, clic derecho y clic sobre “nuevo unit”. Ahora se va a crear una nueva unidad que contendrá señales. Le damos el nombre que queramos, y después se selecciona “unidad virtual”, conectado a bus “virtual1” y las otras opciones según interesen.

III) Para crear la señal propiamente dicha, clic derecho sobre signal y “nuevo signal”. Se le da el nombre deseado, y se selecciona el tipo de señal que se necesite. Se le asigna la unidad que hemos creado, se le indica un “unit mapping” que no esté ya utilizado y el resto de opciones se escogen según las necesidades.

IV) Ahora que ya tenemos la señal creada, habrá que reiniciar el controlador. Para ello menú “controlador”, clic derecho sobre el dispositivo que estemos utilizando, y “reinicio en caliente”.

V) Comprobamos que la señal está bien creada, abriendo el simulador de E/S.

VI) Esta señal ya se puede utilizar en el código RAPID. Pero si lo que se desea es utilizarla para activa mecanismos, otros robots… Hay que interconectarla con la señal adecuada. Por ejemplo, con una señal de activación de una garra.

Para ello en el menú “simulación”, clic en el botón “lógica de estación”.

En la pestaña “señales y conexiones”, añadir conexión de E/S. Se seleccionan las señales de origen y las de destino, y de esta forma quedarán conectadas.

VII) La señal creada y conectada con otra(s) ya está lista para ser usada. Para activarla, o desactivarla en el código RAPID (solo si es tipo output), no es necesario declararla como si fuera otra variable más.

4.3) Gestor de eventos:

El gestor de eventos es una herramienta que permite desencadenar acontecimientos en función de diferentes entradas, posiciones…

Sus opciones más interesantes son las de conectar/desconectar objeto, cambiar entrada/salida, mover un objeto… Para crear un evento los pasos a seguir son los siguientes:

I) En el gestor de eventos (menú de simulación, botón pequeñito situado junto a “configurar”), añadir nuevo evento… Seleccionamos las opciones que nos convengan.

II) Seleccionar la señal que queremos que sea la desencadenante de la acción (o la colisión… en caso de otras opciones que no son señales).

III) Se elige qué tipo de acción se ha de realizar, y luego se van seleccionando las opciones que se deseen.

En el caso de querer unir dos objetos, cuidado con la opción “actualizar posición”, porque dependiendo de la referencia que se esté utilizando, puede provocar que se descoloquen objetos.

5) Controlador:

El controlador es una parte importante de RobotStudio, ya que el programa intenta simular que el robot es en realidad un robot real y no virtual, lo que provoca que en ocasiones el controlador haga cosas como los reales y se quede en modo de fallo, sea necesario reiniciarlo…

Es importante evitar las singularidades o su paso cerca de ellas, porque provocan que el controlador entre en modo de fallo, y puede estropear una simulación larga… Por ello es imprescindible tratar con cuidado las singularidades, mediante las herramientas que se ofrecen en el apartado de modelado.

5.1) Panel de control:

En el menú “controlador”, existe el botón panel de control, que simula el panel de control del robot real. Es importante puesto que aquí es donde se puede seleccionar el modo semi-automático, necesario para realizar la verificación paso a paso del código RAPID.

5.2) Flex Pendant:

Al igual que con el panel de control, RobotStudio dispone de una simulación del panel del robot real, que permite todas las funcionalidades que tendría el susodicho panel.

Entre otras opciones interesantes, dispone de otro modo de controlar las entradas y salidas, crear nuevas e interconectarlas con otras.

5.3) Eventos:

Tras una simulación este botón despliega una pestaña en la cual aparecen todos los sucesos acontecidos durante el tiempo de funcionamiento hasta ese momento.

En general pueden aconteces múltiples sucesos de diferentes tipos y con consecuencias de diferente índole, sin que por ello el robot se pare, o entre en fallo. No obstante a veces es importante revisarlos, ya que advierten de cosas tales como una trayectoria que no se pudo ejecutar con la precisión pedida o posiciones que no se han podido alcanzar con la configuración original.

No obstante en fallos más graves la simulación se detiene y aparecen los indicadores del error.

6) RAPID:

Este menú contiene todo lo necesario para realizar la programación del robot en el lenguaje RAPID, e incluso para poder volcarlo a un robot real.

6.1) Sincronizar:

Este botón permite “transportar” al código RAPID todas las posiciones objetivo, las trayectorias y datos de los objetos de trabajo y herramientas que se han creado en el modelado.

Es imprescindible realizar una sincronización cada vez que se modifiquen trayectorias "a mano" o se cambien o inserten nuevas herramientas.

Cuidado al sincronizar si se han cambiado en el tablero de RAPID los nombres de puntos, archivos, trayectorias… Porque en ocasiones pueden duplicarse, o incluso volver a los antiguos nombres.

6.2) Insertar fragmento o instrucción:

Permite insertar trozos completos de código, que pueden estar guardados en bibliotecas personales, o en packs de expansión.

6.3) Aplicar:

Es la instrucción necesaria para compilar el código, y es necesario darle a este botón tanto cada vez que se modifique el código y se desee cargarlo, como cuando se desee comprobar si hay errores de sintaxis.

6.4) Fallos frecuentes:

"Imposible encontrar el símbolo de RAPID": Falta un procedimiento que se llame "main".

Error de sintaxis llamando a funciones: Las funciones que no requieren pasarles datos se llaman solo por su nombre, no hay que acompañarlas de (), ni de (void).

"Inicio: La operación no está permitida por el sistema de seguridad": Aunque indica que hay que cambiar a modo "auto", basta con cambiar en el panel de control a otro modo y luego volver al modo auto.

El programa no se ejecuta, dice que el controlador está en modo de fallo: Hay que reiniciar el controlador. Para ello se va al menú de controlador, y en la pestaña "Controlador" se hace clic derecho sobre el sistema que tenemos parado y reinicio en caliente.

Imposible alcanzar posición / fallo de trayectoria en esquina: Revisar la alcanzabilidad, y las configuraciones indicadas en el modelado. Asegurarse de que el robot parte siempre de una posición de reposo controlada.

7) Ejemplos, tutoriales y vídeos demostrativos:

7.1) Manual de RobotStudio de ABB:

Contiene la mayoría de procedimientos a utilizar, si bien es bastante largo y no muy explicativo sobre algunos aspectos. No incluye específicamente programación en lenguaje RAPID, aunque sí que explica el procedimiento a seguir, y las instrucciones necesarias para las tareas.

También están incluidas todas las explicaciones necesarias sobre los menús del programa y todas las operaciones que se pueden realizar.

7.2) Tutoriales disponibles por internet:

Existen diversos tutoriales en internet, tanto oficiales de ABB como no oficiales creados por profesionales y estudiantes. Son bastante fáciles de seguir, y algunos son tremendamente ilustrativos. No obstante tienen la desventaja de estar hechos casi todos en vídeo y pueden llegar a ser largos.

Oficiales de ABB:

http://www.abb.com/product/ap/seitp327/04a7e77286113ac1c1257427005a5498.aspx

http://www.youtube.com/playlist?list=PLEF161D764568EC82

Tutoriales no oficiales interesantes:

http://www.youtube.com/playlist?list=PL8AF6B92BF2BC44E6

Índice de figuras

1.1. Gráfico resumen del proyecto. 6

2.1. Tipos de piezas que se producen en la célula industrial. 7
2.2. Vista general de la célula completa. Se aprecian diferentes estaciones de trabajo, cintas transportadoras, y al fondo el robot industrial 8
2.3. Subcélula de preparación y transporte de pedidos 9
2.4. Estación expendedora de palets para colocar las piezas. 9
2.5. Zona de trabajo del robot. Está delimitada por las cintas transportadoras y tres vallas de protección. 11
2.6. Autómata M340 e isla Advantys pertenecientes a la estación 6. 12
2.7. Módulo TSX Momentum, perteneciente a una estación de transporte . . . 12

3.1. Armario del controlador RJ-3 del robot FANUC, con el terminal de enseñanza en primer plano. 16
3.2. Terminal de enseñanza del robot FANUC. Se aprecia el pulsador de emergencia. 17
3.3. Vista del programa de comunicación por puerto serie «Hyper-terminal». . . 18
3.4. Vista del programa «Kfloppy», que emula una disquetera virtual. 18
3.5. Piezas apiladas en tres montones distintos como resultado del programa «Paletizado». 20
3.6. Momento en el que el robot descarta una pieza durante el programa «Desechar». 24
3.7. Colocación de una de las piezas del tren durante la ejecución del programa «Montar tren». 24

4.1. Isla Advantys STB que se instaló inicialmente. 28
4.2. Isla Advantys STB que se encuentra actualmente instalada. 28
4.3. Ejemplo de programa SFC. Se aprecian varias acciones secuenciales, y dos en paralelo. 29
4.4. Ejemplo de programa LD. Se aprecian diversas condiciones a la izquierda, y algunas acciones a la derecha. 29
4.5. Autómata principal, situado junto al robot industrial. 30
4.6. Autómata secundario presente en la estación de expedición de palets. . . . 30
4.7. Ejemplo de SFC, correspondiente a la selección de programa del robot. Se aprecia una selección y diversas acciones secuenciales. 32
4.8. SFC correspondiente a la gestión de la seguridad del robot. 32
4.9. SFC correspondiente al control de los transportes (Cintas, topes, desvíos...). 33

4.10. SFC correspondiente al control de la sub-célula completa (controlada totalmente por el autómata principal). 34
4.11. Botonera física de la estación de transporte. Se aprecian los diferentes pulsadores y selectores a los que se hace referencia. 34
4.12. Diagrama SFC correspondiente al control de la estación de expedición de palets. 35
4.13. Estructura jerárquica del conjunto integrado. 37
4.14. Vista de un módulo NOE 0100 como el existente en el laboratorio. 38

5.1. Ejemplo de terminal de diálogo en una industria. 41
5.2. Diagrama explicativo del acceso a memoria. La posibilidad de acceder a la memoria está condicionada según el selector MAN/AUT. 43
5.3. Pantalla del terminal diseñado para controlar el conjunto. 43
5.4. Pantalla del terminal diseñado para la estación expendedora de palets. . . 43
5.5. Terminal de la estación de expedición de palets en estado habilitado 44
5.6. Terminal de la misma estación en estado deshabilitado. Desaparece la botonera dado que no se permite su uso. 44

6.1. Cámara RGB-D modelo «Asus Xtion Pro Live» utilizada durante este proyecto. 47
6.2. Imagen RGB capturada por la cámara. 48
6.3. Imagen resultado de la combinación del color y la profundidad. Se aprecian puntos con color y posición. 48
6.4. Esquema explicativo del almacenamiento de información en formato «Nube de Puntos». 50
6.5. Imagen introducida al algoritmo 51
6.6. Resultado obtenido del algoritmo. 51
6.7. Proceso seguido para reorientar una nube de puntos. 53
6.8. Ilustración de cómo se eliminan los planos. Se puede observar cómo se van eliminando secuencialmente de mayor a menor. 54
6.9. Imagen original, e identificación realizada por el algoritmo. Se aprecian objetos identificados, y huecos en aquellos que no se han reconocido. 57
6.10. Imagen original, y discretización del color realizada por el algoritmo. Se aprecia la adecuada conversión a colores más puros sin perder apenas información. 57
6.11. Ejemplo del archivo de salida con los objetos que el algoritmo ha logrado identificar. Se incluyen el tipo de objeto, su posición, tamaño, y colores principales. 58
6.12. Ejemplo de la salida por pantalla del algoritmo de control del espacio de trabajo. 59

7.1. Muestra de la pantalla del programa RoKiSim. Se aprecian unos gráficos bien conseguidos y una interfaz lista para el uso. 63
7.2. Ejemplo de la pantalla del simulador V-Rep. Se puede apreciar mayor complejidad que en otros simuladores. 64
7.3. Vista general de la pantalla del simulador Easy-Rob. Se observa un buen tratamiento gráfico y múltiples opciones de acceso rápido. 65

7.4. Vista de la pantalla principal del simulador Robot-Studio. Se aprecian diferentes pestañas, y se intuye el amplio rango de opciones que tiene. 66

Índice de tablas

2.1. Principales características del robot 10

4.1. Resumen de los modos de funcionamiento de la sub-célula. 36

6.1. Características del dispositivo RGB-D. 49

Bibliografía

[1] ABB-Robotics. Abb robot-studio (http://new.abb.com/products/robotics/robotstudio).

[2] ABB-Robotics. *Manual del operador. Robot Studio 5.15.*, 2013.

[3] ASUSTeK-Computer-Inc. Asus xtion pro live. (http://www.asus.com/multimedia/xtion_pro_live/), url = http://www.asus.com/multimedia/Xtion_pro_live/,.

[4] Barrientos, A., Peñin, L. F., Balaguer, C., and Aracil, R. Fundamentos de robótica. Mc-Graw-Hill, Ed.

[5] Beltrán, R., and Romeo, A. Programa tareak, 2003.

[6] École Technologie-Supérieure-Université-Montreal. Información y descarga del programa rokisim (http://www.parallemic.org/rokisim.html).

[7] Coppelia-Robotics. Información y descarga del programa v-rep (http://www.coppeliarobotics.com/).

[8] Easy-Rob-Inc. Información y descarga del programa easy rob (http://www.easy-rob.com/easy-rob/).

[9] FANUC-Robotics. *R–J3iB Mate Controller LR Handling Tool Operator's Manual.*

[10] FANUC-Robotics. *SYSTEM RJ-3 Controller KAREL Reference Manual.*

[11] FANUC-Robotics. *R-J3 Controller CE Maintenance*, 2000.

[12] FANUC-Robotics. *Curso de programación TPE*, 2003.

[13] Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography. In *Magazine Communications of the ACM* (1981).

[14] OpenNI-organization. Openni libraries (http://www.openni.org/), url=http://www.openni.org/,.

[15] robots.com. http://www.robots.com/fanuc/arcmate-50il.

[16] Rusu, R. B., and Cousins, S. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)* (Shanghai, China, May 9-13 2011).

[17] Rusu, R. B., O'Leary, G., and Rublee, E. Point cloud library documentation (www.pointclouds.org).

[18] Schneider-Electric. Información sobre el autómata modicon m340 (http://www.schneider-electric.com/products/co/ls/3900-pac-plc-y-otros-controladores/3950-pacs/1468-modicon-m340/).

[19] Schneider-Electric. Información sobre el autómata tsx premium (http://www.schneider-electric.com/products/ww/en/3900-pac-plc-other-controllers/3950-pacs/537-modicon-premium/).

[20] Schneider-Electric. Información sobre el módulo de entradas y salidas distribuidas «advantys stb» (http://www.schneider-electric.com/products/mx/ls/3900-pac-plc-y-otros-controladores/3930-i-o-distribuida/606-advantys-stb/).

[21] Schneider-Electric. Magelis xbtgt4330 (http://www.schneider-electric.com/site/home/index.cfm/es/).

[22] Schneider-Electric. *Advantys STB, módulo de interfaz de red CANopen básico. Manual de aplicaciones*, 2009.

[23] Schneider-Electric. *Advantys STB, módulos de E/S digitales. Manual de referencia*, 2009.

[24] Schneider-Electric. *Modicon M340 para Ethernet. Procesadores y módulos de comunicaciones. Manual de usuario*, 2009.

[25] Schneider-Electric. *Unity Pro. Lenguajes y estructura del programa. Manual de referencia*, 2009.

[26] Tuerker, S. Euclidean cluster extraction (www.pointclouds.org/).

[27] Universidad-Zaragoza. Sistemas industriales de control. práctica 4. utilización de una red industrial de comunicaciones. programación de una interfaz humano-máquina., 2013.