Universidad Zaragoza
1542

Proyecto Fin de Carrera

Puesta en Marcha de un Robot ABB IRB 120.

Evaluación del Entorno de Programación/Simulación RobotStudio

"Setting up the ABB IRB 120 Robot.

Testing and Evaluating the ABB simulation and offline programming software RobotStudio"

Autor:

Octavio Augusto Ansón Clemente

Director:

Antonio Romeo Tello

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Año: 2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. OCTAVIO AUGUSTO ANSÓN CLEMENTE,

con nº de DNI 17743101-G en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO EN INGENIERÍA ELECTRONICA Y AUTOMÁTICA, (Título del Trabajo) "PUESTA EN MARCHA DE UN ROBOT ABB IRB 120. EVALUACIÓN DEL ENTORNO DE PROGRAMACIÓN/SIMULACIÓN ROBOT STUDIO"

"SETTING UP THE ABB IRB 120 ROBOT. TESTING AND EVALUATING THE ABB SIMULATION AND OFFLINE PROGRAMMING SOFTWARE ROBOT STUDIO",

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 02 de FEBRERO de 2018

Fdo: OCTAVIO AUGUSTO ANSÓN CLEMENTE

Resumen TFG:
Puesta en Marcha de un Robot ABB IRB 120

Este es un **Proyecto de Análisis Crítico del Entorno de Programación de un Robot ABB IRB 120** necesario **para su puesta en marcha,** para ello se ha optado por dividir el análisis en dos Partes:

- **Análisis del Entorno Virtual y/o Simulación**: Entorno Integral de Programación de una Estación Completa
- **Análisis del Entorno Real y/o Inserción de un Robot** real **en una célula de trabajo** industrial

Se ha tenido como **área principal del análisis el entorno integral de simulación desde RobotStudio**, dado que permite extrapolar las destrezas y conocimiento alcanzados con relativa facilidad desde el entorno virtual al real, abordando los aspectos referidos a la: *programación*, *modelado* y *simulación* de entornos virtuales.

Los **objetivos abordados** se han clasificado en 3 bloques:

1. **Evaluación de RobotStudio** en sus 2 vertientes (*herramienta de programación del robot y herramienta de modelado de entornos virtuales*) y diseño, creación y simulación de una Estación Industrial Virtual Completa.
2. **Inserción del ABB IRB 120 en una pequeña célula de fabricación flexible:** Definición y explotación de E/S, por medio del desarrollo de una aplicación que suponga la inserción del robot en una pequeña célula flexible de fabricación.
3. **Análisis crítico de las posibilidades de relativización de localizaciones** que ofrece **RAPID**.

El Robot **objeto del proyecto** ha sido **adquirido por el departamento de Informática e Ingeniería de Sistemas de la EINA en el curso 2016/2017**. Esto ha provocado la **necesidad de su puesta a punto para el curso 2017/2018**, así como **evaluar el potencial de las herramientas**. Para ello, se ha simulado/emulado todas las casuísticas identificadas, en la definición y creación de una Estación Industrial Completa, con el **entorno de programación de los Robots ABB**: *Software RobotStudio*. Clasificadas en el **ANEXO II** y evaluadas en la Memoria.

Índice de Apartados del TFG:

1. Introducción 3

1. 1. Objetivo y Alcance del Proyecto: 3

1. 2. Estación Virtual y Evaluación de Metodologías Enfrentadas 7

1. 3. Inserción de un Robot IRB-120 en una Estación de Trabajo 11

2. Creación y Explotación de Entornos Virtuales 17

2. 1. Descripción de la Estación Virtual 17

2. 2. Implementación de la Herramienta Virtual 19

2. 2. 1. Propiedades Generales y Geométricas 20
2. 2. 2. Implementación con: Sensores 22
2. 2. 3. Implementación con: Detector de Colisiones 23

2. 3. Simulación de PLC's 24

2. 3. 1. Programa Implementado en el Autómata: Grafo de Estados 26
2. 3. 2. PLC's con Controlador IRC-5: Implementado en RAPID 28
2. 3. 3. PLC's con Bloques Lógicos: Implementado con SmartObject 28

2. 4. Aplicaciones de los Robot en RAPID 30

2. 5. Evaluación de RobotStudio para Simular Células de Trabajo 31

3. Inserción del ABB en una Célula de Fabricación Flexible 33

3. 1. Descripción de la Estación de Trabajo 33

3. 2. Descripción de la Herramienta 35

3. 3. Aplicación en el PLC 37

3. 3. 1. Programa Implementado en el Autómata 38
3. 3. 2. Mapeado de E/S y Cableado 42
3. 3. 3. Jerarquía de control 43

3. 4. Aplicación RAPID en el Robot 44

3. 4. 1. Elección de Jerarquía de Niveles 45
3. 4. 2. Análisis crítico de las posibilidades de relativización de localizaciones que ofrece RAPID. 46
3. 4. 3. Programa Principal del Robot ABB IRB 120: Main Program 58

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

4. Conclusiones del Trabajo Fin de Grado 59

4. 1. Creación de Herramientas y Controladores mediante el uso de *SmartObjects* 59

4. 2. Implementación de un PLC en RobotStudio 61

4. 3. Relativización de localizaciones en RAPID 62

Documentos Bibliografía 64

Índice de Figuras 65

Índice de Tablas 71

Índice de Vídeos 72

Índice de Acrónimos 75

Índice de ANEXOS: 76

ANEXO I: Cronograma de Gantt .....

ANEXO II: Compendio de Consideraciones, Limitaciones y Observaciones varias de RobotStudio .....

ANEXO III: Fichas de Ruta: "Cómo se hace…" .....

ANEXO IV: Limitaciones de los Sensores .....

ANEXO V: La Herramienta Virtual"*SR.8*" .....

ANEXO VI: Implementación del PLC de 16 Estados: Automtismo Moore .....

ANEXO VII: La Estación Simulada "*Célula 8*" .....

1. Introducción:

1. 1. Objetivo y Alcance del Proyecto

Considerado un **Proyecto de Análisis Crítico de Herramientas** y de adaptación de material existente necesario **para la puesta en marcha de un Robot de la compañía ABB**, se han abordado los aspectos referidos a programación, modelado y simulación de entornos virtuales con RobotStudio.

Los **objetivos desarrollados**, en el desarrollo del **TFG**[1], se han clasificado en 3 bloques:

1. **Inserción del robot en una pequeña célula de fabricación flexible:** Definición y explotación de **E/S**[2], por medio del desarrollo de una aplicación que suponga la inserción del robot en una pequeña célula flexible de fabricación.
2. **Análisis crítico de las posibilidades de relativización de localizaciones** que ofrece **75**[3].
3. **Evaluación de RobotStudio** en sus 2 vertientes (herramienta de programación del robot y herramienta de modelado de entornos virtuales).

El Robot **objeto del proyecto**, un **ABB IRB 120**, ha sido **adquirido por el departamento de Informática e Ingeniería de Sistemas de la EINA**[4] **en el curso 2016/2017**. Esto ha provocado la **necesidad de su puesta a punto** y **evaluar el potencial de las entorno de programación**. Por ello, se ha simulado/emulado todas las casuísticas clasificadas en el **ANEXO II** y evaluadas en la Memoria.

Metodología seguida

Se ha adoptado las siguientes **directrices**:

1) **El trabajo se ha realizado en el laboratorio L0.06 del DIIS**, donde se encuentra ubicado el robot. Para ello el autor del TFG ha requerido de una autorización administrativa para acceder a dicho laboratorio.
2) **El Proyecto requiere la comprensión y asimilación previa de la herramienta de simulación/programación RobotStudio,** la familiarización con el lenguaje de programación del robot (RAPID) y la interfaz de usuario **FlexPendant**[5] del Robot ABB IRB 120.
3) **El uso del software RobotStudio** implicará tanto la programación del robot real, como la creación y explotación de un entorno virtual.

[1] **TFG**: Trabajo Fin de Grado, también denominado Proyecto o Proyecto Final de Carrera
[2] **E/S**: Entradas y Salidas de un Sistema cualesquiera, pueden ser de carácter Analógico o Digital
[3] **RAPID**: **R**obotics **A**pplication **P**rogramming **I**nteractive **D**ialogue
[4] **EINA**: Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza (UNIZAR)
[5] **FlexPendant**: Dispositivo/Display para manipular directamente el Robot, tanto a nivel de control directo como para configurarlo/programarlo

Titular: Octavio Augusto Ansón Clemente **Director**: Antonio Romeo Tello

A) *Cronograma y estudio temporal*

Se ha contado con un **marco temporal en torno a 4 meses**, comprendidos entre el 25 de septiembre de 2017 y el 30 de enero de 2018. Restricciones impuestas por demanda del Director del TFG y fechas administrativas.

Inicialmente se optó por **dividir el Proyecto en 5 etapas** o **fases de trabajo** ordenadas cronológicamente:

1. Trabajo sobre herramientas software específico y consulta de normativas, para alcanzar el nivel de conocimiento suficiente.
2. Desarrollo de una aplicación que suponga la inserción del robot en una pequeña célula flexible de fabricación.
3. Implementar la aplicación en el sistema real y la exploración de las E/S del Robot ABB IRB 120.
4. Evaluación de resultados obtenidos.
5. Elaboración del informe final.

Dadas dificultades ajenas al desarrollo del TFG, avería técnica de los reductores "*Armonic-Drive*" de los motores del Robot (*ejes del 1 al 3*), fue necesario modificar la planificación final. Detallada en el **ANEXO I**, donde se han indicado los distintos objetivos e hitos alcanzados en el TFG.

B) Planificación Final: Cronograma real resultante del desarrollo del TFG

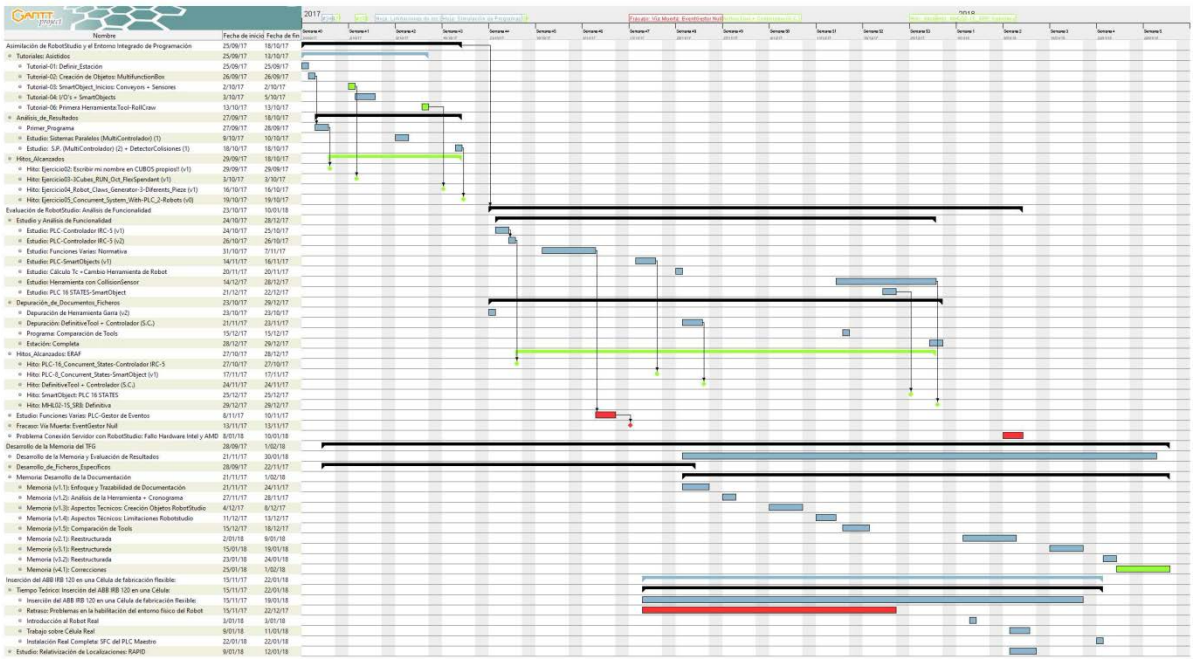


Figure 1: Cronograma GANTT de las Tareas Desarrolladas en el TFG

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Directrices del Análisis del Entorno de Programación

Se ha optado por dividir el análisis en dos *Part*es:

- **Análisis del Entorno Virtual y/o Simulación**: Entorno Integral de Programación de una Estación Completa
- **Análisis del Entorno Real y/o Inserción de un Robot** real **en una célula de trabajo** industrial

Se ha tenido como **área principal del análisis el entorno integral de simulación desde RobotStudio**, dado que permite extrapolar las destrezas y conocimiento alcanzados con relativa facilidad desde el entorno virtual al real, lo que **supuso la necesidad de tomar conciencia de las posibles dificultades y/o limitaciones presentes en el entorno de programación de ABB**, y la forma de sortearlo/solucionarlo.

Así mismo se ha tenido en cuenta, **la funcionalidad y simplicidad del contenido implementado, para poder capturar las limitaciones existentes y poderlos utilizar en procesos instructivos**. A costa de **prescindir del cuidado del entorno gráfico**, que permite RobotStudio, **y centrarse en su funcionalidad**.

Todo ello, ha provocado la **necesidad de crear un elevado número de ficheros específicos,** desarrollados exclusivamente para **analizar cada funcionalidad requerida en la implementación de una Estación Completa**. Esto ha permitido **sintetizar todo un compendio de consideraciones, limitaciones y observaciones varias de RobotStudio,** para la implementación de una Estación Completa, **en un único Anexo**: "**ANEXO II**". Se ha procurado proporcionar la demostración de cada observación y la justificación y exposición de cada peculiaridad detectada, así como posibles soluciones a implementar.

Software: "RobotStudio"

A) Introducción al entorno de programación: RobotStudio

RobotStudio es una aplicación para PC, destinada tanto a modelado como a la programación fuera de línea. Es una herramienta que permite simular células industriales compuestas con robot. **Permite trabajar**:

a. Con un **controlador fuera de línea denominado Controlador Virtual (VC)** o **IRC 5**[6] **virtual,** que se ejecuta localmente en el PC.
b. Con un **controlador físico denominado IRC5 real**, que se puede ejecutar desde un FlexPendant o desde el PC.

[6]**IRC 5**: Controlador IC-5: Unidad de Procesamiento y Control de los Robots de las Series Industriales, entre las que se encuentra el Robot IRB 120

Posee varios modos de funcionamiento:

- "**Modo Online**": Conectado directamente sobre controladores reales.
- "**Modo fuera de línea**": Cuando se trabaja contra un controlador real sin conexión o mientras se está ejecutando/trabajando con un controlador virtual. El "Modo fuera de línea" permite trabajar de forma deslocalizada, desde cualquier terminal en el que esté instalado RobotStudio.

El **software** de los Controladores IRC5 está **basado en el entorno de programación VirtualController de ABB.**

Figure 2: Isotipo RobotStudio

Para más información consultar la **página oficial de ABB**:

- Url: **http://new.abb.com/products/robotics/es/robotstudio**

DataSheet [7] de **RobotStudio**: **RobotStudio_5_datasheet_es_print**

B) Documentación recomendada para Trabajar con RobotStudio: Manuales

Los siguientes manuales han sido una piedra angular en el desarrollo del TFG:

- **"Manual de Referencia Técnica: Descripción General de RAPID" (228 páginas)**:
  Contiene una explicación general del lenguaje de programación, así como la definición de algunos tipos de datos, instrucciones y funciones, clasificadas en bloques de trabajo genéricos.
  Uso recomendado: Consultas iniciales para saber a qué manual hay que ir a buscar la información desarrollada.

- **"Manual del Operador: RobotStudio_3HAC032104-es" (648 páginas)**:
  Explica los términos y conceptos relacionados con la programación tanto fuera de línea como en línea.
  Uso recomendado: Se describe cómo crear, programar y simular, células de trabajo y estaciones completas con RobotStudio.

- **"Manual de Referencia Técnica: Instrucciones, Funciones y Tipos de Datos de RAPID" (1.332 páginas)**:
  Se detallan las instrucciones, funciones y tipos de datos básicos del lenguaje.
  Uso recomendado: Destinado exclusivamente a consulta de funciones propias del lenguaje RAPID.

[7] **DataSheet**: Recopilación de documentación técnica facilitada por el fabricante y/o distribuidor

1. 2. Estación Virtual y Evaluación de Metodologías Enfrentadas

Para el desarrollo de una Estación Completa, ha sido **necesario estudiar** las distintas **funciones que permite implementar RobotStudio,** así como **distintas casuísticas** para abordar cada **objetivo**.

Entre las **necesidades contempladas** se encuentra: creación de objetos simples (*escenografía y cubo multi-usos*), creación de objetos inteligentes (*SmartBuffers y SmartConveyors*), creación de herramientas simples (*con 1 o más TCP's, con alteración de posición y orientación respecto de la brida de montaje*) y complejas y sus respectivos controladores lógicos (*aquellas herramientas con Partes móviles*), creación de autómatas por medio de **PLC's**[8] (*con SmartObject o con controlador IRC-5 virtual*), control y programación de robots…

Finalmente, **se ha ensamblado todo en una única estación** en la que **coexistan y se combinen todas las alternativas abordadas**, siendo el usuario final el que escoja los elementos que interactúen en la simulación, desde un panel selector en la Estación General. Por ello, ha sido **necesario proveerlo de robustez a fallos**, lo que ha provocado saturar el programa con lógica cableada. Adicionalmente, **se han implementado Herramientas Virtuales capaces de manipular y/o simular acciones de mecanizado**, y se han empleado distintos métodos en el proceso de creación.

Equipamiento de la Estación Simulada

Se ha **diseñado una célula industrial compuesta por**: 2 Robots distintos encargados de la manipulación y mecanizado de piezas, 2 Cintas transportadoras ("***Conveyors***[9]"), encargadas de proveer material y extraerlo, y un Autómata Maestro ("PLC") encargado de comandar los distintos elementos y ejecutar el programa principal. Como se ve en la **Figure 3** (*y vídeo*) y en la **Tabla 1**.

Se debe mencionar que: **RobotStudio no permite la implementación directa de PLC Maestros**, pese a ello se ha **conseguido programar un Automatismo Tipo Moore de 16 Estados con Concurrencia** de dos formas distintas:

- **Tipología industrial típica**: empleo de lógica cableada por medio del uso de **Biestables**[10], en un SmartObject
- **Punto de vista de un Programador**: por líneas de código en un Controlador IRC-5 Virtual.

[8] **PLC**: (en inglés: Programmable Logic Controller) Controlador lógico programable
[9] **Conveyors**: (en inglés) Cinta Transportadora Industrial
[10] **Biestable o Flip-Flop** (en inglés): es un multivibrador capaz de permanecer en uno de dos estados posibles durante un tiempo indefinido en ausencia de perturbaciones. Esta característica es ampliamente utilizada en electrónica digital para memorizar información.

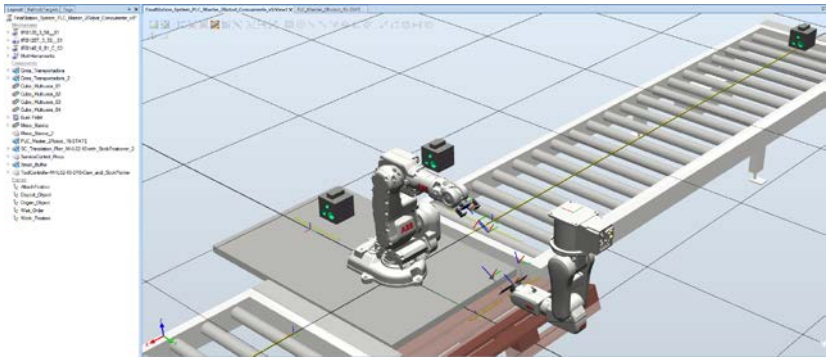


Figure 3: Simulación Célula Industrial Completa (*y Enlace a un Vídeo*)

A) *Resumen de los principales elementos de la Estación Final*

Tabla 1: Principales Elementos Estación Simulada Final

Elemento Principal	Complemento Auxiliar	Usuario Selecciona Modalidad en Simulación
PLC Maestro	Programa Principal	**Ejecución por SmartObject**
		Ejecución por Controlador IRC-5 Virtual
Modo Operación	Programa Principal	**Escritura de letras: sobre "Cara A" del Cubo**
		Pick and Place: sobre "Cara B" del Cubo
Robot: IRB 120 (Figure 5)	Herramienta doble Operación (**Figure 4**)	-
	Programa Principal	-
Robot IRB 140 (Figure 6)	Herramienta Principal	**Herramienta MHL02-10-SR.8 con 2 Sensores (Figure 7)**
		Herramienta MHL02-10-SR.8 con CollisionSenser (Figure 8)
		Herramienta Rotacional con 1 Sensor (Figure 9)
	Programa Principal	-
Conveyor Aprovisionamiento (Figure 10)	Lógica Cableada	-
Conveyor Extracción (Figure 11)	Lógica Cableada	-
Almacén Inteligente	Lógica Cableada	**Posicionamiento "Boxies" n1:n4**
Cajas Manipulables "*Boxies*" (Figure 12)	-	-
Escenografía Varia	Mesas, Pallets,…	-

Figure 4: Herramienta Doble Operación

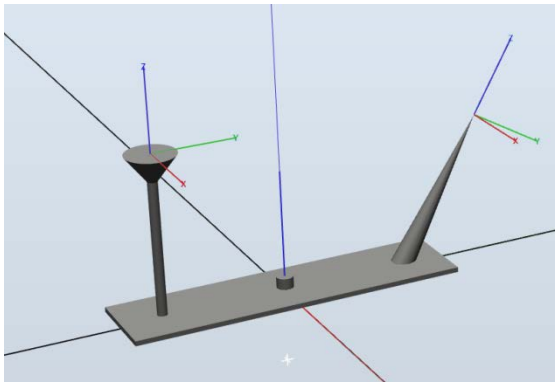


Figure 5: Robot ABB IRB 120

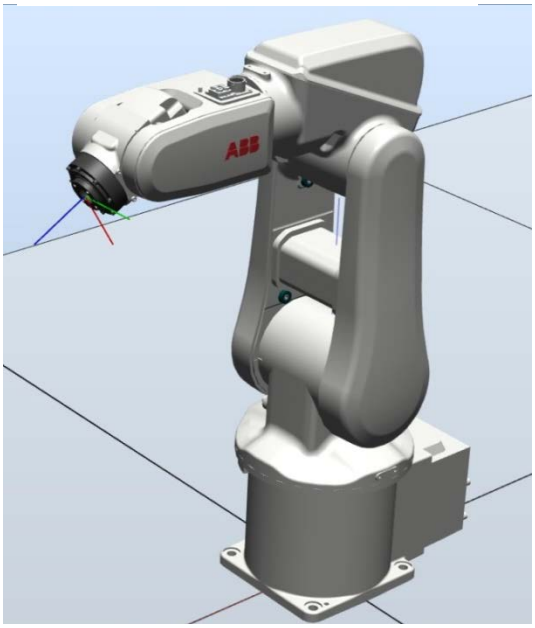


Figure 6: Robot ABB IRB 140

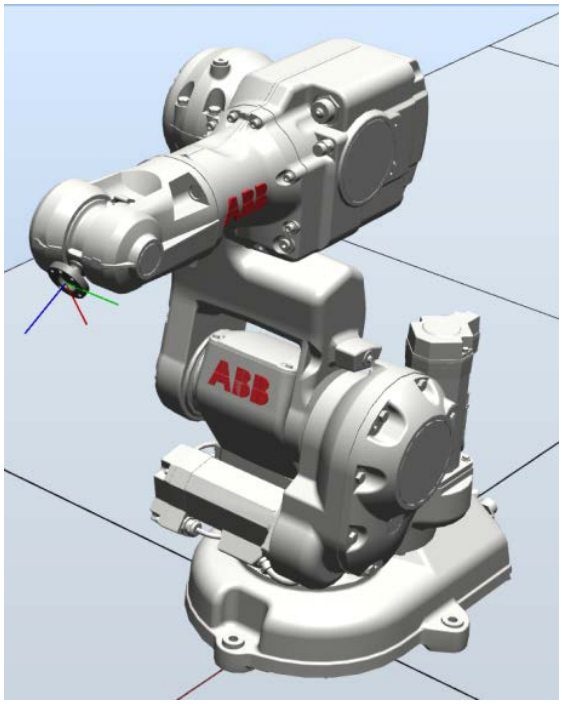


Figure 9: Herramienta MHL02-10-SR.8 con 2 Sensores

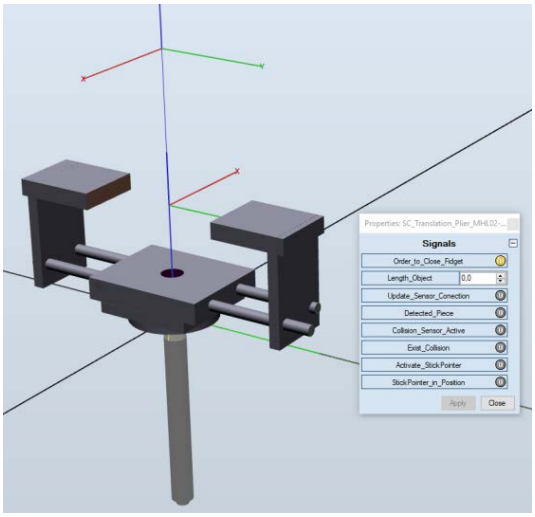


Figure 7: Herramienta MHL02-10-SR.8 con CollisionSenser

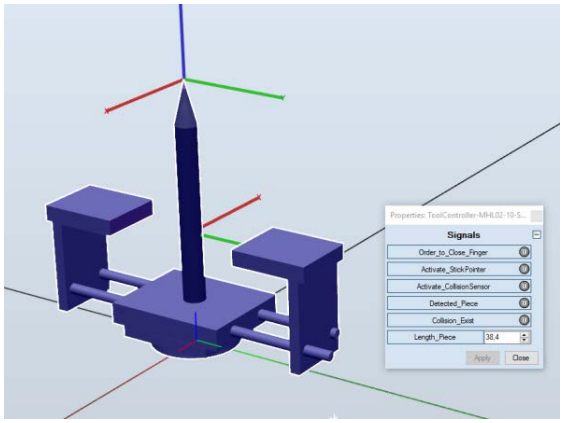
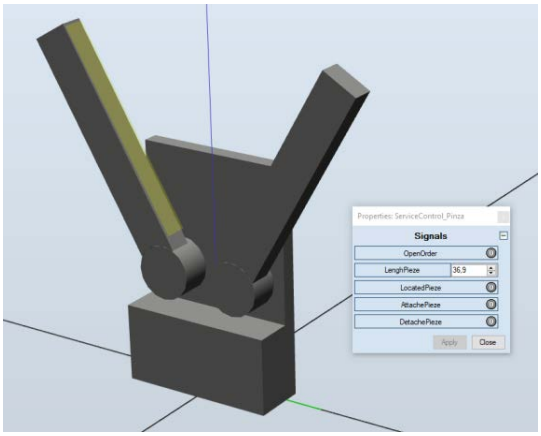


Figure 8: Herramienta Rotacional con 1 Sensor



Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Figure 10: Conveyor de Aprovisionamiento

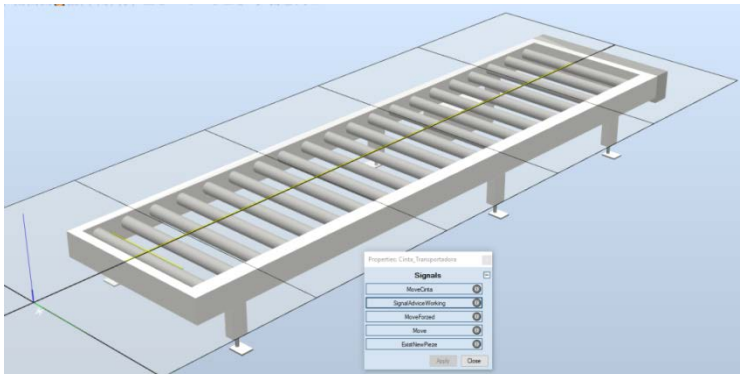


Figure 11: Conveyor de Extracción

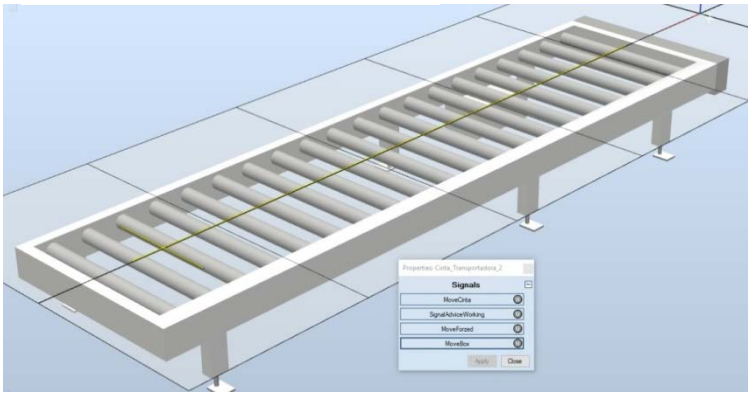
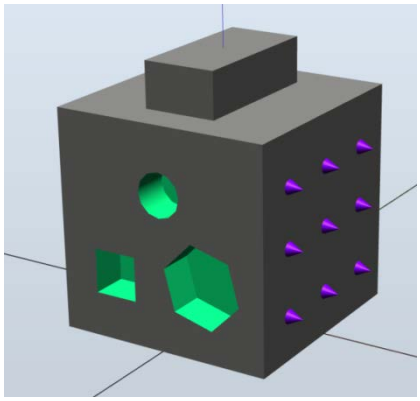


Figure 12: Cajas Manipulables "Boxies"



Comparativa de Procedimientos: Metodologías Enfrentadas

Se han **enfrentado los procedimientos de definición y creación**, así como los **resultados obtenidos** para:

- Los **Controladores de Herramientas complejas (*MHL02-10-SR.8*)**: "**Figure 7: Herramienta MHL02-10-SR.8 con 2 Sensores**" vs "**Figure 8: Herramienta MHL02-10-SR.8 con CollisionSenser**"
- Las distintas **metodologías de creación de** un **PLC o Autómatas Maestros**: "Metodología Biestable (*Flip-Flop*) con SmartObject" vs "Controlador IRC-5 Virtual"

Cabe mencionar que, tanto las versiones de la **Herramienta "MHL-02-10-SR.8"** simuladas, implementadas con Sensores o con el CollisionSensor, como el **PLC de 16 Estados se han depositado en librerías abiertas para el libre uso** de la sociedad universitaria y la comunidad de usuarios de RobotStudio.

1. 3. Inserción de un Robot IRB-120 en una Estación de Trabajo

A) *Objetivo*

Inserción de un Robot ABB IRB 120 en una célula de mecanizado de piezas existente, encargado de 2 operaciones distintas: abastecimiento de piezas y desbarbado de piezas mecanizadas.

La **Estación** es **controlada** por un **PLC Maestro (M 340 de MODICON)** en el que se ha implementado un **Grafcet[11]/SFC en Unity Pro XL**, encargado de comandar:

- Un Robot IRB 120
- Una **máquina taladradora FISCHERTECHNIK** con posicionador circular de 4 piezas para la celda de mecanizado

B) Descripción de la célula

Se puede **resumir por medio del ciclo de vida de las piezas a procesar**:

1. Abastecimiento de Piezas por el Robot (*aprovisionamiento*)
2. La máquina de taladrar opera sobre las piezas depositadas sobre la base móvil
3. El Robot operará sobre las piezas ya mecanizadas (*desbarbado*)
4. Finalmente, las piezas tratadas son extraídas de la Estación por caída libre

Siendo el diagrama:

Aprovisionamiento → Giro → Taladrado → Giro → Desbarbado → Giro → Caída por gravedad

Figure 13: Ciclo de Vida de la Pieza en Régimen Permanente: Célula Real

La **planificación de operaciones de la célula contempla el paralelizado de procesos concurrentes**: accionamiento simultáneo del taladrado y el robot (*aprovisionamiento y desbarbado*).

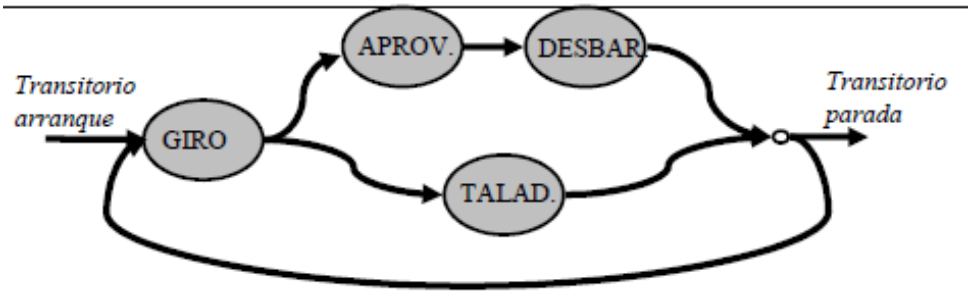


Figure 14: Secuencia Programación de Régimen Permanente: Célula Real

[11] **Grafcet/SFC (*Sequential Function Chart*)**: es el acrónimo tanto de **Gra**ph **F**onctionnel de **C**ommande **E**tape-**T**ransition (*en español, grafo funcional de control de etapa-transición*)

1. 3. 1. Equipamiento Hardware

A) *"Robot ABB IRB 120"*

Es un **robot industrial multiusos, el más pequeño y ligero de los ABB**, pesa alrededor de 25 kg, y puede **soportar una carga de 3 kg** (*4 kg en posición vertical de la muñeca*), dadas sus pequeñas dimensiones posee un área de trabajo limitada, **con un alcance de 580 mm**.

Figure 15: Robot ABB IRB 120 con su Controlador IRC-5 real y FlexPendant

Para más información y/o vídeos de funcionamiento se recomienda accede a la **página oficial de ABB**:

- **url**: http://new.abb.com/products/robotics/es/robots-industriales/irb-120
- **url** Vídeo demostración ABB: https://www.youtube.com/watch?time_continue=1&v=-39W3fdD5WA

Se ha incluido un enlace al DataSheet completo: DataSheet_ABB_IRB_120

a) Resumen de las Especificaciones Técnicas

Specification			
Variants	**Reach**	**Payload**	**Armload**
IRB 120-3/0.6	580 mm	3 kg (4kg)*	0.3 kg
Features			
Integrated signal supply	10 signals on wrist		
Integrated air supply	4 air on wrist (5 bar)		
Position repeatability	0.01 mm		
Robot mounting	Any angle		
Degree of protection	IP30		
Controllers	IRC5 Compact / IRC5 Single cabinet or Panel mounted		
Movement			
Axis movements	**Working range**	**Maximum speed**	
Axis 1 Rotation	+165° to -165°	250 °/s	
Axis 2 Arm	+110° to -110°	250 °/s	
Axis 3 Arm	+70° to -90°	250 °/s	
Axis 4 Wrist	+160° to -160°	320 °/s	
Axis 5 Bend	+120° to -120°	320 °/s	
Axis 6 Turn	+400° to -400°	420 °/s	
Performance			
1 kg picking cycle			
25 x 300 x 25 mm		0.58 s	
Max TCP velocity		6.2 m/s	
Max TCP acceleration		28 m/s²	
Acceleration time 0-1 m/s		0.07 s	
Electrical connections			
Supply voltage	200-600 V, 50/60 Hz		
Rated power			
Transformer rating	3.0 kVA		
Power consumption	0.25 kW		
Physical			
Dimension robot base	180 x 180 mm		
Dimension robot height	700 mm		
Weight	25 kg		
Environment			
Ambient temperature for Robot manipulator:			
During operation	+5°C (41°F) to +45°C (122°F)		
Relative transportation and storage	-25°C (-13°F) to +55°C (131°F)		
For short periods	up to +70°C (158°F)		
Relative humidity	Max 95%		
Noise level	Max 70 dB (A)		
Safety	Safety and emergency stops 2-channel safety circuits supervision 3-position enabling device		
Emission	EMC/EMI-shielded		

* With vertical wrist

Data and dimensions may be changed without notice

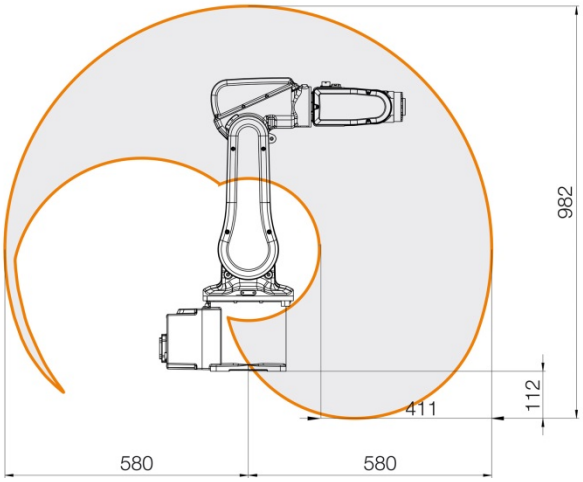


Figure 16: IRB120 Working range Sidways 1700x1406

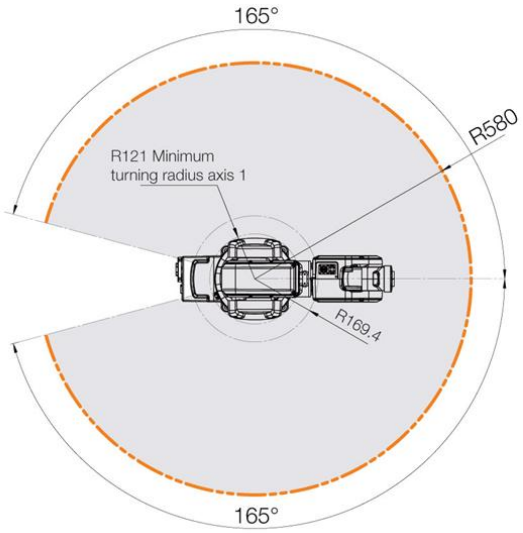


Figure 17: IRB 120 Working range Plane

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

B) Módulo de E/S binario del Robot

Permite al usuario, por medio de **instrucciones RAPID**, **controlar periféricos** mediante la activación de las líneas de salida y/**o responder ante** combinaciones de las líneas de entrada, generada por **eventos externos al robot**.

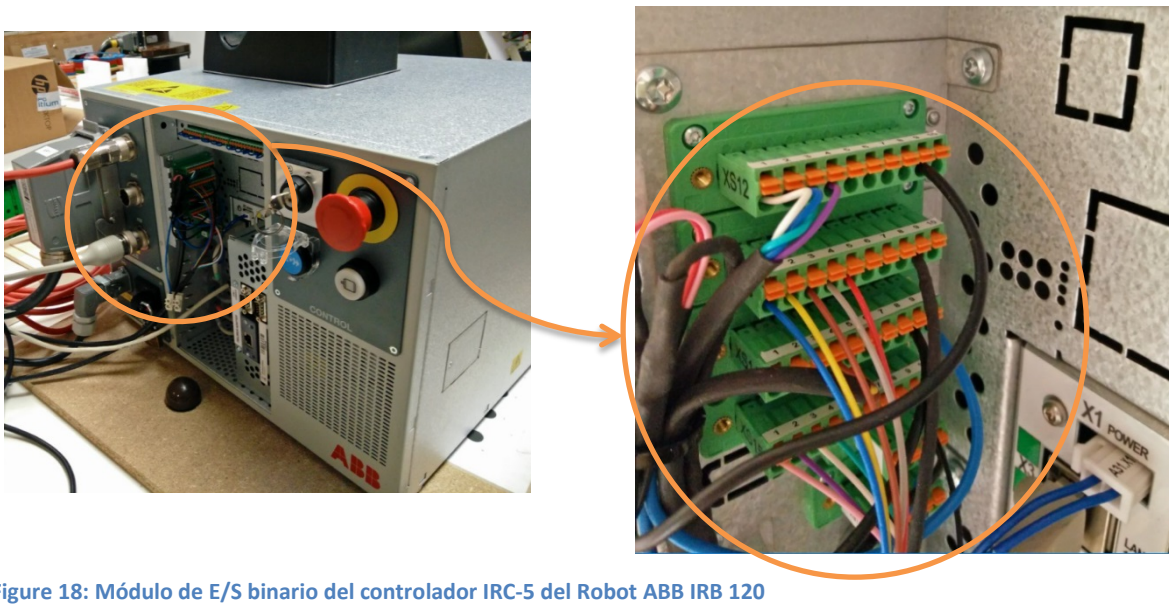


Figure 18: Módulo de E/S binario del controlador IRC-5 del Robot ABB IRB 120

C) Maqueta de máquina de taladrado FISCHERTECHNIK

Compuesta por:

- **Una plataforma giratoria** accionada desde un **motor de corriente continua acoplado a un reductor**.
- **Un taladro** que puede **desplazarse verticalmente** a lo largo de una guía, accionado desde el correspondiente motor-reductor.

La base cuenta con cuatro **posiciones de taladrado desfasadas 90º**, y **detectables mediante** el correspondiente **micro-ruptor**[12]. **El taladro** cuenta con **micro-ruptores que actúan como finales de carrera**, identifica límites superior e inferior.

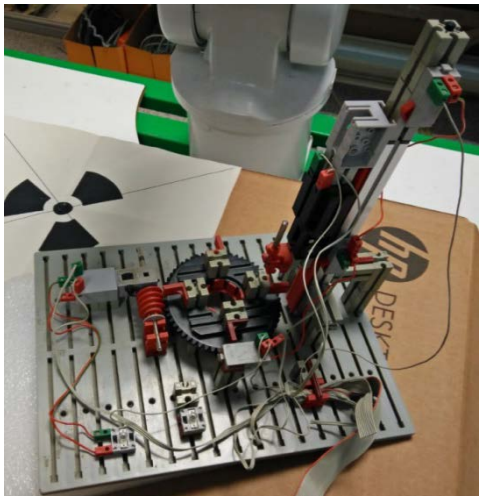


Figure 19: Maqueta de máquina de taladrado FISCHERTECHNIK

[12] **Ruptor**: Dispositivo para abrir o cerrar el paso de corriente eléctrica en un circuito

D) Autómata programable MODICON M 340

El autómata se presenta montado sobre un bastidor, y está formado por diferentes módulos de comunicaciones y E/S, de entre los que destacan los siguientes:

- Módulo BMX DDI 1602, de 16 entradas binarias a 24 V DC
- Módulo BMX DRA 0805, de 8 salidas tipo relé electromecánico
- Módulo BMX DDM 3202K, de 16 entradas binarias manuales o cableadas (interruptor triestado) a 24 V DC

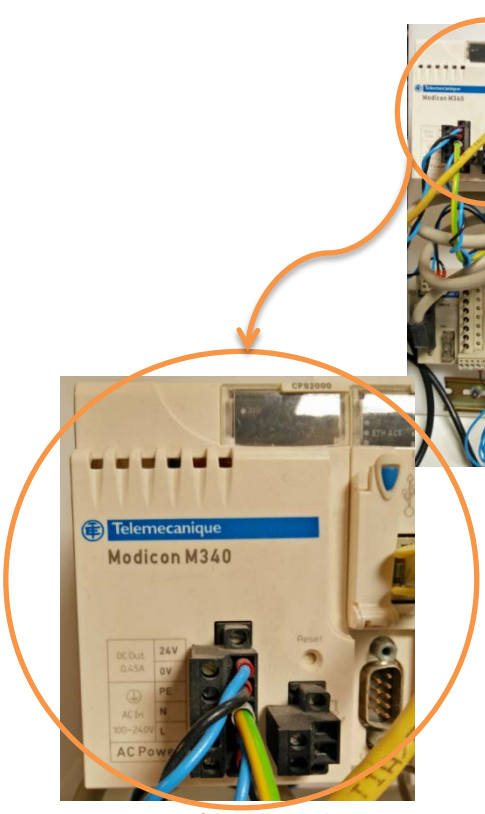


Figure 20: Módulo Central M 340

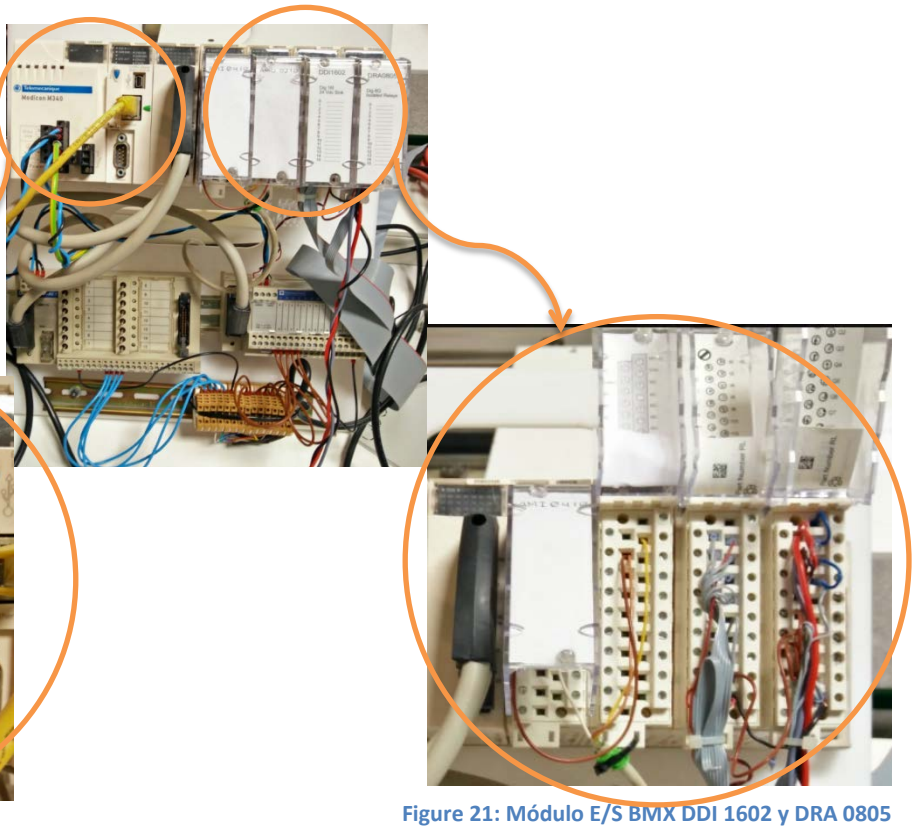


Figure 21: Módulo E/S BMX DDI 1602 y DRA 0805

Alimentado desde una fuente de alimentación con las tensiones a manejar (+24 y +-5 Volt.):

Figure 22: Fuente de Alimentación: +24V y +5V

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

E) Cableado entre el autómata y los robots

Cableado de E/S del autómata, su significación y su correspondencia con E/S del Robot ABB:

- **Entradas del Robot**: **Tabla 5: Mapa de Entradas y Cableado del Robot**
- **Salidas del Robot**: **Tabla 6: Mapeado de Salidas y Cableado del Robot**

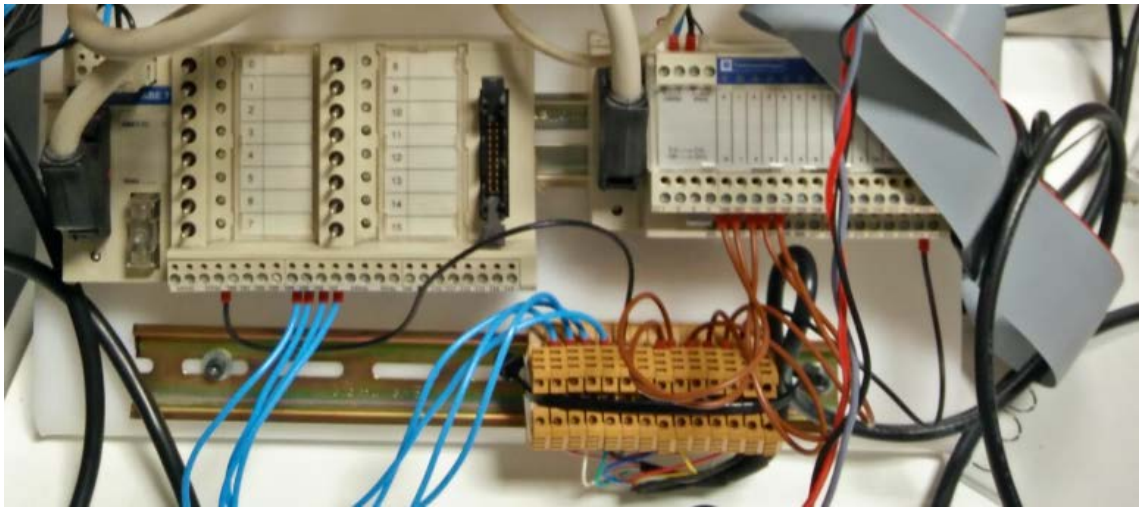


Figure 23: Cableado entre el autómata

Evaluación de RAPID y la Relativización de Localizaciones

La programación de las operaciones/tareas del robot se ha restringido exclusivamente al empleo de funciones propias de **RAPID**, considerando que **permite abordar las localizaciones de distintas formas**, **se han estudiado las diferencias entre algunas de ellas** y su posible recomendación de uso:

1. Basada en herramientas de definición de traslaciones (***RelTool***)
2. Basada en la estructuración de la escena proporcionada por ABB (***Offs/WObject***)
3. Basada en desplazamiento de programa (***PDispSet***)
4. Basada en composición explícita de **TRF**[13] (***PoseMult***)

[13] **TRF**: "Transformaciones", estructura matricial que contiene la información de Posición y Orientación relativas a una referencia, entre otros datos (*escalado, prespectiva*…).

2. Creación y Explotación de Entornos Virtuales:

Se ha concebido la **Estación como una síntesis de resultados del análisis de cada punto descrito** en el "**ANEXO II**" para la implementación de una Estación Completa y con el fin de exponer las diferencias entre los distintos procedimientos a la hora de abordar un mismo problema.

Se ha trabajado sobre un caso genérico, una célula de trabajo industrial donde existe **coordinación entre 2 Robots**, **controlado** todo **por un PLC Maestro** o Autómata de Moore de 16 Estados con Concurrencia.

Para ello, **se han diseñado e implementado los elementos que componen la Estación**, incluido el diseño de una herramienta virtual similar a la herramienta teórica del robot real, ABB IRB 120 ubicado en el laboratorio DIIS L0.06, propiedad del departamento de Informática e Ingeniería de Sistemas de la EINA.

En la **Tabla 1**: "**Principales Elementos Estación Simulada Final"** se encuentran definidos todos los componentes que forman la Estación Simulada y se habla en profundidad de cada uno de ellos en el **ANEXO VII**.

Se ha previsto la interacción del usuario con la Estación, permitiéndole decidir qué elementos participan en la simulación y el modo de operación, por medio de un Display.

2. 1. Descripción de la Estación Virtual

La Estación consta de un PLC Maestro, 2 Robots, 2 Cintas transportadoras ("*Conveyors*"), un almacén inteligente con reposicionamiento de objetos, varios elementos manipulables ("*Boxies*") y escenografía básica. Como se puede ver en la **Figure 3: Simulación Célula Industrial Completa.**

- **PLC Maestro:** encargado de ejecutar el programa principal y lograr que todo interactúe en armonía.
- **Cintas transportadoras (*"Conveyors"*):** elementos encargados de abastecer y sacar de la célula de trabajo el material manipulado por los Robots.
- **Robot IRB 140** dotado con una **Herramienta compleja: MHL02-10-SR.8:** se encarga de transportar y posicionar las piezas que van a ser manipuladas, en función de la operación que haya demandado el usuario.
- **Robot IRB 120** dotado con una **herramienta simple con doble operación**: se encarga de simular las operaciones de mecanizado y/o manipulación de los objetos:
  - "*Escritura de letras O-C-T*": respetando una secuencia específica o escribiendo letras sueltas sobre la Cara A de los "*Boxies*"
  - "*Pick and Place*": sobre la Cara B.

Jerarquía de Control

Si solo se contempla la Estación, **el PLC Maestro es el elemento de mayor rango**. Pudiendo ser alterado en modos de operación por acción directa de los usuarios. Por debajo de él, se encuentran las cintas transportadoras y los robots, seguidos de las herramientas de estos.

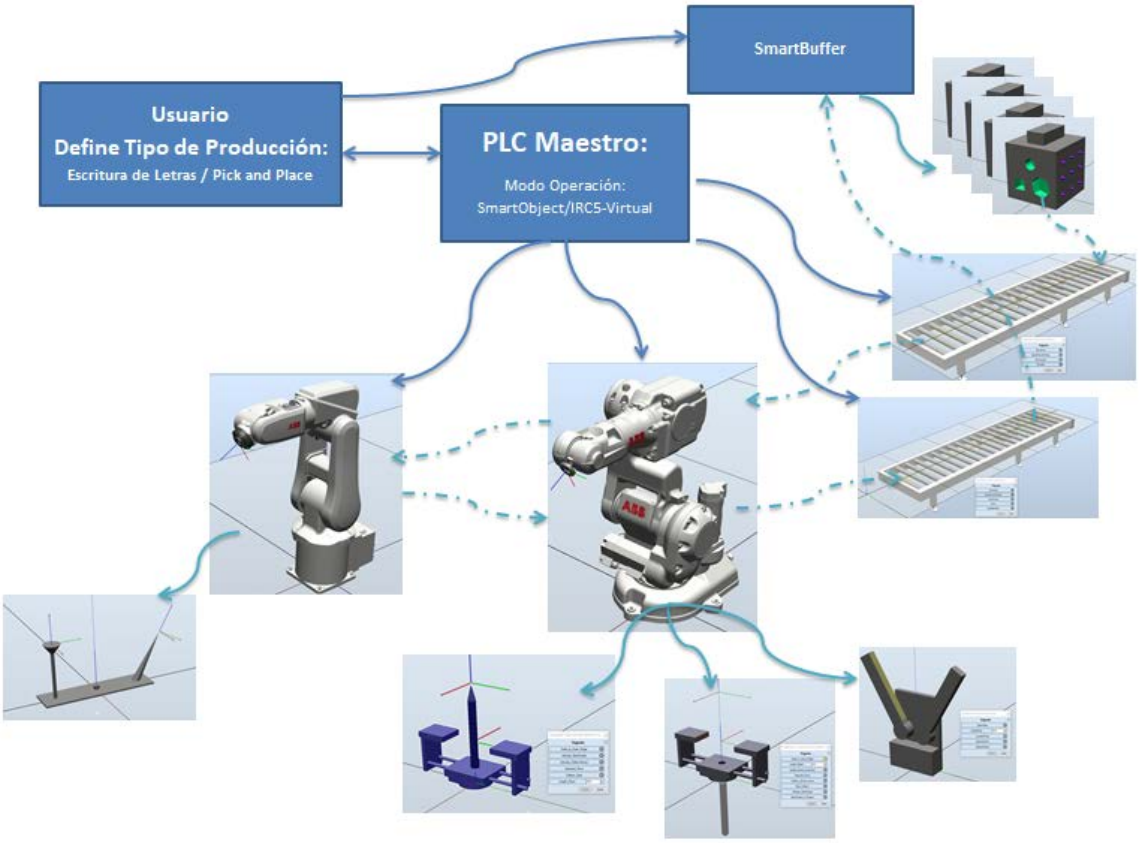


Figure 24: Jerarquía de Control y Relaciones entre componentes de la Estación Simulada

En los enlaces **se pueden ver distintos vídeos del funcionamiento de la Estación**, tanto operando con el PLC en modalidad SmartObject como en modalidad Líneas de Código con un IRC5-Virtual:

- **CompleteStation-IRC5_PLC_Master**
- **CompleteStation-SmartObject_PLC_Master**
- **CompleteStation-SmartObject_PLC_Master_PickAndPlace**

Para más información ver: **ANEXO VII**.

2. 2. Implementación de la Herramienta Virtual

Se ha diseñado una **herramienta,** lo más **versátil** posible, para poder abordar un amplio abanico de **tareas**, tanto de **posicionamiento y seguimiento de trayectorias** como de **manipulación y agarre de objetos**.

La herramienta "MHL02-10-SR8" es una **combinación de una Pinza Neumática** de gran apertura, empresa **SMC**, **con un puntero** acoplable emulando la Herramienta Real que se va a instalar en el Robot IRB 120 del laboratorio L0.06 del DIIS de la EINA.

Partiendo del modelo teórico de la herramienta real, se ha parametrizado y reproducido en un modelo digital, el cual poder exportar a las distintas simulaciones. *Las posibles diferencias entre la herramienta real y la herramienta simulada responden a la secuencia de fechas entre la orden de fabricación de la herramienta real y la necesidad de simular resultados con la herramienta virtual.*

RobotStudio permite **definir características específicas de la herramienta** para cada uno de los Modos de Operación contemplados (*Modo Garra y Modo Puntero*), como son: el ***Tool Center Point*** **(TCP[14]'s)**, el **Centro Geométrico (CG[15]) e Inercias Principales (I_x I_Y e I_z)**.

Se ha diseñado el control lógico de la Herramienta por dos procedimientos distintos:

a. **Control lógico con Sensores** (ejemplo expuesto a continuación):
    a. Sin restricciones de diseño.
    b. El vástago posee 1 Grado de Libertad (desplazamiento traslacional, se esconde en el interior del robot virtual).
b. **Control lógico con Detector de Colisiones**:
    a. Los sólidos que componen la Herramienta NO pueden tocarse ni en situación estática ni en las trayectorias de desplazamiento.
    b. El vástago NO posee Grado de Libertad, se oculta y se visualiza en la misma posición física relativa a la base.

Para más información sobre la implementación de la herramienta simulada, ver: **ANEXO V: Implementación de la Herramienta Virtual: "*SR.8*"**.

[14] **TCP**: Tool Center Point: Punto de ajuste de la herramienta al que se hará referencia para toda acción de la herramienta
[15] **CG**: Centro Geométrico de Masas de un sólido/conjunto de sólidos que operan como uno solo

2. 2. 1. Propiedades Generales y Geométricas

A pesar de haberse diseñado por medio de dos procedimientos distintos, la **estructura geométrica resultante de la Herramienta para las dos configuraciones es muy similar**.

A) Aproximaciones en el Diseño Geométrico

Se han aplicado las siguientes **aproximaciones en el diseño**:

1. **Estructura Geométrica Regular**: Composición de prismas regulares y varillas de masa despreciable
2. Para todo Prisma y Varilla: **Sólidos Rígidos, Indeformables e Impenetrables**
3. A nivel de CG e Inercias: **Posee 2 Planos de Simetría en torno Z ("xz" e "yz")**
4. **Base de la Herramienta**: Centrada en el Origen con la Base de la Brida coincidente con el Centroide de la misma
5. La **Varilla**, sólido extraíble, en la simulación guardada una **representación únicamente visual** en la modalidad garra, aunque con repercusiones en la lógica cableada.
6. **Garra de gran apertura** (*comparación con la garra real*): **64mm** frente a **30mm** de la real. Conveniente para facilitar la manipulación de un mayor número de objetos.

B) Cálculo de CG e Inercias Principales

En el **cálculo del CG e Inercias Principales** se ha diferenciado entre las configuraciones:

a. **Modo Garra**: donde se ha despreciado la existencia del Puntero extraíble y considerado la configuración más desfavorable "completamente abierta"
b. **Modo Puntero**: los dedos agarrando el Puntero para dotarlo de mayor resistencia a flexión.

El cálculo se ha llevado a cabo con Matlab, fichero adjunto en el **ANEXO V**.

C) *Propiedades Implementadas*

Se ha empleado un índice de colores, para la identificación de E/S, en los esquemas generales de las Herramientas implementadas, **Figure 27** y **Figure 28**:

- Azul oscuro: Modo operación Garra
- Violeta: Modo Pntero "*Stick*"
- Naranja: Sólo de carácter informativo: Detector de Colisiones
- Verde: Informa de la distancia de apertura de la garra

D) Modos de Funcionamiento

Utilizando como ejemplo la Herramienta creada con Sensores

La herramienta ha sido diseñada contemplando dos **modos** distintos **de funcionamiento, que el usuario puede seleccionar durante la simulación**, y sus consiguientes propiedades técnicas asociadas:

a) Funcionamiento "Modo Garra"

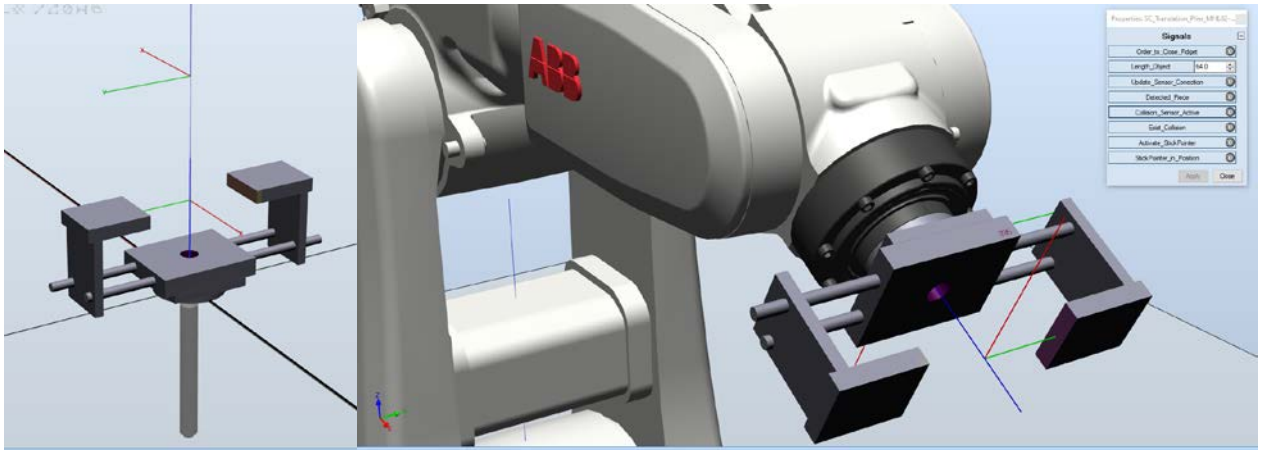


Figure 25: Modelo Virtual de la Herramienta: **Modo Garra**

Tabla 2: Propiedades Técnicas Herramienta Simulada "Modo Garra"

Masa [Kg]	C.G.=(x,y,z) [mm]			Inercias: I_x I_Y I_z [Kg/ m^2]		
1	0	0	27.8750	0.00015932	0.00017696	0.00023658

b) Funcionamiento "Modo Puntero '*Stick*'"

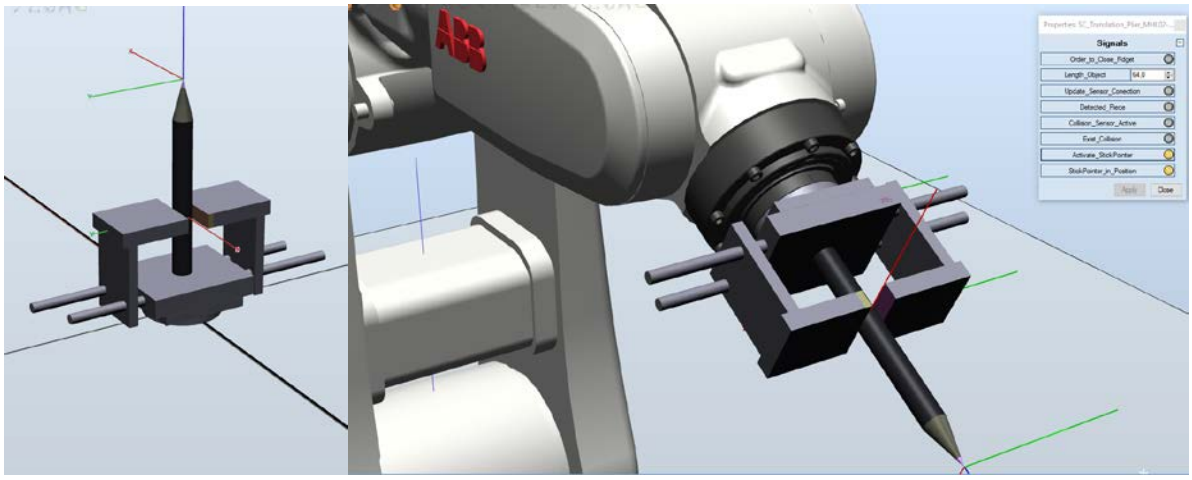


Figure 26: Modelo Virtual de la Herramienta: **Modo Stick**

Tabla 3: Propiedades Técnicas Herramienta Simulada "Modo Puntero"

Masa [Kg]	C.G.=(x,y,z) [mm]			Inercias: I_x I_Y I_z [Kg/ m^2]		
1.1	0	0	31.3068	0.00020401	0.00022165	0.00059966

OBSERVACIÓN: La influencia del vástago/puntero no supone cambios relevantes en la implementación de la herramienta virtual, en las simulaciones de RobotStudio se podría haber despreciado la inclusión de estos cálculos a la hora de definir/crear la herramienta.

2. 2. 2. Implementación con: Sensores

A) *Propiedades principales*

a. **No sujeta a restricciones estructurales**: irrelevancia a penetración entre los sólidos que la componen.
b. No es estrictamente necesario contemplar un Grado de Libertad adicional para el vástago, se puede implementar con bloques de ocultación y visualización.
c. **Sujeta a restricciones propias de los sensores**: Ver **ANEXO IV**.
Se la ha dotado "*para mayor seguridad*" de un pin para forzar un refresco de los sensores, antes de ser utilizada.
d. **Necesarios 2 Sensores planos tangenciales entre sí**: uno principal y otro auxiliar en el funcionamiento en cierre de garra.
e. La **distancia de penetración del Plano del Sensor, en el sólido detectable,** es inversamente proporcional a la relación entre el "Paso de Simulación" (***Step***[16]) y la velocidad relativa de aproximación entre los sensores y el objeto a detectar.
f. **Lógica cableada**: aumenta considerablemente la complejidad y cantidad de bloques lógicos para controlar la operación de agarre seguro.
g. **Mayor tiempo de ciclo** para comprobar los estados lógicos.
h. **Posible aparición de "*Lag*" en simulación**, se aprecian saltos discontinuos en la simulación, aunque la ejecución del programa es correcta.
i. **Menor tamaño del fichero** ".rslib".
j. **Requiere la inclusión de precisión "fine"** en la orden previa de movimiento del robot. No necesita consideraciones adicionales en código RAPID.

B) *Esquema general*

Figure 27: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con Sensores

C) *Funcionamiento de la Garra: Implementada con Sensores*

Se puede ver el Funcionamiento de la Garra en el siguiente vídeo: Tool_Sensor_Claw.

[16] **Step o "Paso de Simulación"**: Tiempo transcurrido entre la actualización de las acciones simuladas. RobotStudio es un programa de simulación discreto.

2. 2. 3. Implementación con: Detector de Colisiones

A) *Propiedades principales*

a. **Sujeta a restricciones estructurales**: obligación de que no exista contacto, ni superposición, entre los sólidos que componen las *Part*es de la Herramienta.
b. No es necesario contemplar un Grado de Libertad adicional para el vástago, se puede implementar con bloques de ocultación y visualización, como ha sido el caso. Lo que evita la necesidad de la salida "output" validando la posición del *StickPointer*.
c. **No utiliza sensores convencionales**: no sujeta a restricciones propias de los sensores.
d. **Mayor robustez a penetración**: menor distancia de penetración de la herramienta sobre el sólido detectable, la penetración es más constante ante la relación de "Velocidad de Aproximación-*Step*". Ver **ANEXO IV**.
e. **Menor cantidad de lógica cableada**: disminuye el número de bloques lógicos para controlar la operación de agarre seguro.
f. **Menor tiempo de ciclo** en la comprobación de los estados lógicos.
g. **No detectada la aparición de "*Lag*[17]"** en simulación.
h. **Mayor tamaño del fichero** ".rslib", un orden 10 veces mayor que la modalidad de sensores.
i. Es **insuficiente** con **la inclusión de precisión "fine" en la orden previa de movimiento** del robot: **Ver vídeo demostración:** Acumulative_Error_CollisionSenser
Para aquellos programas en los que no se recolocan los objetos solos, o no se ejecutan los comandos de creación y destrucción, es **necesario tomar consideraciones adicionales en código RAPID** como dotar al fichero RAPID de "WaitTime xxxx" para evitar problemas en la acción de soltar piezas**. Ver vídeo**: Solution_Acumulative_Error

B) *Esquema general*

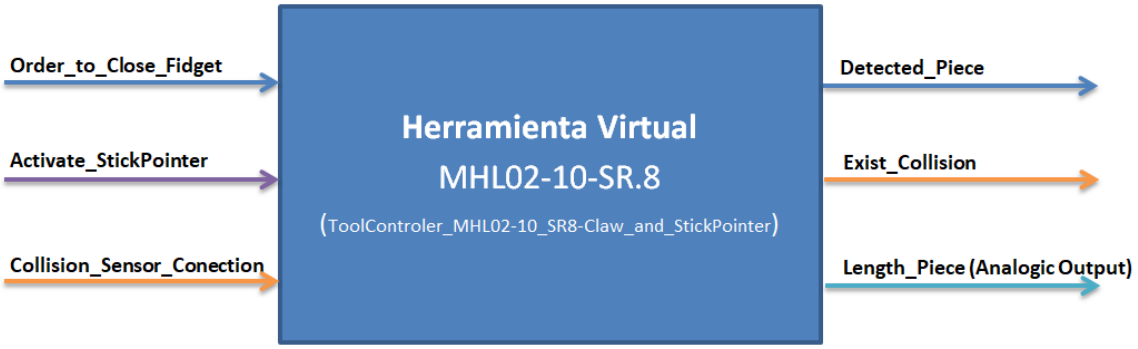


Figure 28: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con CollisionSensor

C) *Funcionamiento de la Garra: Implementada con CollisionSensor*

Se puede ver el Funcionamiento de la Garra en el siguiente vídeo: Tool_Collision_Claw.

[17] **Lag**: Retraso que se produce en una comunicación, generalmente en una red informática, aunque puede aplicarse a cualquier tipo de comunicación.

2. 3. Simulación de PLC's

Hay que mencionar que: **RobotStudio no permite la implementación directa de PLC Maestros**, pese a ello se ha **conseguido implementar un Automatismo Tipo Moore** de varias formas distintas:

- **PLC Programable (***Punto de vista de un Programador***)**: Implementado en código estructurado, un Autómata de MOORE, con la modalidad *Switch-Case*
- **PLC de Lógica Cableada (***Tipología Industrial Típica***)**: al estilo tradicional, por medio de la implementación de "*Flip-Flop*" en un SmartObject
- **PLC Lógica de Contactos**: al estilo tradicional, por líneas lógicas de contactos, en el *EventManager* (*DESECHADA por elevado número de limitaciones y problemática en la gestión de E/S añadido a que es un sistema asíncrono; elemento relevado por los SmartObject*)

A) *Propiedades de la Estación simulada*

Partiendo de la base que la **Estación Simulada cuenta con**:

- **Sistema de suministro de piezas continuo y de distribución aleatoria**: representado por la Cinta Transportadora 1 (ver **Figure 10: Conveyor de Aprovisionamiento**)
- **Sistema de extracción segura y con ausencia de bloqueos**: representado por la Cinta Transportadora 2 (ver **Figure 11: Conveyor de Extracción**)

El **PLC Maestro es el encargado de controlar el estado de la Estación** durante el proceso de producción. Se ha simplificado hasta conseguir un **Automatismo Tipo Moore de 16 Estados con Concurrencia**. Cada estado, representado en el **Grafo**[18], es una configuración absoluta de la Estación que específica qué partes están en funcionamiento y qué tarea/operación se encuentra haciendo.

El PLC Maestro se encargará de coordinar 2 Robots y garantizar la concurrencia entre ambos:

- **Robot 1 "R1P_xC_y"**: responsable de proveer/posicionar las piezas sobre una zona de trabajo controlada y común y sacarlas de dichas zonas cuando ya estén mecanizadas
- **Robot 2 "R2P_x"**: responsable de manipular y mecanizar las piezas posicionadas previamente por el Robot 1.

[18] **Grafo**: Representación simbólica de los elementos constituidos de un sistema o conjunto, mediante esquemas gráficos.

B) *Automatismo Tipo Moore de 16 Estados con Concurrencia*

El **PLC** se encarga de **ordenar al Robot 1 una de las siguientes funciones**:

- Coger y posicionar las piezas de la Cinta Transportadora 1 sobre la Posición 1 de la zona de trabajo (***R111***)
- Coger y posicionar las piezas de la Cinta Transportadora 1 sobre la Posición 2 de la zona de trabajo (***R121***)
- Extraer de la Posición 1 la pieza mecanizada y depositarla en la Cinta Transportadora 2 (***R112***)
- Extraer de la Posición 2 la pieza mecanizada y depositarla en la Cinta Transportadora 2 (***R122***)

Así como **ordenarle al Robot 2 que ejecute una de las dos tareas programadas**:

- Mecanizar sobre la Posición 1 (***R12***)
- Mecanizar sobre la Posición 2 (***R22***)

El **Modo de Operación de la Estación**, escritura de letras sobre la Cara A de los "*Boxies*" o "*Pick and Place*" sobre la Cara B, **o la Herramienta seleccionada del IRB 140**, una de las 3 Garras implementadas, **quedan fuera del funcionamiento del automatismo**. Serán los propios robots los encargados de seleccionar qué tipo de tarea de posicionamiento y tarea de operación han de realizar, en función de las directrices de los usuarios.

Se ha codificado la información:

- **Robot 1 "R1P_xC_y"**: **R1** (*Robot 1*) trabaja sobre $\mathbf{P_x}$ (***1**: Posición 1 ; **2**: Posición 2*) tomando como referencia $\mathbf{C_y}$ (***1**: Suministrar/Coger pieza de la Cinta Transportadora 1 ; **2**: Extraerla pieza mecanizada y depositarla en la Cinta Transportadora 2*).
- **Robot 2 "R2P_x"**: **R2** (*Robot 2*) trabaja sobre $\mathbf{P_x}$ (***1**: Posición 1 ; **2**: Posición 2*).
- **Posición "Pxy"**: **x** es la Posición 1 e **y** es la Posición 2, para ambos casos: (***0**: Posición vacía ; **1**: Posición ocupada y siendo manipulada ; **2**: Posición Ocupada y pendiente de Extracción de pieza mecanizada*).
- **Existencia de Objeto en Cinta Transportadora 1**:
  - **Obj**: Variable que indica la existencia de pieza en la Cinta Transportadora 1, para que sea transportada por el Robot 1 hasta la $\mathbf{P_x}$.
  - $\overline{\boldsymbol{Obj}}$: No hay piezas para producir.

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

2. 3. 1. Programa Implementado en el Autómata: Grafo de Estados

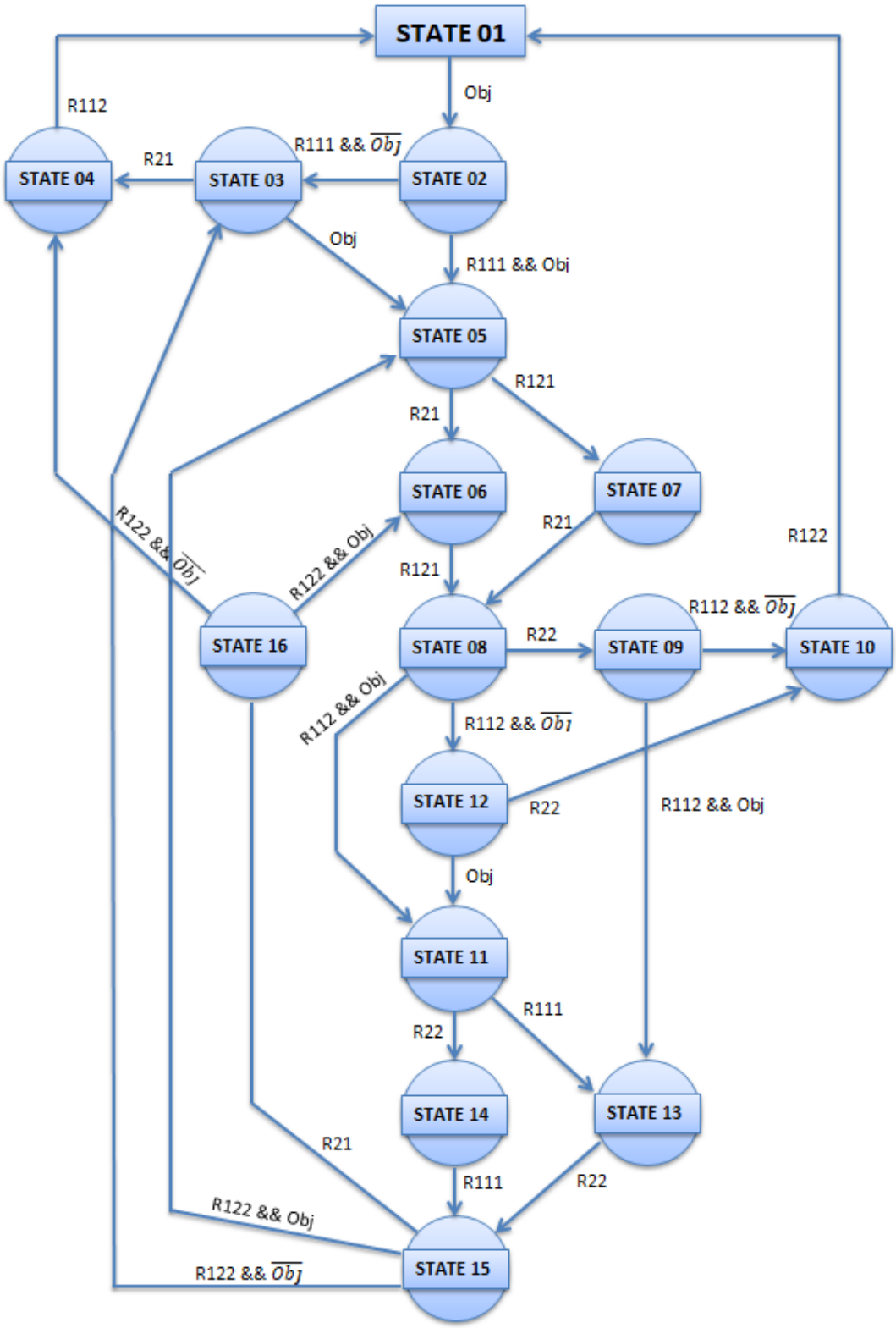


Figure 29: Grafo de Estados del PLC Master de 16 Estados con Concurrencia: Estación Simulada

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Tabla 4: Programa Implementado en el Autómata: Grafo de 16 Estados con Concurrencia

ESTADO del GRAFO:	Descripción	Distribución de la Posición de Trabajo	Salida del Autómata	
			Acción Robot 1	Acción Robot 2
STATE 01	Estación en Reposo	P00	-	-
STATE 02	Suministrando	P00	R111	-
STATE 03	Operando	P10	-	R21
STATE 04	Extrayendo	P20	R112	-
STATE 05	Concurrencia	P10	R121	R21
STATE 06	Suministrando	P20	R121	-
STATE 07	Operando	P11	-	R21
STATE 08	Concurrencia	P21	R112	R22
STATE 09	Extrayendo	P22	R112	-
STATE 10	Extrayendo	P02	R122	-
STATE 11	Concurrencia	P01	R111	R22
STATE 12	Operando	P01	-	R22
STATE 13	Suministrando	P02	R111	-
STATE 14	Operando	P11	-	R22
STATE 15	Concurrencia	P12	R122	R21
STATE 16	Extrayendo	P22	R122	-

2. 3. 2. PLC's con Controlador IRC-5: Implementado en RAPID

El **Controlador IRC-5 Virtual** se encarga de comandar y coordinar todos los elementos que componen la Estación Simulada, respeta la **siguiente secuencia**: Inicialización de todos los componentes de la Estación, puesta en marcha de cintas transportadoras y posicionamiento los objetos "*Boxies*", para finalmente ejecutar el programa principal. El programa principal, ejecutivo cíclico o "*Loop*", se ha implementado con la metodología de **2 *Swich-Case*[19], uno para actualizar el Estado y otro para actualizar las salidas**.

El código de programa principal, ejecutado cíclicamente, se resuelve en dos tiempos:

- **Primero**: Actualiza el Estado del Autómata, con la primera estructura "case", chequeando la variable de "Estado del Autómata" actual y las condiciones de transición asociadas al Estado actual (Configuración de Entradas)
- **Segundo**: Actualiza la configuración de Salidas en función del Estado actual/activo, con un segundo "case" con la variable de "Estado del Autómata" encadenada al primero.

La **activación de salidas** se ha implementado **por nivel**, responde al problema de la gestión de E/S de RobotStudio. *Si se presenciasen "fallos" en la comunicación de E/S, se aconseja forzar sincronismos por código RAPID (inclusión de "Delays" con duración superior al valor de un "Step" e inferior a dos) al final del programa, perdiendo velocidad de procesamiento a cambio de garantizar la comunicación entre los elementos de la Estación.*

Para visualizar el Código implementado en el PLC Maestro, consultar: **ANEXO VI**. Se puede ver un vídeo del funcionamiento en: **Estación_con_PLC_Controlador_IRC-5_Virtual**.

2. 3. 3. PLC's con Bloques Lógicos: Implementado con SmartObject

Esta **opción** es **considerada como una de las metodologías tradicionales de la industria de la última década**. A pesar de ello, la complejidad a la hora de diseñar un programa con esta metodología, sumado a las dificultades de implementarlo con RobotStudio y la problemática en la gestión de E/S entre los elementos que componen la Estación, provoca que **se desaconseje seriamente el empleo de esta metodología** para programar PLC's de más de 8 Estados.

Se han **adjuntado** los **esquemáticos**, tanto de **puertas lógicas** como la **imagen del SmartObject de RobotStudio**, **en** el **ANEXO VI**. RobotStudio presenta diversas deficiencias en el apartado de diseño, ver el **ANEXO II**, entre ellas la dificultad de lectura de los diagramas de bloques lo que ha provocado la necesidad de adjuntar un esquema de los "*Flip-Flop*" implementado con el Programa "***Crocodrile Clips v3.5***".

Ver vídeo del funcionamiento de: **Estación_con_PLC_SmartObject**.

[19] **Switch-Case**: estructura de control empleada en programación. Se utiliza para agilizar la toma de decisiones múltiples. Trabaja de la misma manera que lo harían sucesivos if , if else o until anidados.

A) *Esquemático resultante del Grafo de Estados implementado*

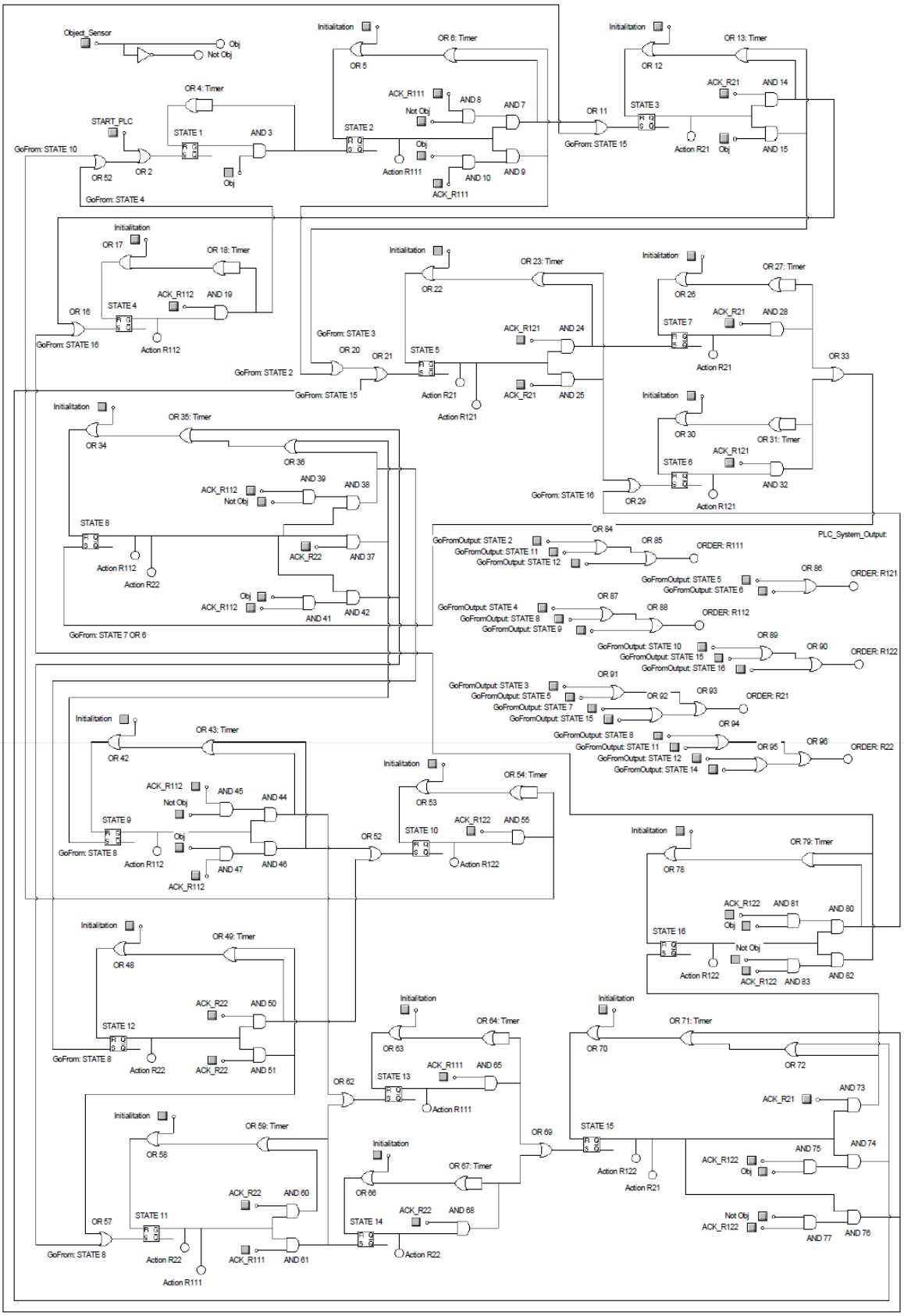


Figure 30: Esquemático del PLC Master 16 Estados: Flip-Flop del SmartObject

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

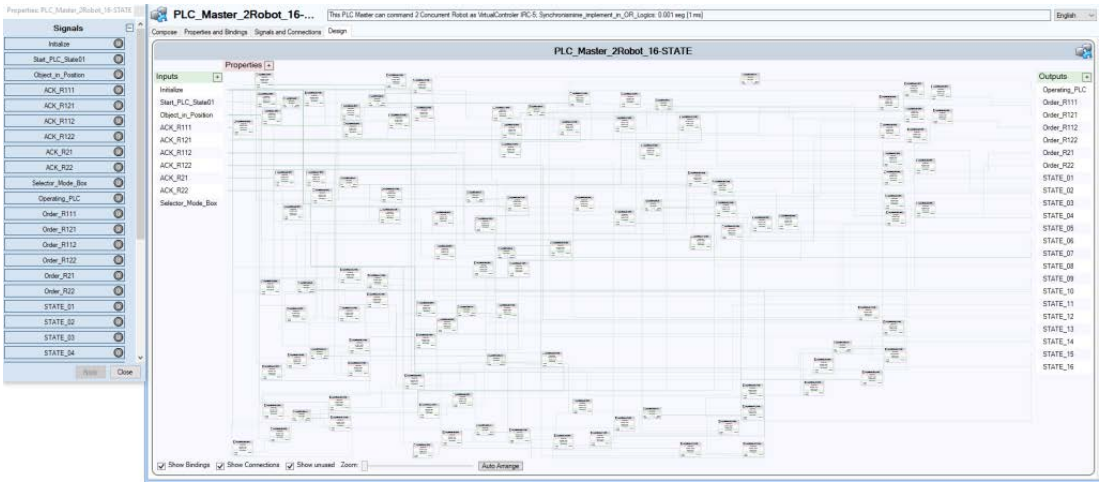


Figure 31: Esquemático del PLC Master 16 Estados: SmartObject

Con intención de facilitar la lectura, se ha intentado hacer agrupaciones visuales tanto con los componentes del SmartObject, formando bloques compactos que representan cada uno de los 16 Estados del Autómata, como en el esquemático, donde cada lazo de realimentación (*entrada Reset de los Flip-Flop*) representa un Estado completo del Autómata.

2. 4. Aplicaciones de los Robot en RAPID

Los Robots **han sido configurados como "*esclavos perfectos*"**, si no están obedeciendo una orden previa estarán esperando una nueva orden del PLC Maestro.

Ambos Robots **constan de dos ficheros**:

- **Módulo Principal**: está formado por un programa de inicialización y un ***Loop***[20], donde se estará esperando una nueva orden momento en el que se chequeará cual de todas las órdenes es, se ejecutará y se informará al Maestro una vez completada con éxito.
- **Módulo de Funciones**: Fichero en el que se encuentran los datos de los elementos del programa y la lista de Procesos/Ordenes programadas.

a) Observación Importante de la Simulación de Robots

Se ha detectado que **es necesario proveer de pequeños "*Delays*" tras las órdenes de cierre de garra en las simulaciones y en la activación de Salidas del IRC-5 Virtual**, caso que difiere del comportamiento de la instalación en la realidad. Se cree que es debido a la problemática en la gestión de E/S, evidenciada en los ANEXOS, ver el **ANEXO II** y **ANEXO IV**.

Los "*Delays*" se deben dimensionar en función del *Step* seleccionado en la configuración del Programa, siendo el $T_{Delay} > T_{Step}$.

[20] **Loop** (*en programación*): Bucle o ciclo, sentencia que ejecuta repetidas veces un trozo de código hasta que la condición asignada a dicho bucle deja de cumplirse.

2. 5. Evaluación de RobotStudio para Simular Células de Trabajo

A) *Consideraciones se Carácter General*

Para ver la información detallada de cada punto se aconseja leer el **ANEXO II**, como **conclusión del TFG**:

a) Dada su **posición de liderazgo** en el **Mercado de la Robótica Industrial**, **RobotStudio** se está convirtiendo en un ***"estándar de facto"*** **en** la **Simulación de Estaciones Robotizadas**. Se puede considerar un **Software completo** que **permite diseñar instalaciones de cierta complejidad**, y que a grandes rasgos cumple las expectativas depositadas en él.

b) La **redundancia del lenguaje RAPID**, posee un extenso repertorio de instrucciones/funciones **para llevar a cabo las mismas acciones**, lo que dificulta la legibilidad del código fuente. Véase el análisis de la **46**.

c) Normalmente las instalaciones robotizadas se componen de diversos elementos que hay que simular de manera concurrente. **RobotStudio no específica cómo gestiona la concurrencia entre los componentes de la Estación**.
La **Gestión Incorrecta de la Concurrencia** puede provocar la **pérdida y/o aparición de retardos de eventos** (*como es la pérdida de activación de una de las Entradas de un SmartObject o un Controlador Virtual IRC 5*), comportamiento/situación que **no se reflejaría en la realidad**. Este fenómeno/deficiencia **se ha detectado en recurrentes ocasiones** en la implementación de Estaciones de cierto grado de complejidad. *Para **solventar** este **problema** se ha tenido que **recurrir a introducir retardos programados "Delays"**, o **"refrescos" forzados de los sensores**, que en una instalación real no hubieran sido necesarios.*

d) Como se ha comentado anteriormente, RobotStudio no define cómo gestiona la concurrencia virtual entre los elementos que componen la Estación, manteniendo como **único parámetro de simulación configurable** por el usuario el "***Step o StepTime***".
Ver ANEXO III: **Ficha de Ruta: "Cómo Configurar y Medir el Tiempo en Simulación"** y ANEXO IV: **Consecuencias de la Velocidad de Movimiento junto con la Frecuencia de Actualización de E/S (*Step*)**.
Se ha observado que cuanto mayor es el Step de Simulación, mayor es la penetración de los cuerpos en los procesos de identificación. Hecho que se materializa en la simulación de las operaciones de agarre, la herramienta invade el volumen del objeto pudiendo llegar a no identificarlo por una parada tardía del movimiento de los dedos.

e) Uno de los elementos más comunes en una Instalación Robotizada es el **PLC**, al que se le suele encomendar el papel de **coordinador de las actividades de todos los elementos que intervienen en la Estación** (*tanto Robots como otros elementos auxiliares o de transporte: cintas, utillajes…*). **RobotStudio NO incorpora PLC's entre sus componentes ni permite una implantación fácil de estos**, lo que obliga a tener que "*parchear*" esta deficiencia. Se ha conseguido solventar este problema de dos formas distintas, como se explica en **ANEXO VI**.
Dada la **importancia de los PLC's en las Instalaciones Robotizadas** su **ausencia** puede ser **considerada** como una **carencia particularmente grave**.

f) Por último, se debería añadir que **una práctica común en los productos ABB** de la **división de robótica** es que **los manuales son bastante crípticos y resultan poco intuitivos**, con **información dispersa entre múltiples documentos que hay que manejar simultáneamente**. A lo que hay que sumar que, el contenido y profundidad de los Manuales y de la "Ayuda de Programa", es exactamente el mismo (*las mismas palabras e imágenes*), contenido insuficiente para reproducirlo rápidamente de forma autodidacta.

B) *Acerca de RobotStudio*

Es una herramienta muy potente para el desarrollo de entornos simulados y trabajo directo como herramienta de edición de programas, en robots reales y simulados. Concluyendo:

1) RobotStudio **presenta varias deficiencias y dificulta seriamente el trabajo para llevar a cabo proyectos orientados al análisis y/o estudio de alternativas existentes**. No facilita la reedición de ficheros para poder comparar los cambios entre ellos y poder así analizar y estudiar las distintas alternativas. *Parece que el programa requiere que los **proyectos se encuentren previamente definidos en formato físico (papel),** y este sea utilizado con el fin de visualizar las Estaciones en formato multimedia.*

2) Existen una serie de herramientas que pretenden auxiliar al usuario, que a menudo no resultan tan beneficiosas. Se recomienda encarecidamente **Deshabilitar las "Auto-ayudas"** Dificultan seriamente el trabajo.

3) Hay que contemplar y prever que **el comportamiento será distinto en la simulación respecto del que se encontraría en una situación real**, debido a la problemática en la gestión de E/S en Simulación. Este hecho obliga a contemplar distintas consideraciones en la implementación de programas, entre los Simulados y los implementados en Instalaciones Reales. **Si la Estación se compone de elementos lógicos** interconectados entre sí, requiere y exige la **existencia de "*Delay's*" de duración superior al "*Step*",** para que se pueda **garantizar la comunicación entre los distintos elementos**.

3. Inserción del ABB IRB 120 en una Célula de Fabricación Flexible:

3. 1. Descripción de la Estación de Trabajo

Inserción de un Robot ABB IRB 120 en una célula de mecanizado de piezas, **encargado de 2 operaciones distintas**: abastecimiento de piezas y desbarbado de piezas mecanizadas.

La **Estación** se encuentra **controlada** por un **PLC Maestro (M 340 de MODICON)** en el que se ha implementado un **Grafcet/SFC en Unity Pro XL**, encargado de comandar:

- Un Robot ABB IRB 120
- Una maqueta o **máquina taladradora FISCHERTECHNIK** con posicionador circular de 4 piezas para la celda de mecanizado

El **objetivo abordado en el TFG** se ha **restringido a la inclusión y programación del Robot en la célula existente**. Para ello, se ha provisto al alumno de la maqueta completamente funcional con el SFC implementado y la instalación cableada.

A) Descripción de la célula

Se puede **resumir por medio del ciclo de vida de las piezas a procesar**: el robot abastece de piezas a la maqueta (*Posición A*), seguidamente la maqueta gira ¼ de vuelta la base y la máquina de taladrar opera sobre las piezas depositadas previamente por el robot (*Posición B*). A continuación, la maqueta gira ¼ de vuelta la base y el robot operará sobre las piezas mecanizadas/taladradas (*Posición C*). Finalmente, la maqueta gira ¼ de vuelta la base para que las piezas tratadas puedan ser extraídas de la Estación, por caída de gravedad a un contenedor (*Posición D*). En otras palabras, ver:

- **Figure 13: Ciclo de Vida de la Pieza en Régimen Permanente: Célula Real**
- **Figure 14: Secuencia Programación de Régimen Permanente: Célula Real**.
- **Figure 32: Descripción de Posiciones de Operación de la Maqueta: Célula Real.**

La **planificación de operaciones de la célula contempla el paralelizado de procesos o concurrencia** entre las operaciones del taladrado junto con las 2 operaciones realizadas por el robot (aprovisionamiento y desbarbado).

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la Célula de Trabajo Completa: [Fischertechnik_Machine_Demostration_Running].

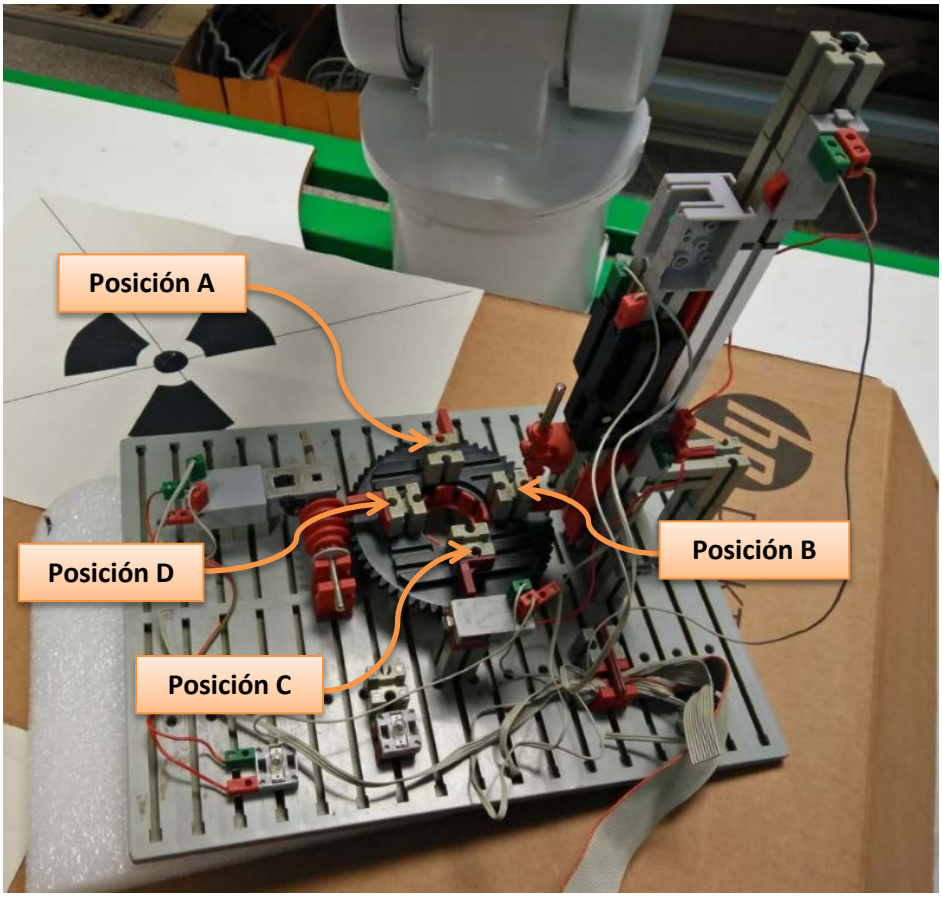


Figure 32: Descripción de Posiciones de Operación de la Maqueta: Célula Real

B) Maqueta de máquina de taladrado FISCHERTECHNIK

Compuesta por:

- **Una Plataforma Giratoria:** accionada desde un **motor de corriente continua acoplado a un reductor**.
- **Un Taladro:** que puede **desplazarse verticalmente** a lo largo de una guía, accionado por un motor-reductor propio.
- **Varios Posicionadores:**
  - **La Base: cuenta con cuatro posiciones** de taladrado desfasadas 90º, y **detectables mediante** el correspondiente **micro-ruptor**.
  - **El Taladro:** cuenta con **micro-ruptores** que **actúan como final de carrera** e informan de la presencia del mismo en los límites superior e inferior.

La maqueta se encuentra controlada por **autómata programable M 340 de MODICON**, comunicada por el cableado preformado para su rápida conexión a las E/S con el autómata.

3. 2. Descripción de la Herramienta

Se ha dotado al Robot de una **herramienta** lo más **versátil** posible para poder abordar un amplio abanico de **tareas**, tanto de **posicionamiento y seguimiento de trayectorias** como de **manipulación y agarre de objetos** existentes.

La pinza real se encuentra instalada en el Robot IRB 120 del laboratorio L0.06 del DIIS de la EINA.

Partiendo de una **Base neumática de la empresa SMC "Modelo MHL2_ES-10"**, **ver Figure 34,** se han instalado **2 dispositivos, que hacen la vez de "dedos"** de la garra, **y** un **orificio roscado provisto de guías mecánicas** de precisión, ubicado en el centro geométrico de la base, sobre el que se puede **acoplar y desacoplar un puntero metálico** acabado en una **punta de 10µm**.

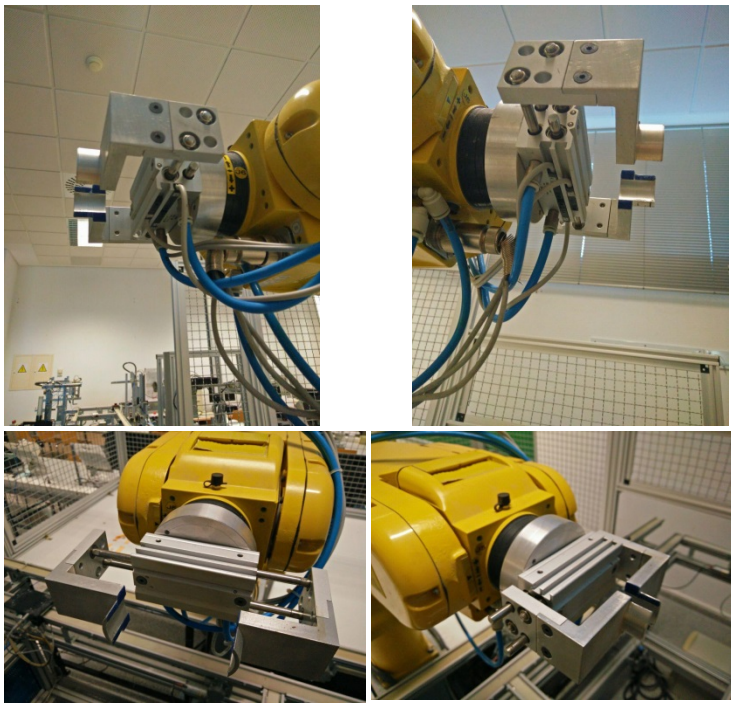


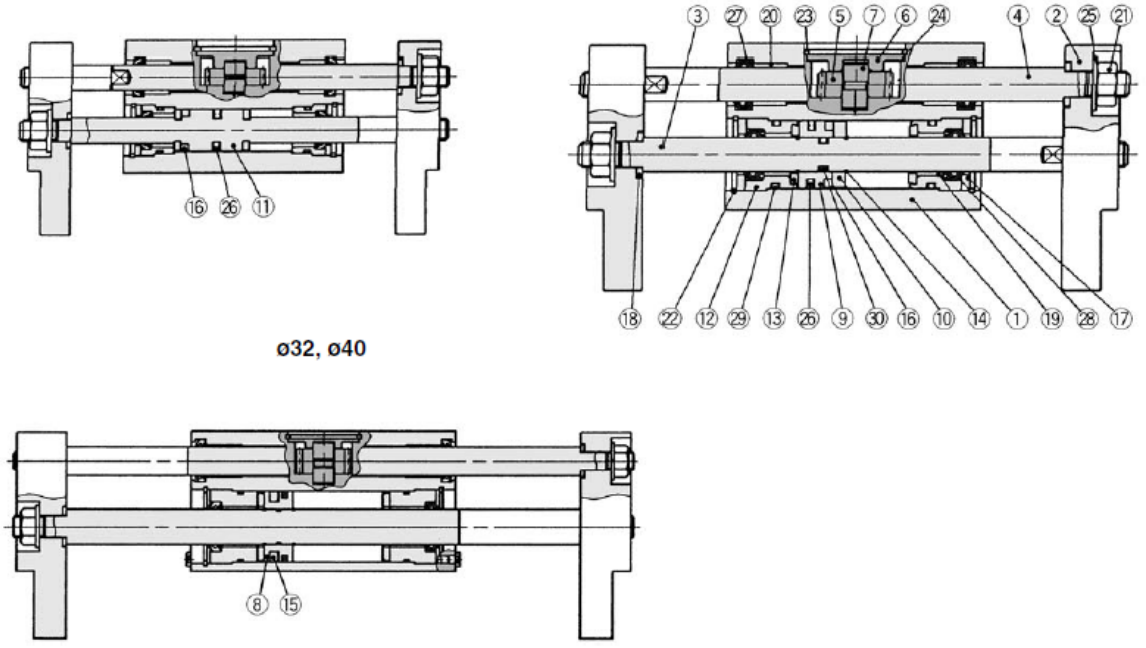
Figure 33: Herramienta Garra Real: Base neumática de la empresa SMC "Modelo MHL2_ES-10"

La **distribución de masas**:

- **1 kg para la pinza completa**, base de la garra y dedos
- **100gr** para el **puntero** desacoplable

Para la configuración del puntero se deberá cerrar la pinza y garantizar así la verticalidad y una mayor resistencia a flexión del puntero.

Es responsabilidad del usuario asegurar el correcto posicionamiento del puntero, así como garantizar las condiciones de seguridad en la manipulación del puntero y del robot.



Lista de componentes

Nº	Designación	Materiales	Observaciones
①	Cuerpo	Aleación de aluminio	Anodizado
②	Dedos	Aleación de aluminio	Anodizado
③	Vástago	Acero inoxidable	
④	Cremallera	Acero inoxidable	
⑤	Piñón	Acero al carbono	
⑥	Cubierta piñón	Acero al carbono	Niquelado electrolítico
⑦	Eje piñón	Acero inoxidable	Nitrurado
⑧	Émbolo	Latón	
⑨	Émbolo A	Latón	
⑩	Piston B	Latón	
⑪	Émbolo A	Acero inoxidable	
⑫	Tapa	Aleación de aluminio	Cromado
⑬	Amortiguador	Caucho uretano	
⑭	Clip	Acero inoxidable para muelles	
⑮	Imán	Goma sintética	

Nº	Designación	Materiales	Observaciones
⑯	Imán	Material magnético	Niquelado
⑰	Cubierta vástago B	Acero laminado frío	Niquelado electrolítico
⑱	Arandela	Acero inoxidable	Nitrurado
⑲	Casquillo	Metal lubricado	
⑳	Casquillo	Metal lubricado	
㉑	Tuerca U	Acero al carbono	Niquelado
㉒	Anillo de cierre R	Acero al carbono	Niquelado
㉓	Anillo de cierre C	Acero al carbono	Niquelado
㉔	Arandela	Acero para muelle	Revestimiento fosfato
㉕	Arandela	Acero al carbono	Niquelado

Figure 34: Cuerpo de la Pinza MHL2-10

Se han incluido las características técnicas de la pinza mecánica, así como de los indicadores de limitaciones de carga máxima, ver el DataSheet completo de la pinza mecánica:

- Ver: **DataSheet_MHL2_ES**

3. 3. Aplicación en el PLC

Se ha provisto al alumno titular del TFG de acceso a un autómata o PLC-Maestro, "MODICON: M 340", y del programa completo.

El programa consta de **3 partes principales**:

- **Transitorio de Arranque**: Hasta que existan piezas en todas las posiciones de la plataforma rotativa (*3 posiciones*)
- **Régimen Permanente**: *Loop* u Operación de Producción Continua
- **Transitorio de Parada**: Hasta que se extraigan todas las piezas existentes en la plataforma rotativa por la posición de caída por gravedad.

La **planificación de operaciones de la célula contempla el paralelizado de procesos o concurrencia entre sus elementos**: accionamiento simultáneo del taladrado y el robot (*aprovisionamiento y desbarbado*).

Ver **Figure 14: Secuencia Programación de Régimen Permanente: Célula Real**

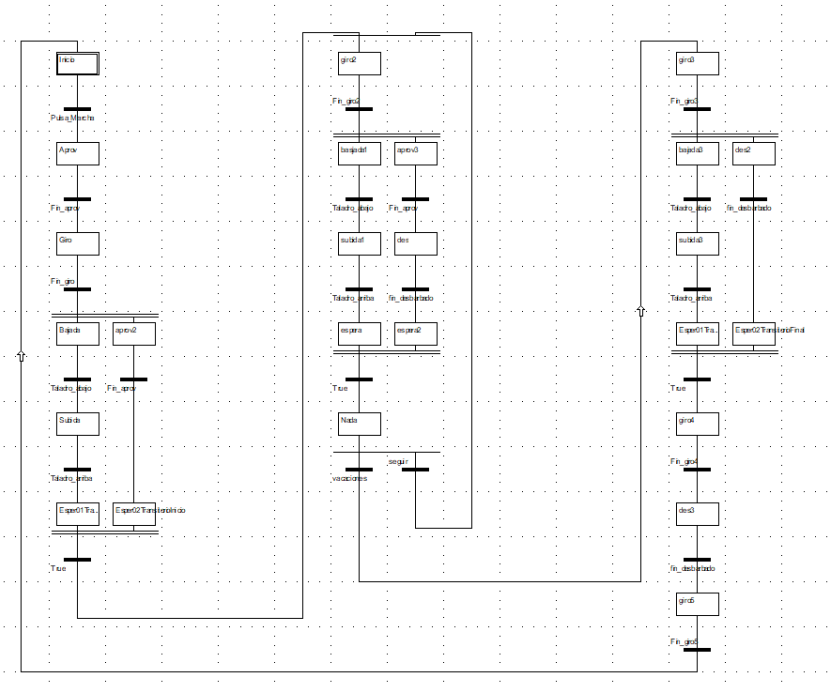


Figure 35: Aplicación en el autómata "MODICON: M 340": Implementación de un PLC

El régimen permanente se resume por medio del ciclo de vida de las piezas a procesar:

1. Abastecimiento de Piezas por el Robot
2. La máquina de taladrar opera sobre las piezas depositadas sobre la base móvil
3. El robot operará sobre las piezas ya mecanizadas
4. Finalmente, las piezas tratadas son extraídas de la Estación por caída libre

Ver **Figure 13: Ciclo de Vida de la Pieza en Régimen Permanente: Célula Real**

3. 3. 1. Programa Implementado en el Autómata

A) *Transitorio de Arranque*

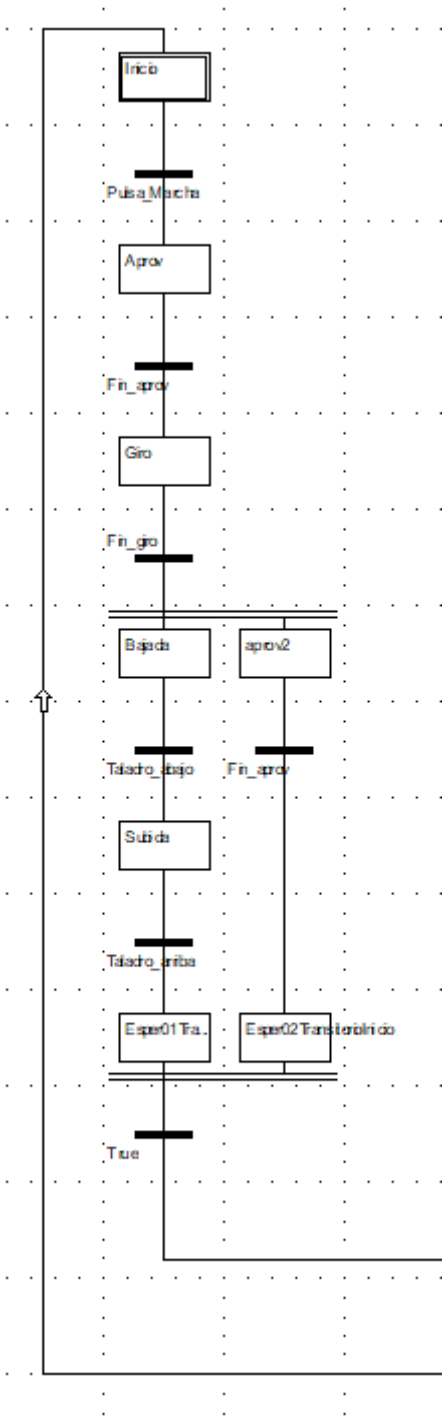


Figure 36: Programa Implementado en el PLC Maestro de la Célula Real: Transitorio de Arranque

El **programa se inicia** con el transitorio de arranque y/o de aprovisionamiento.

El **PLC Maestro "secuestra" el recurso del robot** hasta que finalice todas las operaciones/tareas solicitadas por el PLC y/o salte una emergencia.

Comienzo del Programa: El IRB 120 deposita una pieza en la Posición A de la base del taladro, ver **Figure 32: Descripción de Posiciones de Operación de la Maqueta: Célula Real**.

Seguidamente, la base girará 90º posicionando la pieza provista por el Robot en la Posición B.

A continuación, **el taladro y el Robot IRB 120 operarán en paralelo**, el taladro mecanizará la pieza ubicada en Posición B mientras el Robot deposita una nueva pieza en la maqueta, Posición A.

Finaliza el Transitorio de Arranque y da paso al Régimen Permanente.

B) *Régimen Permanente*

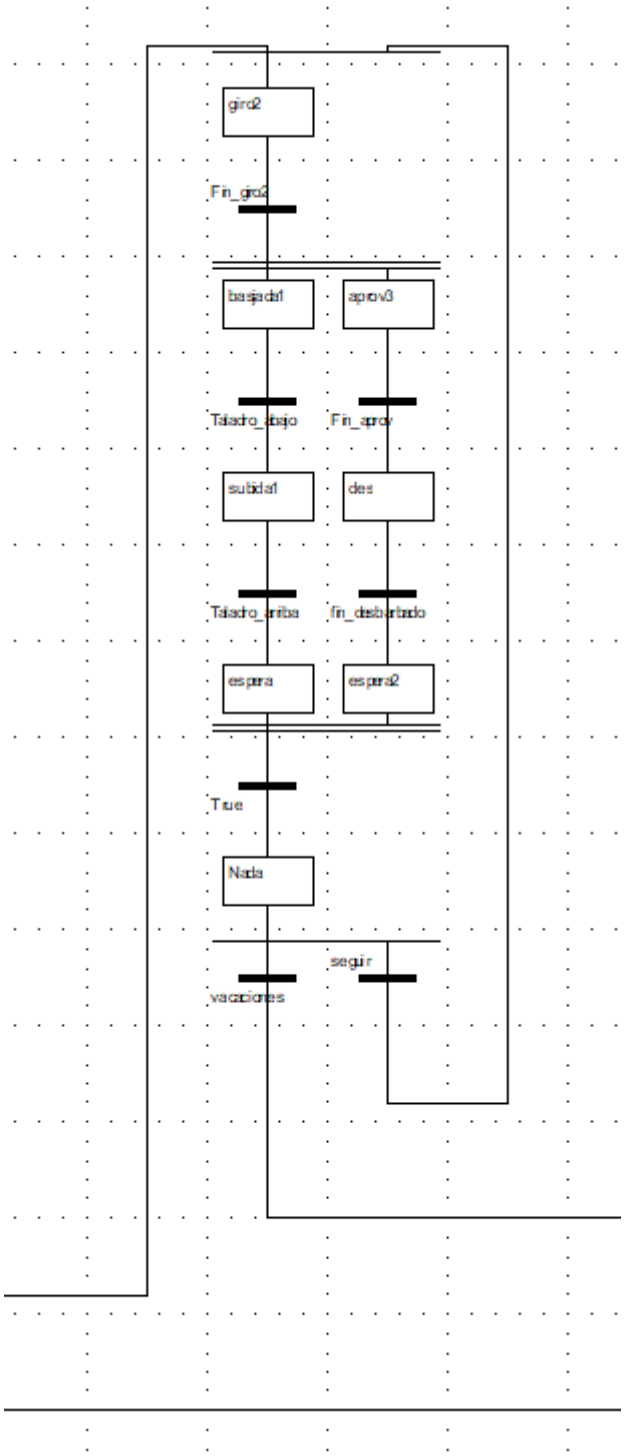


Figure 37: Programa Implementado en el PLC Maestro de la Célula Real: Régimen Permanente

El **Régimen Permanente se inicia** con un **posicionamiento/giro de 90º** de la base, y las **posiciones B y C ocupadas** con piezas.

El **programa** se ejecuta **con concurrencia entre el robot y el taladro**. Tanto el ciclo completo de mecanizado, sobre la Posición B, como las operaciones del robot, abastecimiento en la Posición A y desbarbado en la Posición C, se ejecutan simultáneamente.

A continuación, la base rota 90º y vuelve a repetir las operaciones del taladro y el Robot.

Esta fase permanecerá en ejecución cíclica hasta que el usuario le comunique al PLC el fin de programa, momento en el que acabará de ejecutar el ciclo activo y pasará al régimen transitorio de parada.

C) *Transitorio de Parada*

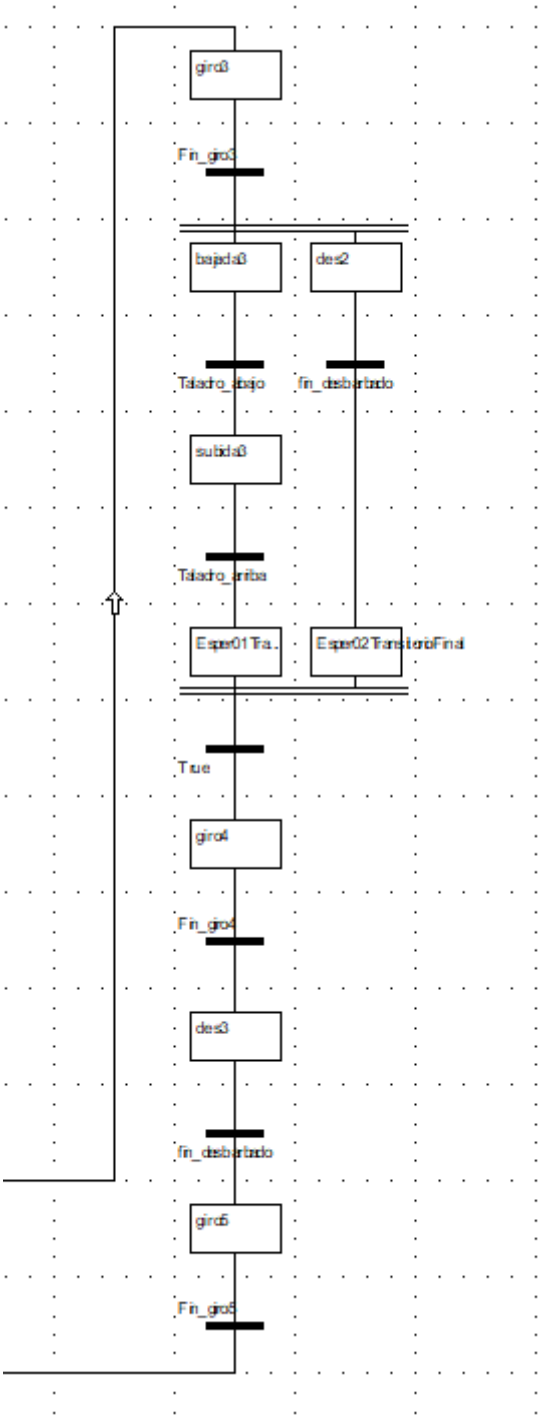


Figure 38: Programa Implementado en el PLC Maestro de la Célula Real: Transitorio de Parada

El Robot dejará de aprovisionar piezas a la maqueta y **restringirá sus operaciones a desbarbar las piezas mecanizadas**.

Tras posicionar las piezas con un nuevo giro de la base, **se ejecutará un ciclo de concurrencia entre la última mecanización del taladro y el desbarbado del Robot**.

A continuación, volverá a girar la base y ubicará la última pieza en la Posición C para que el Robot la pueda desbarbar y ser liberado de las tareas de la célula de trabajo.

D) *Interrupción de Fallo de la Célula*

El autómata "MODICON: M 340" o **PLC Maestro** permite **programar** situaciones singulares, basadas en **interrupciones, activación directa de las entradas del autómata por los usuarios.**

Se ha implementado **un tratamiento de paro de seguridad con rearme manual**, tanto para el PLC cómo para el **Robot IRB 120, quien interrumpirá la acción ejecutada y permanecerá quieto hasta nueva orden**.

Una vez solucionado el problema:

- **El PLC se reiniciará** ubicando el puntero del programa en el estado inicial.
- **El Robot IRB 120 se reposicionará en una configuración segura** (*Home Position*).

a) Código del PLC Maestro

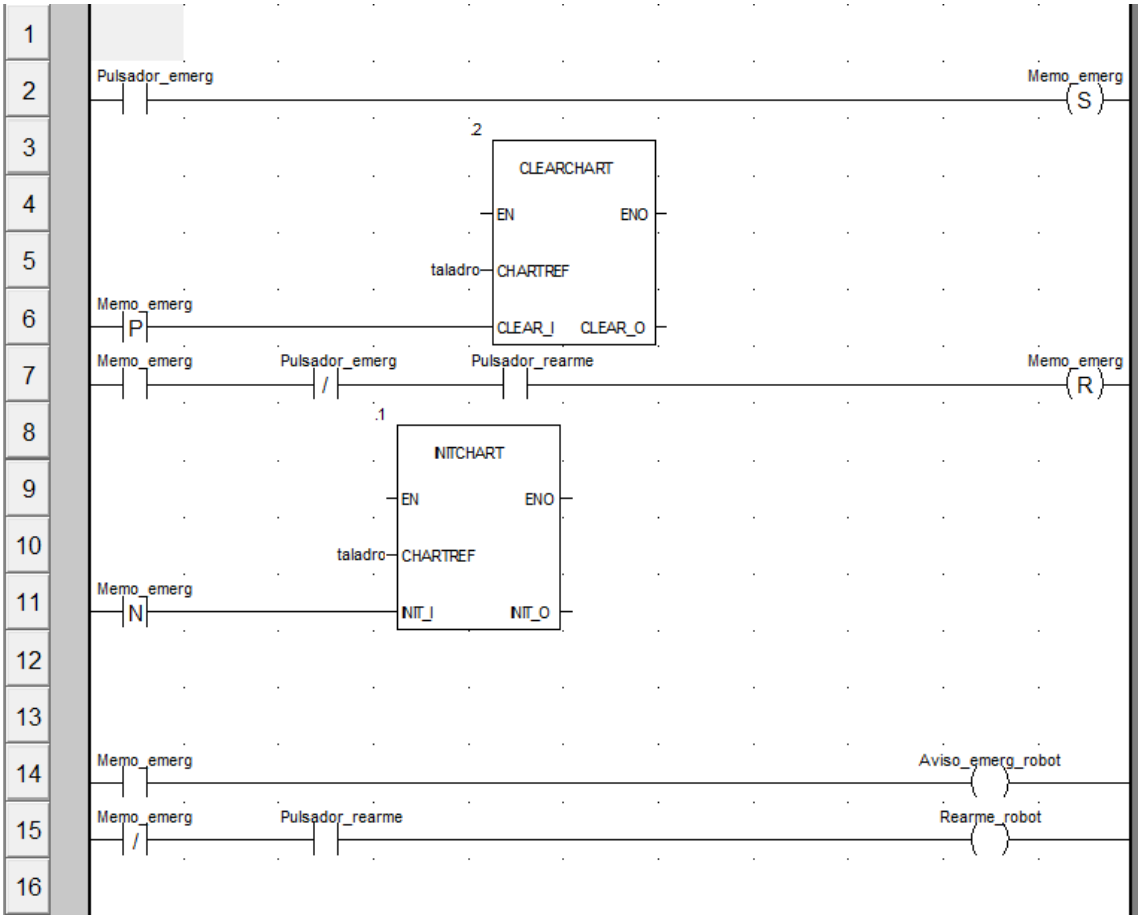


Figure 39: Código de Interrupción-Fallo del PLC Maestro: Célula Real

3. 3. 2. Mapeado de E/S y Cableado

Tabla 5: Mapa de Entradas y Cableado del Robot

Salida Autómata	Entrada Robot	Descripción	Borne cableado Robot	Color Conductor Cableado Robot	Color Conductor Autómata	Borne Cableado Autómata
Q0.1.16	DI9	Orden al Robot de Aprovisionamiento	XS13-pin 1	Azul	Marrón	100
Q0.1.17	DI10	Orden al Robot Desbarbado	XS13-pin 2	Amarillo	Marrón	101
Q0.1.18	DI11	Orden al Robot de Fin Aplicación	XS13-pin 3	Marrón	Marrón	102
Q0.1.19	DI12	Orden al Robot de Emergencia	XS13-pin 4	Gris	Marrón	103
Q0.1.20	DI13	Orden al Robot de Rearme	XS13-pin 5	Rojo	Marrón	104
		Masa	XS13-pin 9	Malla	Negro	0V

Tabla 6: Mapeado de Salidas y Cableado del Robot

Entrada Autómata	Salida Robot	Descripción	Borne cableado Robot	Color Conductor Cableado Robot	Color Conductor Autómata	Borne Cableado Autómata
I0.1.4	DO9	Notificación fin Desbarbado	XS15-pin 1	Rosa	Azul	104
I0.1.5	DO10	Notificación fin Aprovisionamiento	XS15-pin 2	Violeta	Azul	105
I0.1.6	DO11	Notificación Robot Preparado	XS15-pin 3	Blanco	Azul	106
I0.1.7	DO12	Notificación Robot en Emergencia	XS15-pin 4	Verde	Azul	107
		Masa	XS15-pin 9	Negro	Negro	215

Cada línea de E/S se corresponde con un único terminal ubicado en la parte posterior del armario del Controlador del Robot (IRC-5).

Por motivos de operatividad, las líneas de E/S se han extendido de manera transparente hasta las regletas situadas en la parte inferior del bastidor del autómata, **Figure 23**.

3. 3. 3. Jerarquía de control

En las instalaciones industriales convencionales, la maquinaria encargada de mecanizar poseería su propio elemento de control (*típicamente un PLC*), además del PLC Maestro encargado de la coordinación de actividades. A pesar de ello, **dada la sencillez de la célula de trabajo** sobre la que se ha trabajado **y dado que el PLC "MODICON: M 340"** del que se ha dispuesto **posee capacidad suficiente**, se ha decidido **centralizar todas las operaciones** directamente sobre él. Al cual se le ha encomendado el doble papel de:

1. **Elemento de Control de la máquina** de mecanizar
2. **Elemento Coordinador de actividades** de la célula

La jerarquía resultante de la Célula Flexible se resumiría en el siguiente esquema:

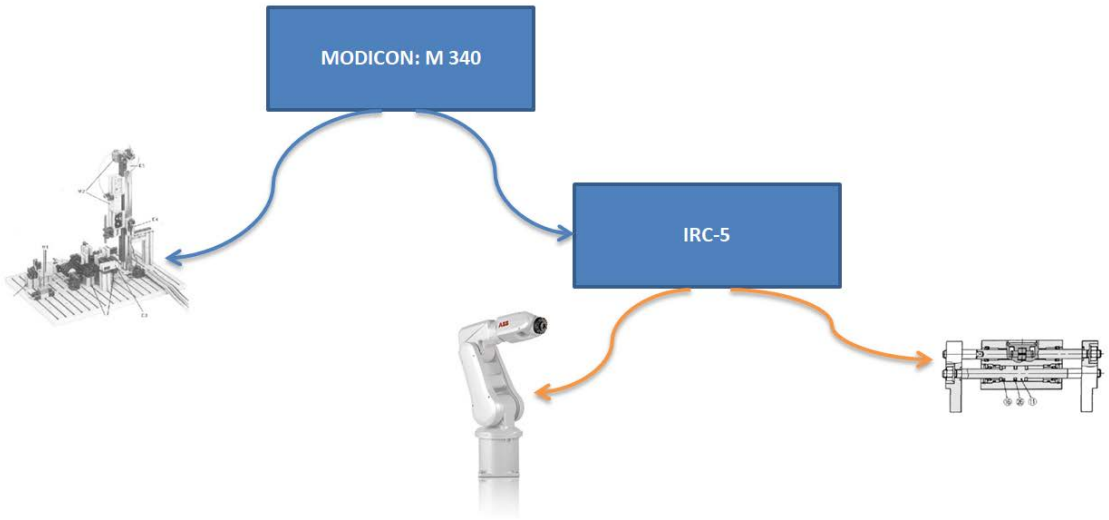


Figure 40: Jerarquía Control: Célula Real

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la Célula de Trabajo Completa: **Fischertechnik_Machine_Demostration_Running**.

3. 4. Aplicación RAPID en el Robot

El **Robot ABB IRB 120** se encuentra **integrado en una célula de trabajo de localización y orientación variable**.

Se ha considerado que **el robot posee una localización absoluta** e invariante en posición y orientación, y **opera como un esclavo perfecto comandado por un PLC** externo, restringidas sus funciones a **2 únicas tareas**:

- Abastecimiento de Piezas en un Punto A
- Desbarbado de Piezas tratadas en un Punto B

Dada la simplicidad de las Tareas que se le requieren al Robot, y aprovechando la versatilidad de RAPID para alcanzar las localizaciones, se han implementado distintas formas de abordarlas, tomando como **referencia principal (*WorkObject*[21]) la Base de la Maqueta y manteniendo una estructura de Jerarquía de Transformaciones de 4 Niveles**, como se muestra en **Figure 42**.

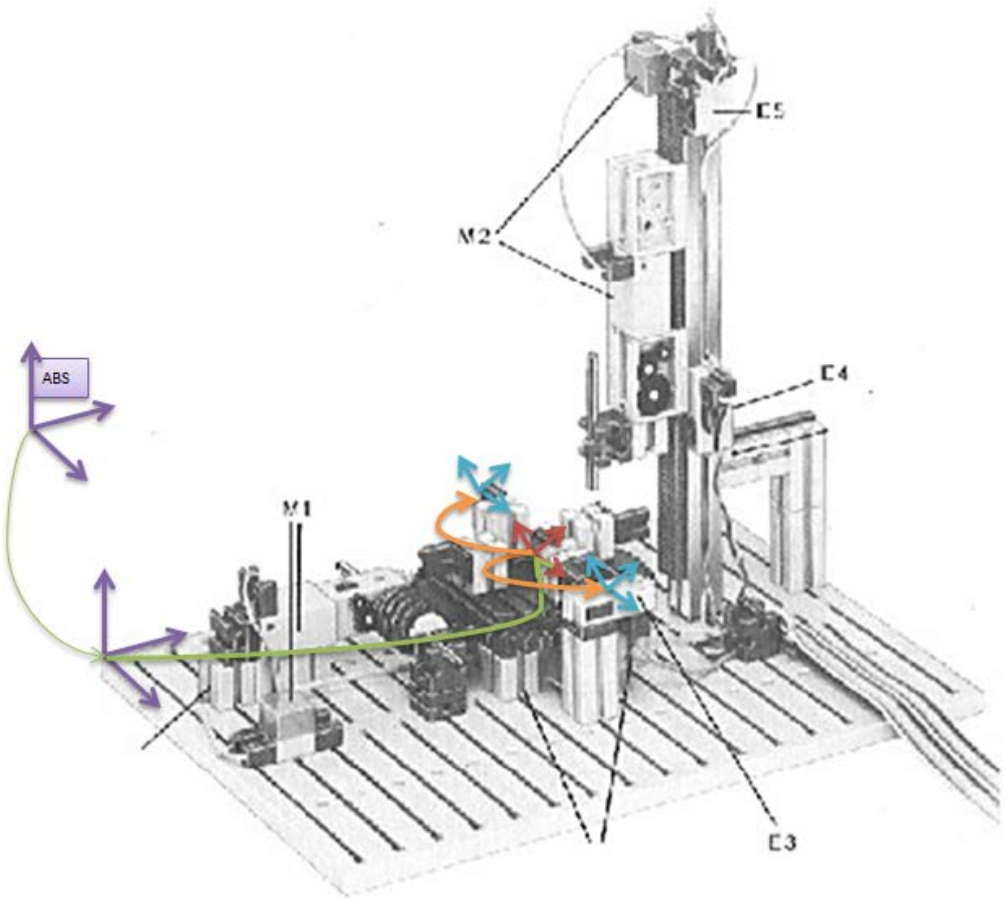


Figure 41: Jerarquía de Localizaciones de la Maqueta: Célula Real

[21] **WorkObject**: Referencia principal de un plano de trabajo específico en el lenguaje RAPID

3. 4. 1. Elección de Jerarquía de Niveles

A) *Jerarquía de Transformaciones Seleccionada: "Estructura de 4 Niveles":*

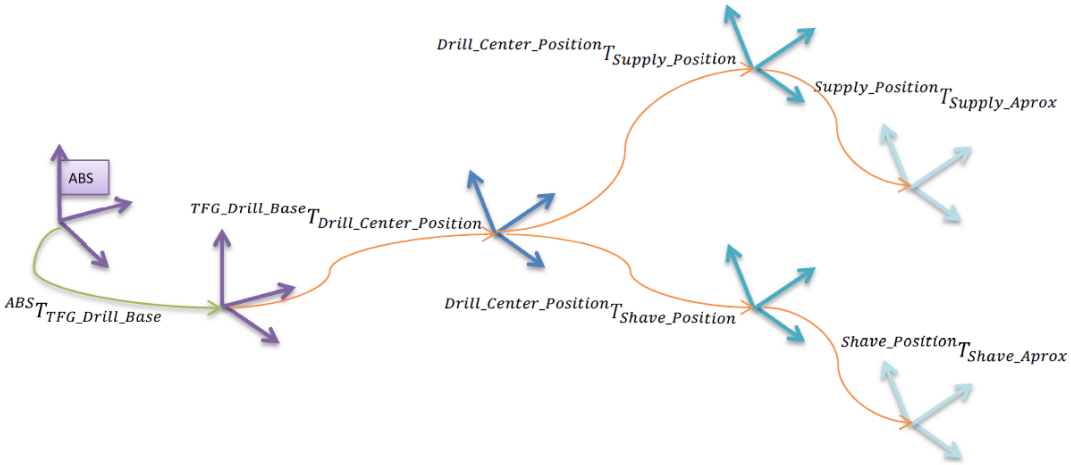


Figure 42: Relativización de Localizaciones: 4 Niveles: Célula Real

B) *Estructura de 3 Niveles (Desaconsejada):*

Si se cuenta con un plano de trabajo, en el que se detalla la localización de todos los puntos ("***Targets***[22]"), se pueden referenciar todos los *Targets* al *WorkObject* de la Base de la Maqueta.

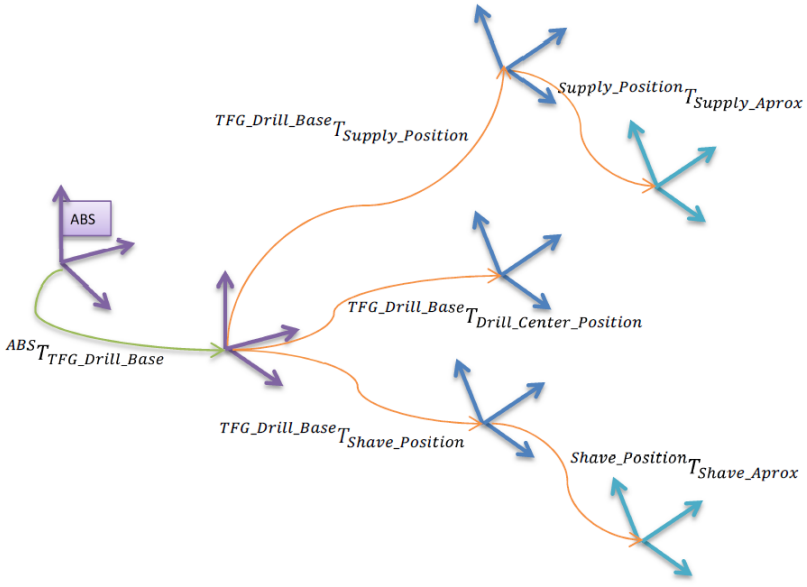


Figure 43: Relativización de Localizaciones: 3 Niveles: Célula Real

Se desaconseja esta opción. Al considerar todos los puntos con el mismo nivel jerárquico, cualquier modificación futura del conjunto exige estudiar y/o aplicar correcciones para todos los puntos referenciados a la base.

[22] **Targets:** "Punto Objetivo" en RobotStudio, definido con Posición y Orientación relativos a la Frame especificada.

Para todas ellas, **el usuario únicamente debe proporcionarle al controlador IRC-5 la información** de ubicación y orientación **del *WorkObject* de la Base de la Maqueta** desde el *FlexSpendant,* a través del Método de los 3 Puntos.

3. 4. 2. Análisis crítico de las posibilidades de relativización de localizaciones que ofrece RAPID.

Entre las distintas posibilidades que existen y/o permiten definir una tarea en RAPID, se ha optado por la relativización:

1. Basada en herramientas de definición de traslaciones (RelTool)
2. Basada en la estructuración de la escena proporcionada por ABB (WObject y Offs)
3. Basada en desplazamiento de programa (PDispSet)
4. Basada en composición explícita de TRF (PoseMult)

Para facilitar una comparativa homogénea, se ha intentado mantener la misma estructura de programación para todas las funciones implementadas. Se facilitan, a modo de ejemplo, los procedimientos de alcanzabilidad del Punto de Abastecimiento (*Supply_Position*).

Basada en herramientas de definición de traslaciones (RelTool)

A) Propiedades de RelTool (Relative Tool):

a) Generalidades:

La función RelTool **se utiliza para añadir un desplazamiento y/o rotación, expresada en el sistema de coordenadas de la herramienta activa**, a una posición del Robot.

Tipo de Dato: *robtarget*.

Permite definir de forma compacta una transformación en coordenadas relativas a un *Target* sin perder la Base de Coordenadas del Sistema de Referencia.

b) Principal contra:

Se oculta la composición de Transformaciones, quedando enmascarada a los usuarios que consulten el código programado.

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la Herramienta **RelTool**.

B) Estructura de 4 Niveles:

```
MODULE RelTool_Function

    !CONFIGURATION DATA::
    !Lebel Reference01:     !wobjdata TFG_Drill_Base: It's a date that the user need to upgrade with the FlexPendant
    TASK PERS wobjdata TFG_Drill_Base:=[FALSE,TRUE,"",[[414.666,-66.9352,281.176],[0.984505,-0.0722476,-0.0116787,0.159357]],[[0,0,0],[1,0,0,0]]];   !New
    !Constant Data: Dont Delete This:
    CONST extjoint Extern_Axis:=[9E9,9E9,9E9,9E9,9E9,9E9];        !Put 9E9 in all term if the Task haven't any extern axis
    CONST confdata Robot_Configuration:=[0,0,0,0];
    CONST jointtarget Joint_Home:=[[0,0,0,0,0,0],Extern_Axis];    !It's define the home position, it's a common place for all tasks
    !The user need to define the quatern terms: q1:q4
    CONST orient RotationXangle30:=[0.96592582,0.25881904,0,0]; !Common Reference !Rot(x,30)
    !ALL THIS BLOCK USE: wobjdata TFG_Drill_Base: RELATIVE LOCATION:
                            !Lebel Reference: 02: Relative to TFG_Drill_Base
    CONST robtarget Drill_Center_Possition:=[[77,133,50],RotationXangle30,Robot_Configuration,Extern_Axis];     !Point00 Absolute: Drill Center Position: Lebel 02
                            !Lebel Reference: 03: Relative to Drill_Center_Position
    PERS robtarget Supply_RelativeToDrillCenter;                                                              !Point01 Relative: Lebel 03
    PERS robtarget Shave_RelativeToDrillCenter;                                                               !Point02 Relative: Lebel 03


    PROC Relative_Supply_RelPosition()
        !Define Relative Point: Lebel Reference: 03   !Point01RelativeToDrillCenter: Supply_Piece_Position: Work Reference  !Rot(x,30)
        Supply_RelativeToDrillCenter:=RelTool(Drill_Center_Possition,-23,0,5);

        !Now the Robot try to Reach the wished position:
        !Move Composition to aproximity: TypeMove RelTool(robtarget NameReference, offset x,offset y, offset z), speed, proximity, NameTool\WObj:=wobjdata NameReference;
        MoveJ RelTool(Supply_RelativeToDrillCenter,0,0,45),v500,z10,Puntero_Celula\WObj:=TFG_Drill_Base;        !Move to Aproximation Supply Position: Lebel 04
        MoveL Supply_RelativeToDrillCenter,v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;                       !Simple Lineal Move Order until wished position: Lebel 03
        Action_SupplyPiece_Pos01;                                                                               !Action Program in Supply Position
        MoveL RelTool(Supply_RelativeToDrillCenter,0,0,45),v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;       !Move to Aproximation Supply Position: Lebel 04
    ENDPROC

ENDMODULE
```

Figure 44: Código Relativización de Localizaciones Función RelTool: 4 Niveles: Célula Real

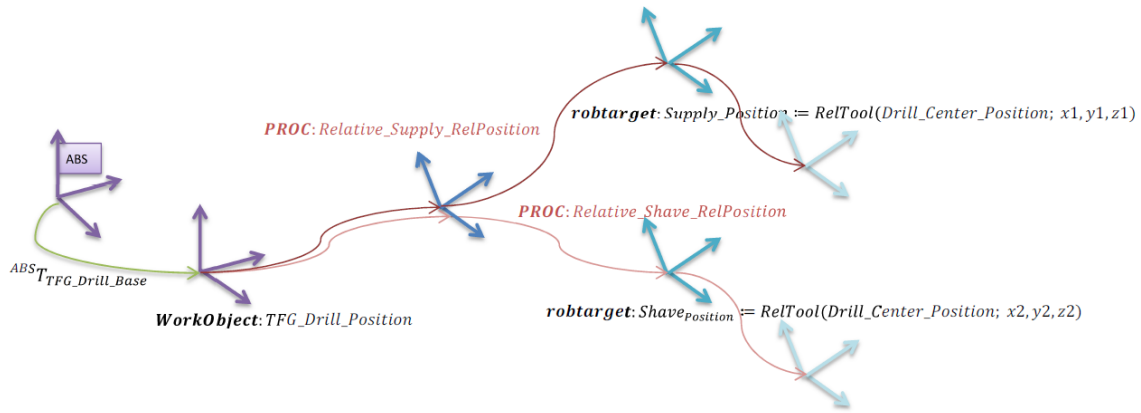


Figure 45: Relativización de Localizaciones Función RelTool: 4 Niveles: Célula Real

a) ¿Cumple las Expectativas?:

Cumple las expectativas.

C) *ATENCIÓN: Estructura de 3 Niveles:*

```
MODULE RelTool_Function

    !CONFIGURATION DATA::
    !Lebel Reference01:     !wobjdata TFG_Drill_Base: It's a date that the user need to upgrade with the FlexPendant
    TASK PERS wobjdata TFG_Drill_Base:=[FALSE,TRUE,"",[[414.666,-66.9352,281.176],[0.984505,-0.0722476,-0.0116787,0.159357]],[[0,0,0],[1,0,0,0]]];   !New
    !Constant Data: Dont Delete This:
    CONST extjoint Extern_Axis:=[9E9,9E9,9E9,9E9,9E9,9E9];        !Put 9E9 in all term if the Task haven't any extern axis
    CONST confdata Robot_Configuration:=[0,0,0,0];
    CONST jointtarget Joint_Home:=[[0,0,0,0,0,0],Extern_Axis];    !It's define the home position, it's a common place for all tasks
    !The user need to define the quatern terms: q1:q4
    CONST orient RotationXangle30:=[0.96592582,0.25881904,0,0]; !Common Reference !Rot(x,30)
    !ALL THIS BLOCK USE: wobjdata TFG_Drill_Base: RELATIVE LOCATION:
                            !Lebel Reference: 02: Relative to TFG_Drill_Base
    CONST robtarget Drill_Center_Possition:=[[77,133,50],RotationXangle30,Robot_Configuration,Extern_Axis];     !Point00 Absolute: Drill Center Position: Lebel 02
                            !Lebel Reference: 03: Relative to Drill_Center_Position
    PERS robtarget Supply_RelativeToDrillCenter;                                                              !Point01 Relative: Lebel 03
    PERS robtarget Shave_RelativeToDrillCenter;                                                               !Point02 Relative: Lebel 03


    PROC Relative_Supply_RelPosition()
        !Now the Robot try to Reach the wished position:
        !Move Composition to aproximity: TypeMove RelTool(robtarget NameReference, offset x,offset y, offset z), speed, proximity, NameTool\WObj:=wobjdata NameReference;
        MoveJ RelTool(Drill_Center_Possition,-23,0,50),v500,z10,Puntero_Celula\WObj:=TFG_Drill_Base;           !Move to Aproximation Supply Position: Lebel 03
        MoveL RelTool(Drill_Center_Possition,-23,0,5),v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;           !Simple Lineal Move Order until wished position: Lebel 03
        Action_SupplyPiece_Pos01;                                                                             !Action Program in Supply Position
        MoveL RelTool(Drill_Center_Possition,-23,0,50),v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;          !Move to Aproximation Supply Position: Lebel 03
    ENDPROC

ENDMODULE
```

Figure 46: Código Relativización de Localizaciones Función RelTool: Transformaciones 3 Niveles: Célula Real

Lo que aparentemente es un cambio mínimo, **eliminar la "etiqueta" del *Target*** y hacer la composición de transformaciones directamente en el interior de la instrucción de movimiento, **provoca una alteración importante en el nivel jerárquico** dando como resultado:

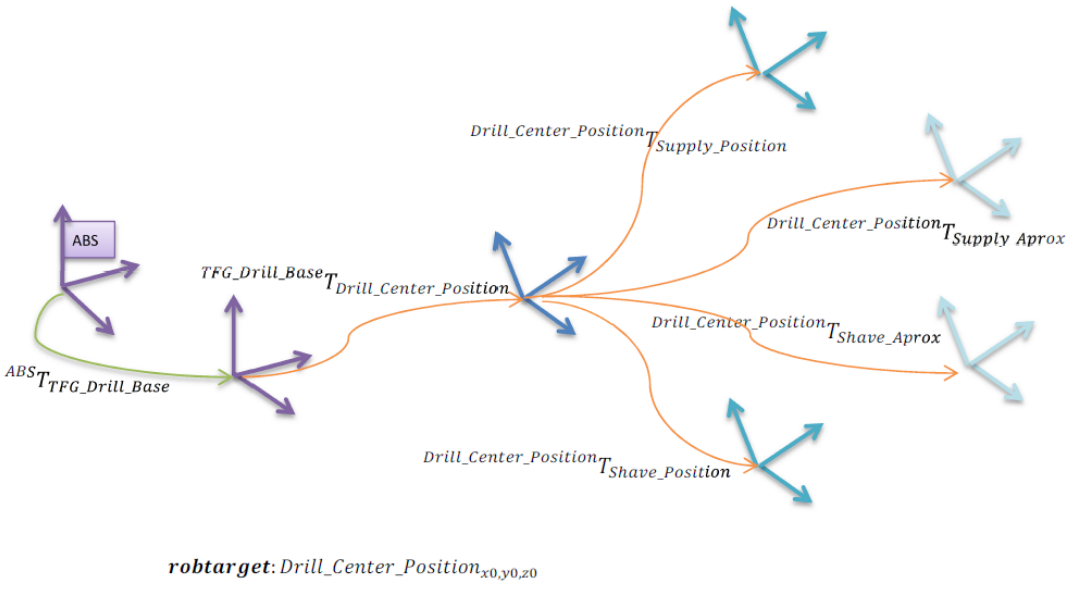


Figure 47: Relativización de Localizaciones: Transformaciones 3 Niveles: Célula Real

Siendo las funciones implementadas:

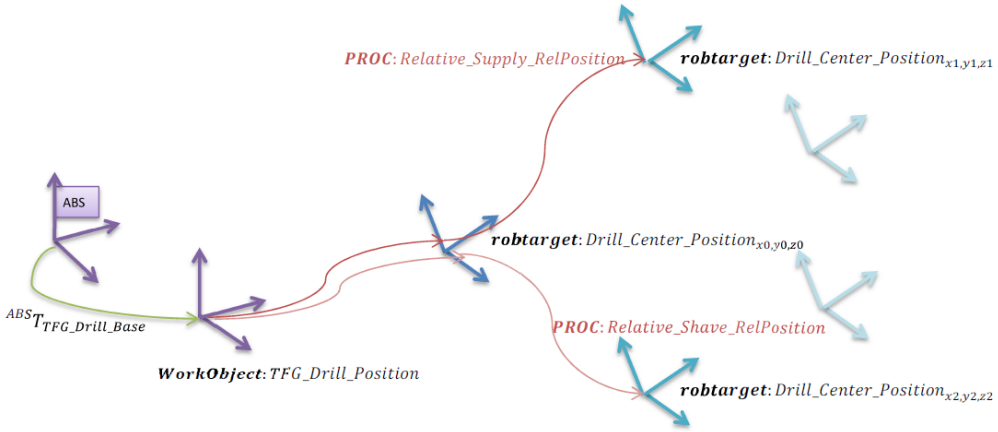


Figure 48: Relativización de Localizaciones Función RelTool: 3 Niveles: Célula Real

a) ¿Cumple las Expectativas?:

Cumple la alcanzabilidad de los *Targets*, pero no cumple el nivel jerárquico. Para todo *Target* dependiente de "*Drill_Center_Position*" corresponde el nivel 3 en la jerarquía.

Basada en la estructuración de la escena proporcionada por ABB (WorkObject y Offs)

A) *Propiedades de Offs (Offset):*

a) Generalidades:

La función *Offs* se utiliza para **añadir un desplazamiento expresado en el sistema de coordenadas del objeto activo a una posición del robot**, desplazamiento referenciado a los ejes del *WorkObject* que pertenece el objeto.

Tipo de Dato: *robtarget*.

ATENCIÓN: No permite definir de forma compacta una transformación en coordenadas relativas a un *Target*. Las traslaciones se hacen respecto de la Base de Coordenadas del Sistema de Referencia (*WorkObject*) *Part*iendo de la posición del TCP en sincronismo con la posición y orientación del *Target*.

Observación: Aprovecha la estructuración de 2 niveles (*user Frame + work object Frame*) aportada por defecto en RAPID.

Es necesario explicitar la referencia utilizada para poder aplicarla en la instrucción de movimiento.

b) Observaciones Propias del Ejemplo:

La **existencia del *Target* como referencia "Nula" sirve para respetar el nivel jerárquico** del problema, aunque pudiera haber sido sustituido directamente por el *Target* "Drill_Center_Position" y utilizado este punto como referencia pura.

Los ejes "X" de la referencia del *WorkObject* y la referencia del *Target* son paralelos. Los ejes "Z" e "Y" se encuentran reorientados +30º sobre el eje "X". Los desplazamientos aplicados sobre el eje "X" serán los mismos tanto mirados desde el *WorkObject* como mirados desde la referencia *Target*, sin embargo, para los otros dos ejes las traslaciones serán distintas.

c) Principales contras:

Se oculta la composición de Transformaciones, queda enmascarada a los usuarios que consulten el código programado.

Requiere de la existencia de un *Target* de referencia sobre el que aplicar los *Offsets*.

No hereda las orientaciones relativas al *Target* de nivel inmediatamente anterior, la traslación se efectúa sobre la referencia *WorkObject*. Si la **referencia del *WorkObject* y la referencia del *Target* se encuentran con una orientación distinta**, sus **ejes (X,Y,Z) no son paralelos entre sí**. Esto puede provocar desplazamientos no deseados por el plano de trabajo y la necesidad de apoyar este procedimiento con cálculos adicionales y/o funciones complementarias.

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la Herramienta **Offs**.

Como se aprecia en el vídeo, la acción de aproximación al punto describe una trayectoria distinta que en los supuestos anteriores, dado que los ejes z no son paralelos. Este efecto no afecta al eje "X" que sí guarda paralelismo entre la referencia *Target* y la referencia *WorkObject*.

B) *Estructura de 4 Niveles:*

```
MODULE WOffs_Function

    !CONFIGURATION DATA::
    !Lebel Reference01:     !wobjdata TFG_Drill_Base: It's a date that the user need to upgrade with the FlexPendant
    TASK PERS wobjdata TFG_Drill_Base:=[FALSE,TRUE,"",[[414.666,-66.9352,281.176],[0.984505,-0.0722476,-0.0116787,0.159357]],[[0,0,0],[1,0,0,0]]];   !New
    !Constant Data: Dont Delete This:
    CONST extjoint Extern_Axis:=[9E9,9E9,9E9,9E9,9E9,9E9];        !Put 9E9 in all term if the Task haven't any extern axis
    CONST confdata Robot_Configuration:=[0,0,0,0];
    CONST jointtarget Joint_Home:=[[0,0,0,0,0,0],Extern_Axis];    !It's define the home position, it's a common place for all tasks
    !The user need to define the quatern terms: q1:q4
    CONST orient RotationXangle30:=[0.96592582,0.25881904,0,0]; !Common Reference !Rot(x,30)
    !ALL THIS BLOCK USE: wobjdata TFG_Drill_Base: RELATIVE LOCATION:
                            !Lebel Reference: 02: Relative to TFG_Drill_Base
    CONST robtarget Drill_Center_Possition:=[[77,133,50],RotationXangle30,Robot_Configuration,Extern_Axis];     !Point00 Absolute: Drill Center Position: Lebel 02
                            !Lebel Reference: 03: Relative to Drill_Center_Position
    PERS robtarget Supply_RelativeToDrillCenter;                                                              !Point01 Relative: Lebel 03
    PERS robtarget Shave_RelativeToDrillCenter;                                                               !Point02 Relative: Lebel 03


    PROC WOffs_Supply_RelPosition()
        !Define Relative Point: Lebel Reference: 03   !Point01RelativeToDrillCenter: Supply_Piece_Position: Work Reference  !Rot(x,30)
        Supply_RelativeToDrillCenter:=Offs(Drill_Center_Possition,-23,0,5);

        !Now the Robot try to Reach the wished position:
        !Move Composition to aproximity: TypeMove RelTool(robtarget NameReference, offset x,offset y, offset z), speed, proximity, NameTool\WObj:=wobjdata NameReference;
        MoveJ Offs(Supply_RelativeToDrillCenter,0,0,45),v500,z10,Puntero_Celula\WObj:=TFG_Drill_Base;          !Move to Aproximation Supply Position: Lebel 04
        MoveL Supply_RelativeToDrillCenter,v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;                      !Simple Lineal Move Order until wished position: Lebel 03
        Action_SupplyPiece_Pos01;                                                                              !Action Program in Supply Position
        MoveL Offs(Supply_RelativeToDrillCenter,0,0,45),v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;         !Move to Aproximation Supply Position: Lebel 04
    ENDPROC

ENDMODULE
```

Figure 49: Código Relativización de Localizaciones Función WOffs: 4 Niveles: Célula Real

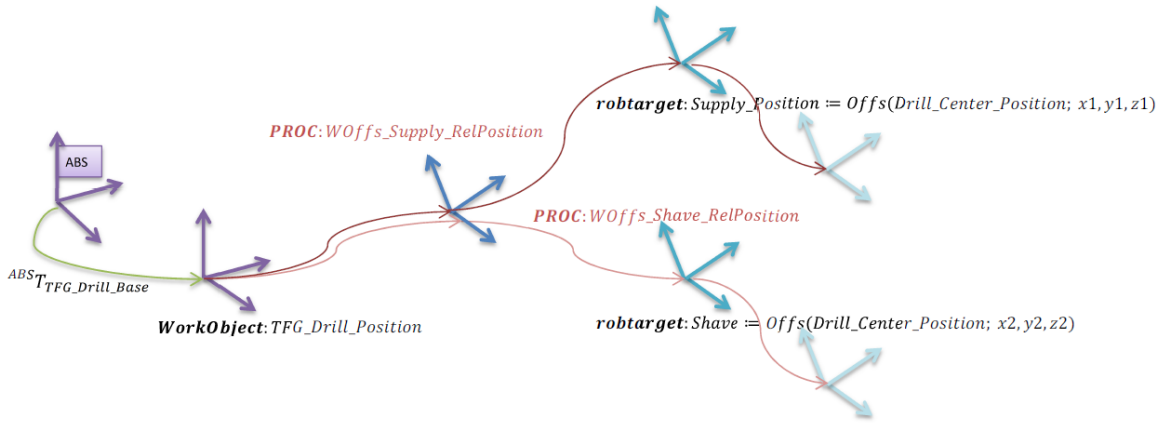


Figure 50: Relativización de Localizaciones Función WOffs: 4 Niveles: Célula Real

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Se ha tomado como referencia común el *Target* "Drill_Center_Possition" en vez de una referencia "Nula", dado que la referencia de este punto es la misma que la referencia del *WorkObject* con una rotación de 30º sobre "X". Todos los desplazamientos desde el Punto son traslaciones en "X" y un par de aproximaciones sobre "Z".

a) ¿Cumple las Expectativas?:

NO cumple la alcanzabilidad de los *Target* en el eje "Z". Es necesario apoyar este método con cálculos y/o funciones complementarias.

C) Alternativa: Reposicionamiento del WorkObject en sincronismo con el Centro del Taladro:

Con el fin de **aplicar todos los *Offsets* desde una referencia común y "Nula" respecto de la referencia *WorkObject***, se ha optado por desplazar el *WorkObject* hasta la posición del centro del taladro "TFG_Drill_Base".

```
MODULE WOffs_Alternative_Function

    !CONFIGURATION DATA::
    !Lebel Reference01:     !wobjdata TFG_Drill_Base: It's a date that the user need to upgrade with the FlexPendant
    TASK PERS wobjdata TFG_Drill_CenterBase:=[FALSE,TRUE,"",[[443.927,89.1351,311.217],[0.984515,-0.0722916,-0.0116478,0.159275]],[[0,0,0],[1,0,0,0]]];   !New
    !Constant Data: Dont Delete This:
    CONST extjoint Extern_Axis:=[9E9,9E9,9E9,9E9,9E9,9E9];          !Put 9E9 in all term if the Task haven't any extern axis
    CONST confdata Robot_Configuration:=[0,0,0,0];
    CONST jointtarget Joint_Home:=[[0,0,0,0,0,0],Extern_Axis];      !It's define the home position, it's a common place for all tasks
    !The user need to define the quatern terms: q1:q4
    CONST orient RotationXangle30:=[0.96592582,0.25881904,0,0];     !Common Reference !Rot(x,30)
    !ALL THIS BLOCK USE: wobjdata TFG_Drill_Base: RELATIVE LOCATION:
                            !Lebel Reference: 02: Relative to TFG_Drill_Base
    !The used Offs methodes need the Specific Reference "robtarget Null_Location_WOffs" how be used by origin of position and rotation by the Tool(TCP)
    CONST robtarget Null_Location_WOffs:= [[0,0,0],RotationXangle30,Robot_Configuration,Extern_Axis];!Point00 Absolute: Drill Center Position: Lebel 02
                            !Lebel Reference: 03: Relative to Drill_Center_Position
    PERS robtarget Supply_RelativeToDrillCenter;                                                    !Point01 Relative: Lebel 03
    PERS robtarget Shave_RelativeToDrillCenter;                                                     !Point02 Relative: Lebel 03


    PROC WOffs_Supply_RelPosition()
        !Define Relative Point: Lebel Reference: 03   !Point01RelativeToDrillCenter: Supply_Piece_Position: Work Reference  !Rot(x,30)
        Supply_RelativeToDrillCenter:=Offs(Null_Location_WOffs,-23,0,5);

        !Now the Robot try to Reach the wished position:
        !Move Composition to aproximity: TypeMove RelTool(robtarget NameReference, offset x,offset y, offset z), speed, proximity, NameTool\WObj:=wobjdata NameReference;
        MoveJ Offs(Supply_RelativeToDrillCenter,0,0,45),v500,z10,Puntero_Celula\WObj:=TFG_Drill_CenterBase;          !Move to Aproximation Supply Position: Lebel 04
        MoveL Supply_RelativeToDrillCenter,v100,fine,Puntero_Celula\WObj:=TFG_Drill_CenterBase;                      !Simple Lineal Move Order until wished position: Lebel 03
        Action_SupplyPiece_Pos01;                                                                                   !Action Program in Supply Position
        MoveL Offs(Supply_RelativeToDrillCenter,0,0,45),v100,fine,Puntero_Celula\WObj:=TFG_Drill_CenterBase;         !Move to Aproximation Supply Position: Lebel 04
    ENDPROC

ENDMODULE
```

Figure 51: Código Relativización de Localizaciones Función WOffs Alternativa: 4 Niveles: Célula Real

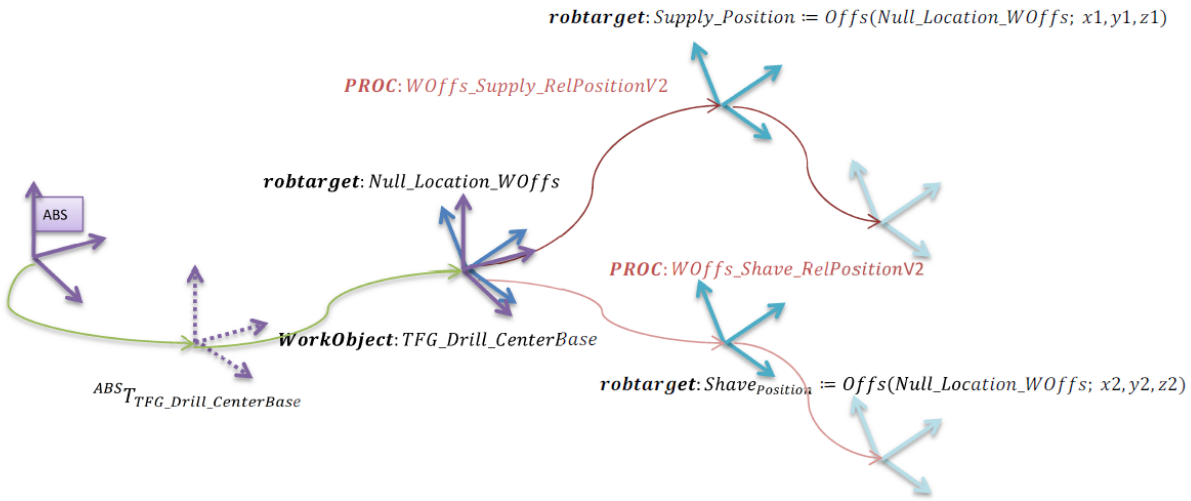


Figure 52: Relativización de Localizaciones Función WOffs Alternativa: 4 Niveles: Célula Real

a) ¿Cumple las Expectativas?:

Mismo resultado que en el punto anterior. **NO cumple la alcanzabilidad de los *Target*** en el eje Z. Es necesario apoyar este método con cálculos y/o funciones complementarias.

b) Observaciones propias de este apartado:

El ***WorkObject* "TFG_Drill_CenterBase" no es el mismo que el de la Base de la Maqueta**, aunque se le ha dotado de un paralelismo exacto en todos los ejes del *WorkObject* "TFG_Drill_Base".

Definir la ubicación y orientación de este *WorkObject* es notablemente más complicado, dada la estructura geométrica de la maqueta.

Ha sido necesario **crear un *Target* como referencia "Nula"**, con una **orientación de +30º sobre "X"**, para evitar la colisión del robot con la maqueta a la hora de alcanzar los *Target*.

Esta estrategia, asumiendo los sacrificios mencionados, **permitiría solucionar el problema de alcanzabilidad** que tiene la solución del apartado anterior. Una posible solución sería dotar de una inclinación +30º sobre el eje "X" directamente sobre el *WorkObject* manteniendo las mismas líneas de código implementado.

Basada en desplazamiento de programa (PDispSet)

A) *Propiedades de PDispSet (Program Displacement Set):*

a) Generalidades:

La **función "*PDispSet*"** se puede usar para **definir y activar un desplazamiento de la referencia *WorkObject,* usando una base de coordenadas conocida procedente de una transformación "pose" de un *Target***.

A pesar de que el "PDispSet" permite definir una composición explícita de transformaciones, respecto de una "pose" especificada por el usuario, el uso que se le pretendió dar desde RobotStudio es radicalmente distinto. **Se recomienda para programas con patrones de movimiento similares entre sí**, con el fin de optimizar y reducir el código implementado.

Tipo de Dato: *pose*

Permite transformar la posición y orientación de la "pose" de un *Target* en una referencia similar a un *WorkObject*, sobre el que poder efectuar traslaciones con *Offset*s puros. **Soluciona las limitaciones de la función "*Offs*"** en trayectorias sobre ejes no paralelos.

Exige de la existencia de una transformada de referencia (pose) "Location_CenterBase", con posición y orientación conocidas respecto de la referencia *WorkObject* "TFG_Drill_Base".

ATENCÓN: Es importante no olvidarse de anular la orden *PDispSet*() con un *PdispOff*, al finalizar las zonas de código específicas.

b) Observaciones Propias del Ejemplo:

No evita la recomendación de recurrir a una referencia "nula", pero sí de trampear al programa con una rotación de +30º en "X" sobre dicha referencia.

Transformación explicitada en la instrucción de "desplazamiento", incluida la rotación.

c) Principales contras:

Se oculta la composición de Transformaciones, queda enmascarada a los usuarios que consulten el código programado.

Se **requiere de la definición de un "*robtarget* nulo"** ("Null_Loc_WOffs_V2").

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la Herramienta **PDispSet**.

B) Estructura de 4 Niveles:

```
MODULE PDispSet_Function

    !CONFIGURATION DATA::
    !Lebel Reference01:     !wobjdata TFG_Drill_Base: It's a date that the user need to upgrade with the FlexPendant
    TASK PERS wobjdata TFG_Drill_Base:=[FALSE,TRUE,"",[[414.666,-66.9352,281.176],[0.984505,-0.0722476,-0.0116787,0.159357]],[[0,0,0],[1,0,0,0]]];   !New
    !Constant Data: Dont Delete This:
    CONST extjoint Extern_Axis:=[9E9,9E9,9E9,9E9,9E9,9E9];        !Put 9E9 in all term if the Task haven't any extern axis
    CONST confdata Robot_Configuration:=[0,0,0,0];
    CONST jointtarget Joint_Home:=[[0,0,0,0,0,0],Extern_Axis];    !It's define the home position, it's a common place for all tasks
    !The user need to define the quatern terms: q1:q4
    CONST orient RotationXangle30:=[0.96592582,0.25881904,0,0]; !Common Reference !Rot(x,30)
    CONST pose Location_CenterBase:= [[77,133,50],RotationXangle30];
    !ALL THIS BLOCK USE: wobjdata TFG_Drill_Base: RELATIVE LOCATION:
                            !Lebel Reference: 02: Relative to TFG_Drill_Base
    !The used Offs methodes need the Specific Reference "robtarget Null_Location_WOffs" how be used by origin of position and rotation by the Tool(TCP)
    CONST robtarget Null_Loc_WOffs_V2:= [[0,0,0],[1,0,0,0],Robot_Configuration,Extern_Axis];     !Point00 Absolute: Drill Center Position: Lebel 02
                            !Lebel Reference: 03: Relative to Drill_Center_Position
    PERS robtarget Supply_RelativeToDrillCenter;                                                !Point01 Relative: Lebel 03
    PERS robtarget Shave_RelativeToDrillCenter;                                                 !Point02 Relative: Lebel 03


    PROC PDispSet_Supply_RelPosition()
        !PDispSet Location_CenterBase(how translation and orientation reference)
        PDispSet(Location_CenterBase);

        !Define Relative Point: Lebel Reference: 03   !Point01RelativeToDrillCenter: Supply_Piece_Position: Work Reference  !Rot(x,30)
        Supply_RelativeToDrillCenter:=Offs(Null_Loc_WOffs_V2,-23,0,5);

        !Now the Robot try to Reach the wished position:
        !Move Composition to aproximity: TypeMove RelTool(robtarget NameReference, offset x,offset y, offset z), speed, proximity, NameTool\WObj:=wobjdata NameReference;
        MoveJ Offs(Supply_RelativeToDrillCenter,0,0,45),v500,z10,Puntero_Celula\WObj:=TFG_Drill_Base;          !Move to Aproximation Supply Position: Lebel 04
        MoveL Supply_RelativeToDrillCenter,v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;                      !Simple Lineal Move Order until wished position: Lebel 03
        Action_SupplyPiece_Pos01;                                                                              !Action Program in Supply Position
        MoveL Offs(Supply_RelativeToDrillCenter,0,0,45),v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;         !Move to Aproximation Supply Position: Lebel 04

        !Ending PDispSet(Location_CenterBase)
        PDispOff;
    ENDPROC

ENDMODULE
```

Figure 53: Código Relativización de Localizaciones Función PDispSet: 4 Niveles: Célula Real

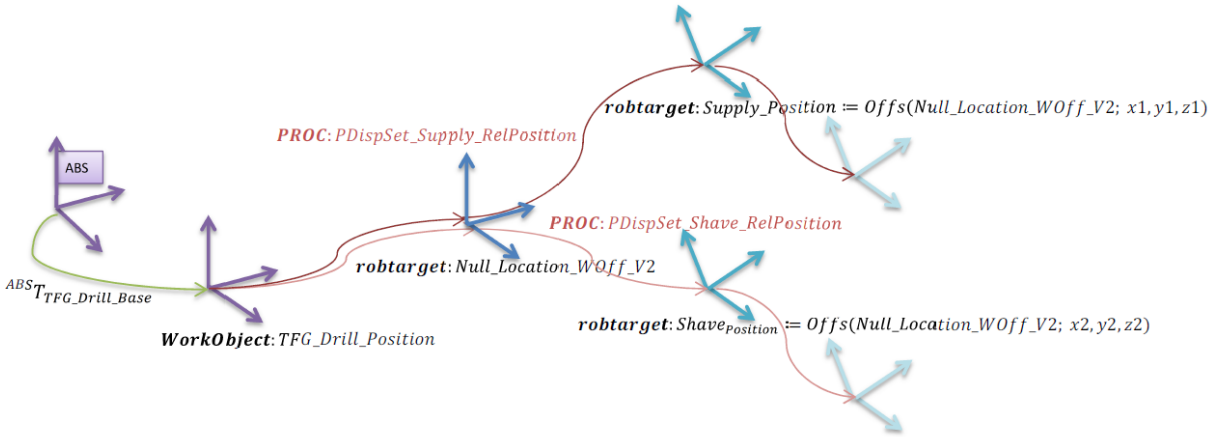


Figure 54: Relativización de Localizaciones Función PDispSet: 4 Niveles: Célula Real

a) ¿Cumple las Expectativas?:

Cumple las expectativas, aunque aumenta el nivel de código e introduce la necesidad de definir una transformación específica y una referencia "nula".

Composición explícita de TRF (PoseMult)

A) *Propiedades de TRF-PoseMult (Pose Multiply):*

RAPID permite la composición explícita de transformaciones por medio de la función "***PoseMult***", no obstante, **generalmente incita a seguir los otros procedimientos, expuestos con anterioridad, enmascarándola bajo** diferentes **instrucciones/herramientas** de programación.

a) Generalidades:

La función *PoseMult* **se utiliza para calcular el producto de dos transformaciones de "pose".** Permite definir transformaciones de nivel jerárquico superior como composición de transformaciones conocidas.

Tipo de Dato: pose

Explicita la composición de Transformaciones, no la enmascara y permite que el usuario defina la composición jerárquica exacta.

b) Principales contras:

Procedimiento complicado: Obliga a definir todas las transformaciones que existen con el tipo de dato "pose" y configurar los *Target* como la composición de las transformaciones descritas.

RAPID no favorece el uso de la composición explícita de transformaciones "*PoseMult*", pues su uso lleva aparejada la escritura de más líneas de código fuente de las estrictamente necesarias mediante el uso de "*RelTool*" o "*PDispSet*.".

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la Herramienta **TRF**.

B) *Estructura de 4 Niveles:*

```
MODULE TRF_Function

    !CONFIGURATION DATA::
    !Lebel Reference01:     !wobjdata TFG_Drill_Base: It's a date that the user need to upgrade with the FlexPendant
    TASK PERS wobjdata TFG_Drill_Base:=[FALSE,TRUE,"",[[414.666,-66.9352,281.176],[0.984505,-0.0722476,-0.0116787,0.159357]],[[0,0,0],[1,0,0,0]]];   !New
    !Constant Data: Dont Delete This:
    CONST extjoint Extern_Axis:=[9E9,9E9,9E9,9E9,9E9,9E9];        !Put 9E9 in all term if the Task haven't any extern axis
    CONST confdata Robot_Configuration:=[0,0,0,0];
    CONST jointtarget Joint_Home:=[[0,0,0,0,0,0],Extern_Axis];    !It's define the home position, it's a common place for all tasks
    !The user need to define the quatern terms: q1:q4
    CONST orient RotationXangle30:=[0.96592582,0.25881904,0,0];   !Common Reference !Rot(x,30)
    !Data Transform Composition:
        CONST pose Location_CenterBase:= [[77,133,50],RotationXangle30];
        CONST pose Null_Drill_Base:= [[0,0,0],[1,0,0,0]];         !Only for Show all Transformation way: (*1)
        PERS pose Center_Drill_Position;
        PERS pose Supply_Position;
        PERS pose Shave_Position;

    !ALL THIS BLOCK USE: wobjdata TFG_Drill_Base: RELATIVE LOCATION:
                            !Lebel Reference: 02: Relative to TFG_Drill_Base
    PERS  robtarget Drill_RelativeToBase;                         !Only for Show all Transformation way: (*1)   !Point00 Absolute: Drill Center Position: Lebel 02
                            !Lebel Reference: 03: Relative to Drill_Center_Position
    PERS robtarget Supply_RelativeToDrillCenter;                                                                !Point01 Relative: Lebel 03
    PERS robtarget Shave_RelativeToDrillCenter;                                                                 !Point02 Relative: Lebel 03


    PROC PDispSet_Supply_RelPosition()
        !Define Pose Relative Point: Lebel Reference: 02
        Center_Drill_Position:=PoseMult(Null_Drill_Base,Location_CenterBase);      !(*2) This Composition isn't obligatory: Only for show all Transformation ways
        !If you can see the consecution Transform Moves: Uncomment this Lines
               !Drill_RelativeToBase:=[Center_Drill_Position.trans,Center_Drill_Position.rot,Robot_Configuration,Extern_Axis];
               !MoveJ RelTool(Drill_RelativeToBase,0,0,45),v200,z10,Puntero_Celula\WObj:=TFG_Drill_Base;
        !Define Relative Point: Lebel Reference: 03   !Point01RelativeToDrillCenter: Supply_Piece_Position:
        Supply_Position:=PoseMult(Center_Drill_Position,[[-23,0,5],[1,0,0,0]]);     !If line (*1) be remove: Must change Center_Drill_Position by Location_CenterBase
        !Define the target Position:
        Supply_RelativeToDrillCenter:=[Supply_Position.trans,Supply_Position.rot,Robot_Configuration,Extern_Axis];

        !Now the Robot try to Reach the wished position:
        !Move Composition to aproximity: TypeMove RelTool(robtarget NameReference, offset x,offset y, offset z), speed, proximity, NameTool\WObj:=wobjdata NameReference;
        MoveJ RelTool(Supply_RelativeToDrillCenter,0,0,45),v200,z10,Puntero_Celula\WObj:=TFG_Drill_Base;       !Move to Aproximation Supply Position: Lebel 04
        MoveL Supply_RelativeToDrillCenter,v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;                      !Simple Lineal Move Order until wished position: Lebel 03
        Action_SupplyPiece_Pos01;                                                                              !Action Program in Supply Position
        MoveL RelTool(Supply_RelativeToDrillCenter,0,0,45),v100,fine,Puntero_Celula\WObj:=TFG_Drill_Base;      !Move to Aproximation Supply Position: Lebel 04
    ENDPROC

ENDMODULE
```

Figure 55: Código Relativización de Localizaciones Función TRF: 4 Niveles: Célula Real

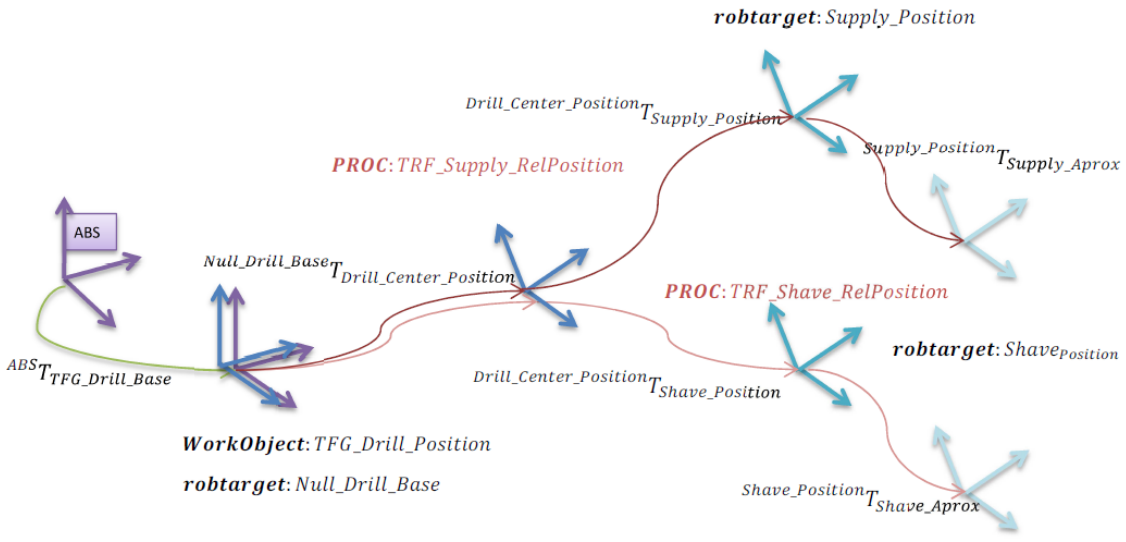


Figure 56: Relativización de Localizaciones Función TRF: 4 Niveles: Célula Real

a) ¿Cumple las Expectativas?:

Cumple las expectativas, aunque aumenta el nivel de complejidad del código

3. 4. 3. Programa Principal del Robot ABB IRB 120: Main Program

Consta de una inicialización del sistema y de un "*Loop*" donde permanecerá esperando hasta que se le explicite una de las dos órdenes de movimiento programadas, siempre y cuando el PLC siga demandando el uso del recurso del Robot o no salte la emergencia del sistema. Una vez activada alguna de las Entradas del Robot que lo comunica con el autómata, chequeará qué orden se le está solicitando y la ejecutará, tras ello le comunicará el fin de tarea al PLC Maestro y volverá al estado "esperando orden".

120-100975 (10.3.17.204) ×
T_ROB1/MainModule ×

```
1   MODULE MainModule
2       VAR intnum Interruption_Error;
3       !signalgo
4       PROC main()
5           !Initialitation Program:
6           TFG_OctAugusto_Initialitation;
7           !Initialitation Interruption TRAP
8           IDelete Interruption_Error;
9           CONNECT Interruption_Error WITH Error_Routine;
10          ISignalDI D652_10_DI12,1,Interruption_Error;
11          !Robot ready to work:
12          SetDO D652_10_DO11,1;
13
14          !Wait Order PLC Master: Perfect Slave: "Yes My Lord!"
15          WHILE NOT TestDI(D652_10_DI11) DO
16              WaitUntil (TestDI(D652_10_DI9) OR TestDI(D652_10_DI10) OR TestDI(D652_10_DI11));
17              IF TestDI(D652_10_DI9) THEN
18                  Relative_Supply_RelPosition;
19                  SetDO D652_10_DO10,1;
20                  WaitTime 0.2;
21                  SetDO D652_10_DO10,0;
22              ELSEIF TestDI(D652_10_DI10) THEN
23                  Relative_Shave_RelPosition;
24                  SetDO D652_10_DO9,1;
25                  WaitTime 0.2;
26                  SetDO D652_10_DO9,0;
27              ENDIF
28          ENDWHILE
29          !The robot returns home and will be disconected:
30          SetDO D652_10_DO11,0;
31          MoveAbsJ Joint_Home\NoEOffs,v500,fine,Puntero_Celula;
32      ENDPROC
33
34      TRAP Error_Routine
35          TPWrite "Error: You must: 1.- Check the Station; 2.- Identity the error";
36          TPWrite "3.- Rearm the PLC Master with UnityPro and ream the Robor with RobotStudio";
37          WaitDI D652_10_DI13,1;
38          MoveAbsJ Joint_Home\NoEOffs,v500,fine,Puntero_Celula;
39          SetDO D652_10_DO11,0;
40          Exit;
41      ENDTRAP
42
43  ENDMODULE
```

Figure 57: Programa Principal del Robot ABB IRB 120: Main Program: Célula Real

Adicionalmente, se le ha programado una **Rutina de Interrupción (*Error_Routine*)** encargada de bloquear los movimientos del Robot, hasta que se haya solucionado la emergencia y rearmado el PLC Maestro. Así mismo, para **continuar con la producción**, será **necesario rearmar/arrancar el programa principal desde RAPID-RobotStudio**.

En el siguiente enlace se puede ver un Vídeo del Funcionamiento de la **Célula_de_Trabajo_Completa**.

4. Conclusiones Finales del Proyecto:

4. 1. Creación de Herramientas y Controladores mediante el uso de *SmartObjects*

A) Definición de una Herramienta Virtual o Simulada

Si no se va a trabajar sobre un soporte real y **sólo se opera a nivel virtual**, el cuidado de **detalles y cálculo de CG e Inercias Principales** queda a **expensas** del tiempo que desee invertir el **usuario**. **No salpica al contenido de los programas** implementados **en RobotStudio-RAPID**. A la vista de los resultados obtenidos, **puede concluirse que**:

- Si no se va a trabajar sobre Herramientas Reales, y el modelo de esta es geométricamente sencillo, **es innecesario calcular el CG e Inercias Principales de la Herramienta**.

B) Comparación del Procedimiento de Implementación de un Controlador Lógico para una Herramienta dada

Para una misma estructura geométrica, en la acción de apertura y cierre de una Garra Mecanizada se obtienen los resultados expuestos en la **Tabla 7**.

Las consideraciones que hay que tomar en el código RAPID, cuando se emplea un Detector de Colisiones, van contra lógica. Es engorroso tener que mantener todas las partes de la Herramienta separadas entre sí. Se considera que no es un supuesto predecible a-priori.

Para programas iterativos y trabajando siempre sobre un mismo objeto, es decir, que no sea reposicionado por un elemento externo, **es recomendable utilizar el control por Sensores. Presenta una mayor robustez a repetitividad**, aunque la lógica implementada es notablemente más compleja, crece de forma exponencial al número de sensores utilizados.

A pesar de que, **siempre que se pueda por diseño geométrico** de la herramienta "*Tool*" **y se trabaje de forma no iterativa sobre los mismos objetos**, sería aconsejable **usar el detector de colisiones "*CollisionSensor*"**. Presenta un comportamiento más ajustado a la realidad, en cuanto a lógica programada del SmartObject, y una notable mejora en consumo de tiempo de cómputo en ejecución de tareas. El principal inconveniente es que necesita del apoyo de esperas programadas "*Delays*" por código RAPID, sumado al excesivo "*tamaño*" del fichero.

A la vista de la **Tabla 7** y de los argumentos expuestos**, puede concluirse que**:

1) **Se recomienda implementar el Controlador de la Herramienta por medio de Sensores**.

Tabla 7: Comparación del Procedimiento de Implementación de un Controlador Lógico para una Herramienta dada: "Implementación Por Sensores" Vs "Implementación por CollisionSenser"

Propiedad	Implementación con Sensores	Implementación con *CollisionSenser*
Restricciones Estructurales en diseño	No	**Los sólidos** que componen la herramienta **no deben tocarse entre si**
Tipo número de Sensor	2 Planos	1 Detector de Colisiones
Restricciones propias de Sensores	Sí, *ver* ***ANEXO IV*** **(Problema GRAVE!)**	No
Relación Distancia de Penetración	Elevada **(Problema GRAVE!)**	Menor distancia de penetración, más constante ante la relación: "Velocidad de Aproximación-*Step*"
Complejidad Lógica Cableada	Nivel Elevado **(Proceso intuitivo)**	Nivel Moderado (*Exige conocimientos específicos*) **(Problema Moderado…)**
Tiempo de Ciclo Ejecución Lógica cableada	Mayor (*Sorprendentemente el funciona mejor un sensor volumétrico que 2 planos!!!)*	Menor
Aparición de "*lag*" en simulación	Sí	No detectado
Tamaño del fichero ".rslib"	Entre **15kB y 40kB**	A *Part*ir de **150kB** **(Problema GRAVE!)**
Consideraciones en RAPID (para *Task* cíclicas sin reposicionamiento externo)	No *Suficiente con la orden "fine" en la acción previa de movimiento.*	Sí, *inclusión de "WaitTime XXXX" tras las acciones: agarrar/abrir.* *Insuficiente con la orden "fine" en la acción previa de movimiento.* **Ver vídeo demostración:** Acumulative_Error_CollisionSenser **(Problema MUY GRAVE!)**

Se puede ver el Funcionamiento de la Garra implementada con Sensores en el siguiente vídeo: Tool_Sensor_Claw.

Se puede ver el Funcionamiento de la Garra implementada con *CollisionSenser* en el siguiente vídeo: Tool_Collision_Claw.

4. 2. Implementación de un PLC en RobotStudio

RobotStudio NO es un programa centrado en la simulación de PLC's, facilita alguna herramienta simplificada que puede "*intentar*" hacer la vez de estos en una simulación, aunque con serias deficiencias. A cambio, recomiendan/fomentan recurrir a la comunicación con elementos/programas externos que se encarguen de ello.

Dado que se pretendía trabajar íntegramente con el entorno de programación de ABB, se han experimentado y obtenido resultados en la implementación de PLC's concurrentes de varias formas distintas:

- **PLC Programable**: Programado en código estructurado con la modalidad *Switch-Case*
- **PLC de Lógica Cableada**: al estilo tradicional, por medio de la implementación de "*Flip-Flop*", en un *SmartObject*

*Part*iendo de la base, **los IRC-5 tienen que restringir sus funciones exclusivamente para calcular las transformaciones de las trayectorias programadas** y procesamiento de órdenes mínimas, como activación/desactivación de E/S. **No se debería hacer uso de un IRC-5 para gestionar un PLC Maestro**. Por procedimiento de trabajo, es aconsejable tratar a los Robots como "*esclavos perfectos*" y deslocalizar las funciones de coordinación, a elementos externos de los Controladores de los Robots.

Se desaconseja implementar PLC's con la metodología de biestables "*Flip-Flop*", a pesar de ser la más extendida en la industria contemporánea. Las razones son:

- RobotStudio **presenta deficiencias en la gestión de las E/S**, lo que provoca serios problemas en la interacción de los SmartObject con el resto de la Estación. **Problemática de pérdida de flancos y/o identificar la actualización de las Salidas de los elementos**. Como se aprecia en "**PLC_SmartObject_con_Fallos_IOs**".
- RobotStudio **opera de forma Asíncrona y simula de forma Síncrona**. Esto ha provocado la **necesidad de "*Delays*" para desconectar los Estados Activos** (*implementados con Puertas Lógicas "OR"*) y poder así **garantizar** el **sincronismo por Pulso mínimo** (T_p*=0.001 seg*). Esto, así mismo, genera un **Problema muy Grave "¡***Existe concurrencia de 2 Estados consecutivos Activos durante el tiempo del ancho de pulso***!**". Comportamiento que **difiere notablemente de la realidad** de la operación de los PLC's.
- **Limitaciones en la edición de *SmartObjects*,** en aquellos objetos que poseen un grado de complejidad elevado en la Lógica Implementada. **Ejemplo**: *Dificultad en identificación de bloques lógicos que componen el PLC*, ver **ANEXO II** y **ANEXO VI**.

A la vista de los argumentos expuestos, **puede concluirse que**:

1) Sin lugar a dudas, **se recomienda encarecidamente la implementación por líneas de código en un IRC-5 Virtual** antes que recurrir a la implementación por "*Flip-Flop*" directamente implementada sobre un SmartObject.

4. 3. Relativización de localizaciones en RAPID

A) Análisis General de la Relativización de Localizaciones

RAPID permite la composición explícita de transformaciones ("*PoseMult*"), si bien queda **generalmente enmascarada bajo** diferentes **instrucciones/herramientas** de programación **como son**:

a. El **uso de "*WorkObjects*"**
b. La **definición de traslaciones** con "*Offs*" y "*RelTool*"
c. El **replicado de localizaciones/bloques de programa** por medio de "*PDispSet*".

Es **posible combinar estas herramientas**, abriendo todavía más el abanico de posibilidades para relativizar localizaciones, a costa de dificultar la legibilidad del código (*por ejemplo, combinando el uso de referencias de objeto y de Offs/RelTool*).

Cabe mencionar que **RAPID**, ya sea desde RobotStudio o desde el FlexPendant, favorece el uso de alguna de las alternativas comentadas previamente, enmascarando la composición explícita de transformaciones:

- ***Estructura** predefinida de 2 niveles (referencias de usuario y de objeto de trabajo: "**WorkObject**") apoyada con **diferentes Targets y Offset**.*)

A lo que hay que añadir que, como se comentará más adelante, alguna de las alternativas citadas no supone en realidad una composición de transformaciones.

B) Comparación aplicada a las Tareas de la Maqueta

En la **Tabla 8**, se comparan los procedimientos seleccionados para relativizar las localizaciones. **Puede concluirse que**:

1) La **definición de traslaciones mediante "*Offs*" no supone una verdadera relativización de coordenadas**, dado que el *Offset* especificado no se define en la referencia incluida en la instrucción. Dicho esto, su uso podría llegar a ser satisfactorio en el caso particular de que la referencia incluida en la instrucción tuviese exactamente la misma orientación que la referencia del objeto de trabajo que se esté utilizando.
2) La **definición de traslaciones** mediante **"*RelTool*" sí supone una relativización efectiva de coordenadas**, pues el *Offset* especificado sí se define en la referencia incluida en la instrucción. Por ello, la denominación de la instrucción no parece las más adecuada, pues resulta poco intuitiva (*hubiera resultado más intuitiva una denominación como "OffsRel" o similar, por contraposición a "Offs"*).
3) Por último, cabe señalar que **RAPID no favorece el uso de la composición explícita de transformaciones "*PoseMult*"**, pues su uso lleva aparejada la escritura de más líneas de código fuente de las estrictamente necesarias mediante el uso de "*RelTool*" o "*PDispSet*.".

Tabla 8: Análisis crítico de las posibilidades de relativización de localizaciones que ofrece RAPID

Procedimiento:	Estructura Jerárquica	Consideraciones Propias:	Composición de Transformaciones
Herramienta definición de Traslaciones (*RelTool*)	4 Niveles	Permite **definir de forma compacta una transformación en coordenadas relativas** a un *Target*	**Sí** (*Enmascarada*)
Herramienta definición de Traslaciones (*RelTool*)	3 Niveles	Definir el **RelTool directamente en la función de movimiento**	**Sí** (*Enmascarada*)
Estructuración de la escena (*WorkObject + Offs*)	4 Niveles	**Aprovecha la estructuración de 2 niveles** (user *Frame* + work object *Frame*) **aportada por defecto en RAPID**	No
Estructuración de la escena (*WorkObject + Offs*)	4 Niveles-Alternativa	**Desplazamiento del WorkObject** para aplicar todos los *Offset*s desde una referencia común y "Nula"	No
Desplazamiento de Programa	4 Niveles	**Exige de la existencia de una transformada de referencia** (pose), respecto de la referencia WorkObject	**Sí** (*Enmascarada*)
Composición Explicita TRF	4 Niveles	**Procedimiento farragoso**, el lenguaje no da facilidades	**Sí (*Explícita*)**

En el siguiente enlace se puede ver un Vídeo Comparativo del Funcionamiento de las distintas alternativas estudiadas: **Comparative_Proccess_Relativitation**.

Bibliografía

Documentos Bibliografía:

Para Introducción al Robot

[1] **DataSheet:** "*RobotStudio_5_datasheet_es_print*". ABB Automation Technology Products AB. Noviembre 2005. DataSheet del Programa RobotStudio.

[2] **DataSheet:** "*DataSheet_ABB_IRB_120_Complet*". ABB Automation Technology Products AB. Publicado 2017-10-17. DataSheet del Robot ABB IRB 120.

[3] **DataSheet:** "*DataSheet_ABB_IRB_120_Sumary*". ABB Automation Technology. Agosto 2009. DataSheet del Robot ABB IRB 120.

[4] **DataSheet**: "*DataSheet_MHL2_ES*". SMC Corporation. November 2001

[5] **Canal de YouTube**: "RobotStudio". ABB Robotics. Descripción del Programa RobotStudio. https://www.youtube.com/channel/UCMNYO6IutHCnpyQevcnpACw

Para Introducción a los Fundamentos del programa

[6] **Vídeos Tutoriales de RobotStudio**: Canal YouTube "*Juan Carlos Martín Castillo*". Profesor de FP II de Electricidad, electrónica y Automática (http://reea-blog.blogspot.com.es/). Autor: Juan Carlos Martín Castillo https://www.youtube.com/channel/UCsRQGtuqCm1-d-whRYOSV2Q

[7] **Vídeos varios Tutoriales de RobotStudio**: "*alonsoprofe*: Curso de RobotStudio*. Tema* (nº de Tema)". Autor: Alonso https://www.youtube.com/user/alonsoprofe/videos

[8] **Manuales**: "*Manual de referencia de RAPID: Descripción general de RAPID: Software de controlador para IRC5: RobotWare 5.0*". ABB Automation Technology Products AB. Publicado 2004

[9] **Manuales**: "*Manual del Operador: RobotStudio*". ABB Automation Technology Products AB. Publicado 2017

[10] **Documentación Complementaria**: "*Automatización Industrial II: Introducción al lenguaje RAPID*". Autor Dr. Alaa Khamis (Docente de la UC3M)

[11] **Documentación Complementaria**: "*Introducción a la creación de escenarios y de sólidos con RobotStudio de ABB*". Autor Antonio Romeo Tello

Para funciones avanzadas y analizar la relativización de localizaciones

[12] **Manuales Referencias Técnicas**: "*Manual de referencia técnica: Instrucciones, funciones y tipos de datos de RAPID: Software de controlador para IRC5: RobotWare 5.13*". ABB Automation Technology Products AB. Publicado 2010

[13] **Documentación Complementaria**: "*Relativización de localizaciones*". Autor Antonio Romeo

Índices Complementarios

Índice de Figuras:

a) Memoria

Figure 1: Cronograma GANTT de las Tareas Desarrolladas en el TFG 4
Figure 2: Isotipo RobotStudio 6
Figure 3: Simulación Célula Industrial Completa (*y Enlace a un Vídeo*) 8
Figure 4: Herramienta Doble Operación 9
Figure 5: Robot ABB IRB 120 9
Figure 6: Robot ABB IRB 140 9
Figure 7: Herramienta MHL02-10-SR.8 con CollisionSenser 9
Figure 8: Herramienta Rotacional con 1 Sensor 9
Figure 9: Herramienta MHL02-10-SR.8 con 2 Sensores 9
Figure 10: Conveyor de Aprovisionamiento 10
Figure 11: Conveyor de Extracción 10
Figure 12: Cajas Manipulables "Boxies" 10
Figure 13: Ciclo de Vida de la Pieza en Régimen Permanente: Célula Real 11
Figure 14: Secuencia Programación de Régimen Permanente: Célula Real 11
Figure 15: Robot ABB IRB 120 con su Controlador IRC-5 real y FlexPendant 12
Figure 16: IRB120 Working range Sidways 1700x1406 13
Figure 17: IRB 120 Working range Plane 13
Figure 18: Módulo de E/S binario del controlador IRC-5 del Robot ABB IRB 120 14
Figure 19: Maqueta de máquina de taladrado FISCHERTECHNIK 14
Figure 20: Módulo Central M 340 15
Figure 21: Módulo E/S BMX DDI 1602 y DRA 0805 15
Figure 22: Fuente de Alimentación: +24V y +5V 15
Figure 23: Cableado entre el autómata 16
Figure 24: Jerarquía de Control y Relaciones entre componentes de la Estación Simulada 18
Figure 25: Modelo Virtual de la Herramienta: Modo Garra 21
Figure 26: Modelo Virtual de la Herramienta: Modo Stick 21
Figure 27: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con Sensores 22
Figure 28: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con CollisionSensor 23
Figure 29: Grafo de Estados del PLC Master de 16 Estados con Concurrencia: Estación Simulada 26
Figure 30: Esquemático del PLC Master 16 Estados: Flip-Flop del SmartObject 29
Figure 31: Esquemático del PLC Master 16 Estados: SmartObject 30
Figure 32: Descripción de Posiciones de Operación de la Maqueta: Célula Real 34

Figure 33: Herramienta Garra Real: Base neumática de la empresa SMC "Modelo MHL2_ES-10" 35
Figure 34: Cuerpo de la Pinza MHL2-10 36
Figure 35: Aplicación en el autómata "MODICON: M 340": Implementación de un PLC 37
Figure 36: Programa Implementado en el PLC Maestro de la Célula Real: Transitorio de Arranque 38
Figure 37: Programa Implementado en el PLC Maestro de la Célula Real: Régimen Permanente 39
Figure 38: Programa Implementado en el PLC Maestro de la Célula Real: Transitorio de Parada 40
Figure 39: Código de Interrupción-Fallo del PLC Maestro: Célula Real 41
Figure 40: Jerarquía Control: Célula Real 43
Figure 41: Jerarquía de Localizaciones de la Maqueta: Célula Real 44
Figure 42: Relativización de Localizaciones: 4 Niveles: Célula Real 45
Figure 43: Relativización de Localizaciones: 3 Niveles: Célula Real 45
Figure 44: Código Relativización de Localizaciones Función RelTool: 4 Niveles: Célula Real 47
Figure 45: Relativización de Localizaciones Función RelTool: 4 Niveles: Célula Real 47
Figure 46: Código Relativización de Localizaciones Función RelTool: Transformaciones 3 Niveles: Célula Real 48
Figure 47: Relativización de Localizaciones: Transformaciones 3 Niveles: Célula Real 48
Figure 48: Relativización de Localizaciones Función RelTool: 3 Niveles: Célula Real 49
Figure 49: Código Relativización de Localizaciones Función WOffs: 4 Niveles: Célula Real 51
Figure 50: Relativización de Localizaciones Función WOffs: 4 Niveles: Célula Real 51
Figure 51: Código Relativización de Localizaciones Función WOffs Alternativa: 4 Niveles: Célula Real 52
Figure 52: Relativización de Localizaciones Función WOffs Alternativa: 4 Niveles: Célula Real. 53
Figure 53: Código Relativización de Localizaciones Función PDispSet: 4 Niveles: Célula Real ... 55
Figure 54: Relativización de Localizaciones Función PDispSet: 4 Niveles: Célula Real 55
Figure 55: Código Relativización de Localizaciones Función TRF: 4 Niveles: Célula Real 57
Figure 56: Relativización de Localizaciones Función TRF: 4 Niveles: Célula Real 57
Figure 57: Programa Principal del Robot ABB IRB 120: Main Program: Célula Real 58

b) ANEXO I

Figure 58: Cronograma de Gantt Completo 3

c) ANEXO II

Figure 59: Ejemplo Modelaje 6
Figure 60: Ubicación de Referencias en Pieza Ejemplo 8
Figure 61: Aplicación de Acción de Unión de Bodies 9
Figure 62: Ubicación de FramePart y FrameBody pieza resultante de Acción de Unión 9

Figure 63: Pantalla “Offset Position: Name_Part” 10
Figure 64: Pantalla “Offset Position: Name_Body” 10
Figure 65: Ejemplo de Bibliotecas 12
Figure 66: Secuencia de Zoom en la Pestaña de Edición de SmartObject 18
Figure 67: Activada la opción de "Ocultar Conexiones" 19
Figure 68: Pestaña de Edición y Diseño de SmartObject de Referencia 19
Figure 69: Activación de la Función "Auto Arrange" en un SmartObject 20
Figure 70: Consecuencias de "Eliminar" un Bloque de la Pestaña de Diseño: Aparición de "Cronembergs" 21
Figure 71: Tipos de Sensores que existen en RobotStudio 23
Figure 72: Sensores en *Conveyors*: Sensor Guia Movimiento 26
Figure 73: Sensores en *Conveyors*: Sensor Posicionamiento: Plano 26
Figure 74: Herramienta de Comparación de Control por: Sensores vs CollisionSenser 27
Figure 75: Control Lógico Garra CollisionSenser 28
Figure 76: Control Lógico Garra Sensores 29
Figure 77: Herramienta Simple con Puntero Desplazado en X 37
Figure 78: Herramienta Simple con 2 TCP's, uno de ellos con orientación distinta a la base 37
Figure 79: Creación de Mecanismos 40
Figure 80: Estación de Ejemplo Cambio de Herramienta (Tool) 43
Figure 81: Pantallas de Datos de Herramienta "Tool" 44
Figure 82: Pantalla de Buscar/Encontrar de RobotStudio 45
Figure 83: Menú de opciones de la Herramienta 46
Figure 84: Estación Análisis Detector de Colisiones 49
Figure 85: Configuración Parámetros Específicos de Detección de Colisiones: Estación 49
Figure 86: Estación Análisis Selector de Colisiones: Recorrido como Agarre transporte de piezas 50
Figure 87: Estación Análisis Selector de Colisiones: Recorrido como Agarre transporte de piezas y Zoom 50
Figure 88: onfiguración Parámetros Específicos de Detección de Colisiones: SmartObject y Zoom de BlockEditor Properties CollisionSensor 51
Figure 89: Editor de Propiedades del CollisionSensor 51
Figure 90: Indicador de detección de colisión en el agarre 52
Figure 91: Indicador de detección de colisión en el seguimiento con el StickPointer 53
Figure 92: Configuración de Parámetros "EventManager" 54
Figure 93: Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation 57

d) ANEXO III

Figure 94: Menú de la Pestaña "Modeling": RobotStudio 3
Figure 95: Menú de la Pestaña "Modify Part": RobotStudio 3
Figure 96: Menú de la Pestaña "Modify Body": RobotStudio 3

Figure 97: Menú Pestañas: "Signals and Properties", "Parametric Primitives" y "Sensors": SmartObject: RobotStudio 6
Figure 98: Menú Pestañas: "Actions", "Manipulators" y "Other": SmartObject: RobotStudio 7
Figure 99: Menú Asistente de Creación: "Create Tool": RobotStudio 8
Figure 100: Menú Asistente de Creación: "Create Mechanism": RobotStudio 9
Figure 101: Menú Asistente de Creación: "Create Link": "Create Mechanism": RobotStudio 9
Figure 102: Menú Asistente de Creación: "Create Link": "Create Joint": RobotStudio 10
Figure 103: Menú Asistente de Creación: "Create Link": "Create Tooldata": RobotStudio..... 11
Figure 104: Configuración Parámetros Generales de la Estación 15
Figure 105: Configuración Parámetros Específicos de Detección de Colisiones: Estación 16
Figure 106: Configuración Parámetros Específicos de Detección de Colisiones: SmartObject y Zoom de BlockEditor Properties CollisionSensor 17
Figure 107: Configuración de Parámetros "EventManager": RobotStudio 18
Figure 108: Configuración de Parámetros Temporales en Simulación 21
Figure 109: Configuración del Modo de Gestión de Tiempo en Simulación..... 22
Figure 110: Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation 23

e) ANEXO IV

Figure 111: Limitaciones de los Sensores: Sensor Suficientemente Grande 5
Figure 112: Limitaciones de los Sensores: Sensor Insuficientemente Grande 7
Figure 113: Step dado para comparativa de Velocidad de Aproximación entre Objetos..... 12
Figure 114: Estación de Prueba: Limitaciones de la Velocidad de Aproximación..... 12
Figure 115: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18.24mm*)..... 13
Figure 116: Velocidad de Aproximación: 0.15 seg/20 mm: Length final (*15.20mm*)..... 13
Figure 117: Velocidad de Aproximación: 0.25 seg/20 mm: Length final (*14,59mm*)..... 14
Figure 118: Velocidad de Aproximación: 0.5 seg/20 mm: Length final (*14,68mm*)..... 14
Figure 119: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18.24mm*)..... 15
Figure 120: Velocidad de Aproximación: 0.15/20 mm: Length final (*15.20mm*)..... 15
Figure 121: Velocidad de Aproximación: 0.25 seg/20 mm: Length final (*16.42mm*)..... 16
Figure 122: Velocidad de Aproximación: 0.5 seg/20 mm: Length final (*15.50mm*)..... 16
Figure 123: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*15.96mm*): TimStep 12ms (*6ms*)..... 17
Figure 124: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*15.96mm*): TimStep 12ms (*3ms*)..... 18
Figure 125: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*15.96mm*): TimStep 12ms (*12ms*)..... 19
Figure 126: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18,24mm*): TimStep 24ms (*12ms*)..... 20
Figure 127: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18,24mm*): TimStep 24ms (*3ms*)..... 21

f) ANEXO V

Figure 128: Herramienta Garra Real: Base neumática de la empresa SMC "*Modelo MHL2_ES-10*" 2
Figure 129: Cuerpo de la Pinza MHL2-10 3
Figure 130: Diseño Geométrico de la Herramienta Virtual: SR.8 (*modalidad completa: Sticker deslizante*) 5
Figure 131: Datos Generales de CG's e Inercias Principales aplicables a los dos Modos de Simulación de Herramienta Virtual SR.8 6
Figure 132: Script Matlab: CG's e Inercias Principales: Modo Garra: Herramienta Virtual SR.8 .. 7
Figure 133: Propiedades Técnicas Herramienta Simulada "Modo Garra" 7
Figure 134: Script Matlab: CG's e Inercias Principales: Modo Stick: Herramienta Virtual SR.8 8
Figure 135: Propiedades Técnicas Herramienta Simulada "Modo Stick" 9
Figure 136: Propiedades Constructivas de la Herramienta Virtual Implementada con Sensores 10
Figure 137: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con Sensores 11
Figure 138: Esquema General: Implementación con Sensores 12
Figure 139: Esquema Específico Sensores: Modo Garra: Identificación de Objetos y Apertura/Cierre de la Garra 12
Figure 140: Esquema Específico Sensores: Modo Garra: Enlazamiento/Agarre de Objetos y Cálculo de Distancia de Apertura de Garra 13
Figure 141: Esquema Específico Sensores: Control del Modo Sticker 13
Figure 142: Esquema Específico Sensores: Detección de lo Colisiones doble Modo (*funcionalidad opcional*) 14
Figure 143: Propiedades Constructivas de la Herramienta Virtual Implementada con CollisionSenser 15
Figure 144: Diferencia Física: El Vástago NO posee GL, y por consiguiente se puede Ocultar/mostrar 16
Figure 145: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con CollisionSensor 17
Figure 146: Esquema General: Implementación con CollisionSensor 17
Figure 147: Esquema Específico CollisionSensor: Modo Garra: Apertura/Cierre de la Garra, Identificación de Objetos y Enlace/Desenlace de Objetos 18
Figure 148: Esquema Específico CollisionSensor: Control del Modo Sticker 18
Figure 149: Esquema Específico CollisionSensor: Detección de lo Colisiones doble Modo (funcionalidad opcional) 19

g) ANEXO VI

Figure 150: Grafo de Estados del PLC Master: Estación Simulada 4
Figure 151: Esquemático del PLC Master 16 Estados: SmartObject 13
Figure 152: Identificación de las partes del Esquemático del PLC Master 16 Estados: SmartObject 14
Figure 153: Esquemático del PLC Master 16 Estados: Flip-Flop del SmartObject 15
Figure 154: Identificación Partes Esquemático Flip-Flop del SmartObject 16

h) ANEXO VII

Figure 155: Simulación Célula Industrial Completa (*y Enlace a un Vídeo: Ctrl+click izquierdo*) ... 3
Figure 156: Herramienta Doble Operación 5
Figure 157: Robot ABB IRB 120 5
Figure 158: Robot ABB IRB 140 5
Figure 159: Herramienta MHL02-10-SR.8 con CollisionSenser 5
Figure 160: Herramienta Rotacional con 1 Sensor 5
Figure 161: Herramienta MHL02-10-SR.8 con 2 Sensores 5
Figure 162: Conveyor de Aprovisionamiento 6
Figure 163: Conveyor de Extracción 6
Figure 164: Cajas Manipulables “Boxies” 6
Figure 165: Implementación Lógica: *Conveyor* de Abastecimiento 7
Figure 166: Implementación Lógica: Conveyor de Extracción 7
Figure 167: Implementación Lógica: SmartBuffer 8
Figure 168: Jerarquía de Control y Relaciones entre componentes de la Estación Simulada 9
Figure 169: Configuración de E/S: Controlador PLC Master 10
Figure 170: Configuración de E/S: Controlador Robot IRB 120 (Encargado de Mecanizar) 10
Figure 171: Configuración de E/S: Robot IRB 140 (encargado de Abastacer/Extraer) 10
Figure 172: Cableado Lógico General de la Estación: "*Célula 8*" 11
Figure 173: Cableado Lógico Específico de la Estación "Célula 8": Selector de PLC Master 12
Figure 174 : Cableado Lógico Específico de la Estación "Célula 8": Robot IRB 140 (Aprovisionamiento) y Selector de Herramienta (MHL2_ES_10-SR.8) 13
Figure 175: Cableado Lógico Específico de la Estación "Célula 8": Robot IRB 120 (Mecanizado) 14
Figure 176: Panel Selector del Usuario 14
Figure 177: Esquemático del PLC Master 16 Estados: SmartObject 22
Figure 178: Esquemático del PLC Master 16 Estados: Flip-Flop del SmartObject 23
Figure 179: Trayectorias de los Robots: Estación Simulada "Célula 8" 24
Figure 180: Código del Robot 1: IRB 120 (Abastacemiento): Estación Simulada 26
Figure 181: Código del Robot 2: IRB 120 (Mecanizado): Estación Simulada 27

Índice de Tablas

Tabla 1: Principales Elementos Estación Simulada Final 8
Tabla 2: Propiedades Técnicas Herramienta Simulada "Modo Garra" 21
Tabla 3: Propiedades Técnicas Herramienta Simulada "Modo Puntero" 21
Tabla 4: Programa Implementado en el Autómata: Grafo de 16 Estados con Concurrencia 27
Tabla 5: Mapa de Entradas y Cableado del Robot 42
Tabla 6: Mapeado de Salidas y Cableado del Robot 42
Tabla 7: Comparación del Procedimiento de Implementación de un Controlador Lógico para una Herramienta dada: "Implementación Por Sensores" Vs "Implementación por CollisionSenser" 60
Tabla 8: Análisis crítico de las posibilidades de relativización de localizaciones que ofrece RAPID 63
Tabla 9: Extensiones Importación de Geometrías y Bibliotecas ..... 11 (ANEXO II)
Tabla 4: Programa Implementado en el Autómata: Grafo de 16 Estados con Concurrencia ..... 5 (ANEXO VI)
Tabla 11: Elementos que componen la Estación Simulada: Célula 8 ..... 4 (ANEXO VII)

Índice de Vídeos

Link del Vídeo	Categoría	Descripción	Tipo de Formato
PLC_IRC5_Virtual	Funcionamiento Estación Virtual: Célula 8	Vídeo Demostrativo: **funcionamiento de la Estación Simulada** (*carácter general*)	.Wmv
CompleteStation-IRC5_PLC_Master	Funcionamiento Estación Virtual: Célula 8	Vídeo Demostrativo: **funcionamiento de la Estación Simulada**: PLC Master implementado por Líneas de Código (**IRC 5 Virtual**)	.Wmv
CompleteStation-SmartObject_PLC_Master	Funcionamiento Estación Virtual: Célula 8	Vídeo Demostrativo: **funcionamiento de la Estación Simulada**: PLC Master implementado por la **Metodología Biestables (SmartObject)**	.Wmv
CompleteStation-SmartObject_PLC_Master_PickAndPlace	Funcionamiento Estación Virtual: Célula 8	Vídeo Demostrativo: **funcionamiento de la Estación Simulada (*Modo de funcionamiento Alternativa B: "Pick and Place"*)**: PLC Master implementado por la Metodología Biestables (SmartObject)	.Wmv
PLC_SmartObject_with_IOs_Error	Funcionamiento Estación Virtual: Célula 8	Vídeo Demostrativo: **funcionamiento de la Estación Simulada**: PLC Master implementado por la **Metodología Biestables (SmartObject)** evidenciando **problemas en la gestión de E/S** entre elementos de la Estación	.Wmv
Tool_Sensor_Claw	Funcionamiento Herramienta Virtual: SR.8	Vídeo Demostrativo: funcionamiento de la **Herramienta Virtual: SR.8 implementada con 2 Sensores perpendiculares entre si**	.Wmv
Tool_CollisionSensor_Claw	Funcionamiento Herramienta Virtual: SR.8	Vídeo Demostrativo: funcionamiento de la **Herramienta Virtual: SR.8 implementada con "*CollisionSensor*"**	.Wmv

Link del Vídeo	Categoría	Descripción	Tipo de Formato
SensorLimitis_Parallel_Plane	Limitación de Sensores	Vídeo Demostrativo de las **limitaciones de los Sensores**: Ubicación del **Plano del Sensor Paralelo al Plano del Objeto** a identificar	.Wmv
SensorLimits_Parallel_Plane_Solution	Limitación de Sensores	Vídeo Demostrativo de una **posible solución a las limitaciones de los Sensores**: Ubicación del **Plano del Sensor Paralelo al Plano del Objeto** a identificar	.Wmv
AcumulativeError_NO_Delay-CollisionSensor_vs_2Sensor	Limitación de Sensores	Vídeo Demostrativo de las l**imitaciones de los Sensores**: **Acumulación de Error de Posición** al usar un "**CollisionSensor**" para identificar objetos	.Wmv
AcumulativeError_Consecuence-CollisonSensor_vs_2Sensor	Limitación de Sensores	Vídeo Demostrativo de las **limitaciones de los Sensores: Acumulación de Error de Posición** al usar un "**CollisionSensor**" para identificar objetos	.Wmv
AcumilativeError_Solution_Intermidle_Delay-CollisionSensor_vs_2Sensor	Limitación de Sensores	Vídeo Demostrativo de una **posible solución a las limitaciones de los Sensores**: **Acumulación de Error de Posición** al usar un "**CollisionSensor**" para identificar objetos	.Wmv
Fischertechnik_Machine_Demostration_Running	Funcionamiento Célula Flexible Real	Vídeo Demostrativo: **funcionamiento de la Célula Flexible Real Completa**: un PLC Maestro ("*MODICON M 340*"), una Maqueta ("*FISCHERTECHNIK*") y un Robot ABB IRB 120	.mp4

Link del Vídeo	Categoría	Descripción	Tipo de Formato
Comparative_Proccess_Relatitation	Relativización RAPID	Vídeo Comparativo: **Enfrenta alguna de las distintas formas de Relativización que ofrece RAPID** para alcanzar un mismo punto programado (*Punto de Abastecimiento "Supply_Position"*)	.mp4
Relativitation_RelTool	Relativización RAPID	Vídeo Demostrativo: **funcionamiento de la herramienta de relativización RelTool**	.mp4
Relativitation_WOffs	Relativización RAPID	Vídeo Demostrativo: **funcionamiento de la herramienta de relativización WOffs**	.mp4
Relativitation_PDispSet	Relativización RAPID	Vídeo Demostrativo: **funcionamiento de la herramienta de relativización PDispSet**	.mp4
Relativitation_TRF	Relativización RAPID	Vídeo Demostrativo: **funcionamiento de la herramienta de relativización TRF**, Composición Explícita de Transformaciones	.mp4

Índice de Acrónimos

Los siguientes acrónimos se encuentran utilizados a lo largo de los documentos Memoria y ANEXOS:

- **TFG**: Trabajo Fin de Grado, también denominado Proyecto o Proyecto Final de Carrera
- **E/S**: Entradas y Salidas de un Sistema cualesquiera, pueden ser de carácter Analógico o Digital
- **RAPID**: **R**obotics **A**pplication **P**rogramming **I**nteractive **D**ialogue
- **EINA:** Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza (UNIZAR)
- **FlexPendant**: Dispositivo/Display para manipular directamente el Robot, tanto a nivel de control directo como para configurarlo/programarlo
- **IRC 5**: Controlador IC-5: Unidad de Procesamiento y Control de los Robots de las Series Industriales, entre las que se encuentra el Robot IRB 120
- **DataSheet**: Recopilación de documentación técnica facilitada por el fabricante y/o distribuidor
- **PLC**: (en inglés: Programmable Logic Controller) Controlador lógico programable
- **Conveyors**: (en inglés) Cinta Transportadora Industrial
- **Biestable o Flip-Flop** (en inglés): es un multivibrador capaz de permanecer en uno de dos estados posibles durante un tiempo indefinido en ausencia de perturbaciones. Esta característica es ampliamente utilizada en electrónica digital para memorizar información
- **Grafcet/SFC (*Sequential Function Chart*)**: es el acrónimo tanto de Graph Fonctionnel de Commande Etape-Transition (*en español, grafo funcional de control de etapa-transición*)
- **Ruptor**: Dispositivo para abrir o cerrar el paso de corriente eléctrica en un circuito
- **TRF**: "Transformaciones", estructura matricial que contiene la información de Posición y Orientación relativas a una referencia, entre otros datos (*escalado, prespectiva*…).
- **TCP**: Tool Center Point: Punto de ajuste de la herramienta al que se hará referencia para toda acción de la herramienta
- **CG**: Centro Geométrico de Masas de un sólido/conjunto de sólidos que operan como uno solo
- **Step o "Paso de Simulación"**: Tiempo transcurrido entre la actualización de las acciones simuladas. RobotStudio es un programa de simulación discreto.
- **Lag**: Retraso que se produce en una comunicación, generalmente en una red informática, aunque puede aplicarse a cualquier tipo de comunicación
- **Grafo**: Representación simbólica de los elementos constituidos de un sistema o conjunto, mediante esquemas gráficos.
- **Switch-Case**: estructura de control empleada en programación. Se utiliza para agilizar la toma de decisiones múltiples. Trabaja de la misma manera que lo harían sucesivos if , if else o until anidados.
- **Loop** (*en programación*): Bucle o ciclo, sentencia que ejecuta repetidas veces un trozo de código hasta que la condición asignada a dicho bucle deja de cumplirse

- **WorkObject**: Referencia principal de un plano de trabajo específico en el lenguaje RAPID
- **Targets:** "Punto Objetivo" en RobotStudio, definido con Posición y Orientación relativos a la Frame especificada.
- **CAD**: Computer-Aided Design (CAD): diseño asistido por computadora (DAC)
- **Frame**: Sistema de coordenadas genérico que puede usarse como referencia a la hora de posicionar objetos

ANEXOS:

Índice de ANEXOS:

ANEXOS I: Cronograma de Gantt

ANEXOS II: Compendio de Consideraciones, Limitaciones y Observaciones varias de RobotStudio

ANEXOS III: Fichas de Ruta: "Cómo se hace…"

ANEXOS IV: Limitaciones de los Sensores

ANEXO V: La Herramienta Virtual SR.8

ANEXO VI: Implementación del PLC de 16 Estados: Atutomatismo Moore

ANEXO VII: La Estación Simulada

ANEXO I

1. GANTT: Cronograma del Proyecto e Hitos alcanzados

1. 1. Requisitos Previos de la Planificación del TFG

Se ha contado con un **marco temporal en torno a 4 meses**, comprendidos entre el 25 de septiembre de 2017 y el 30 de enero de 2018. Restricciones impuestas por demanda del Director del TFG y fechas administrativas.

Inicialmente se optó por **dividir el Proyecto en 5 etapas** diferentes, arrancando con la comprensión y asimilación de las herramientas de simulación RobotStudio y el lenguaje propio del sistema RAPID. La idea original era que una vez alcanzados los objetivos de conocimiento del sistema, se procedería a operar directamente sobre el Sistema Real "Robot ABB IRB 120" desde la interfaz de usuario FlexPendant y uno de los terminales del laboratorio DIIS-Lab06. Por problemas técnicos varios, ajenos al desarrollo del TFG, se tuvo que modificar la planificación de tareas.

1. 1. 1. Fases de Trabajo:

Fases de trabajo ordenadas cronológicamente:

1. **Trabajo sobre la herramienta software específica** y consulta de manuales y normativas para alcanzar el nivel de conocimiento suficiente en el entorno de programación de Robots ABB.
2. **Desarrollo** de una **aplicación** que suponga la **inserción de robots en** una pequeña **célula flexible de fabricación**.
3. **Implementar** una **aplicación** en el **sistema real** y la exploración de las E/S del Robot ABB IRB 120.
4. **Evaluación de resultados** obtenidos.
5. Elaboración del informe final.

Se ha tenido, como **área principal del análisis, el entorno integral de simulación desde RobotStudio** dado que permite extrapolar las destrezas y conocimiento alcanzados con relativa facilidad desde el entorno virtual al real.

1. 1. 2. Restricciones Temporales propias de este TFG

Las variaciones en los tiempos asignados (*secciones rojas de la imagen Cronograma Gantt*) responden a problemas externos a la gestión del mismo, tales como:

- **Detección de anomalía en el funcionamiento del Robot a altas velocidades**, y las consiguientes revisiones del personal Técnico de ABB (*inutilización del Robot ABB IRB 120*). Tras varias visitas del servicio técnico, se descubrió que las reductoras de los motores de los ejes 1, 2 y 3 (*ArmonicDrive*) se encontraban defectuosas, y se procedió al cambio de estas.
- **Instante temporal en el que se da por habilitado el entorno real del Robot** (*Cableado de la maqueta, Programa del PLC Maestro,* Jaula de protección, "Botonera" de E/S y creación de la Herramienta definitiva del Robot)

1. 2. Cronograma de Gantt de las Tareas e Hitos

Nombre	Fecha de inicio	Fecha de fin
Asimilación de RobotStudio y el Entorno Integrado de Programación	25/09/17	18/10/17
Tutoriales: Asistidos	25/09/17	13/10/17
Tutorial-01: Definir_Estación	25/09/17	25/09/17
Tutorial-02: Creación de Objetos: MultifunctionBox	26/09/17	26/09/17
Tutorial-03: SmartObject_Inicios: Conveyors + Sensores	2/10/17	2/10/17
Tutorial-04: I/O's + SmartObjects	3/10/17	5/10/17
Tutorial-06: Primera Herramienta:Tool-RollCraw	13/10/17	13/10/17
Análisis_de_Resultados	27/09/17	18/10/17
Primer_Programa	27/09/17	28/09/17
Estudio: Sistemas Paralelos (MultiControlador) (1)	9/10/17	11/10/17
Estudio: S.P. (MultiControlador) (2) + DetectorColisiones (1)	18/10/17	18/10/17
Hitos_Alcanzados	29/09/17	18/10/17
Hito: Ejercicio02: Escribir mi nombre en CUBOS propios!! (v1)	29/09/17	29/09/17
Hito: Ejercicio03-3Cubes_RUN_Oct_FlexSpendant (v1)	3/10/17	3/10/17
Hito: Ejercicio04_Robot_Claws_Generator-3-Diferents_Pieze (v1)	16/10/17	16/10/17
Hito: Ejercicio05_Concurrent_System_With-PLC_2-Robots (v0)	19/10/17	19/10/17
Evaluación de RobotStudio: Análisis de Funcionalidad	23/10/17	10/01/18
Estudio y Análisis de Funcionalidad	24/10/17	28/12/17
Estudio: PLC-Controlador IRC-5 (v1)	24/10/17	25/10/17
Estudio: PLC-Controlador IRC-5 (v2)	26/10/17	26/10/17
Estudio: Funciones Varias: Normativa	31/10/17	7/11/17
Estudio: PLC-SmartObjects (v1)	14/11/17	16/11/17
Estudio: Cálculo Tc +Cambio Herramienta de Robot	20/11/17	20/11/17
Estudio: Herramienta con CollisionSensor	14/12/17	28/12/17
Estudio: PLC 16 STATES-SmartObject	21/12/17	22/12/17
Depuración_de_Documentos_Ficheros	23/10/17	29/12/17
Depuración de Herramienta Garra (v2)	23/10/17	23/10/17
Depuración: DefinitiveTool + Controlador (S.C.)	21/11/17	23/11/17
Programa: Comparación de Tools	15/12/17	15/12/17
Estación: Completa	28/12/17	29/12/17
Hitos_Alcanzados: ERAF	27/10/17	28/12/17
Hito: PLC-16_Concurrent_States-Controlador IRC-5	27/10/17	27/10/17
Hito: PLC-8_Concurrent_States-SmartObject (v1)	17/11/17	17/11/17
Hito: DefinitiveTool + Controlador (S.C.)	24/11/17	24/11/17
Hito: SmartObject: PLC 16 STATES	25/12/17	25/12/17
Hito: MHL02-1S_SR8: Definitiva	29/12/17	29/12/17
Estudio: Funciones Varias: PLC-Gestor de Eventos	8/11/17	10/11/17
Fracaso: Vía Muerta: EventGestor Null	13/11/17	13/11/17
Problema Conexión Servidor con RobotStudio: Fallo Hardware Intel y AMD	8/01/18	10/01/18
Desarrollo de la Memoria del TFG	28/09/17	1/02/18
Desarrollo de la Memoria y Evaluación de Resultados	21/11/17	30/01/18
Desarrollo_de_Ficheros_Específicos	28/09/17	22/11/17
Memoria: Desarrollo de la Documentación	21/11/17	1/02/18
Memoria (v1.1): Enfoque y Trazabilidad de Documentación	21/11/17	24/11/17
Memoria (v1.2): Análisis de la Herramienta + Cronograma	27/11/17	28/11/17
Memoria (v1.3): Aspectos Tecnicos: Creación Objetos RobotStudio	4/12/17	8/12/17
Memoria (v1.4): Aspectos Técnicos: Limitaciones Robotstudio	11/12/17	13/12/17
Memoria (v1.5): Comparación de Tools	15/12/17	18/12/17
Memoria (v2.1): Reestructurada	2/01/18	9/01/18
Memoria (v3.1): Reestructurada	15/01/18	19/01/18
Memoria (v3.2): Reestructurada	23/01/18	24/01/18
Memoria (v4.1): Correcciones	25/01/18	1/02/18
Inserción del ABB IRB 120 en una Célula de fabricación flexible:	15/11/17	22/01/18
Tiempo Teórico: Inserción del ABB IRB 120 en una Célula:	15/11/17	22/01/18
Inserción del ABB IRB 120 en una Célula de fabricación flexible:	15/11/17	19/01/18
Retraso: Problemas en la habilitación del entorno físico del Robot	15/11/17	22/12/17
Introducción al Robot Real	3/01/18	3/01/18
Trabajo sobre Célula Real	9/01/18	11/01/18
Instalación Real Completa: SFC del PLC Maestro	22/01/18	22/01/18
Estudio: Relativización de Localizaciones: RAPID	9/01/18	12/01/18

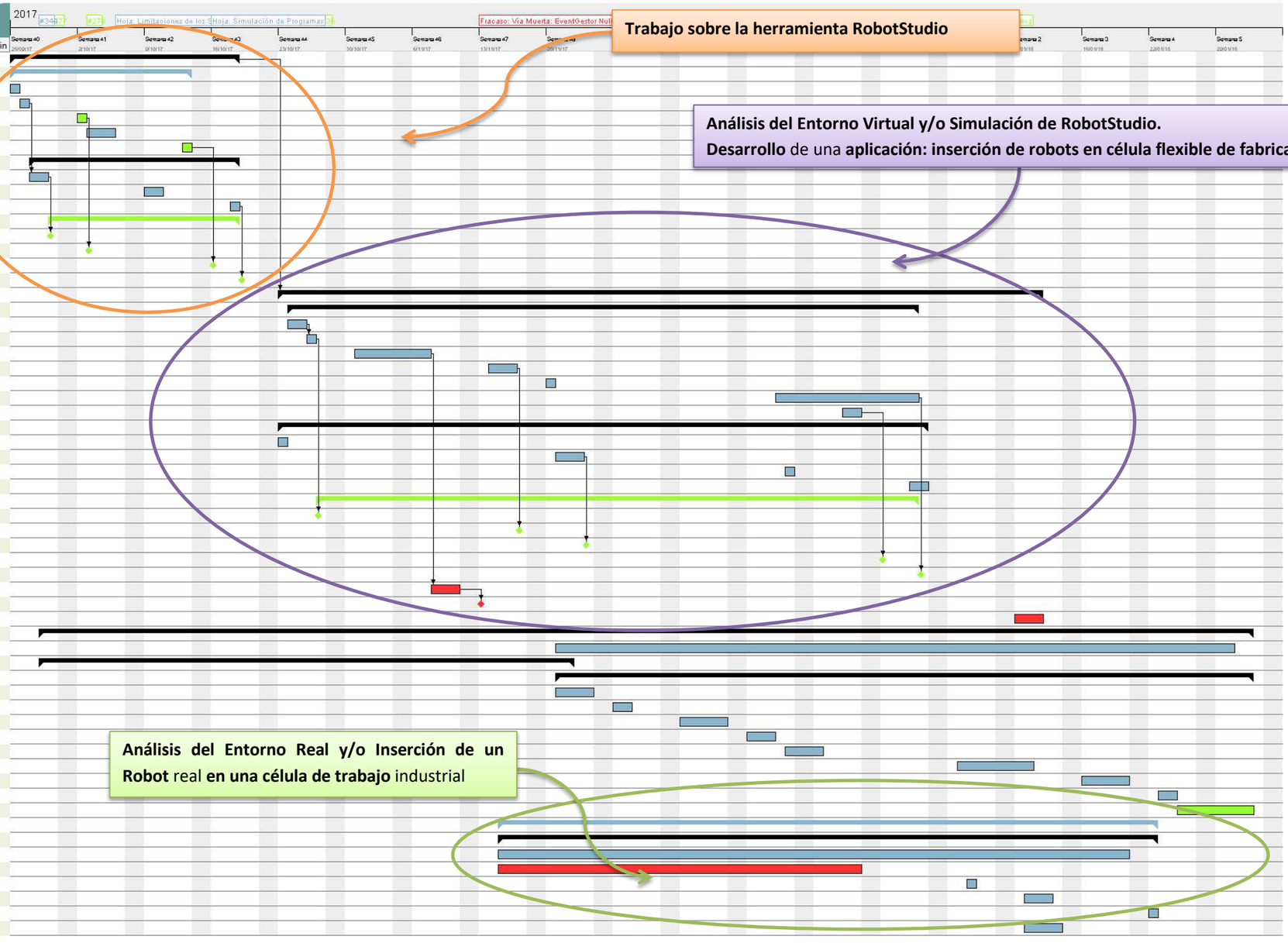


Figure 58: Cronograma de Gantt Completo

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

ANEXO II

2. "Compendio de Consideraciones, Limitaciones y Observaciones varias de RobotStudio"

Índice: ANEXO II

2. 1. Introducción al programa 4

2. 2. RobotStudio: Herramienta de Modelado de Entornos Virtuales.....5

2. 2. 1. Consideraciones de la Definición de un Modelo Físico 5
Observaciones de la Creación de Objetos: 6
Alternativas e Importación de Modelos 11
Consideraciones Generales o Limitaciones de la Creación de Objetos 13

2. 2. 2. Consideraciones de los Objetos Inteligentes "*SmartObjects*" 15
Consideraciones Generales de la creación de un SmartObject 15
Problemática de la Sincronización E/S 22
Uso de los Sensores en los SO 23
Alternativa a los Smart Objects 34
Consideraciones en la definición y creación de una Herramienta y su Controlador Lógico 35
Consideraciones de la creación de una Herramienta: Asistente de creación de Herramientas (Tool Creator) 35
Consideraciones de la creación de una Herramienta: Creación de un Mecanismo (*Mechanism*) 38
Consideraciones de la creación de un Controlador Lógico de una Herramienta ... 41
Consideraciones Generales de la Definición y Creación de una Herramienta 42

2. 2. 3. Consideraciones en el Proceso de Cambio de Herramientas desde RobotStudio 43
Consideraciones Generales 44
Conclusión Cambio de Herramienta simulada desde RobotStudio: 47

2. 2. 4. Consideraciones de los Detectores de Colisiones 48
Configuración Parámetros Específicos de Detección de Colisionesr 49
Alternativa a la Configuración Parámetros Específicos de Detección de Colisiones: SmartObject 51
Alternativa a la Configuración Parámetros Específicos de Detección de Colisiones: EventManager 54
Análisis de la función "Detector de Colisiones" Vs "Alternativas" 55

2. 2. 5. Consideraciones del Manejo del Tiempo en Simulación 56

Modos de Simulación de Tiempo y Configuración de Parámetros 56

Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation..... 57

Control de Tiempos Específicos en Simulación de Programas: Función Clock's RAPID: Cálculo de Tiempos de Ciclo de Funciones (Tc)..... 58

Conclusión del Control de Tiempos con RobotStudio 59

2. 3. Corolario de Observaciones Generales de RobotStudio 60

2. 1. Introducción al programa

RobotStudio es una aplicación de PC destinada al modelado, la programación fuera de línea y la simulación de células de robot.

RobotStudio **permite trabajar**:

a. Con un **controlador fuera de línea**, que constituye un controlador **IRC-5 virtual** que se ejecuta localmente en su PC, conocido cómo **Controlador Virtual (VC)**.
b. Con un **controlador IRC5 físico real**, que simplemente se conoce como el controlador real.

Posee varios modos de funcionamiento:

- "**Modo Online**": Conectado directamente sobre controladores reales.
- "**Modo fuera de línea**": Cuando se trabaja contra un controlador real sin conexión o mientras se está ejecutando/trabajando con un controlador virtual.

El "Modo fuera de línea" permite trabajar de forma deslocalizada, desde cualquier terminal en el que esté instalado RobotStudio.

Según los propios técnicos de ABB, el entorno de programación de RobotStudio proporciona las herramientas para incrementar la rentabilidad de un sistema robotizado mediante tareas como formación, programación y optimización, sin afectar la producción, lo que proporciona algunas ventajas: reducción de riesgos, arranque más rápido, transición más corta e incremento de la productividad.

Se ha construido basado en el VirtualController de ABB, una copia exacta del software real que hace funcionar el Robot por medio del Controlador IRC5. Ello permite simulaciones "*realistas*", con archivos de configuración y programas de robot reales e idénticos a los utilizados en su instalación.

Para más información se recomienda consultar la página oficial de ABB:

- Url: http://new.abb.com/products/robotics/es/robotstudio

DataSheet RobotStudio: **RobotStudio_5_Datasheet**

Se ha evaluado RobotStudio en sus 2 vertientes: "Herramienta de modelado de entornos virtuales" y "Herramienta de programación del robot".

A pesar del gran número de manuales facilitados por ABB, tanto en número de ediciones como en cantidad de contenido (*de 300 a 1.700 hojas*), se ha echado en falta un mayor nivel de profundidad del contenido, así como la justificación del cómo se han implementado cada una de las *Part*es.

Por ello, se ha intentado enfocar el análisis considerando los posibles problemas y/o consideraciones que debiera tener un usuario que se inicie en la herramienta.

2. 2. RobotStudio: Herramienta de Modelado de Entornos Virtuales

La **consecución de los puntos analizados** responde al **orden de aparición de las dudas en el desarrollo de creación de un fichero/programa completo**.

2. 2. 1. Consideraciones de la Definición de un Modelo Físico

Durante la programación o simulación es necesario usar modelos virtuales de piezas, de trabajo y/o equipos. Para ello, **RobotStudio facilita bibliotecas y/o geometrías propias**. Por medio de una **herramienta de diseño 75**[23], que permite elaborar modelos geométricos simplificados.

Otra **opción de trabajo** es **importarlos**, tanto **como geometrías como** en forma de **biblioteca** (*fichero externo asociado*), en caso de disponer de los **modelos de CAD de las piezas**, ver **Tabla 9: Extensiones Importación de Geometrías y Bibliotecas**.

Si se desea saber cómo crear un objeto geométrico ir a: **ANEXO II** a la **Ficha de Ruta: "Cómo hacer un Objeto"**.

[23] **CAD**: Computer-Aided Design (CAD): diseño asistido por computadora (DAC)

Para el análisis de este punto, se ha creado un modelo pseudo-complejo:

- *Part* compuesta por 2 *Body*: "*Part* (Cubo_con_Puntero_Saliente):
  - *Body*1(Base_Agujereada)
  - *Body*2 (Cono)"

Y así poder ser utilizado para explicar "curiosidades" de RobotStudio:

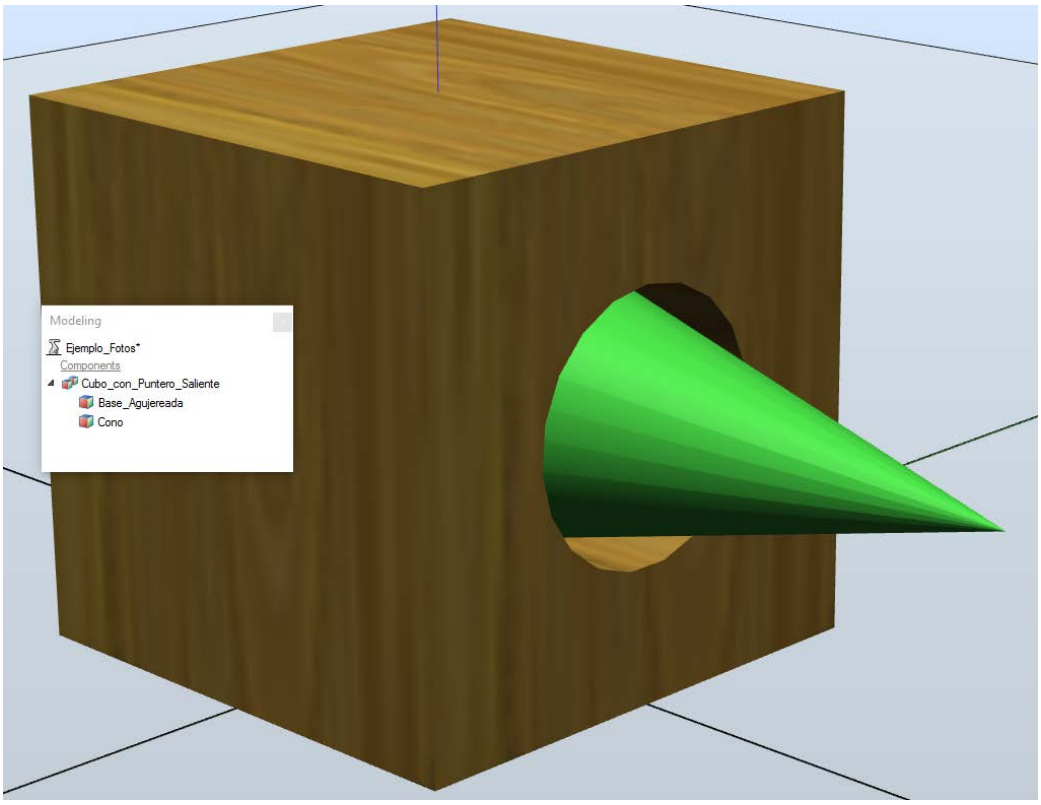


Figure 59: Ejemplo Modelaje

Observaciones de la Creación de Objetos:

A) Construcción de la Geometría

Los **Modelos Físicos**, o representaciones parametrizadas de objetos físicos reales, **están definidos por** un **nodo superior** de la geometría, denominada "Pieza" (*Part*), y por uno o varios "Cuerpos" (*Body*), que pueden ser de los tipos: sólido, superficie o curva.

RobotStudio, además de permitir crear cuerpos, facilita **herramientas para asociar propiedades y/o interacciones** entre los *Bodies* de una misma *Part* (Union, substract e Intersection) y/o entre *Bodies* de *Part*s diferentes (añadiendo, trasladando o eliminando *Bodies*).

- La **aplicación de estas interacciones *SE DEBE HACERDE 2 EN 2***

a) Geometría matemática:

Su representación gráfica, se genera a *Part*ir de la representación matemática al importar la geometría desde RobotStudio. A *Part*ir de ese momento, la geometría recibe la denominación de "*Part*".

a. **El nivel de detalle gráfico es configurable**, **se reduce el tamaño de archivo**, y el tiempo de representación de los modelos de gran tamaño, y **se mejora la visualización de los modelos pequeños**, que quizá desee ampliar.

b. El **nivel de detalle sólo afecta a la visualización**

c. **Los objetos NO son objetos sólidos propiamente dicho**. El software opera teniendo en cuenta las intersecciones con los planos que definen los objetos, esto tiene consecuencias en que: *SALPICA A LA DETECCIÓN DE OBJETOS CON SENSORES*.

B) Asociación de Referencias "*Frame*" al objeto

Una base de coordenadas es un **sistema de coordenadas genérico que puede usarse como referencia** a la hora de posicionar objetos.

En la creación de objetos con RobotStudio, **se dota de una referencia ("Frame[24]") tanto a la *Part* como a cada uno de los *Bodies*$_i$**, quedando "colgados" (*efecto "padre"-"hijo/s"*) cada *Body* de la *Part*.

Como se muestra en el ejemplo de la **Figure 60**: *Part* compuesta por 2 *Body*, "*Part*(Cubo_con_Puntero_Saliente):*Body*1(Base_Agujereada) y *Body*2(Cono)", ha sido creado sobre el origen y se ha sometido a una traslación x,y,z: (100,300,0), además de aplicarle un *Offset* Position sobre la *FramePart* x,y,z:(100,50,0).

[24]**Frame**: Sistema de coordenadas genérico que puede usarse como referencia a la hora de posicionar objetos

En las imágenes se muestra la **ubicación de la *FramePart* y las *FrameBody*$_j$ para cada *Body***:

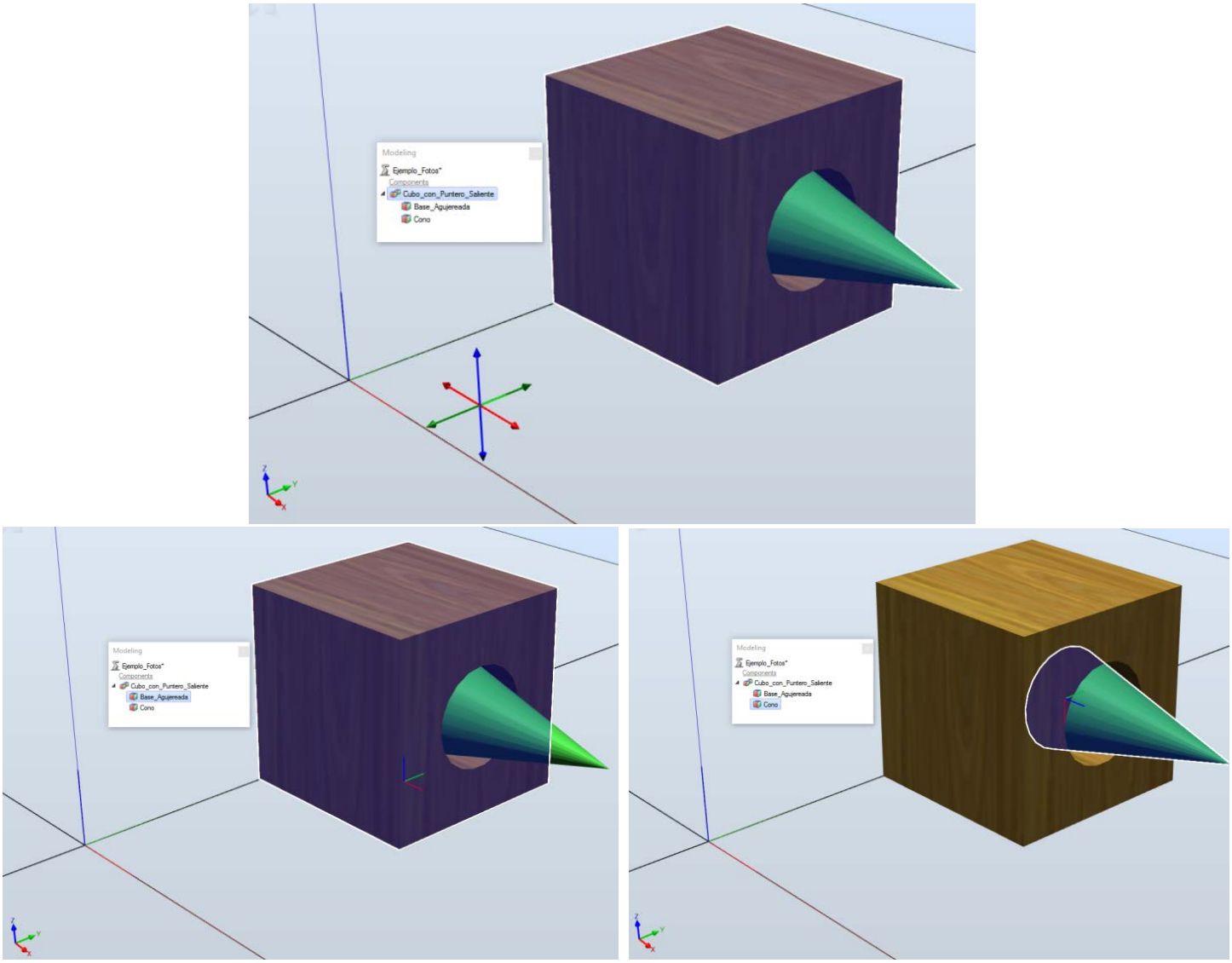


Figure 60: Ubicación de Referencias en Pieza Ejemplo

Por el contrario, **si el objeto es exportado, el software entiende que NO existen cuerpos colgando de la *Part*e principal**, provocando que **solo exista** una referencia, la ***FramePart***.

ATENCIÓN: ES **MUY IMPORTANTE la ubicación de las piezas cuando se están creando**, el *FramePart* de la pieza final se asociará directamente al origen de coordenadas de la estación.

- Al aplicar acciones de interacciones entre *Parts/Bodies*, como pueden ser las: *adition/substract/intersection*, **la pieza resultante hereda como *FramePart* o *FrameBody* EL ORIGEN DE COORDENADAS**, y será necesario eliminarlo con *Offset*s.

Se ha sometido a una acción/interacción de unión de los *Bodies* desde la ubicación trasladada, conservando los modelos originales.

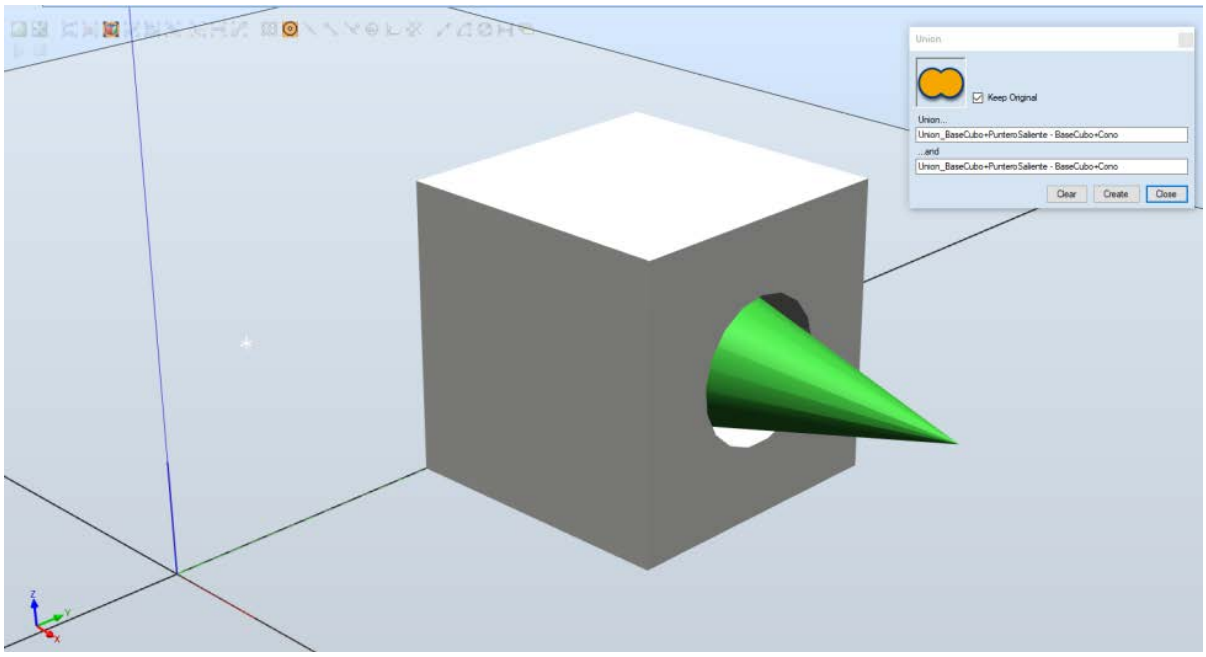


Figure 61: Aplicación de Acción de Unión de Bodies

El **resultado** es:

- Un **único cuerpo asociado a una nueva *Part***
- Se **han perdido las propiedades asociadas,** obsérvese el cambio de color de al menos uno de los *Bodies*
- Se **han alterado las ubicaciones de las *FrameBody* y la *FramePart***, ambas **reubicadas en el origen** de coordenadas como se ven en las imágenes.

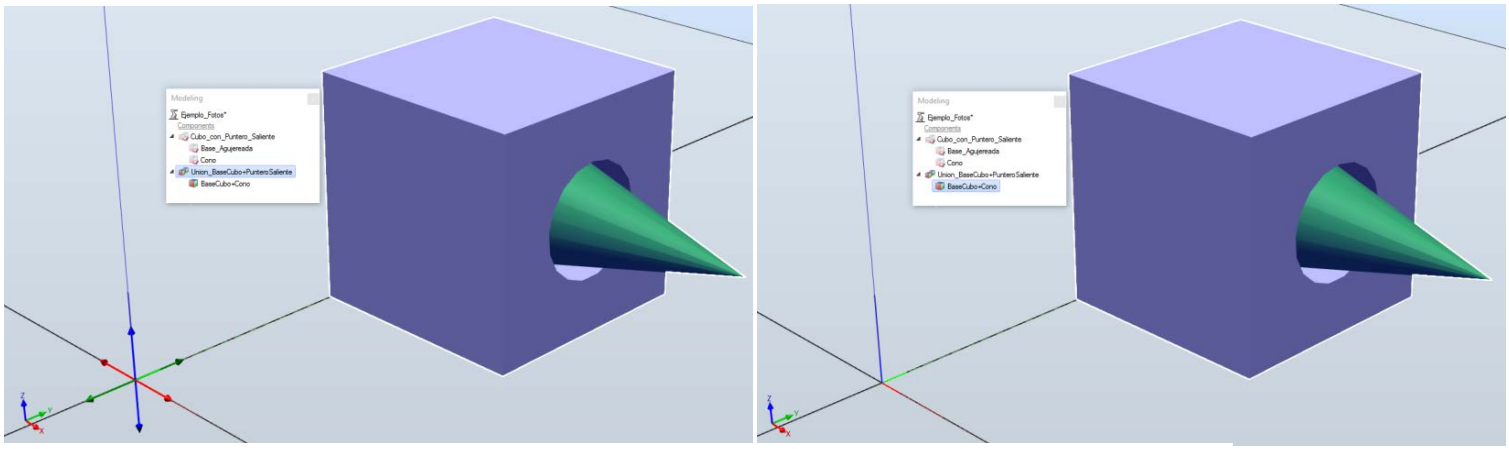


Figure 62: Ubicación de FramePart y FrameBody pieza resultante de Acción de Unión

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

a) Consideraciones de las referencias "*Frame*"

a. **Cada objeto posee un sistema de coordenadas "*FramePart*" propio**, conocido como "Sistema de Coordenadas Local", en el que se define las coordenadas del objeto respecto del origen de la Estación.

b. **Para la manipulación de los objetos** (*traslación y rotación*) **únicamente se considera la *FramePart***. RobotStudio aplica las transformaciones pertinentes entre cada una de las *FrameBody* y la *FramePart*.

c. **IMPORTANTE**: La importación de Geometría y Biblioteca ha de hacerse antes de las traslaciones y rotaciones o tendrá como consecuencia que se arrastrará más adelante los *Offset*s que existan.

d. **Es posible corregir la ubicación de los *FramePart* y los *FrameBody***, aunque estos últimos NO poseen relevancia en la ejecución de las simulaciones. Por medio del comando "Establecer Origen Local", comando "*Offset*", se reposiciona el sistema de coordenadas local del objeto, no el objeto en sí.

e. **La modificación de los *FrameBody* pasa por un PROCESO NO INTUITIVO**: Asignar una *Part* y seleccionar modificar su *Frame* con "*Offset* Position", seguidamente en la Sub-pestaña "Modeling" pulsar sobre el *Body* que se desea modificar la *FrameBody*, se comprueba que en la Pantalla "*Offset* Position: *Name_Part*" se modifica el título por "*Offset* Position: *Name_Body*" y ya se puede proceder a modificar el *Offset* de la *FrameBody* (*inaccesible desde los Menús de Usuario*), este paso se puede ver en las siguientes imágenes:

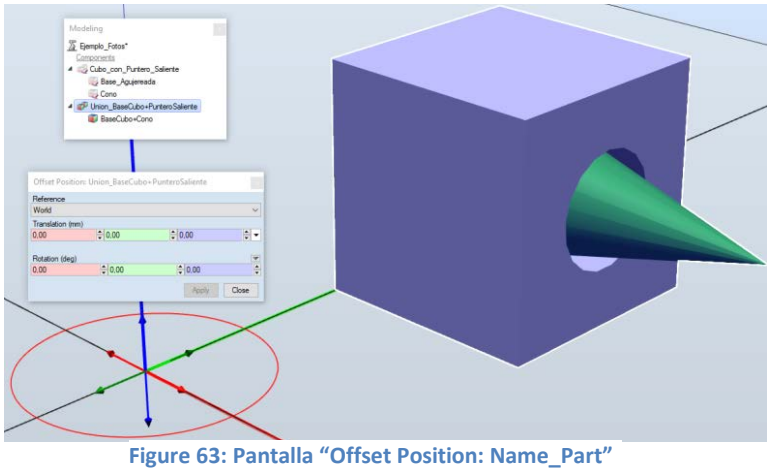


Figure 63: Pantalla "Offset Position: Name_Part"

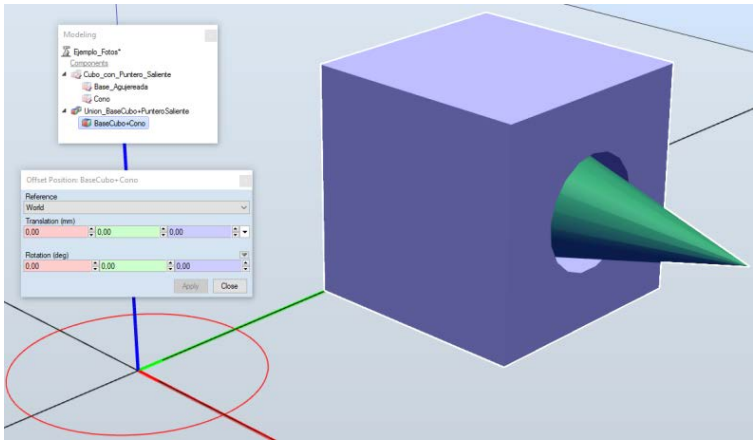


Figure 64: Pantalla "Offset Position: Name_Body"

Alternativas e Importación de Modelos

RobotStudio **permite importar los siguientes formatos**:

Tabla 9: Extensiones Importación de Geometrías y Bibliotecas

Formato	Extensión de Archivo	Opción Requerida
3DStudio	.3ds	-
ACIS	.sat, .sab, .asat, .asab	-
CATIA V4	.del, .exp, .sessionmo	CATIA V4
CATIA V5/V6	.CAT*Part*, .CATProduct,.CGR, .3DXML	CATIA V5
COLLADA 1.4.1	.dae	-
DirectX	.x	-
DXF/DWG	.dxf, .dwg	AutoCAD
FBX	.fbx	-
IGES,	.igs, .iges	IGES
Inventor,	.ipt, .iam	Inventor
JT	.jt	JT
LDraw	.ldr, .ldraw, .mpd	-
NX	.prt	NX
OBJ	.obj	-
Parasolid,	.x_t, .xmt_txt, .x_b, .xmt_bin	Parasolid
Pro/E /	.prt, .prt.*, .asm, .asm.*	Pro/ENGINEER
Solid Edge	.par, .asm, .psm	SolidEdge
SolidWorks,	.sldprt, .sldasm	SolidWorks
STEP,	stp, step, p21	STEP
STL,	stl	-
VDA-FS	vda, vdafs	VDA-FS
VRML,	wrl, vrml, vrml2	-

La columna de “Opción Requerida” responde a módulos “extra” que se ha de contratar a ABB.

- **RobotStudio conserva estructuras de montaje en la pieza de CAD importada**.
- **La importación puede requerir bastante tiempo** en las piezas con muchas entidades. Para solucionar este problema, en la pestaña Inicio haga clic en: Importar geometría y, a continuación, seleccionar Convertir geometría de CAD en una sola pieza.

C) Librery VS Geomtry (Export Geometry): Memoria del Programa

Los objetos importados desde una estación pueden ser geometrías o bibliotecas.

a. Las **geometrías** son **básicamente archivos de CAD** que, al importarlos, se copian a la estación de RobotStudio (Geometry*Name*.sat).

b. Las **bibliotecas son objetos guardados desde RobotStudio como archivos externos**. Al importar una biblioteca se crea un enlace entre la estación y el archivo de biblioteca. Por tanto, el archivo de la estación no aumenta de tamaño como ocurre al importar geometrías (Library*Name*.rslib).

Además, aparte de datos geométricos, **los archivos de biblioteca pueden contener datos específicos de RobotStudio**. Por ejemplo, al guardar: herramientas "Tool", mecanismos "Mechanism", Objetos inteligentes "SmartObject"… como bibliotecas, los **datos asociados** a la herramienta **se guardan junto con los datos de CAD**.

Las **bibliotecas importadas permiten un nivel de edición escaso, carecen de la información asociada a la pestaña "Modelado"**. Esto se traduce en que no existen los *Bodies* de las *Part*s que lo componen ni los *FrameBody* asociados. Para acceder a ese nivel de edición/modelado es necesario "romper el vínculo" y volver a reestablecerlo/guardarlo posteriormente, lo que supone perder las ventajas que se asocian al uso de librerías. Mientras el vínculo esté roto, automáticamente se generará una copia del contenido de la biblioteca en la estación.

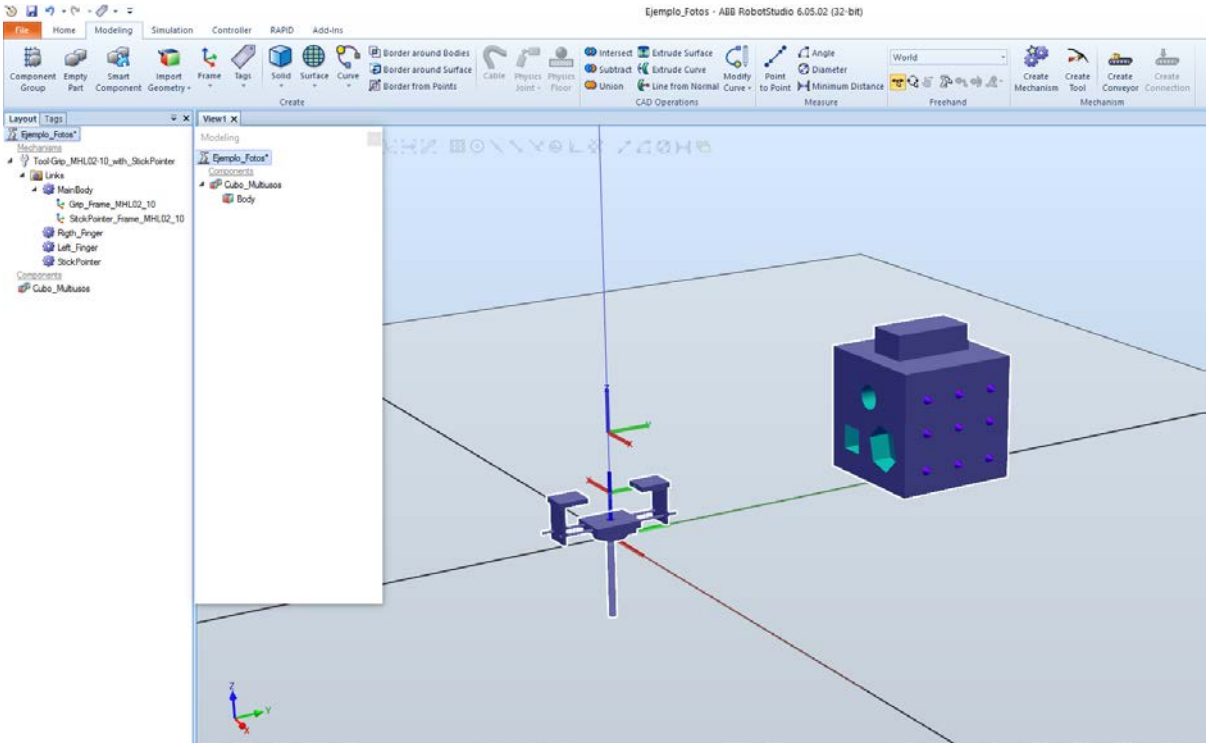


Figure 65: Ejemplo de Bibliotecas

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Consideraciones Generales o Limitaciones de la Creación de Objetos

a) RobotStudio **NO ES UNA HERRAMIENTA DE MODELADO**: a pesar de facilitar herramientas para modelar geometrías sencillas, o de complejidad moderada, acaba siendo un procedimiento costoso en tiempo y limitado en posibilidades.
Si se desea entrar en detalles de diseño visual es necesario recurrir a herramientas externas indicadas en la **Tabla 9**: Extensiones Importación de Geometrías y Bibliotecas.

b) **ATENCIÓN: Es necesario SER MUY RIGUROSO en el PROCESO DE CREACIÓN DE OBJETOS**: **NO PERMITE CORREGIR**. Si se selecciona "eliminar originales" nos encontramos ante **UN PROCESO IRREVERSIBLE**.

c) **ATENCIÓN: NO ES POSIBLE modificar propiedades geométricas una vez "creado" un objeto**, por muy simple que sea el objeto: "¡NO PERMITE ESCALAR OBJETOS!"

d) **Se recomienda NO ASIGNAR atributos a las piezas inacabadas**: CONSECUENCIA: cuando se aplican acciones de interacción entre *Parts*/*Bodies* se restaura por defecto las propiedades de los materiales asignadas. Ver **Figure 61: Aplicación de Acción de Unión de *Bodies***.

e) En la creación de objetos complejos **se recomienda MINIMIZAR el USO DE ACCIONES DE INTERACCIÓN entre *Bodies*/*Parts*, es preferible que una *Part* esté compuesta por un conjunto de *Bodies***, facilita la edición de objetos y posibles modificaciones futuras elevando la complejidad del objeto futuro. Ver **Figure 59: Ejemplo Modelaje** (*Ejemplo de una Part con 2 Body*) y **Figure 60: Ubicación de Referencias en Pieza Ejemplo** (*Ejemplo misma Part y 1 único Body combinación de los dos anteriores*)

f) En la **creación de objetos con Grados de Libertad (GL)** (*traslación y/o rotación*) **ES NECESARIO definir todos los distintos *Bodies*, que componen una única *Part*, para cada sub-objeto que posea GL**, así como otro sub-objeto que funcione como base.

g) **Se recomienda guardar en formato biblioteca los objetos complejos considerados como objetos finales**. Y guardar como geometrías todos los objetos que se prevea volver a editar o completar en detalle (Efecto copia de seguridad).

h) **ATENCIÓN**: **Las Herramientas "Tools" exportadas en formato biblioteca y desacopladas: NO SON editables desde la pestaña de Modelado**: ¡*PART* y *BODIES* inaccesibles!

i) **ATENCIÓN**: Al **guardar un objeto como biblioteca asocia la "*Frame*0" como la *FramePart***.

j) **Para copiar *Part*s o *Bodies* se recomienda** guardar el original en una biblioteca o fichero de exportar geometría **e importar las copias desde las bibliotecas**. Otra opción, aunque aumentará el tamaño del fichero de la estación es Copy-X:Paste-X, **SOLO ENTIENDE EL PEGAR-X si se selecciona LA ESTACIÓN** (*pinchar sobre el nombre de la estación en la pantalla izquierda ya sea desde la Pestaña "Home" o desde "Modeling"*), **no sirve seleccionar la información que contiene la Estación**. Ver **Figure 65: Ejemplo de Bibliotecas** (*LA ESTACIÓN se encuentra subrayado en azul en la ventana izquierda*)

k) **ATENCIÓN**: Los **objetos NO responden a las leyes físicas**. Se consideran **"Sólidos Rígidos Indeformables y Penetrables"**. "AFECTA A LA DETECCIÓN DE COLISIONES y PRECISIÓN EN LAS SIMULACIONES/FIDELIDAD RESPECTO DE LA RESOLUCIÓN REAL".

l) **Se recomienda** desde ABB **el uso de "Modelos Simplificados NO SATURADOS"** de detalle: "PROBLEMÁTICA EN SIMULACIÓN: Aparición de "Lag" en la simulación y problemática en la gestión /actualización de E/S".

m) **No es posible seleccionar las piezas "*Bodies*" que formen *Part*e de una biblioteca o un mecanismo**.

2. 2. 2. Consideraciones de los Objetos Inteligentes "*SmartObjects*"

Un componente inteligente **es un objeto de RobotStudio, con o sin representación gráfica, que representa el comportamiento de otros componentes "inteligentes"** presentes en el mundo real.

La complejidad del objeto inteligente queda supeditada al nivel que el usuario desee implementar y a las limitaciones propias del software **y procesador utilizado**.

Sí se desea saber cómo crear un objeto geométrico ir a: **ANEXO III** a la **Ficha de Ruta: "Cómo hacer un SmartObject**".

Consideraciones Generales de la creación de un SmartObject

a. Es una **alternativa al uso de un compilador de bibliotecas basado en texto estructurado, XML**. *Obsérvese que puede implementarse mediante la clase code-behind (xml based library compiler: .NET class)*.

b. **Es posible dotar de protección a un componente inteligente**: ocultar su estructura interna y protegerlo contra la edición, protegiendo su funcionalidad.
   - Los componentes subordinados, de un componente inteligente protegido, permanecen ocultos en todos los navegadores de RobotStudio, incluido el "Analizador de señales".
   - Esta opción, es una forma de ocultar la complejidad, **no tiene como fin aportar seguridad ni protegerlo de forma inalterable**.

f. **RobotStudio NO permite una edición plena de los componentes/bloques inteligentes** en la implementación de *SmartObjects*, **NO SE PUEDE ASIGNAR NOMBRES a voluntad para cada bloque**, entre otras especificaciones/atributos, diferencia entre bloques "editables" (*los que aparecen en el árbol de la Estación*) y "bloques por defecto" (*el resto*) cuya programación es mínima, **ejemplo puertas lógicas**.
   - **Problema**: limpieza visual en la implementación de objetos inteligentes corregidos/modificados, nuevamente invita a que lo hagas bien a la primera y sin corregir.

c. **ATENCIÓN**: Si se requiere **extraer y/o usar información analógica**, de un bloque "Expression()" por ejemplo, es **necesario tratarla posteriormente con un bloque "*Converter*"**. De cualquier otra forma NO servirán os datos analógicos extraídos por el SmartObject, *ver ejemplo variable output "Length_Object" de la Herramienta implementada en* **ANEXO V**.

d. **Los bloques que dependen de una función de Activación**, como por ejemplo los Sensores, elementos de información de avance de articulaciones o bloques de cálculo de funciones, se encuentran con las siguientes **limitaciones**:

- **Activación manual**: Si se coloca en activado por defecto, un 1 para siempre activo o un 0 para siempre apagado, al iniciar una simulación nueva volcará la información del último estado alcanzado en la última simulación corrida. **Problemática de "refrescado de bloques"**, directamente proporcional a la cantidad de bloques con activación que se hayan implementado, será **necesario configurarlos manualmente en el estado inicial** (*para cada uno de ellos: activar-desactivar-activar*) **antes de cada simulación**.
- **Activación Automática**: por lógica del SmartObject o la Estación:
  - **Refresco condicionado a activación**: Mismo problema de memoria del último estado como estado inicial. **Problemática en robustez de lógica programada**, configuración inicial distinta a la esperada y distinta en cada ocasión que se lance la simulación.

    Ejemplo: limpieza del contenido de la SensedPart() de un Sensor que solo se actualiza cuando se alimenta una entrada del SmartObject. Los sensores solo actualizan el estado del SensedPart() cuando existe una variación en el plano/s del sensor. **En el ejemplo**, para una entrada desactivada y ningún objeto tocando el sensor, el sensor detecta SensedPart($Body_i$) de cuando detecto el $Body_i$ antes de ser desactivado el sensor y posteriormente retirado el $Body_i$. Generando problemas nuevos en la lógica programada aguas abajo, como **por ejemplo** enlazara "Attach" el objeto sensado por el Sensor a la herramienta: Se estaría enlazando un objeto que NO se encuentra ubicado en el sensor cuando este vuelve a ser alimentado.
  - **Refresco por Bit Específico**: El usuario fuerza la actualización de los bloques, ya sea manualmente o por código RAPID, y, por consiguiente, de los campos vulnerables a memoria de último estado: SensedPart(), *Mechanism*(), *Expression*(),…

    Es una alternativa que complica la utilización de los SmartObject por terceros, a pesar de dar muy buenos resultados, al poder controlar la actualización del estado de los bloques a voluntad del usuario, en paralelo con la lógica implementada. *Ver ejemplo variable intput "Update_Sensor_Conection" de la Herramienta implementada en ANEXO V, se ha implementado la lógica del SmartObject intentando contemplar todos los estados posibles y su consiguiente actualización de bloques susceptibles a Activación. A pesar de ello si el usuario quiere refrescar los bloques se le da permiso para hacerlo por medio de la variable de entrada*.

- **Refresco continuo**: equivale a dejar anclado el bit de activación manualmente. Consume muchos recursos y puede salpicar a la velocidad de simulación.
  Única que garantiza eliminación de la memoria del último estado de la última simulación, refleja mejor la realidad inicial o instantánea de la Estación al lanzar la simulación.
  RobotStudio NO PERMITE implementar un 1 ó 0 lógico en la creación de los SmartObject, *se ha optado por la implementación de un 1 lógico con una puerta NOT y una OR sobre una de las E/S del SmartObject, y un 0 lógico con una NOT y una AND.*

e. **ATENCIÓN**: **La ejecución de los eventos internos se resuelve de forma atómica y asíncrona**. Si se desea implementar programas lógicos procedurales hay que dotar al sistema de pequeños retrasos de tiempo o "*Delay*s" y poder actualizar el estado de las E/S. Entre las opciones disponibles están:
   - **"*Delay*" con Temporizadores**: Lógica complicada y no intuitiva, ni en implementación ni en lectura.
   - **"*Delay*" con puertas "OR"**: Se propone utilizar el "*Delay*" a la conexión del bloque OR, visualmente es más cómodo y de programación directa.
   - **El "*Delay*" mínimo,** que se ha podido **implementar en el SmartObject** para garantizar una buena comunicación con la Estación y actualización de estados propios, **es de 1ms**.

g. RobotStudio **NO PERMITE ORDENAR visualmente la lógica cableada**, o conexiones entre bloques, **solo permite situar los bloques sobre una plantilla blanca 2D**. Considera que su solución es la idónea, aunque cruce cables o los ponga todos por la misma línea.
   - **Problema 01**: **Limpieza visual** en la implementación de objetos inteligentes corregidos/modificados, nuevamente invita a que lo hagas bien a la primera.
   - **Problema 02**: La "maravillosa" **gestión del Zoom y reordenación de cableado y bloques**, todo el zoom lo hace sobre el mismo cuadrante, *Part*e central superior (*debajo del nombre principal*) obligando a tener que usar las barras de desplazamiento vertical y horizontal con el zoom ya activado, sumado a que si te dejas pulsado un bloque al hacer un zoom el bloque te lo recoloca donde le place. Función más que necesaria cuando se trabaja con *SmartObjects* ya "grandecitos" como el del ejemplo de la **Figure 66**.
   - **Problema 03**: **Ocultar el cableado NO ES UNA OPCIÓN**, **no aparecen etiquetas que aclaren/especifiquen los puntos interconectados**, solo se ve un popurrí de bloques sin sentido, como se puede ver en la **Figure 68**.
   - **Problema 04**: **Función "Auto Arrange"** Esta es la propuesta de auto-colocación que proponen sobre el ejemplo de la Herramienta. A esto hay que añadir que ¡ES IRREVERSIBLE! ¡No permite volver atrás una vez ejecutada!

Utilizando el ejemplo de la Herramienta implementada se ha generado una **Secuencia Zoom. Hay que mencionar que RobotStudio no otorga el control sobre la zona resultante**, el resultado del Zoom siempre es el mismo a pesar de hacer el *Scroll* del ratón en zonas distintas de la pantalla:

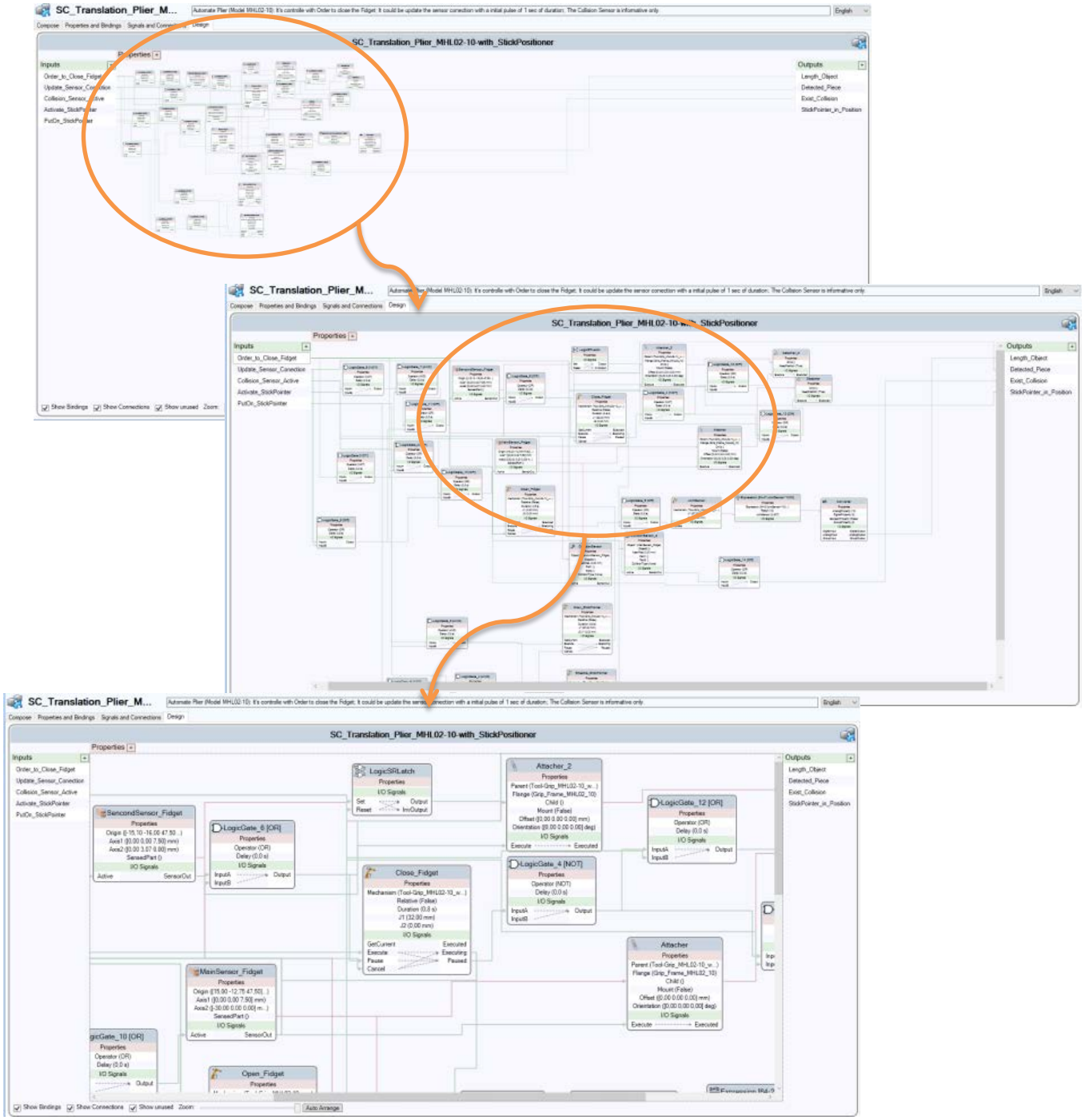


Figure 66: Secuencia de Zoom en la Pestaña de Edición de SmartObject

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

Continuando con el Ejemplo anterior, para las **Conexiones Lógicas y Efecto de la Ocultación de estas** se puede apreciar el resultado de activar la opción de ocultación y **pérdida directa de toda referencia existente**: Desaparece toda conexión entre elementos y E/S.

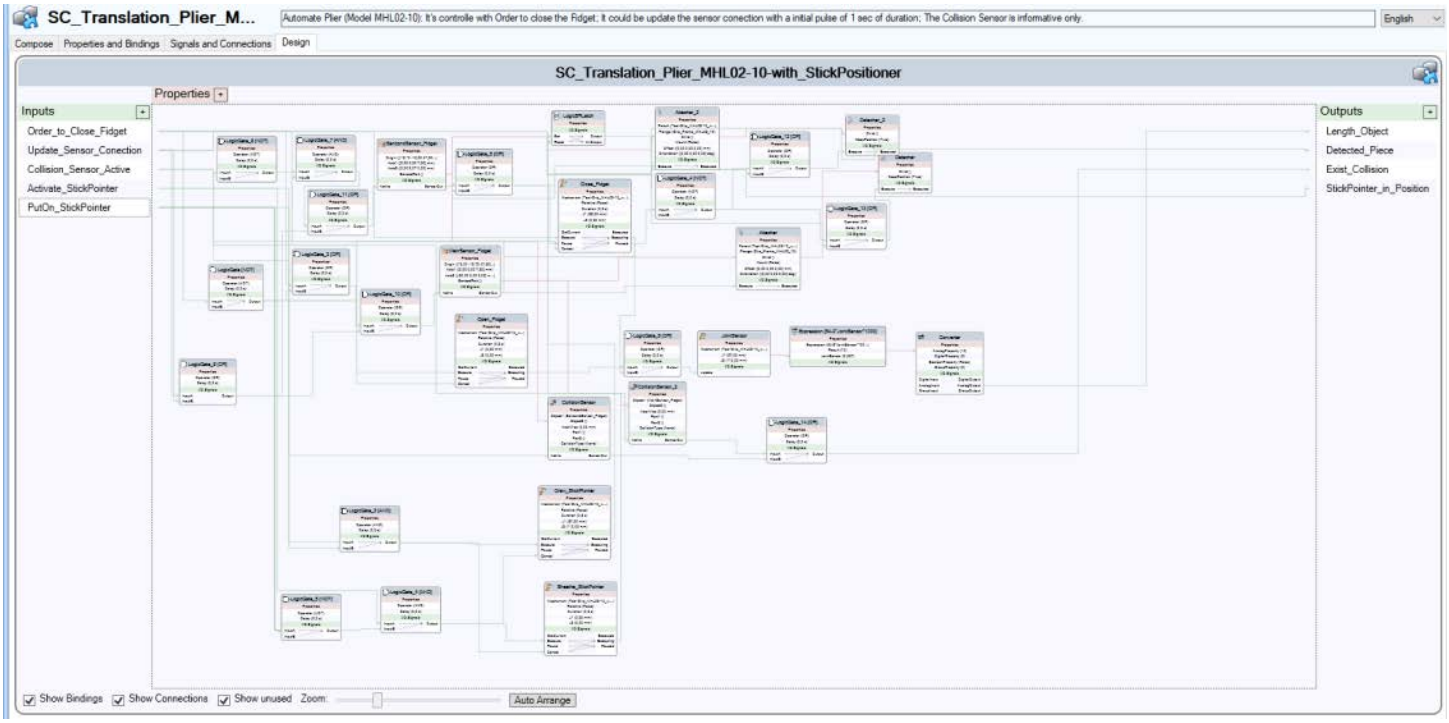


Figure 68: Pestaña de Edición y Diseño de SmartObject de Referencia

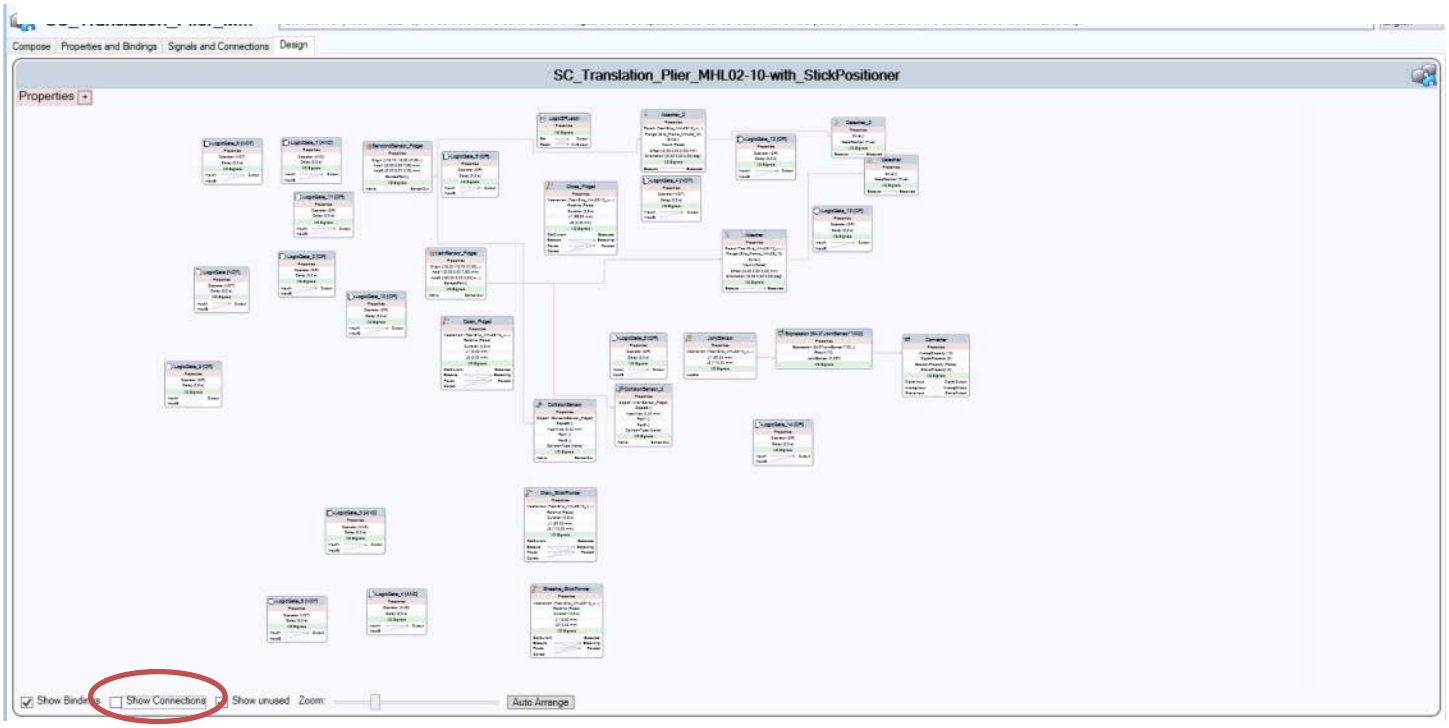


Figure 67: Activada la opción de "Ocultar Conexiones"

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

Tomando como referencia la **Figure 67**, si se activa la **Función "Auto Arrange"** se obtiene como resultado las siguientes imágenes:

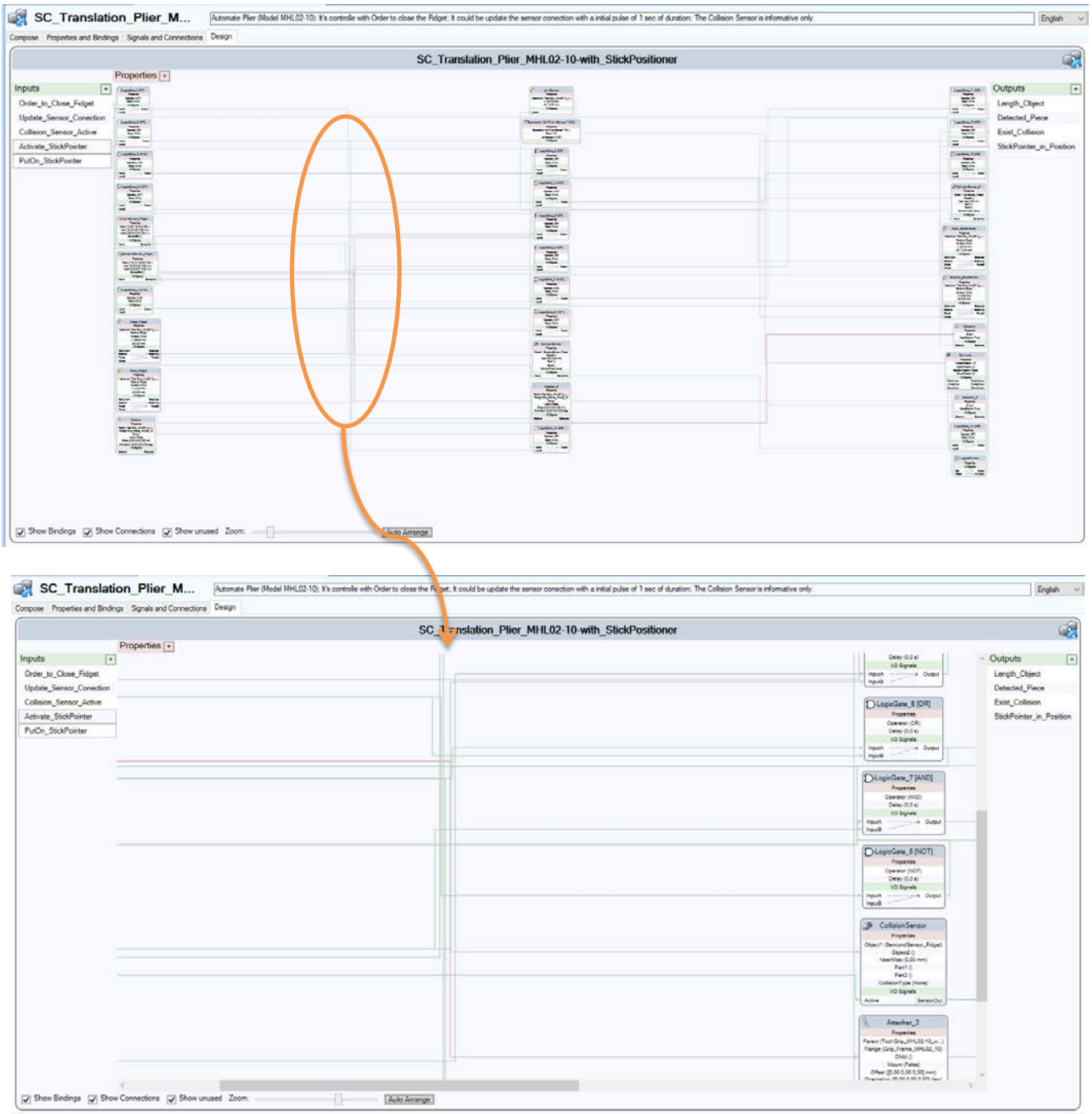


Figure 69: Activación de la Función "Auto Arrange" en un SmartObject

Imposibilidad para saber qué y cuántas conexiones pasan por el mismo cable.

h. **ATENCIÓN**: **Si se elimina algún bloque** o variable E/S d**e la Pestaña de "Design" o "Signal and Conections"** RobotStudio:

- **Eliminará las conexiones de la pestaña "Design"**, pero **NO ELIMINARÁ ninguna de las conexiones que se hubieran hecho previamente sobre el bloque** reflejadas en la pestaña **"Signal and Conections"**
- **No informará por pantalla de la existencia de conexiones residuales en el momento de eliminar el bloque**, más allá de un "**!**" en rojo en el lugar de definición de la conexión, pestaña: **"Signal and Conections"**
- Se genera el **problema "riesgo de reutilización de nombres"**. Si se crea un segundo bloque con el mismo nombre se efectuarán los links con las conexiones residuales del bloque anterior con el que comparte nombre

Tomando como ejemplo la **Figure 67**, se ha eliminado un Bloque: LogicGate_3[AND], lo que ha provocado (sin la aparición de ningún aviso por pantalla) la eliminación de los elementos de conexiones manteniendo las conexiones residuales pendientes de reasignación.

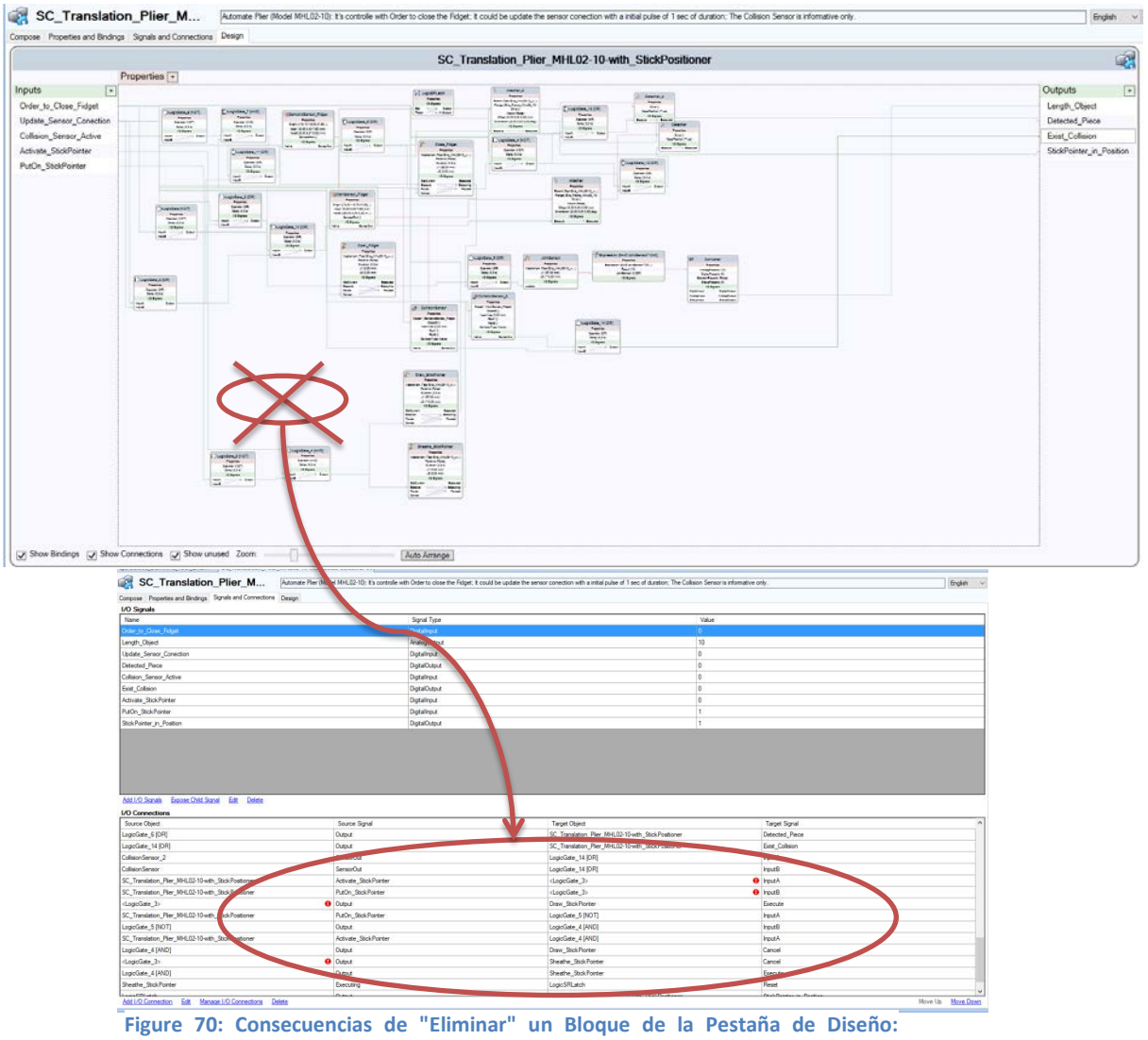


Figure 70: Consecuencias de "Eliminar" un Bloque de la Pestaña de Diseño: Aparición de "Cronembergs"

Problemática de la Sincronización E/S

En la manipulación de RobotStudio, con fin de exprimir la herramienta, se ha detectado que:

1. **Presenta problemas con la comunicación entre componentes de una misma Estación**, en la utilización de protocolos por medio de pulsos, *ejemplo implementación de un Autómata Meally*.
   Problemática: ¿Cuánto tiempo es necesario mantener una señal de salida en alto para asegurar recepción (*límite inferior*) y evitar que confunda la transición que ha de pasar de un estado a otro (*límite superior*)? **La respuesta**: Hay que mantener el pulso activo una unidad de tiempo superior al "Step de Simulación" configurado, y no más de 2 Step. **Nuevo Problema**: EL STEP ES VARIABLE/CONFIGURABLE por el usuario, y el tiempo de activación de las E/S no se puede configurar con una unidad de tiempo relativa al Step seleccionado.
   A esto se suma que RobotStudio dificulta la reutilización de bit en la comunicación, requiriendo un bit distinto para cada transición, malgastando canales/puertos en la comunicación.
   Problema: Es muy probable que **RobotStudio** haya sido implementado como **un sistema asíncrono** cuando la robótica industrial es síncrona por naturaleza.

2. **Se desconoce cómo se resuelve la ejecución de la lógica de los SmartObject y el EventManager**.
   ABB mantiene el funcionamiento de RobotStudio como si se tratase de una caja negra.
   Problemática: en la implementación de biestables lógicos ("Flip-Flop") para máquina de estados, es necesario incluir un "*Delay*" después de cada avance de estado.
   Problema: No se ha podido confirmar si al resolver la lógica se utiliza una secuenciación o una resolución paralela de golpe de toda la lógica implementada. Provocando tener que aplicar estrategias de control asíncrono.

3. **Se desconoce el tiempo de cómputo medio de los SmartObject**, no se facilitan referencias en los manuales y las funciones de control de tiempo son de precisión poco generosa (ms). Ver **Consideraciones del Manejo del Tiempo en Simulación**.
   Problemática: Cálculo de tiempo de ciclo medio de un ejecutivo cíclico programado, creación de un PLC por ejemplo y su correspondiente control cíclico.
   Problema: sincronización con otros elementos cíclicos de la Estación, asignación de "*Delays*" por procedimiento empírico.

Todos estos problemas han provocado la necesidad de implementar soluciones/"*ñapas*", que en situaciones reales NO SERÍAN NECESARIAS, **para asegurar una comunicación óptima entre distintos elementos, se han forzado sincronismos o "paradas controladas" en la ejecución de los elementos**, y su consiguiente aumento en el tiempo de ciclo medio de producción.

Estas posibles soluciones se han obtenido de forma empírica ante la falta de información facilitada por los Manuales de ABB.

Uso de los Sensores en los SO

Entendiendo un Sensor como: Dispositivo que capta magnitudes físicas (*existencia de objetos y/o propiedades específicas*) u otras alteraciones de su entorno para poder interpretar y utilizar dicha información. RobotStudio permite simular y definir las siguientes categorías:

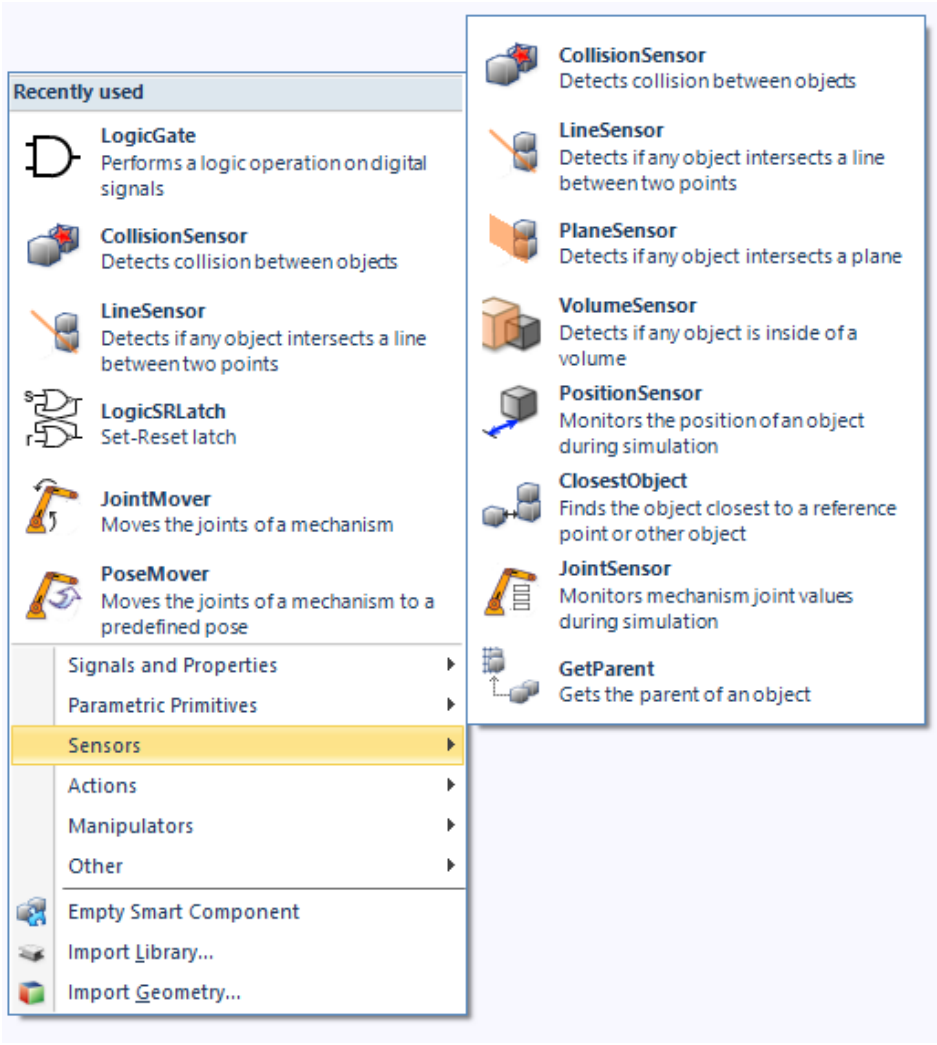


Figure 71: Tipos de Sensores que existen en RobotStudio

Para la explicación detallada de cada bloque se recomienda ver el "ManualOperdor-RobotStudio": "*Capítulo 9: Pestaña Modelado: 9.4. Componente Inteligente: 9.4.7 Componentes inteligentes básicos: Sensores: (334/648)*"

Este apartado ha sido enfocado para cumplir las necesidades de las tareas propias de detección de sólidos (*localización de ubicación o posicionamiento*), tanto para el controlador de una herramienta de agarre, como para control de ejes externos o cintas transportadoras, "*Conveyors*".

A) Diferenciar CollisionSensor vs Line/Plane/VolumenSensor

Existen algunas diferencias de funcionamiento, aunque **ninguno de estos sensores garantiza una ubicación exacta del objeto, sólo informa de que existe una intersección en algún punto del plano/s** que representa el sensor utilizado:

a) CollisionSensor:

Detecta colisiones y, si se le explicita, **eventos** de casi colisión **entre un objeto A y otro B**. Si no se especifica uno de los objetos, el otro se comprueba frente a toda la Estación.

1. La **comparación** la hace **por medio de la intersección entre el volumen de los objetos A y B**. Se posibilita la Activación/Desactivación del bloque en función de la Lógica implementada.
    a. **Representa un menor tiempo de cómputo** que las operaciones propias de los sensores convencionales.
2. **ATENCIÓN**: el sensor detecta la colisión de un objeto/pieza "*Part*" como conjunto en sí mismo, es decir detectará las colisiones entre los propios cuerpos "*Body*" que la componen en aquellos casos en que se estén tocando.
    a. Esto **genera la necesidad de crear la herramienta "*Tool*" con piezas suspendidas en el espacio y asegurar que no se encuentren superpuestas entre sí** para ninguna de las configuraciones o trayectorias posibles, ni siquiera apoyadas.
3. **Este sensor INFORMA de qué pieza está en colisión** con la pieza comprobada.
4. Es **posible comparar entre objetos específicos**, seleccionados de una lista, **u objetos relativos a una posición** en la que se encuentre un sensor identificando a estos (*a través de la conexión lógica-analógica, cable "morado"*).
5. **No posee representación gráfica en la simulación**, no se encuentran asociados a una posición espacial.

Si se desea más información del funcionamiento de un detector de colisiones o de más consideraciones a tener en cuenta para su implementación: Ver "**Consideraciones de los Detectores de Colisiones**".

b) Line/Plane/VolumenSensor

Bloques inteligentes utilizados para detectar los objetos que presentan intersección con el plano del sensor.

1. **El tiempo de computo varía** en función del tipo de sensor escogido: lineal (*inmediato*), planar (*poco*) o volumétrico (*elevado*).
   No obstante para todos ellos, es mayor que el tiempo de cómputo del CollisionSensor.
2. **Cada sensor identifica un único objeto que presente intersección con su plano** siguiendo el siguiente orden de prioridad:
   1. **Temporal** (*seleccionará el último en entrar*)
   2. **Ubicación** (*si se encuentran varios objetos detectados, el que se encuentre más cerca de la referencia del sensor*)
   3. **Tamaño** (*el más grande de todos los objetos que hagan intersección en el mismo punto, mismo principio que Ubicación*)
3. Se **posibilita la Activación/Desactivación del bloque** en función de la Lógica implementada. Mientras se encuentre en posición Activado y exista una variación del estado de las entradas, el sensor actualizará la información de "*SensedPart(*Name_object_sensed)*"
4. **Son Bloques lógicos con representación gráfica en la simulación**: una línea, un plano o un volumen.

ATENCIÓN: Si se oculta la representación gráfica el sensor deja de detectar intersecciones.

5. **Deben tener un tamaño que garantice que ocupan un espacio superior al objeto a identificar**. Existe problemas con la velocidad de simulación, velocidad relativa de los objetos y gestión de E/S.

c) Tipo de Sensor recomendado para posicionamiento y "*Conveyors*"

Para la "ubicación" de objetos en el espacio: bastaría con **sensores lineales o planares** colocados en el punto espacial en el que se desea localizar el objeto, **siempre que el sensor ocupe un espacio superior al objeto a identificar**, y que no exista más de un objeto en la misma área.

ATENCiÓN: Si en la configuración inicial hay más de un objeto superpuesto encima de un sensor y la información sensada tiene por objeto reposicionar objetos, se mandarán TODOS los objetos al punto de reposicionamiento aunque el Display del sensor solo detecte un único objeto.

En el caso de los "*Conveyors*", ya sea para utilizarlo de guía de movimiento o para indicar que el objeto ha alcanzado la situación deseada, misma conclusión que para la "ubicación" de objetos, **los sensores más idóneos son los *LineSensor* o los *PlaneSensor***.

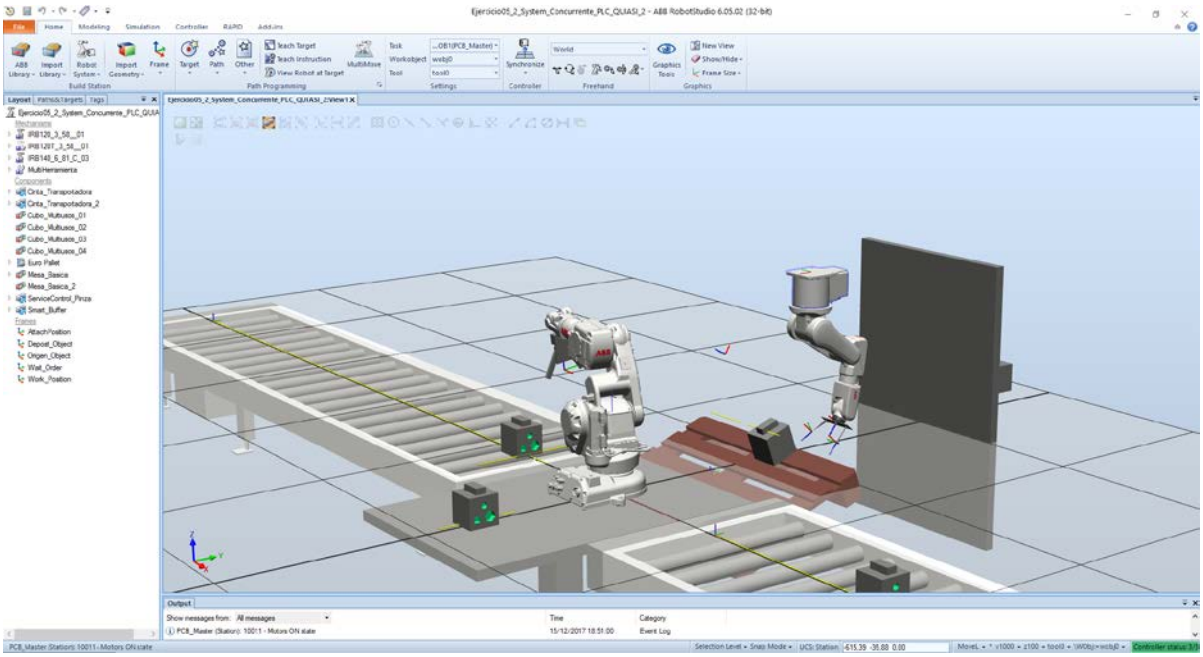


Figure 72: Sensores en *Conveyors*: Sensor Guia Movimiento

OBSERVACIÓN: El sensor lineal se ha utilizado para garantizar que el objeto identificado sea el único que se asocia al bloque de movimiento, el objeto se moverá, si y solo si, se encuentra sobre el sensor lineal.

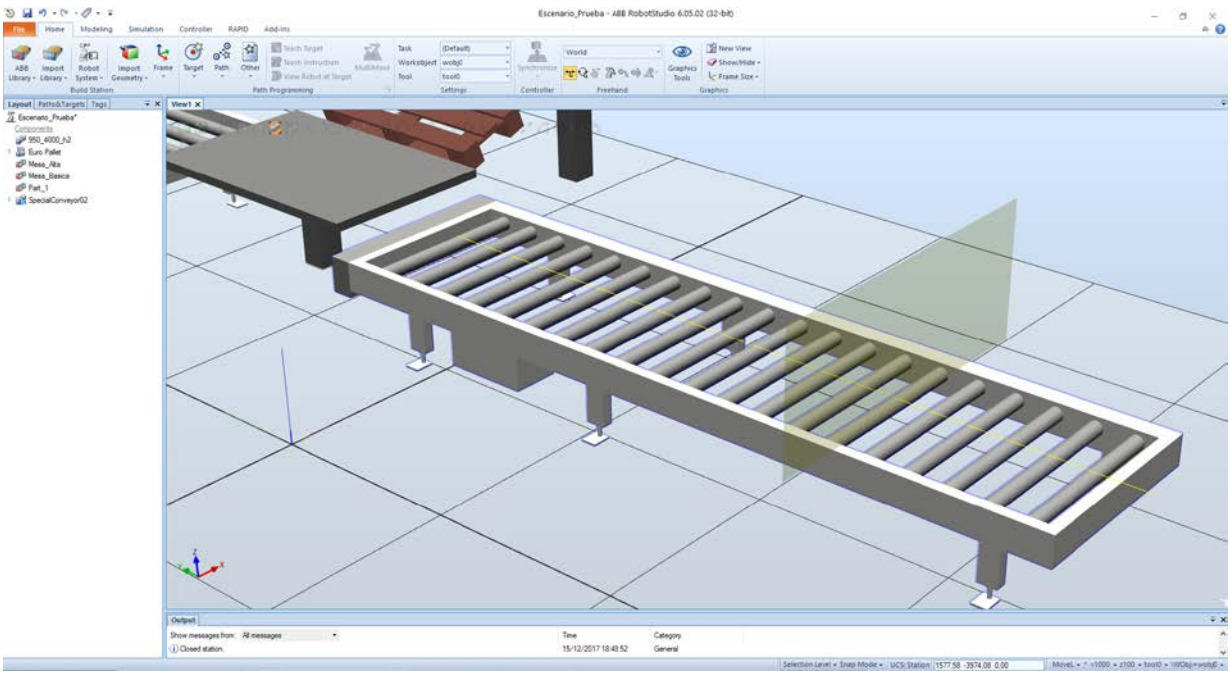


Figure 73: Sensores en *Conveyors*: Sensor Posicionamiento: Plano

OBSERVACIÓN: El sensor plano sirve para posicionar y ordenar el paro de la cinta, en modelos posteriores se ha sustituido por un sensor lineal transversal al sensor guía.

d) Tipo de Sensor recomendado para "*Tools*"

Objetivo: Utilizar la información de colisión o intersección con un sensor para comandar la orden de paro/cierre de la garra, y aplicar la acción de enlazar.

Para poder estudiar la diferencia entre CollisionSensor y los PlaneSensor, en el Controlador de la herramienta, se ha creado una herramienta con los sólidos separados entre sí, y garantizando la ausencia de cualquier posible colisión entre sus *Part*es durante las traslaciones y rotaciones. Ambas herramientas se han superpuesto en el Robot, manteniendo siempre oculta una de las dos herramientas y gestionando su operación desde RAPID y un Switch selector de herramienta.

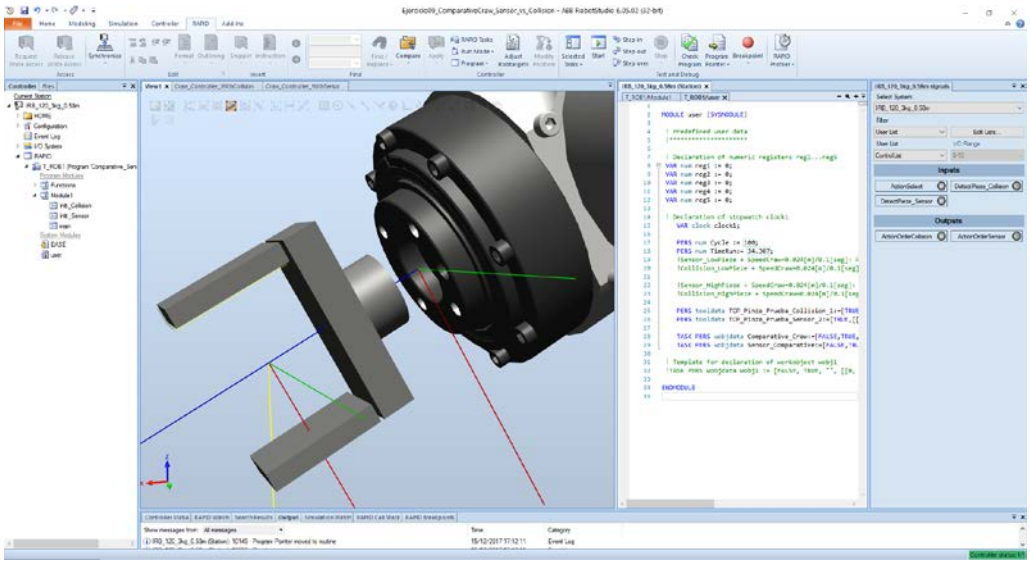


Figure 74: Herramienta de Comparación de Control por: Sensores vs CollisionSenser

i. CollisionSensor

CollisionSensor: Detecta existencia de colisión o casi colisión. Activo Siempre, aunque sólo relevante cuando se ordena cierre de garra.

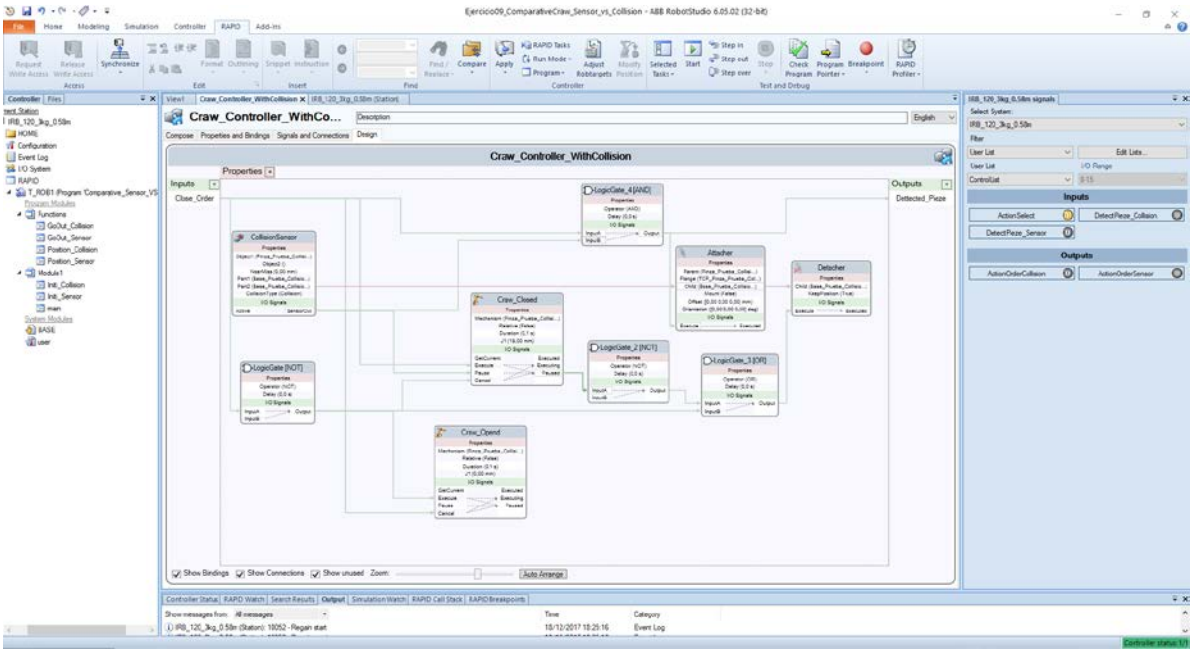


Figure 75: Control Lógico Garra CollisionSenser

Observaciones del Detector de Colisiones:

1. **Robusto a tamaño de piezas a detectar**, no necesario apoyo de sensores auxiliares para discriminar tamaños.
2. No Limpia contenido identificado por el bloque CollisionSenser
3. Menor tiempo de cómputo/ciclo
4. Consume más recursos
5. **ATENCIÓN**: La Herramienta **"Tool" tiene que ser diseñada con mucho cuidado para garantizar que los sólidos que la componen NO se toquen entre sí** tanto en estático como en trayectorias, ni siquiera la brida con el robot
6. Simplicidad de la Lógica implementada
7. Si el objeto es invisible, no lo detecta
8. **ATENCIÓN**: Si la Tool es invisible, sigue consumiendo recursos del sistema, sigue comparando/detectando. Activa salidas de la herramienta
9. "APARENTEMENTE" penetra menos en la pieza
10. **ATENCIÓN**: Si no se contempla un "*Delay*" tras parar el movimiento en la apertura de garra, aparece un FALLO ACUMULATIVO, insuficiente con la orden "fine". Efecto de recortar distancia en cada iteración. **REALIZADO VÍDEO**!!
    SOLUCIÓN: Necesario implementar un *Delay* en código RAPID, SUPERIOR AL STEP "Simulation Timestep", para que le dé tiempo a liberar la pieza del Attach!!! No basta con los comandos FINE!! (Innecesario en la realidad…)

ii. Line/Plane/VolumenSensor

Sensor Principal: Detecta piezas más pequeñas que sensor principal. Activo siempre.

Sensor secundario: Detecta piezas más grandes que el sensor principal. Solo se activa durante el cierre y siempre que el principal no detecte.

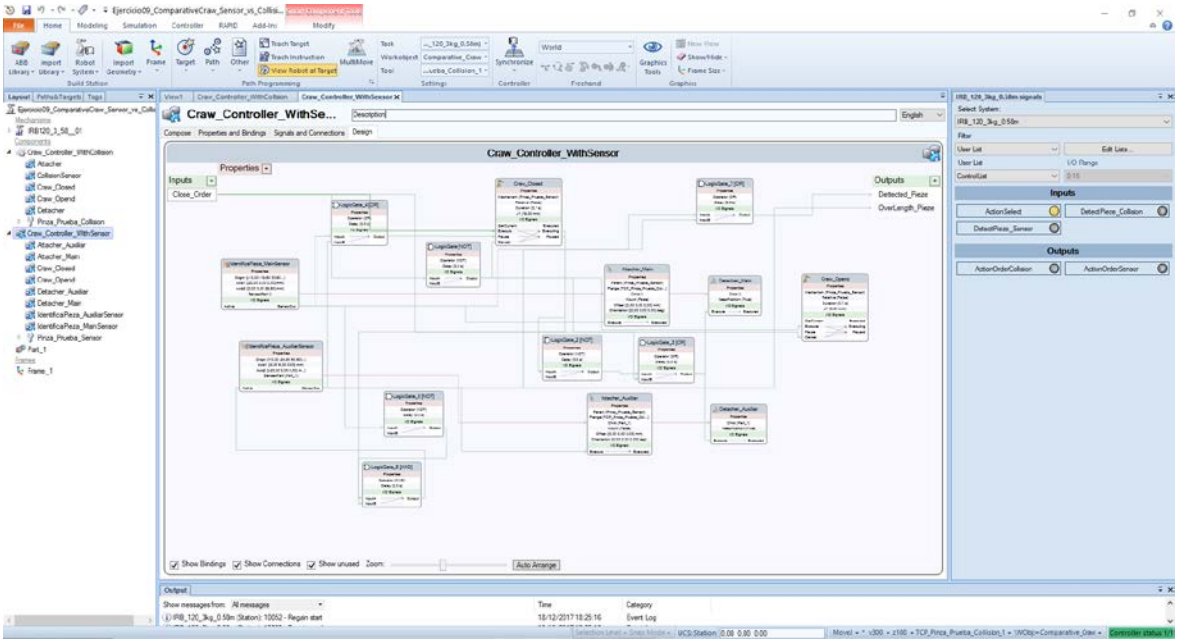


Figure 76: Control Lógico Garra Sensores

Observaciones de los Sensores:

1. Identifica qué objeto es el colisionador
2. Aparentemente, mayor tiempo de cómputo/ciclo
3. Consumo de recursos variable según lógica implementada
4. Recomendable ubicar los sensores tangenciales entre sí para abarcar un espacio volumétrico.
5. **NO NECESARIO CUIDADO en creación de la Herramienta "Tool"**
6. **Aumenta la complejidad de lógica implementada**. *Aparece la necesidad de contemplar el caso especial de "pieza a detectar de mayor volumen que el área de los sensores implementados"*.
7. **Si la lógica implementada es correcta NO FALLA en simulación, más robusto a repetitividad y no necesarios *Delay*s en código RAPID.**
8. Si el objeto es invisible, no lo detecta
9. **ATENCIÓN**: Si la Tool es invisible, sigue consumiendo recursos del sistema en el cómputo de la comparación, PERO NO Activa salida (compara con objeto vacío sensor)
10. Susceptible ante variaciones de Step de Simulación, inversamente proporcional a la relación de Step y velocidad de aproximación, a mayor Step mayor penetración.

iii. Conclusión: Sensor a utilizar en una herramienta de agarre "Craw Tool"

Se recomienda implementar el controlador de la Herramienta por medio de Sensores.

Las consideraciones que hay que tomar en el código RAPID, cuando se emplea un Detector de Colisiones, van contra lógica. Es engorroso tener que mantener todas las partes de la Herramienta separadas entre sí, considerando que no es un supuesto predecible a-priori.

A pesar de que, **siempre que se pueda por diseño geométrico** de la herramienta "*Tool*" **y se trabaje de forma no iterativa sobre los mismos objetos**, es decir, sean manipulados no solo por un único robot, sería aconsejable **usar el detector de colisiones "*CollisionSensor*"**. Presenta un comportamiento más ajustado a la realidad en cuanto a lógica programada del SmartObject y una notable mejora en consumo de tiempo de cómputo en ejecución de tareas. Pero necesita del apoyo de esperas programadas en código RAPID.

Para programas iterativos y trabajando siempre sobre un mismo objeto, es decir no sea reposicionado por un elemento externo, **se recomienda utilizar el control por Sensores, presenta una mayor robustez a repetitividad**. Aunque la lógica implementada es notablemente más compleja, crece de forma exponencial al número de sensores utilizados.

B) Limitaciones de los Sensores

Considerando:

i. Implementación del tiempo de Simulación:

A la vista de los resultados obtenidos en las simulaciones, se cree que **RobotStudio** ha sido diseñado con una **gestión de tiempos no continua (*discreta*)**, para gestionar/actualizar las E/S (*en función del "TimeStep"*), **y una simulación continua (*asíncrona*)**.

Véase los vídeos y contenido del **ANEXO IV**.

ii. Los objetos a identificar NO SON SÓLIDOS

Los objetos se encuentran definidos como tipología CAD o Geometría Matemática: Todo sólido representado en la pantalla de edición se encuentra compuesto exclusivamente de las caras que se visualizan. Es decir, el contenido informativo se encuentra restringido a las superficies de sus caras, que asimismo son los planos que se comprobarán y detectarán cuando se verifique que existe un corte entre el plano que compone la cara del objeto y el plano del sensor.

iii. Posición y Orientación del Planos del Sensor

Consideración del Plano de referencia sobre el que se compara:

- **Evitar planos Paralelos a las superficies de sujeción o tangenciales al vector de movimiento del objeto a identificar**. Se busca la **intersección progresiva** para dar tiempo a actualizar los registros de E/S.
- **Combinar planos tangenciales entre sí**, en los supuestos en que la variable principal a respetar sea la velocidad de aproximación de los objetos. **Evitar problema de pieza más grande que plano a identificar**.

iv. Restricción de Velocidad:

Condicionada por:

- **Tiempo de Actualización de registros de E/S**, en función del *TimeStep*
- **Tipo de sensor escogido y posición** del plano del sensor

v. Ubicación exacta del Objeto en el Espacio/Estación:

Ninguno de estos sensores garantiza una ubicación exacta en el objeto, sólo informa de que existe una intersección en algún punto del plano/s que representa el sensor utilizado.

La suma de estos factores ha provocado:

a) Intersección de Planos

Problema: Supuesto en que el **Plano de Sensor es paralelo**, o cuasi-paralelo, **al Plano de la cara del Objeto** a detectar, **y de menor tamaño/superficie que el plano a detectar**, es decir que el Plano del Sensor **quede completamente contenido en el Plano de la Cara del Objeto**.

Consecuencia: Si el momento de la intersección se da fuera del instante del "*Step*" (*que se adelanta que será en la práctica totalidad de las ocasiones, el 99,99% de las ocasiones*), es decir, fuera del periodo de muestreo de las E/S, el resultado será la pérdida del flanco de activación del sensor, ergo no se detectara la existencia de colisión aunque el sensor se encuentre completamente contenido en el objeto a detectar, es decir, la representación del Sensor se encontrará en el interior del volumen del objeto que se dese detectar.

Ver ANEXO: **ANEXO V: Limitaciones de los Sensores:**

b) Necesidad de Forzar el "Refresco" de los Sensores en el instante deseado:

Problema: Configurar el Estado Inicial, o un Estado por defecto, un SmartObject.

Si se ha diseñado un SmartObject condicionado a la activación de una orden (*ejemplo: uso de sensores para detectar objetos*), el comportamiento encontrado es el de que **mantendrá como valores iniciales la configuración de Salidas registrada en el último estado en el que se encontró**, es decir en la última activación simulada. Dicho de otra forma, configuración de los valores de salida de cada bloque posee memoria entre simulaciones, son consideradas variables persistentes. Y se mantendrá en dicha configuración **hasta que no exista una variación de las E/S del SmartObject**, lo que **se traduce en que no calculará la configuración del Estado del Instante en el que se encuentra inicialmente.** A esto hay que añadir que, una vez cumplida alguna condición de variación de Entradas, **sólo calculará el nuevo estado de configuración interna para las variables reasignadas**, para el resto de variables no actualizará la configuración de Salidas. ***Todo este "detalle" no tendría relevancia si el programa identificase la gestión de E/S por nivel, en vez de hacerlo por variación de nivel.*** Lo que provoca tener que aplicar consideraciones en la elaboración del programa que difieren del comportamiento real de una instalación.

Por otro lado, si se ha optado por dejar la configuración inicial a expensas del usuario que simule, es decir **activar/refrescar/configurar CADA BLOQUE SENSIBLE manualmente**, o lo que es lo mismo, dejar el **bit Activo en 1 por defecto**, puede suponer un verdadero problema. Problema especialmente costoso en tiempo, a la hora de poder pre-condicionar el Estado Inicial para todos los bloques que componen la Estación, en función de la cantidad de bloques utilizados y de la complejidad del SmartObject. Y sí, habrá que hacer "esto" antes de CADA UNA DE LAS SIMULACIONES LANZADAS.

Alternativa: **Refrescar por código desde RAPID o implementando una variable de entrada manual,** que se deberá manipular una única vez y antes de lanzar las simulaciones. Aviso que requiere de **lógica cableada propia** y elevará notablemente la complejidad de la lógica cableada final del SmartObject.

Tanto para la opción de RAPID, como para la activación/desactivación manual, pasa por implementar una variable **externa, que haga la función de interruptor on/off,** conectada directamente a la casilla "Active" de cada bloque sensible.

Consecuencia: *Aparición de verdaderos CROMENBERGS en la simulación.*

c) Consecuencias de la Velocidad de Movimiento

Durante la simulación, **en cada "*Step*", RobotStudio calcula, predice y simula la velocidad de los objetos y la posición alcanzada para el siguiente "*Step*"**. Aunque no se ha podido corroborar cómo lo resuelve, nos hemos encontrado en las simulaciones que, **a la hora de intentar detectar un objeto,** el resultado dependerá de si el instante de actualización del "*Step*" coincide con el momento temporal en el que se da la intersección entre alguno de los planos de objeto con el plano/s del sensor. Dicho instante **se encuentra íntimamente ligado a las velocidades de los objetos, sensor y objeto a detectar**, que *Part*icipan en la identificación de colisión. **A mayor velocidad relativa entre los cuerpos, menor tiempo de intersección entre planos y viceversa**.

Este fenómeno puede forzarnos la **necesidad de configurar "*Delays*" en la lógica cableada** de detección de objetos, y/o gestor de movimiento de objetos, para **garantizar el sincronismo de E/S entre los SmartObject y el resto de la Estación**, por ejemplo un controlador IRC5 virtual encargado de comandar un robot.

Todo ello se traduce en que, **será necesario estudiar la velocidad máxima y mínima de los objetos móviles**, excluidos los robots, para un correcto funcionamiento de la Estación. El estudio no es extrapolable entre Estaciones, siendo soluciones individuales para cada una de las Estaciones que se estudien.

Alternativa a los Smart Objects

RobotStudio posibilita la coordinación con programas externos, como pueden ser programas que se encarguen de la gestión/configuración de las E/S, haciendo las veces de PLC y convirtiendo a los Robots en esclavos puros, como por ejemplo:

- Comunicación con dispositivos/programas externos desde el mismo PC: Unity Pro 8

ATENCIÓN: El estudio de la coordinación con elementos externos a RobotStudio, no ha sido objeto de este TFG.

A esto hay que mencionar que RobotStudio presume de facilitar "alternativa" a los SmartObject que, aun habiendo caído en desuso, aún quedan resquicios de viejas resistencias existentes en la industria y foros de la comunidad de RobotStudio:

- **Event Manager**: Relegado a un segundo plano/reemplazados con la aparición de los *SmartObjects*.
- **Manual Controller**: NO se considera una opción viable en la era computada

Consideraciones en la definición y creación de una Herramienta y su Controlador Lógico

Es posible escoger varias vías distintas en la creación de una **Herramienta (*Tool*)**, en función de la cantidad de **TCP's** y la existencia de *Part*es móviles o no.

a. **Asistente de Creación de Herramientas**: No existen G.L.
b. **Creación de Mecanismos**: Existe 1 o más G.L. en la herramienta.

Para ambos casos, no existe restricción de TCP's o modos de operación. Para cada uno de ellos RobotStudio permite asociar características propias, pudiendo dotar a la Herramienta de distintas formas de trabajo, sin tener que pasar por la simulación de un cambio físico o virtual de Herramienta.

Ver el ejemplo desarrollado de herramienta del TFG: "**ANEXO V**"

Para ver proceso de creación: **ANEXO III: FICHA DE "How to make a Tool and Tool-Controller"**

Consideraciones de la creación de una Herramienta: Asistente de creación de Herramientas (Tool Creator)

El Asistente de Creación de Herramientas: permite definir con facilidad las características de una herramienta a *Part*ir de una pieza ya existente, o usando una pieza simulada para representarla.

Ver **"ManualOperdor-RobotStudio"**:

- "*9 Pestaña Modelado:9.18 Crear Herramienta: (378/648)*"

A) Observaciones generales de la Creación de Herramientas Simples

a. **Se recomienda abrir un fichero nuevo/en blanco para definir con facilidad la herramientas y poder así ubicar los sólidos en el origen de coordenadas,** considerando la Referencia de la Estación como el punto de contacto de la base de la Herramienta.

a. **ATENCIÓN**: Cualquier desplazamiento de los Sólidos de la Herramienta respecto del origen de coordenadas será entendido como un *Offset* propio de la definición de la herramienta y se arrastrará hasta en la colocación de la Herramienta al Robot. Ver apartado **Consideraciones de las referencias "*Frame*"**.

b. **Definición de Estructura Física**: La similitud de la herramienta virtual con la realidad no tiene por qué ser "milimétrica", es más **se aconseja recurrir a modelos simplificados**. Si no se conocen las dimensiones exactas de la herramienta física, **se puede** seguir

implementando el controlador de la herramienta, tanto para programar movimientos como acciones propias. Más adelante, **cuando se disponga del modelo exacto**, **se podrá re-asociar** el nuevo modelo físico sin excesivos problemas **al controlador diseñado**.

c. **Definición de la Masa de la Herramienta**: Para aquellas herramientas diseñadas de materiales con una densidad similar, RobotStudio permite **encontrar el centro de gravedad haciendo clic en el modelo de la herramienta con el modo de ajuste "Centro de gravedad"**.

d. **Definición de TCP's**:
   - El **nombre predeterminado es el mismo que el nombre de la herramienta**. Si se desea crear varios TCP's para una misma herramienta, cada TCP debe tener un nombre exclusivo. Ver "**Figure 78**: Herramienta Simple con 2 TCP's, uno de ellos con orientación distinta a la base"
   - **No existe la necesidad de que la referencia del TCP sea paralela con la referencia Base** de la Herramienta. Ver el TCP de la derecha de la "**Figure 78**: Herramienta Simple con 2 TCP's, uno de ellos con orientación distinta a la base".
   - **No existe un convenio para definir TCP's** propiamente dicho, pero se recomienda colocar la referencia del TCP siguiendo las siguientes directrices:
     - **Herramientas de Posicionamiento y Acción de Penetrar**: "eje Z del TCP saliente de la Herramienta"
     - **Herramientas de Manipulación de Posición y Rotación (*acción de agarre*)**: "Eje Z entrante en el TCP de la Herramienta"

   - **ATENCIÓN: Al introducir/definir la posición del TCP**: se toma la referencia respecto del sistema de coordenadas mundo, donde RobotStudio considera que está representado el punto de montaje de la herramienta.

e. Se recomienda **guardar la herramienta como fichero biblioteca, antes de mover la Herramienta de la posición (0,0,0)**. En el menú Archivo, hacer clic en Guardar como biblioteca.

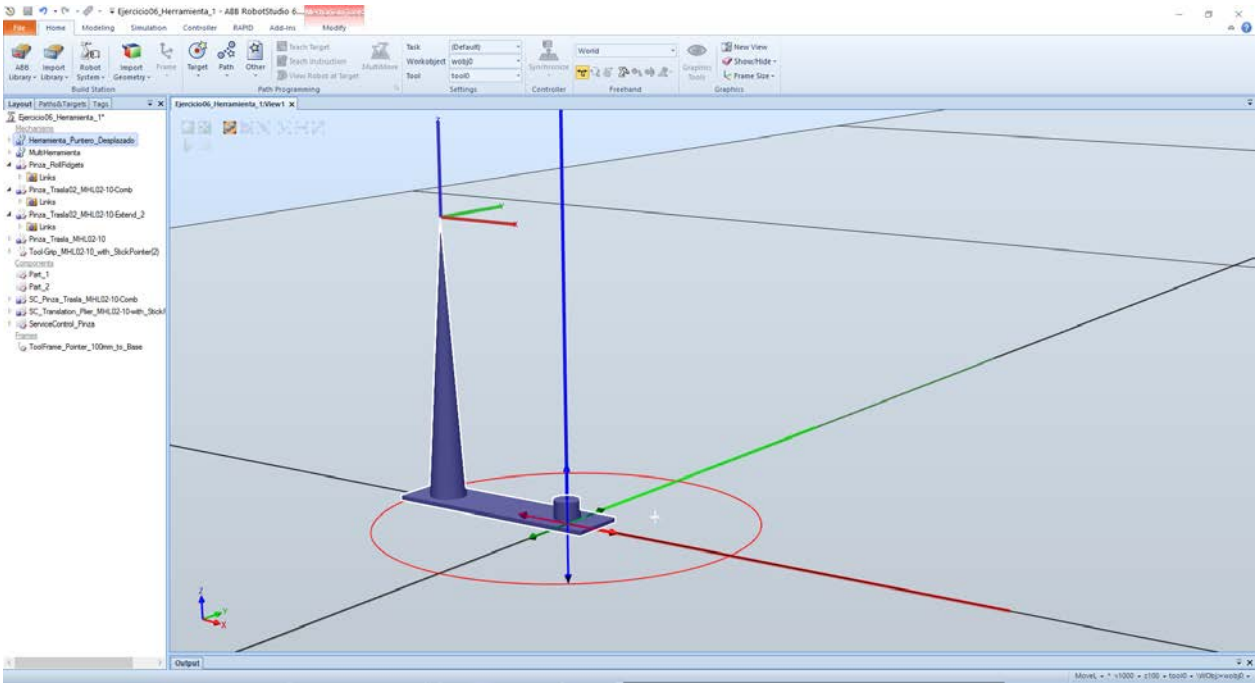


Figure 77: Herramienta Simple con Puntero Desplazado en X

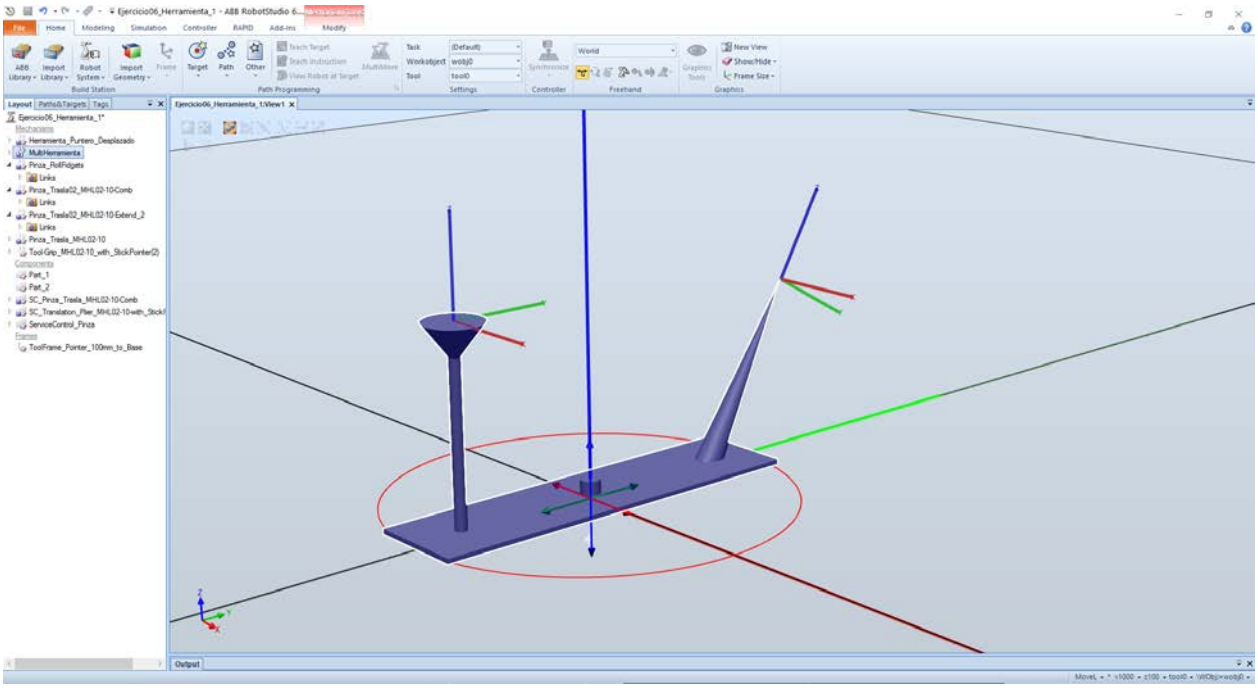


Figure 78: Herramienta Simple con 2 TCP's, uno de ellos con orientación distinta a la base

B) Creación de Datos Herramienta: Asignación de las herramientas al robot

a. **ATENCIÓN:** RobotStudio es es**pecialmente sensible a los elementos que se tienen "señalados" cuando se está definiendo/creando nuevos componentes**, se recomienda prestar atención a que se tenga **seleccionado el robot** en el que se crean los datos de herramienta.

b. **IMPORTANTE: Hay que "Arrastrar" el isotipo de herramienta HASTA el NOMBRE DEL ROBOT**, si se arrastra hasta alguno de los eslabones del robot SÓLO SE CREARÁ UN ENLACE DE DEPENDENCIA "Attach".
Comprobar en: "*Home: Pestaña 'Paths&Targets':Station Elements:Default Task: Tooldata:*ExistData*".

c. **ATENCIÓN: Para lograr que el robot sostenga la herramienta**, hay que conectar la herramienta al robot y aceptar la actualización de la "*Frame*Tool". La Herramienta toma como nueva referencia/"*FrameTool*" la base de coordenadas de la estación junto con el *Offset* que existiese cuando fue guardada en forma de biblioteca.

C) Observaciones de la Simulación

Solo aplicable para la versión de RobotStudio de 64 bits: Se puede utilizar la función Física para incluir simulaciones físicas, junto a las herramientas de programación y simulación tradicionales de los robots, como: *ejes, cables, piezas y demás*, los cuales seguirán las reglas de la física durante la simulación.

Consideraciones de la creación de una Herramienta: Creación de un Mecanismo (*Mechanism*)

Si se tiene la necesidad de **dotar a una Herramienta de algún G.L.: será necesario** pasar por la **Creación de un Mecanismo**, proceso más largo y laborioso que la modalidad anterior, Tool Creator, e irreversible una vez compilado.

Para ver el proceso de creación de un Mecanismo se recomienda:

Ver **"ManualOperdor-RobotStudio"**:

- "*9 Pestaña Modelado:9.17 Crear Mecanismo: (370-378/648)*"

En el proceso de creación de una Herramienta como Mecanismo, pasa por un asistente de creación de relativo fácil seguimiento. Ver el ejemplo desarrollado de herramienta del TFG: "**ANEXO V**"

No perder de vista las recomendaciones expuestas en los puntos anteriores: **Consideraciones de la creación de una Herramienta: Asistente de creación de Herramientas (Tool Creator)**

A) Observaciones generales de la Creación de Herramientas Complejas

Apoyar estas recomendaciones con la **Figure 79**:

a. **ATENCIÓN**: Es un proceso IRREVERSIBLE. Cuando se compila la Herramienta, remueve de la lista de objetos las *Part*es que componen el mecanismo, impidiendo volver a recuperarlas, salvo que estuviesen guardadas como librerías o geometrías.

b. **IMPORTANTE**: Definir/Especificar que el Mecanismo que se crea ES TIPO HERRAMIENTA. Los mecanismos NO NECESARIAMENTE SON HERRAMIENTAS.

c. **Se recomienda acotar el rango de libertad de los ejes rotacionales y traslacionales**. Evita tener que controlarlos por código, o delimitarlos con los bloques de acciones de los SmartObject, posteriormente.

d. **Sí existen G.L. dependientes**: Es **más que recomendable especificar las dependencias entre los ejes en esta fase de diseño**, de otra forma habrá que asumir el control de "esa" dependencia por código o lógica cableada en el controlador y tratar a todos los ejes como absolutos.

e. **Al compilar**:
   - **Mecanismos Nuevos**: Se añade un nuevo mecanismo generado en la Estación, con el nombre predeterminado "Mecanismo_" seguido de un número de índice.
   - **Mecanismos editables existentes**: modificado en el modo de modificación del modelador de mecanismos, se guarda sin ninguna pose, correlación de ejes ni tiempos de transición.
     - El mecanismo se inserta en la estación activa. Las *Part*es del eslabón son clonadas con nombres nuevos, pero los eslabones actualizan sus referencias de pieza. Al cerrar el modelador de mecanismos, estas piezas clonadas se eliminan.
     - El modelador de mecanismos pasa ahora al modo de modificación

f. **ATENCIÓN: No es posible seleccionar las piezas** que formen *Part*e de una biblioteca o un mecanismo.

g. **Es posible comprobar las características de los ejes traslacionales y rotacionales,** una vez creada la herramienta.

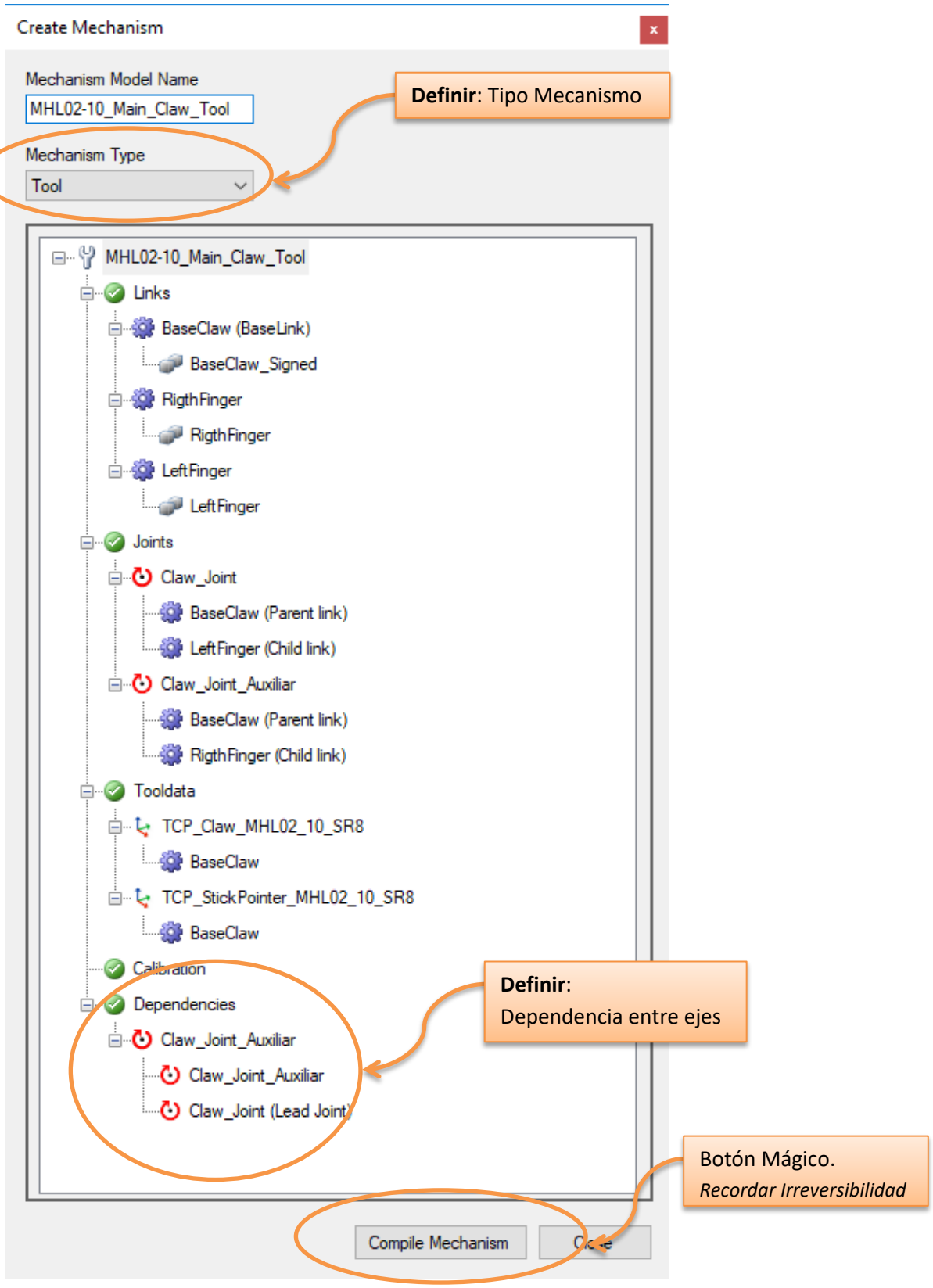


Figure 79: Creación de Mecanismos

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Consideraciones de la creación de un Controlador Lógico de una Herramienta

a) **Solo es obligatorio** implementar un Controlador de Herramienta **si la Herramienta ha sido creada como Mecanismo y posee al menos 1 G.L.,** sobre el que se desea actuar en la simulación, y/o el robot posea el control de las acciones de este.

b) Resumiendo, **se recomienda pasar por el uso de un SmartObject**, encargado de comandar todas las operaciones lógicas y de control de la herramienta. Aliviando de esta forma al controlador IRC5 y/o al programa RAPID implementado, dado que en condiciones reales no poseen las funciones de control de funcionamiento de la Herramienta y se limitan a solicitar la activación/desactivación de las acciones.
Las consideraciones pasarán por las mismas que los SmartObject, aplicando las restricciones impuestas por la definición de la Herramienta. Ver Capitulo **Consideraciones de los Objetos Inteligentes "*SmartObjects*"**.

c) Es **posible reutilizar un Controlador Lógico, SmartObject existente,** en otra nueva Herramienta definida así mismo como Mecanismo.
   - Se deberá poner especial cuidado en eliminar la herramienta anterior junto con todas las conexiones que tuviera.
   - Cuidado al re-cablear las conexiones con la nueva Herramienta.
   - Este proceso puede NO ser recomendable si el nivel de complejidad del Controlador Lógico es muy elevado, el tiempo que exige el cambiar una herramienta de controlador puede exceder el tiempo de creación de uno nuevo.
   - Posibilidad de aparición de errores en el SmartObject (*fácilmente corrompibles*).

d) **Las propiedades que haya que implementar estarán íntimamente ligadas al tipo de Herramienta, y de las restricciones que se deseen simular**, al menos se recomienda contemplar la existencia de:
   - 1 Variable de Entrada: encargada de hacer de Actuardor/Accionador de Orden de Activar el G.L., mínimo una por cada G.L. no dependiente
   - 1 Sensor (*lineal o planar*) encargado de informar de alcanzado destino deseado en el avance del G.L., al menos 1 por cada G.L. no dependiente que se prevé no alcance su recorrido/rotación completa.
   - 1 Variable de Salida: Informar de alcanzado destino en la activación del $G.L._i$, al menos 1 por cada G.L. no dependiente

Para más información de la implementación de un Controlador de Herramienta ver la Herramienta Simulada en el TFG donde se entra en detalle de su Controlador Lógico: **ANEXO V**.

Consideraciones Generales de la Definición y Creación de una Herramienta

a) **La creación de Herramientas es un proceso Irreversible** que **destruye los sólidos utilizados**, se recomienda **ser escrupuloso en la implementación**, así como **tener recopilada toda la información necesaria para su implementación**, a ser posible en formato físico (*papel*).
Se dificulta la posibilidad de corregir errores y/o hacer modificaciones futuras.

b) El **nivel de complejidad del diseño gráfico de la Herramienta y/o del control de la Herramienta Mecanismo** puede salpicar a la visualización de la simulación y al aumento del tiempo de respuesta entre la comunicación Herramienta-Robot (RAPID).
La elección de implementar una Herramienta con un Mecanismo queda restringida a que exista la necesidad de dotarla de Grados de Libertad..

c) **La complejidad de la lógica de control dependerá del nivel de robustez** que se desee implementar, suele ser un objetivo asegurar la simplicidad de uso para terceros.

d) Las características propias de la Herramienta se encuentran contempladas por código en RAPID, y así poder ser utilizadas para calcular las trayectorias de los movimientos del Robot real. Es **estrictamente necesario implementar la herramienta con los datos reales**, CG e Inercias principales, **cuando se vaya a exportar el programa a un controlador IRC5 real con una herramienta similar a la virtual**.

2. 2. 3. Consideraciones en el Proceso de Cambio de Herramientas desde RobotStudio

En primer lugar, hay que mencionar que, **el cambio de herramienta en RobotStudio no es inmediato**, el **grado de complejidad es directamente proporcional a** lo **completa** que se encuentre la **Estación y** la **dependencia de la *Frame*Tool$_i$ con los *WorkObjects*$_j$**, pudiendo llevarse unas pocas horas.

En segundo lugar, es **necesario ser cuidadoso con el procedimiento.**

Para más información ver: **ANEXO III: Ficha de Ruta: "Cómo hacer un Cambio de** Herramienta"

Se ha tomado como ejemplo: La Estación de la imagen "**Figure 80**", donde se ha implementado un programa sencillo en el que se manipula un objeto modificándolo de posición, de PosA a PosB y viceversa, con una pieza de tamaño más pequeño que la Garra (*para más información de la Herramienta ver "***La Herramienta Virtual: "SR.8"***"*).

A continuación, se ha procedido a cambiar la Herramienta-1 por una Herramienta-2, de mismas características constructivas pero distinta implementación lógica.

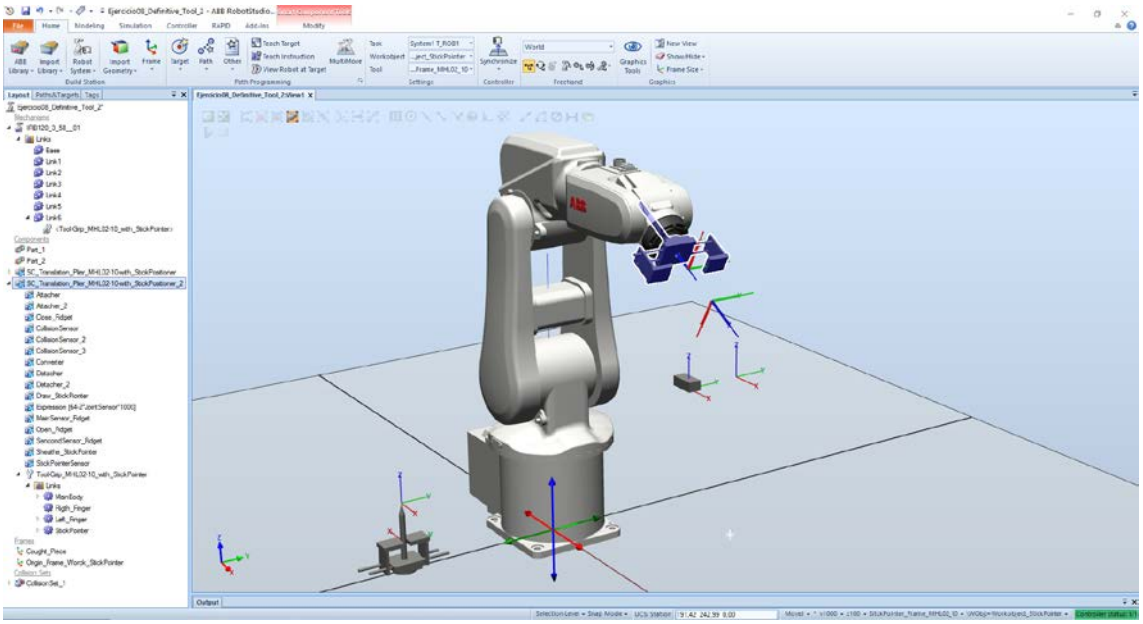


Figure 80: Estación de Ejemplo Cambio de Herramienta (Tool)

Al manipular Herramientas (*asignar y/o modificar características*) hay que diferenciar entre Tareas Disponibles (*"Task"*), Planos de Trabajo *("WorkObjects"*) y TCP's de la Herramienta (*"Tool" o "ToolFrame"*). RobotStudio efectuará enlaces/guardará la información de la Herramienta (*"Links"*) en ubicaciones distintas de la Estación, en función de la *Parte* de la Estación que tengamos seleccionada en cada una de sus pestañas. Si se desea cambiar una Herramienta en RobotStudio habrá que hacer toda la modificación y eliminación de Links referidas a la Vieja

Herramienta. **Todo ello habrá que hacerlo a mano, NO EXISTEN ayudas ni a nivel informativo ni a nivel ejecutivo**. En el ejemplo se contempla:

ATENCIÓN: Los datos de la "Tool" responden a la última Herramienta instalada.

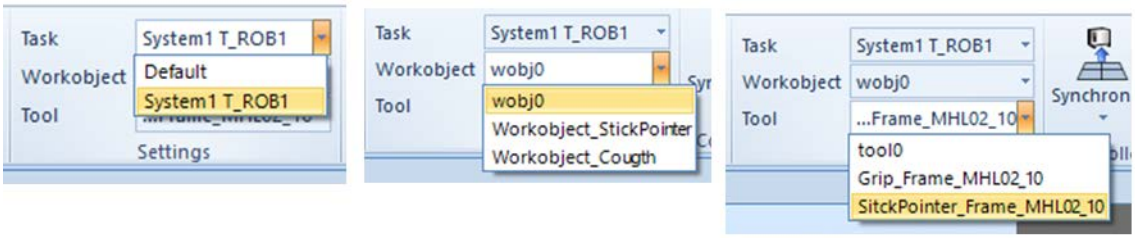


Figure 81: Pantallas de Datos de Herramienta "Tool"

Consideraciones Generales

a) **Comprobar si es necesario deshacerse de la vieja Herramienta ("OldTool")**, o se puede simplemente "*ocultar*" del plano de visión.

b) Si el cambio de Herramienta es por otra herramienta de mismas características físicas, y ubicación exacta de TCP's, el procedimiento se simplifica considerablemente.

c) **Si la nueva Herramienta tiene distinta ubicación de TCP's** es necesario no perder de vista que **al desacoplar la "*OldTool*" RobotStudio**:
   - No Avisa ni Modifica los "Links" entre los distintos *Paths-OldTool*
   - No Avisa ni Modifica el contenido de los Programas RAPID que contiene los *Paths(tool)*
   - Sigue existiendo la *FrameOldTool* en el Robot: "*System1:T_ROB1:Tooldata:*ToolFrame*", y sigue siendo seleccionable por el usuario y el programa.

d) Será necesario cambiar a mano todas las conexiones "*Links-OldTool*" de los *WorkObjecs* y *RAPID_Programs*.

e) **Si se elimina la "*OldTool*" del "*tooldata*"**:
   - **ATENCIÓN RobotStudio modificará sin avisar la "*FrameOldTool*" por "*Tool0*" en todos los *Paths* y *RAPID_Programas*** en los que apareciese "*FrameOldTool*".

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

f) **Recordatorio**: Para cargar ("*Load*") y vincular ("*Link*") a una "*Tool*":

- Es OBLIGATORIO DESACOPLAR la "*Tool*" DE LA LIBRERÍA para poder enlazar "*Link*" la "*NewTool*"
- **ATENCIÓN**:: La "*NewTool*" se debe acoplar al Robot, no al Link6(Brida) del Robot. ACOPLAR ("*Link*") NO ES ENLAZAR ("*Attach*"), acoplar implica cargar la información de la Herramienta en el Controlador del Robot y poder así hacer las Transformaciones de Matrices.

g) En **el Proceso de Alcanzabilidad "*Reachability*" de "*Target$_n$*"**:

- **ATENCIÓN**: Si se ha Eliminado la "*OldTool*" de la "*tooldata*" antes de reasignar los "*Traget$_n$*" a la "*NewTool*": Para todo Objetivo ("*Target$_n$*"): *Tool ="Tool0"*, necesario cambiarlo desde RAPID y/o eliminar todos los *Path$_i$* y volver a definir todos los Objetivos.
- Si la "*NewTool*" es física y estructuralmente distinta, posee distinta ubicación y/u orientación de TCP, será necesario Adaptar/Corregir la ubicación y/u orientación de los *Traget$_n$*.

h) **ATENCIÓN**: **No se trata igual las Herramientas que ejercen una Acción(*Soldar/Pintar/Taladrar/…*) que a las que se encargan de Manipular(*Traslación-Orientación*)**, se toma por Convenio consensuado, no oficial, que la orientación de los "*Target$_n$*" es coincidente en el eje "Z" tanto en dirección como en sentido en las Herramientas de Acción y coincidente en dirección y sentido opuesto en las Herramientas de Manipulación. Esto supone que en los cambios de Herramienta de Acción a Manipulación y viceversa se invertirá mucho tiempo actualizando la orientación de todos los "*Target$_n$* " afectados manualmente.

i) **Existe una función de búsqueda y reemplazar ("*Find/Replace*") en la Pestaña RAPID**: ayuda/aligera en tiempo el proceso de Modificar para toda "*OldFrame*" transformar en "*NewFrame*".

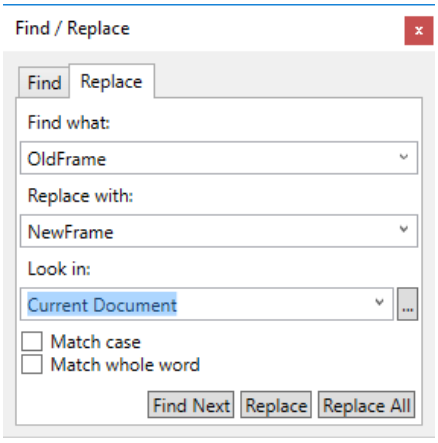


Figure 82: Pantalla de Buscar/Encontrar de RobotStudio

f) Si no se desea Sincronizar la Estación con RAPID: *sentido de RAPID a la Estación* para volcar el contenido modificado/actualizado de los $Path_i$ y poder comprobar el funcionamiento de la nueva Herramienta:

- Se puede seleccionar "***Apply***": **Aplicará los cambios de código RAPID al Controlador del Robot sin modificar contenido de la Estación**, permitiendo simular la ejecución tanto desde la Pestaña de RAPID/Controlador como desde la pantalla visual.

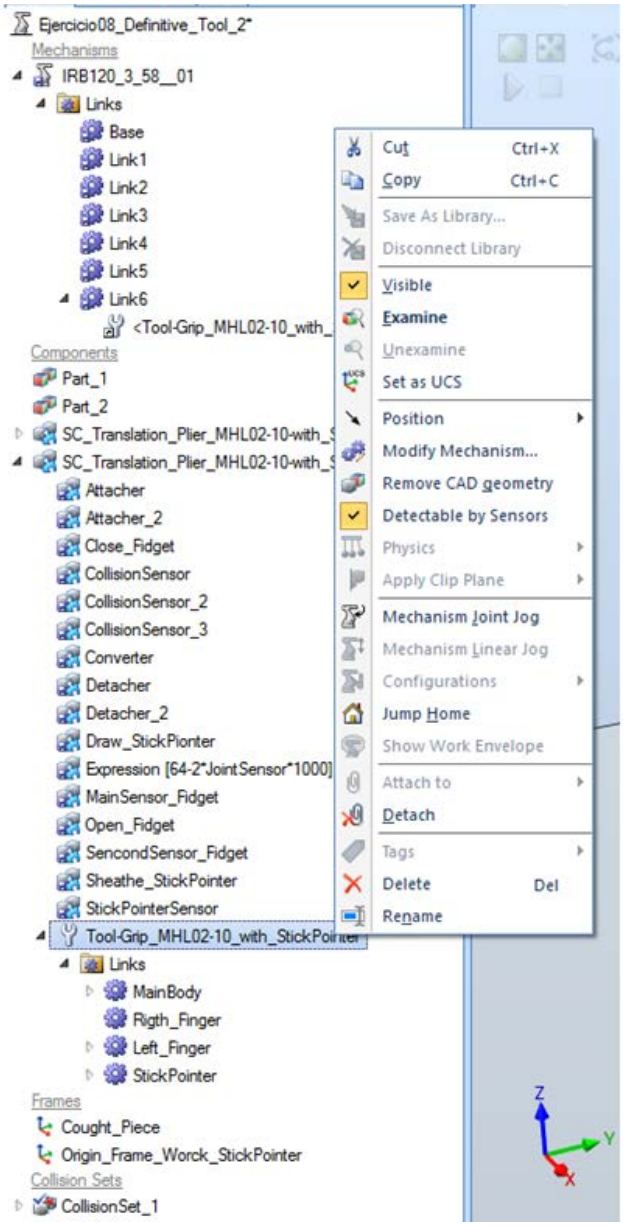


Figure 83: Menú de opciones de la Herramienta

Conclusión Cambio de Herramienta simulada desde RobotStudio:

En primer lugar hay que tener en cuenta que el menor y más sencillo cambio exige bastante tiempo y además escribir "a priori" los distintos pasos en papel, por la complejidad del sistema a implementar.

En el desarrollo de este TFG se ha tiendo que cambiar varias veces las Herramientas, por lo que se cree que:

- **ES PREFERIBLE HACER UN FICHERO NUEVO a reeditar uno existente**, acaba siendo más rápido, si se guardar la Estación en mitad de los cambios NO SOLO no obtiene un fichero con una nueva Herramienta sino que además se perderá el fichero viejo. RobotStudio guarda por partes (*la configuración de la Estación Gráfica por un lado; los programas RAPID por otro; el estado del Controlador Virtual por otro lado; y, todos ellos, guardados sobre una Estación Base que, por defecto, solo guarda lo último que fue implementado…*). Con ello se evita la aparición de verdaderos "Cronenbergs" y/o "Frankesteins".

2. 2. 4. Consideraciones de los Detectores de Colisiones

RobotStudio permite **detectar y registrar las colisiones que se producen entre los objetos** de la estación.

Un **conjunto de colisión** contiene dos grupos de objetos, **Objetos A y Objetos B**, en los que puede especificar objetos para **detectar las colisiones o casi-colisión** que existen entre ellos.

- Cuando cualquier objeto de Objetos A colisiona con cualquier objeto de Objetos B, la colisión **se representa en la vista gráfica y se registra en la ventana de salida**.
- Puede tener **varios conjuntos de colisión en la estación**, con la restricción de que cada conjunto de colisión sólo puede contener dos grupos.
- Cada conjunto de colisión **puede activarse y desactivarse** separadamente.
- Se puede **controlar las situaciones en las que casi se produce una colisión**, lo que se produce cuando un objeto de Objetos A está a una distancia menor de una distancia especificada respecto de un objeto de Objetos B.
- Se considera una herramienta restringida sólo a un nivel visual, información para el usuario para las primeras fases de diseño, aconsejable en el depurado de trayectorias

En general, **se recomienda seguir los principios siguientes** para facilitar la detección de colisiones:

a. Utilizar **conjuntos de colisión lo más pequeños posible**, dividiendo las piezas de gran tamaño y recopilando en los conjuntos de colisión sólo las piezas relevantes.
b. **Activar el nivel de detalle aproximado al importar la geometría**.
c. **Limitar el uso de casi colisiones**.
d. **Activar la última detección de colisión si los resultados son aceptables**.

RobotStudio comprueba las posiciones de los grupos de objetos e indica cualquier colisión existente entre ellos, de acuerdo con las opciones de color definidas.

Para **más información** ver:

- "**ManualOperdor-RobotStudio**":
  "*5 Simulación de Programas: 5.2 Detector de Colisiones: (161/648)*"
- **ANEXO III Ficha de Ruta: "Cómo Configurar un Detector de Colisiones**"

Tomando como ejemplo un fichero simple, mismo programa que el explicado en *apartado*: **Consideraciones en el Proceso de Cambio de Herramientas desde RobotStudio**

A) Ejemplo de la Estación

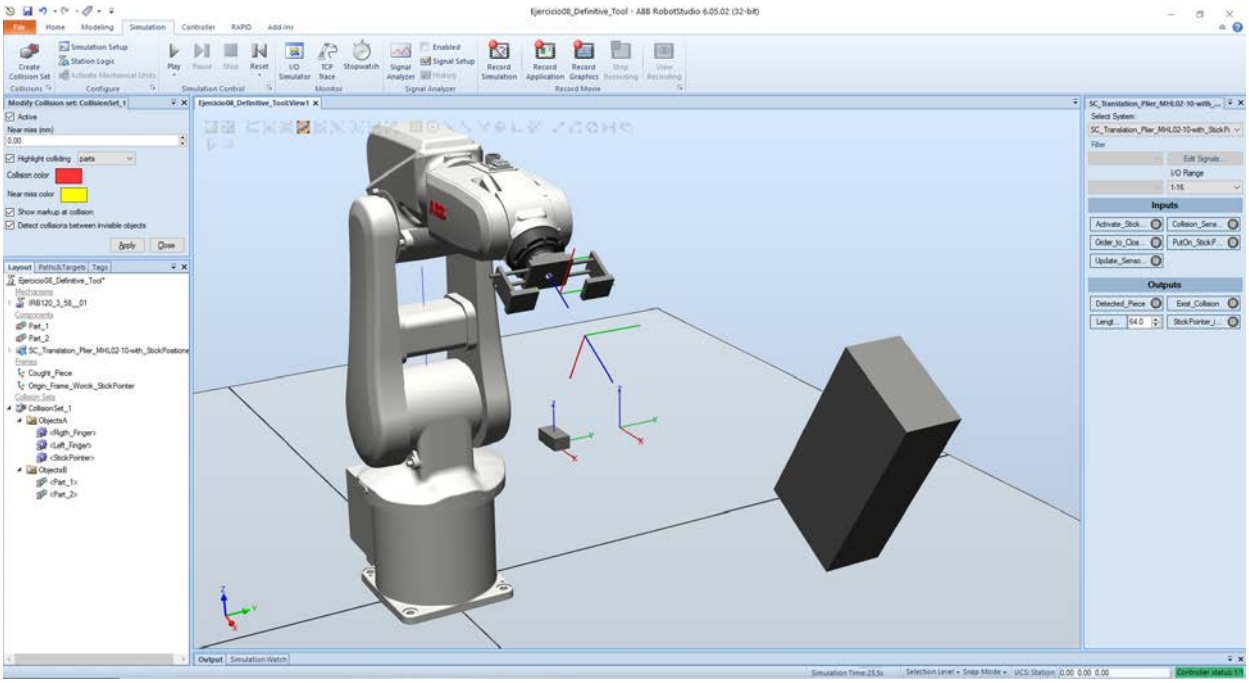


Figure 84: Estación Análisis Detector de Colisiones

La garra es capaz de ajustarse a cualquiera de las dos piezas suspendidas en el aire, el único requisito es que se encuentren ubicadas/situadas inicialmente en la posición A (*ubicación actual de la pieza pequeña*) en la **Figure 84: Estación Análisis Detector de Colisiones**.

Configuración Parámetros Específicos de Detección de Colisionesr

Para poder analizar este a*part*ado, se han configurado **distintas modalidades para Detectar Colisiones**, **todas** ellas se han configurado **con una proximidad de 10 micras**:

- Para el ***Ejemplo:*** La Estación, se ha configurado como se muestra en la **Figure 85**

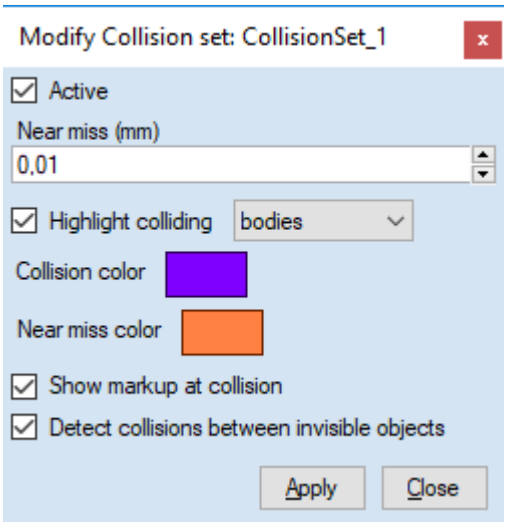


Figure 85: Configuración Parámetros Específicos de Detección de Colisiones: Estación

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

A) Recorrido como Agarre transporte de piezas:

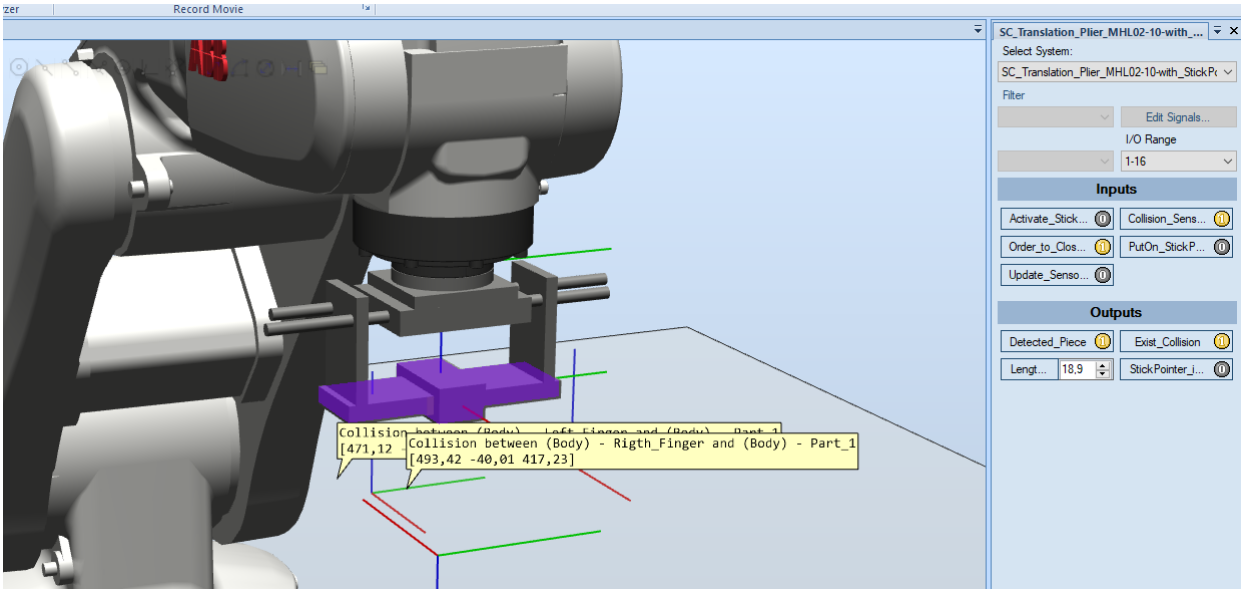


Figure 86: Estación Análisis Selector de Colisiones: Recorrido como Agarre transporte de piezas

B) Recorrido como StickPosition:

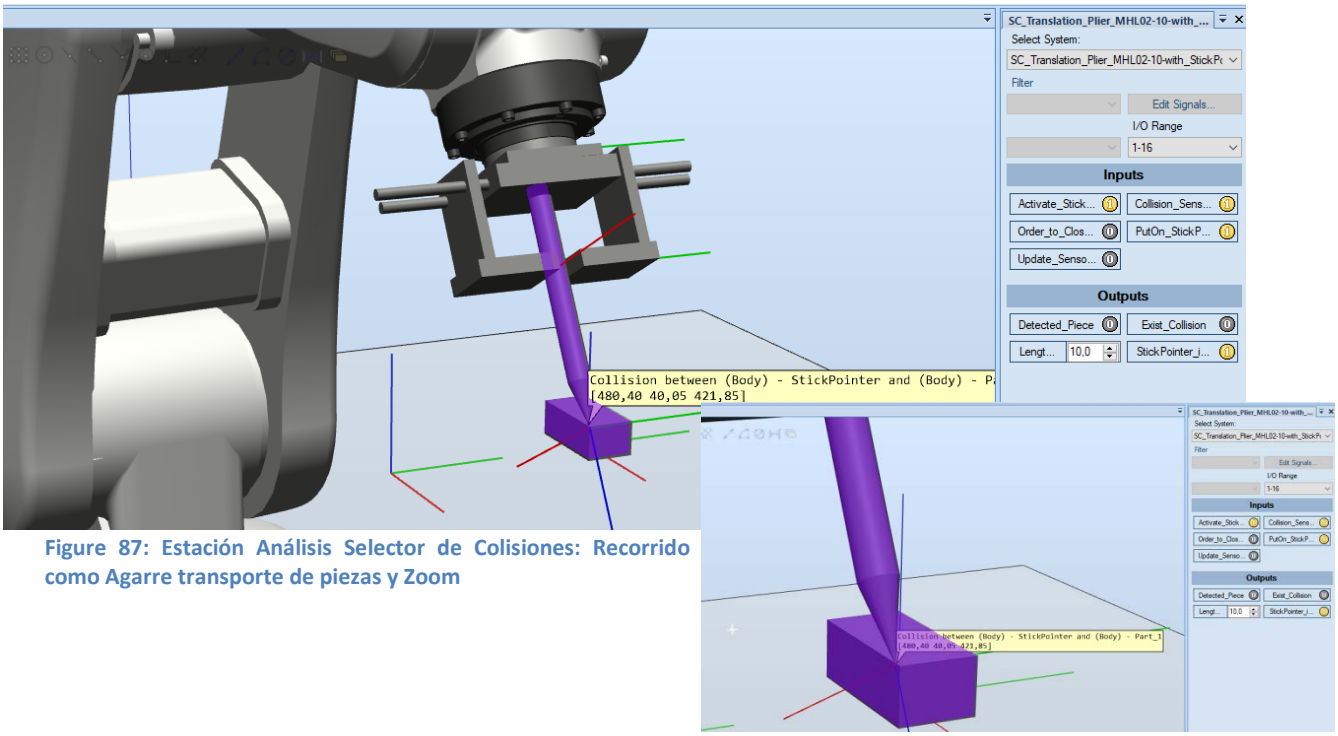


Figure 87: Estación Análisis Selector de Colisiones: Recorrido como Agarre transporte de piezas y Zoom

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Alternativa a la Configuración Parámetros Específicos de Detección de Colisiones: SmartObject

Uso del "*CollisionSensor*" Detector Interno del Controlador de la Herramienta (*SmartObject*):

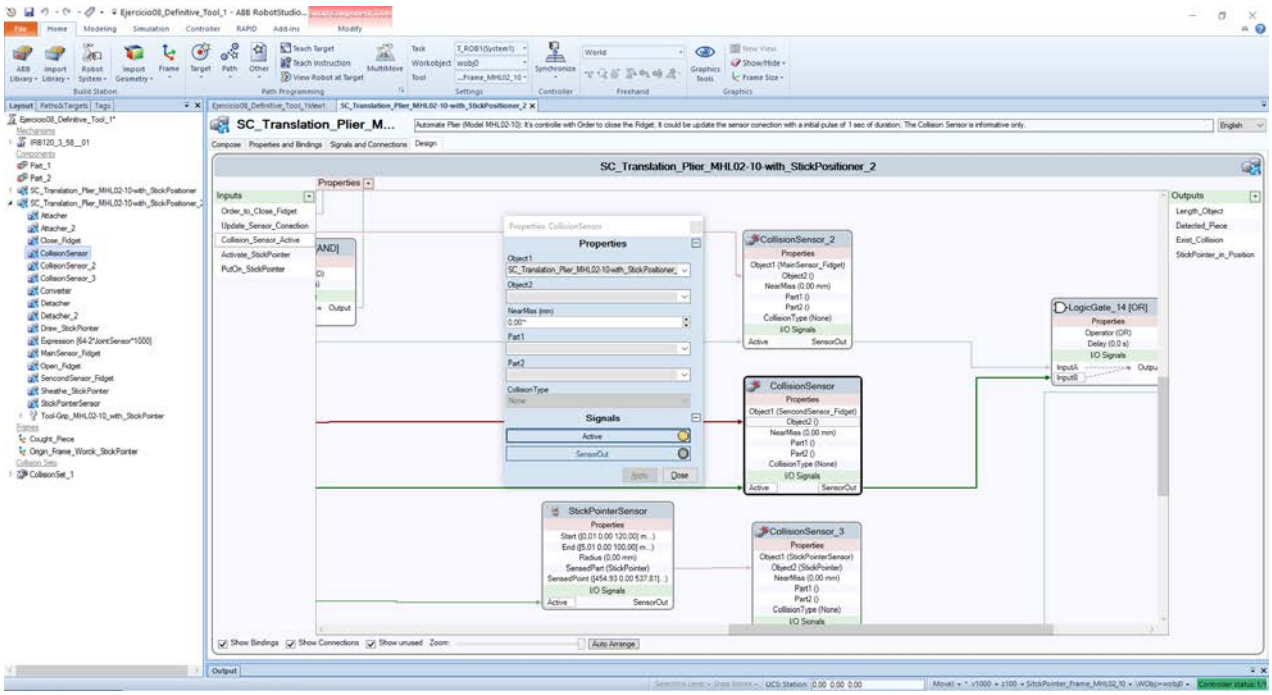


Figure 88: onfiguración Parámetros Específicos de Detección de Colisiones: SmartObject y Zoom de BlockEditor Properties CollisionSensor

Es **necesario configurar** tanto las **interacciones de lógicas del SmartObject** y de la estación, como **editar/definir unas propiedades mínimas en el bloque de propiedades**, edición limitada:

- **No posibilita apoyo gráfico.** No se colorea la colisión en la simulación
- **No aparece información por pantalla**
- **No es posible definir grupos** de "Objetos A" y "Objetos B", **solo puede hacer la comparación entre 2 objetos**

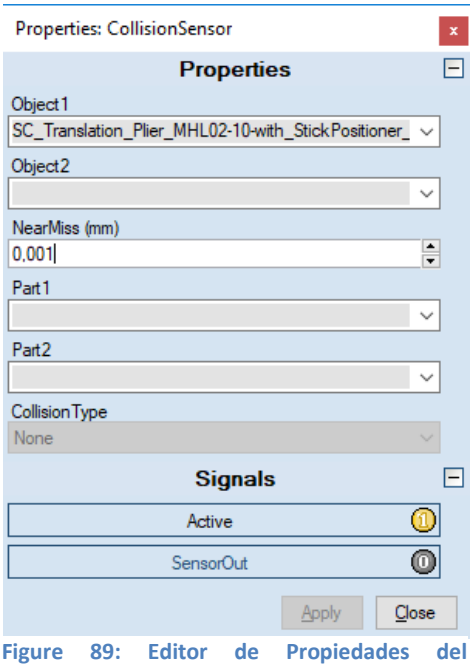


Figure 89: Editor de Propiedades del CollisionSensor

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

A cambio, **posibilita**:

- **Poder crear una variable propia en el SmartObject**, y por ende en la Estación, que **poder interconectar con el controlado y reutilizar en el código de RAPID**
- **Aprovechar la información procedente de otros bloques del SmartObject**, como por ejemplo los objetos sensados/detectados por uno o dos sensores
- **Crear funciones lógicas complejas de detección de objetos**, más fieles y próximas a las interacciones de la realidad
- Comprobación **más finita que el *CollisionSet*** (*CollisionSet: Se basa en la comparación por esfera, excluye los elementos terminados en punta*)

Para más observaciones del *CollisionSensor* , ver el a*part*ado: "**CollisionSensor**:".

A) Indicador de detección de colisión en el agarre:

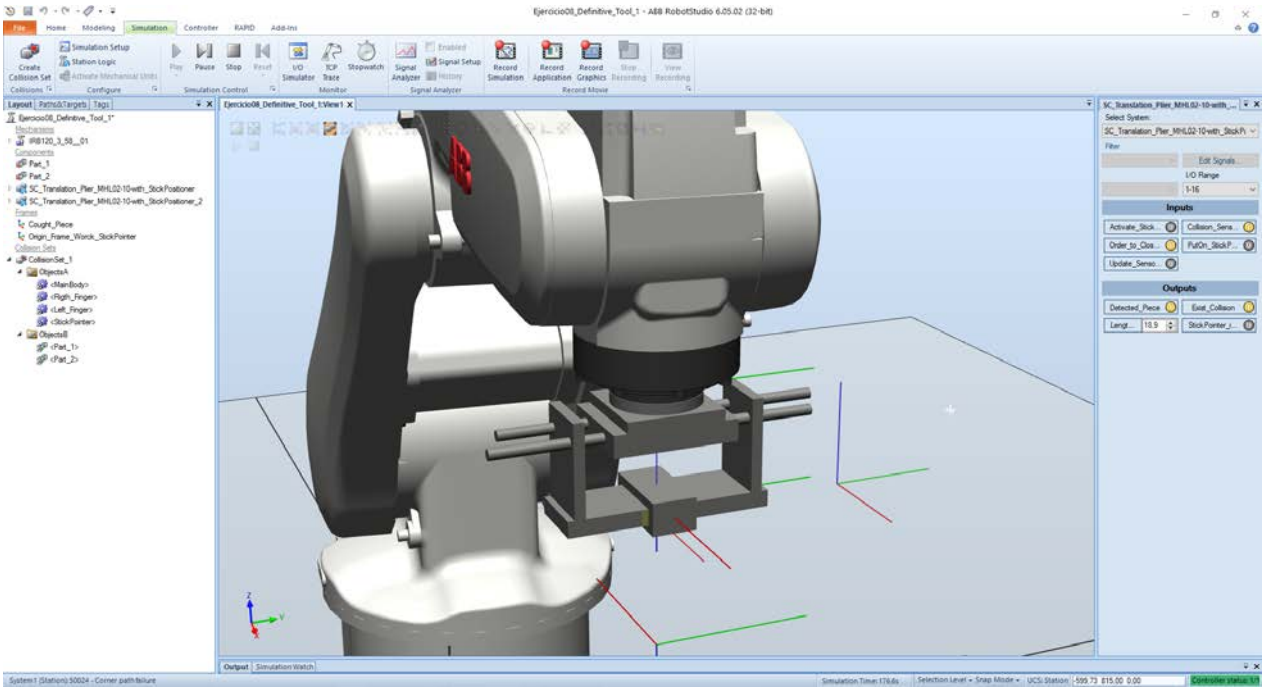
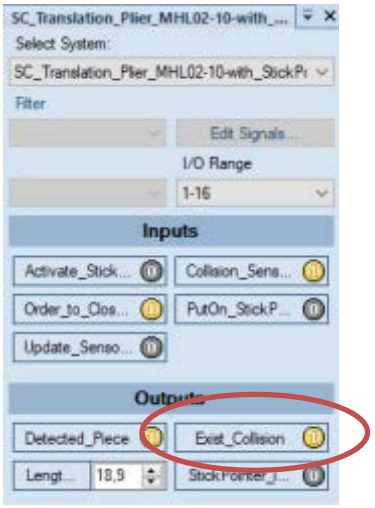


Figure 90: Indicador de detección de colisión en el agarre



Situación Lógica: El Robot se encuentra comprobando si alguno de los sensores de la garra está siendo cortado por la pieza, requisito necesario para el agarre.

Como se aprecia en la imagen, detecta colisión entre los dedos, la pieza y el objeto, ver la activación de la variable "***Output: Exist_Collision=1***".

ATENCIÓN: Existen las "eternas" Problemáticas de la gestión de E/S, (*actualización de los registros de las mismas*) y sincronización con el resto de elementos lógicos de la Estación.

B) Indicador de detección de colisión en el seguimiento con el StickPointer:

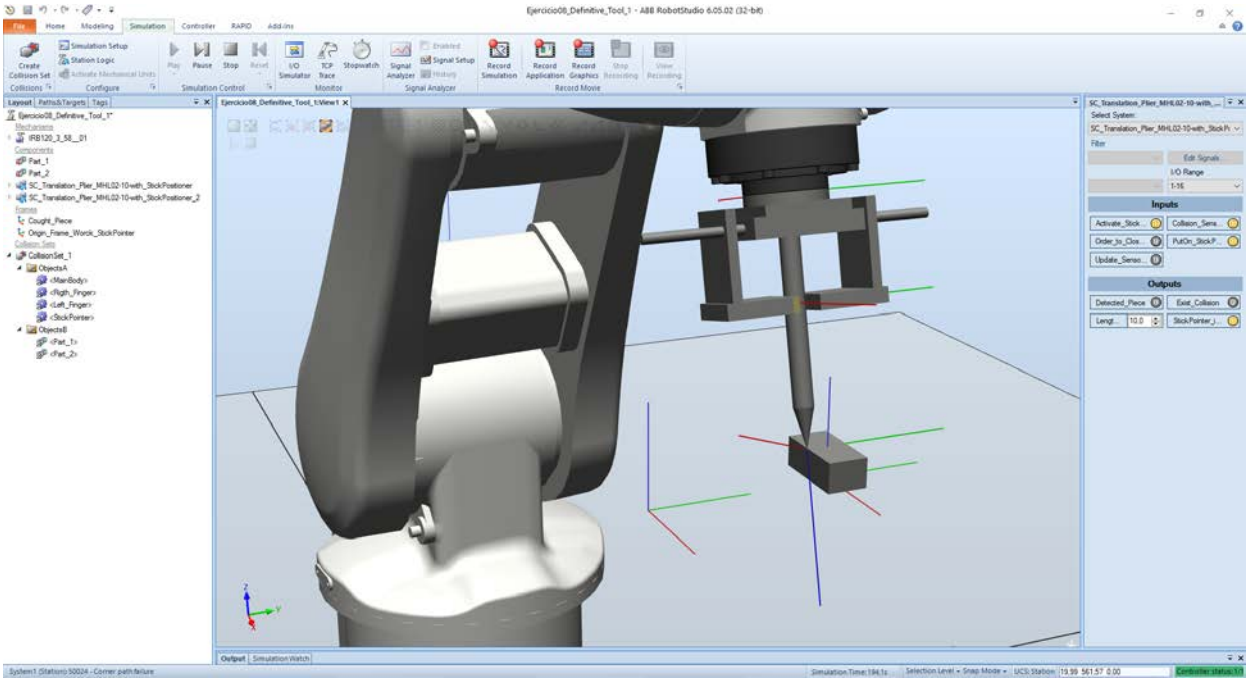
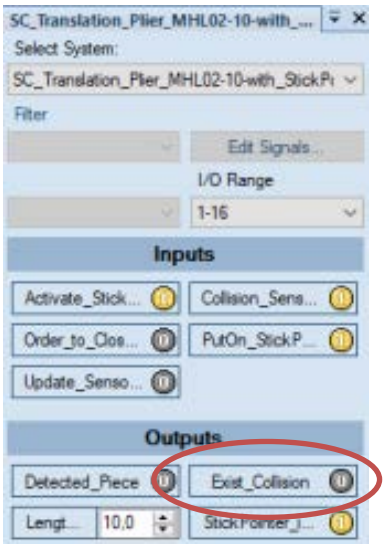


Figure 91: Indicador de detección de colisión en el seguimiento con el StickPointer



Adicionalmente para este a*part*ado, se ha dotado al StickPointer de la Herramienta Simulada, con un sensor lineal. La ubicación del sensor es a lo largo de la punta, comienza en el vértice de la misma y recorre paralelamente la Generatriz del cono hasta alcanzar el cuerpo del cilindro.

Notese que, el TCP se encuentra en el vértice de la punta lo que provoca la necesidad de separar ligeramente el sensor de la punta, esto es debido a que no tiene que ser el mismo punto el del comienzo del sensor que el del TCP (*distancia atómica pero distinta*).

Obsérvese que el resultado es que no detecta Colisión: El controlador IRC-5 sitúa al robot en el punto del TCP, distinto del punto del sensor, mientras el puntero NO ENTRE en el sólido de la pieza NO se detectará colisión, lo que garantiza que NO existe colisión en este modo. **¡EN CONTRAPOSICIÓN CON LA INFORMACIÓN DEL COLLISIONSET! Ver Figure 87: Estación Análisis Selector de Colisiones: Recorrido como Agarre transporte de piezas y Zoom.**

Alternativa a la Configuración Parámetros Específicos de Detección de Colisiones: EventManager

Ejecutando la **funcionalidad "Collision trigger"**:

Simulation: "Configure: *Recuadro esquina inferior izquierda* (*sí está muy escondido*): EventManager:Add: Simulation:Collision:*Define_Parametres"

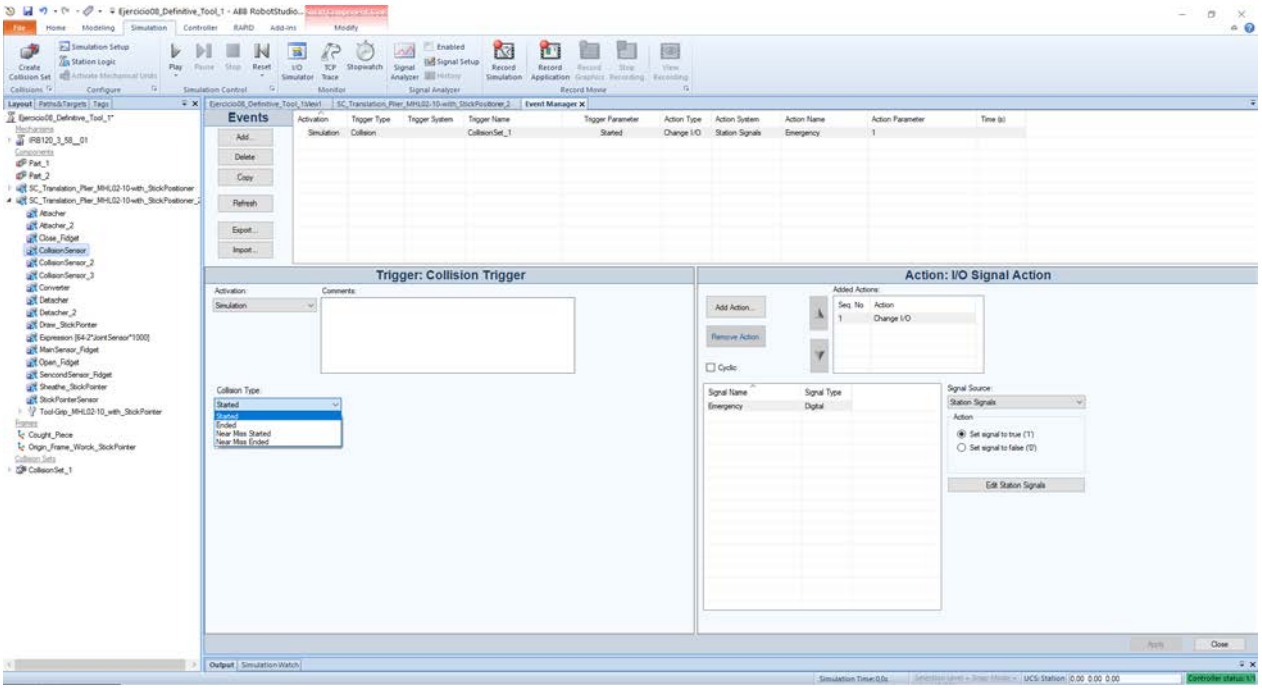


Figure 92: Configuración de Parámetros "EventManager"

Procedimiento limitado y obsoleto, ha sido **reemplazado por los SmartObject**. Aunque si solo se va a utilizar para esto es **relativamente cómodo y rápido**.

Edición limitada:

- **No posibilita apoyo gráfico.** No se colorea la colisión en la simulación
- **No aparece información por pantalla.**
- **No posible definir grupos** de "Objetos A" y "Objetos B", solo puede hacer la **comparación entre 2 objetos**
- **Interacciones lógicas con el resto de la Estación limitadas a acciones binarias simples**
- Se trata de una **gestión asíncrona**, exige ser escrupuloso sincronizándolo con el resto del sistema, o se introducirán nuevos problemas como la perdida de señal o indicador de que ha existido colisión

Análisis de la función "Detector de Colisiones" Vs "Alternativas"

a. **CollisionSet: Elemento limitado que se restringe a la visualización** tanto **en simulación** como **en edición**. Si se desea aprovechar la información de existencia de colisión, o proximidad de esta, y utilizarlo en el código del programa RAPID, exige programar un SmartObject o configurar el *EventManager*, así como definir/ajustar las interacciones en la *LogicStation* y configurar las variables/*flags* en el Controlador Virtual de la Estación.

b. **SmartObject-CollisionSensor:** Se encuentran **supeditados a los problemas de los SmartObject**, gestión de tiempo de ciclo y gestión de las E/S, a lo que hay que añadir el tiempo que cuesta diseñar y pulir uno, hasta que sea funcional y robusto. **Requiere de apoyo auxiliar (*sensores*)** salvo que solo se desee comparar 2 objetos de una lista. Es necesario indicar que **no permite comparar Grupo de Objetos A con Grupo de Objetos B**.

c. **EventManager-CollisionDetect:** La Comunidad de RobotStudio la considera una **herramienta obsoleta** y **limitada en funcionalidad comparada** con otras alternativas. Fue reemplazada cuando se implantaron los SmartObject, sujeta a los mismos fallos que estos, más algún otro. A cambio, es de **programación muy rápida y sencilla**.

d. **La detección de colisiones**: Es una función que puede **ralentizar las simulaciones, consume un elevado número de recursos del sistema**. Se recomienda tener especial cuidado en acotar apropiadamente los objetos a ser analizados/comprobados. Se aconseja, **una vez configurada la Estación** entera, **y comprobado el buen funcionamiento** con los detectores de colisiones, que **sea DESACTIVADA**.

e. **Los mensajes por pantalla**: Pueden llegar a ocultar información en la simulación. Una vez registrada la existencia de colisión, **se recomienda** ocultarlos para las futuras simulaciones.

2. 2. 5. Consideraciones del Manejo del Tiempo en Simulación

RobotStudio utiliza de **forma predeterminada** el "Modo de Divisiones de Tiempo", no obstante es posible cambiar el Modo de Ejecución de tiempo a "Modo Ejecución Libre", en caso necesario.

ATENCIÓN: Se recomienda calcular/estudiar el paso de tiempo ("*TimeStep*"), en función de la sincronización de elementos en la Estación. Dado que salpica directamente al código RAPID implementado para las simulaciones, siendo necesario forzar sincronismo: "*WaitTime [seg]*".

Sí se desea saber cómo crear un objeto geométrico ir a: **ANEXO III** a la **Ficha de Ruta: "Cómo Configurar y Medir el Tiempo en Simulación**

Modos de Simulación de Tiempo y Configuración de Parámetros

A) Tiempo de Ejecución Libre

Este Modo de ejecución de Tiempo "prioriza" el que sea correcto el tiempo de ciclo, a costa de que el establecimiento de señales y el disparo de eventos puedan ser inexactos.

Es "posible" que el comportamiento, es decir la sincronización entre elementos, entre la Estación Simulada y la Célula Real no sea exactamente la misma que si se ejecutasen de forma paralela e independiente todos los sistemas.

B) División de Tiempo: (*Por defecto*): "Time Slice"

Es posible utilizar divisiones de tiempo para **garantizar que la temporización de las señales y las demás interacciones entre los controladores sean exactas**.

RobotStudio sincroniza los controladores dividiendo un segmento de tiempo en pequeños intervalos y esperando a que todos los controladores completen la división de tiempo en curso, antes de que ninguno pueda empezar con una nueva. Por tanto, los controladores se sincronizan "correctamente".

Desventaja: no es posible abrir el FlexPendant virtual y la simulación puede resultar algo lenta, se aprecia cierto "*lag*" viendo pequeños saltos en la simulación.

ATENCIÓN: Si la simulación utiliza eventos o implica a varios controladores diferentes, DEBE USARSE EL MODO DE TIEMPO VIRTUAL DIVISIÓN DE TIEMPO (TIME SLICE) para **asegurar que la temporización entre los controladores se simule correctamente**.

Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation

La **función Cronómetro de la pestaña Simulación** se utiliza para **medir el tiempo transcurrido entre dos puntos de disparo de una simulación/proceso,** denominados **disparador de inicio y disparador de fin**. Además de permitir medir la simulación/proceso en su conjunto.

RobotStudio permite configurar varios cronómetros para una misma simulación.

Se pueden controlar los cronómetros tanto usando *SmartObject* como desde el *EventManager*, o directamente desde Código RAPID y apoyándose en el conexionado lógico de la *LogicStation*.

Para Acceder a la Configuración de los Clocks y poder así hacer mediciones en la Simulation: *Pestaña_Simulation:Monitor:Stopwatch: *to_define_parametre.*

- Procedimiento recomendado para poder medir/estimar los tiempos de ejecución de los propios *SmartObjects*.

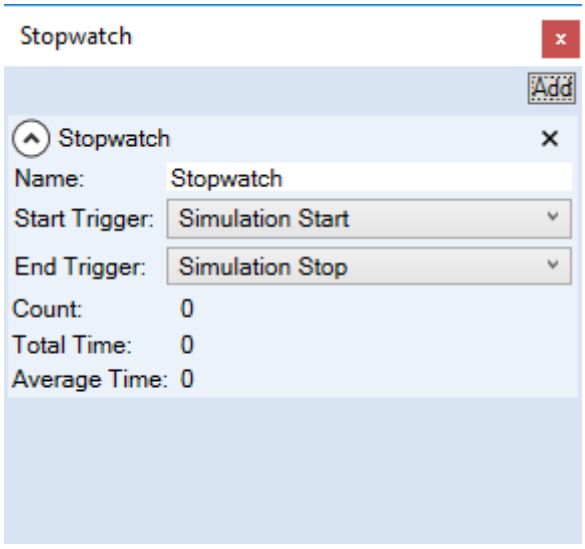


Figure 93: Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation

Control de Tiempos Específicos en Simulación de Programas: Función Clock's RAPID: Cálculo de Tiempos de Ciclo de Funciones (Tc)

Otra forma de **medir el tiempo, o tiempos de ciclo de elementos cíclicos, es desde RAPID**. Permite definir *Clock's* directamente en el código, en otras palabras permite programar el control de los *Stopwatch* "*junto con su maravillosa precisión*", para ello se han utilizado dos estrategias distintas:

a. Crear un algoritmo que Inicializa un Clock01 y pone en marcha el ejecutivo cíclico hasta que completa X ciclos, instante en el que pausa el Clock01 y muestra por pantalla su valor.
b. Crear un algoritmo que Inicializa un contador de ciclos, el Clock01 y pone en marcha el ejecutivo cíclico hasta que alcanza un EndClock=10min, instante en el que pausa el programa y muestra por pantalla el valor del contador de ciclos.

Para ambos supuestos se ha utilizado el Programa de la Estación Simulada: "**ANEXO VII**". Se ha **ejecutado 5 veces cada modo y** se ha **calculado el valor del Tiempo de Ciclo medio (T_c)** y **su desviación (σ)**. Se ha obtenido le T_c como resultado de dividir el tiempo final del programa "i" entre el número de ciclos transcurridos en el mismo programa, como se muestra en la Formula.

a) Formula del T_{CMedio}:

$$\overline{T_{c(\text{valor medio aritmético})}} = \frac{\sum_{i=0}^{j} \frac{\text{Tiempo}_i}{\text{Número de Ciclos}_i}}{j}$$

b) Datos del Análisis:

Nº Ciclos	Tiempo Total [seg]	Tc_i [ms]
*5.000	5	1,0039
10.000	21.8	2.1751
*20.000	31,167	1.55835
30.000	60,2709	2.00903
150.000	308,039	2,05359
300.000	616,2709	2,00903
500.000	1028,12	2.05621
*1.000.000	1.931,7	1,9317
*53.781	100	1,85939
*154.364	300	1,94346
*301.416	600	1,9906
450.777	900	1,99655
746.042	1.500	2.0161

Los datos en rojo se han clasificado como "espurios" y han sido removidos/eliminados del cálculo de T_C.

c) Resultados del Análisis:

$\overline{\mathbf{T_c}} = \mathbf{2,007989\ ms}$: Eliminados los valores con una **Desviación superior al 30%**

σ=0,00672145 y σ^2=0,081985

A) *Observaciones del cálculo T_C de Ejecutivos Cíclicos programados en un IRC 5 Virtual:*

- **PROBLEMA**: Estar ejecutando más programas desde el mismo procesador: Pruebas hechas en 2 tandas: "*: 20/11; frente a las de: -: 6/12"
- **Diferencia notable entre usar el procedimiento "A" y "B"**. El tiempo de computo de "A" consume más recursos obteniendo un T_{ci} mayor para valores similares en "B"
- **El valor medio de T_{ci} del Controlador que simula un PLC tiende a 2ms de media**, salvo que se le aplique una estrategia de sincronismo forzado por "*Delays*" al finalizar el "Loop".

Conclusión del Control de Tiempos con RobotStudio

Las Funciones "*Clock's*" de RAPID **están orientadas a facilitar una simulación teórica**.

El "*TimeStep*" CONDICIONA TODO lo implementado en la Estación, viéndose salpicados los códigos RAPID implementados, situación NO ACORDE A LA REALIDAD.

La duración de ejecución, y con ello **el tiempo medio de ciclo**, **de cada programa varía de una simulación a otra en función de la ocupación del procesador**. Esto es fruto de que RobotStudio está supeditado al procesador del PC utilizado y a la demanda de este (*Windows en el caso de este trabajo*). Este hecho dificulta seriamente el control y cálculo temporal de los ejecutivos cíclicos programados para controlar distintos elementos que componen la Estación.

No es una función orientada a la precisión temporal y a poder permitir una gestión de tiempos del sistema exacto, no permite tener un control finito de los ejecutivos cíclicos.

- En la práctica el procedimiento no resulta suficientemente fino ni exacto (*varia notablemente entre las simulaciones y lo qué se está ejecutando paralelamente al programa*): Equivalente a MEDIR el Grosor de alfileres con un METRO
- Se puede hacer lo mismo con un reloj en la mano y midiéndolo a ojo de buen cubero…

Las Gestiones internas llevadas a cabo por RobotStudio o el Controlador IRC 5, se encuentran bloqueadas a consulta de usuarios finales, lo que dificulta seriamente el poder exprimir esta herramienta con fines experimentales y/o aprendizaje empírico y autodidacta.

2. 3. Corolario de Observaciones Generales de RobotStudio

a) El software está provisto de un módulo de edición y modelado de objetos, aunque de carácter limitado. Acaba invitando a que se utilicen otras herramientas más adecuadas, y exportar los documentos en librerías. Cuidado, existen restricciones inherentes a las licencias de usuario (*la licencia universitaria NO permite exportar ficheros de SolidWork entre otros*).

b) Se han encontrado numerosos problemas con la gestión de Entradas y Salidas (E/S) en la simulación de programas.
Se piensa que o bien la simulación es Síncrona mientras el programa resuelve los eventos de forma instantánea/atómica, o la otra opción es que el programa estima la simulación en cada Step y entre Steps no existen actualizaciones (principio de funcionamiento de rectángulos mayorantes).

c) El contenido de los Manuales y de la "Ayuda de Programa" es el mismo (*sí, las mismas palabras e imágenes*), contenido insuficiente para reproducirlo de forma autodidacta.

d) Es un lenguaje PSEUDO INTUITIVO, algunas funciones poseen una implementación obvia y de muy fácil interpretación, otras no tanto

e) **ATENCIÓN**: Es necesario informar que existen distintas formas de guardar el contenido de la Estación Simulada. Hay que diferenciar entre la Estación Completa y las *Part*es específicas: Estación Gráfica; Configuración del Controlador; Programas Rapid (*Independiente PARA CADA Controlador*).
Deshabilitar el "Auto Guardado (AutoSave)" desde: File:Options:General:Autosave: *enables
Es un proceso irreversible o punto de retorno que te tirará por tierra muchas horas de trabajo, se ejecuta de forma cíclica y cada vez que se hace una modificación "sustancial" sobre alguna de las Partes de la Estación.

f) **ATENCIÓN**: Los ficheros NO son portables a otros terminales, a menos que se guarden en formato PackAndGo.
ATENCIÓN: Es necesario guardar aparte todas las copias de los programas RAPID de la Estación. PackAndGo sólo guarda los ficheros activos de la Estación Completa (*Estación Gráfica+Controlador+Programas*), el resto de las alternativas guardadas en las carpetas de la Estación Completa NO son clonadas en el fichero PackAndGo.

g) En las Simulaciones de RobotStudio se puede necesitar implementar "*Delay*s"/"WaitTime X" que en la Célula de Trabajo Real no serían necesarios con el Controlador IRC-5. Provoca posibles diferencias entre el Código implementado en Simulación del Código del programado en los Robots Reales.

h) **ATENCIÓN**: ¡Con la dirección de la Sincronización entre la ESTACIÓN y RAPID! Por defecto la dirección es de Estación a RAPID, incluso estando en la pestaña de RAPID Programa. *Si no se tiene en cuenta esto, se puede originar la perdida de horas de trabajo*.

i) **CONSEJO de Veterano**: Antes de meterse **en el cambio de una Herramienta** "Tool" es preferible **valorar "¡HACER UN FICHERO NUEVO!"**. Puede llegar a ser más rápido que el proceso de Cambio de Herramienta.

j) **ATENCIÓN**: Cuando se carga un bloque nuevo en la pestaña de "Modelado", SIEMPRE asigna las características por defecto, NO guarda las propiedades editadas de bloques anteriores, ni ofrece posibilidad de exportar contenido de un bloque a otro.

k) **ATENCIÓN**: Sólo es posible tener una pestaña de edición de propiedades para cada tipo de elemento, y no es posible exportar las propiedades de un elemento a otro del mismo tipo. A esto hay que añadir que, durante el transcurso de una misma sesión, NO se guarda el contenido de una pestaña abierta, si se selecciona otra pestaña del mismo tipo de elemento se perderá el contenido de información de la que se estaba editando.

l) **ATENCIÓN**: Las "Pestañas" señaladas, tienen prioridad de edición por encima de los objetos remarcados/señalados tanto en la Zona Gráfica como en el árbol de objetos. "Layout"/"Paths&*Targets*"/"Tags"… EJEMPLO: TaregtPoint with WorkObject…

m) **ATENCIÓN**: Si como consecuencia de una acción sobre algo existente provoca la creación de un elemento nuevo: No se guardarán las propiedades de los objetos padres.
Ejemplo: *CreateFrame (+Attach to) + Convert Frame to WorkObject: New WorkObjects not memory "Attach to"*

n) **ATENCIÓN**: El programa no emite ningún aviso por pantalla cuando al implementar un controlador nuevo, o se añade/edita funcionalidades nuevas al controlador existente, la consecuencia es que RobotStudio carga un fichero RAPID nuevo y vacío, eliminando todo el contenido NO GUARDADO en el fichero RAPID anterior de la pestaña "ProgramRAPID".

o) **ATENCIÓN**: Los objetivos programados "*Target*", definidos en la Estación, SOLO se pasan al programa RAPID por medio de la Sincronización y, si y solo si, pertenecen a una trayectoria activa "Path" que se esté sincronizando desde la Estación a RAPID.
"**Esto es una restricción muy importante dado que:** ***¡No se pueden exportar los Target sin un Path que los contenga!***"

p) **ATENCIÓN**: El Robot NO ve las clases *Frame* ni *Object*, solo es capaz de entender los *Target* que están en un *WorkObject*.

q) **ATENCIÓN (*RESTRICCIÓN MUY IMPORTANTE*):** En la creación de un Programa RAPID:
El Computo de una orden Lógica es notablemente menor que el Computo de Transformadas de Movimiento, el Programa puede estar comunicando antes de haber acabado un movimiento, es decir antes de haber alcanzado un punto deseado.
Solución: Forzar Sincronización "*Delay*" o usar ordenes específicas en las funciones de movimiento "fine**".**

ANEXO III

3. FICHAS de Ruta: "*Cómo se hace…*"

Ficha de Ruta: "Cómo hacer un Objeto"

Definición de un Modelo Físico:

Durante la programación o simulación es necesario usar modelos virtuales de piezas, de trabajo y/o equipos. Para ello, **RobotStudio facilita bibliotecas y/o geometrías propias**. Por medio de una herramienta de diseño CAD que permite elaborar modelos geométricos simplificados.

Otra opción de trabajo es importarlos, tanto como geometrías como en forma de biblioteca (*fichero externo asociado*), en caso de disponer de los modelos de CAD de las piezas. Para ver los tipos de formato permitidos ver **Tabla 9**: **Extensiones Importación de Geometrías y Bibliotecas** , del **ANEXO II**.

Propiedades de la Creación de Objetos:

A) Construcción de la Geometría

Los **Modelos Físicos**, o representaciones parametrizadas de objetos físicos reales, **están definidos por** un **nodo superior** de la geometría, denominada "Pieza" (*Part*), y por uno o varios "Cuerpos" (*Body*).

a) Piezas Primitivas:

Las *Parts* Primitivas, **compuestas por un único *Body***, pueden ser de diversos tipos (*sólidos, superficies* o *curvas*):

- Los **cuerpos de tipo Sólido** son objetos de 3D creados a *Part*ir de Caras.
  Se reconocen en la Pestaña Grafica por aparecer representados como un cuerpo con varias caras. Se consideran sólidos primitivos (*Tetraedros, Conos, Cilindro, pirámides* o *Esferas*) que se pueden combinar para crear cuerpos más complejos.
- Los **cuerpos de tipo Superficie** son objetos de 2D con una sola cara.
  Las piezas compuestas por varios cuerpos 2D unidos (*Círculo, Cuadrado, Polígono o Superficie con Curva*), es decir con una superficie 2D para cada cara que represente el sólido, no se consideran auténticos sólidos de 3D.
- Los **cuerpos de tipo Curva**, representados únicamente con el nodo del cuerpo en el navegador Modelado Gráfico, no contienen ningún nodo hijo.
  Tipos: *Línea, Círculo, Arco, Arco Elíptico, Elipse, Rectángulo, Polígono, Polilínea* o *Spline*.

a) Piezas Compuestas:

Parts* compuestas por más de un *Body, se pueden crear gracias a herramientas encargadas de asociar interacciones entre *Bodies* (*Union, substract e Intersection*), ya sean *Bodies* de una misma *Part* y/o pertenecientes a distintas *Part*s (*añadiendo, trasladando o eliminando Bodies*).

- **ATENCIÓN**: La aplicación de estas interacciones **SE HACE DE 2 EN 2!!!**

B) *Pestañas de Menú:*

RobotStudio diferencia entre las Pestañas de Creación o Modelado y las pestañas de Modificar una *Part* o un *Body* específicos.

a) Menú de la Pestaña "Modeling":

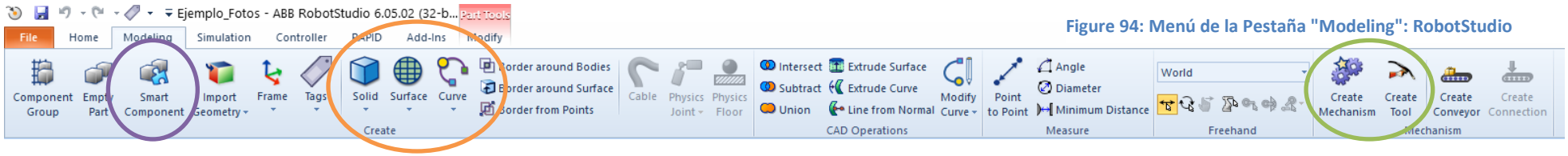


Figure 94: Menú de la Pestaña "Modeling": RobotStudio

Pertenece a las Pestañas Principales de RobotStudio, accesible en todo momento desde el Menú de Programa.

b) Menú Pestaña "Modify *Part*":

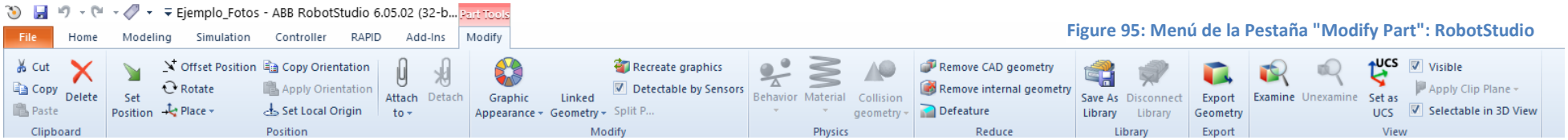


Figure 95: Menú de la Pestaña "Modify Part": RobotStudio

Para poder desbloquear la Pestaña de Modificar una *Part* es necesario seleccionar previamente la *Part* a Modificar, ya posea un único *Body* o varios.

c) Menú Pestaña "Modify *Body*":

Figure 96: Menú de la Pestaña "Modify Body": RobotStudio

Para desbloquear, la Pestaña de Modificar un *Body*, es necesario seleccionar el *Body* deseado perteneciente a una *Part* concreta.

Para crear un objeto nuevo únicamente hay que seleccionar una de las opciones contenidas en el Óvalo naranja de la **Figure 94**: Menú de la Pestaña "Modeling": RobotStudio.

Para Modificar un Objeto existente: Como se aprecia en las imágenes (**Figure 94**, **Figure 95** y **Figure 96**), la capacidad de Edición de Objetos en RobotStudio es muy limitada, los cambios autorizados son:

- **Modificación de la *Part* representada**, por medio de interacciones entre *Bodies* (*Union, substract e Intersection*)
- **Reasignación y composición de *Bodies* en nuevas *Parts*** (*añadiendo, trasladando o eliminando Bodies*)
- **Modificaciones Estéticas y/o de propiedades físicas asociadas** (*material del que se compone y color visualizable*)
- **Modificaciones de las *Frame*** asociadas
- **Modificar la Ubicación de la *Part* en la Estación** o de los *Bodies* respecto de la *Part* padre.

ATENCIÓN: RobotStudio NO permite Modificar ni Escalar las dimensiones físicas asignadas a los Objectos creados en la Pestaña de "Modeling".

C) *Geometría matemática:*

La **geometría de un archivo de CAD** siempre tiene una **representación matemática subyacente**. Su representación gráfica, mostrada en la ventana de gráficos, se genera a *Part*ir de la representación matemática al importar la geometría desde RobotStudio. A *Part*ir de ese momento, la geometría recibe la denominación de "*Part*".

- **El nivel de detalle de la representación gráfica,** de este tipo de geometría, **es configurable**.
  - Se reduce el tamaño de archivo y el tiempo de representación de los modelos de gran tamaño.
  - Se mejora la visualización de los modelos pequeños que quizá desee ampliar.
- **El nivel de detalle sólo afecta a la visualización**. Las trayectorias y curvas, creadas a *Part*ir del modelo, serán exactas con los ajustes de visualización, ya sea visualización aproximada o detallada.
- **Los objetos NO son sólidos**, propiamente dicho. E**l software opera teniendo en cuenta las intersecciones con los planos que definen los objetos**.

Para más información de "Cómo crear un Objeto" o de las distintas funcionalidades de las Herramientas de Modelado y/o Modificación, se recomienda ver el "***ManualOperdor-RobotStudio***":

- "*2 Creación de Estaciones:2.7 Modelado: 2.7.1 Objetos (107/648)*"
- "*9 Pestaña Modelado (311/648)*"

Ficha de Ruta: "Cómo hacer un SmartObject"

Un componente inteligente **es un objeto de RobotStudio, con o sin representación gráfica, que representa el comportamiento de otros componentes "inteligentes"** presentes en el mundo real.

Es una alternativa al uso de un compilador de bibliotecas basado en XML.

Para crear un SmartObject nuevo únicamente hay que seleccionar la opción contenida en el **Óvalo morado** de la **Figure 94**: Menú de la Pestaña "Modeling": RobotStudio, y configurar la Lógica Cableada deseada.

Como se puede ver en las imágenes **Figure 97** y **Figure 98**, es posible diseñar un **objeto inteligente como una combinación de bloques** de distinta naturaleza, catalogados en las siguientes clases:

- **Señales y Propiedades "Signal and Properties"**: Gestión de Señales Lógicas, ya sean de carácter lógico Binario y/o Analógico
- **Estructuras Primitivas "Parametric Primitives"**: Creación e interacción con sólidos locales (generados y gestionados por el propio SmartObject)
- **Sensores "Sensor"**: Gestión de elementos de detección y/o ubicación de los elementos móviles gestionados por el SmartObject, ya sean directamente de creación del SmartObject o la Estación
- **Acciones "Actions"**: Generación e interacción entre elementos gestionados por el SmartObject, como: enlazar "Attach"/ desenlazar "Detach", vincular "SetParent", Mostrar "Show" / esconder ""Hide,…
- **Manipulación "Manipulation"**: Gestiona las acciones de traslaciones y/o rotaciones de los elementos gestionados por el SmartObject, ya sean sólidos externos y/o los sólidos/ejes que componen un mecanismo/robot
- **Otros "Other"**: Compendio de bloques de naturaleza diversa orientados a la simulación de sistemas

La complejidad del objeto inteligente queda supeditada al nivel que el usuario desee implementar y a las limitaciones propias del software, y procesador utilizado. Ver las Consideraciones de trabajar con **Consideraciones de los Objetos Inteligentes "*SmartObjects*"** en el **ANEXO II**.

ATENCIÓN: Es posible dotar de protección a un componente inteligente, oculta su estructura interna y lo protege contra la edición, protege su funcionalidad. Los componentes subordinados de un componente inteligente protegido permanecen ocultos en todos los navegadores de RobotStudio, **CUIDADO** también se encontrará oculto en el navegador "Analizador de señales". Esta opción, es una forma de ocultar la complejidad, **no tiene como fin aportar seguridad ni protegerlo de forma inalterable**.

La **explicación de cada uno de los bloques se encuentra en** la *Ayuda "F1" del software RobotStudio* y/o en el "***ManualOperdor-RobotStudio***":

- "*9 Pestaña Modelado:9.4 Componentes Inteligentes: (314-347/648)*"

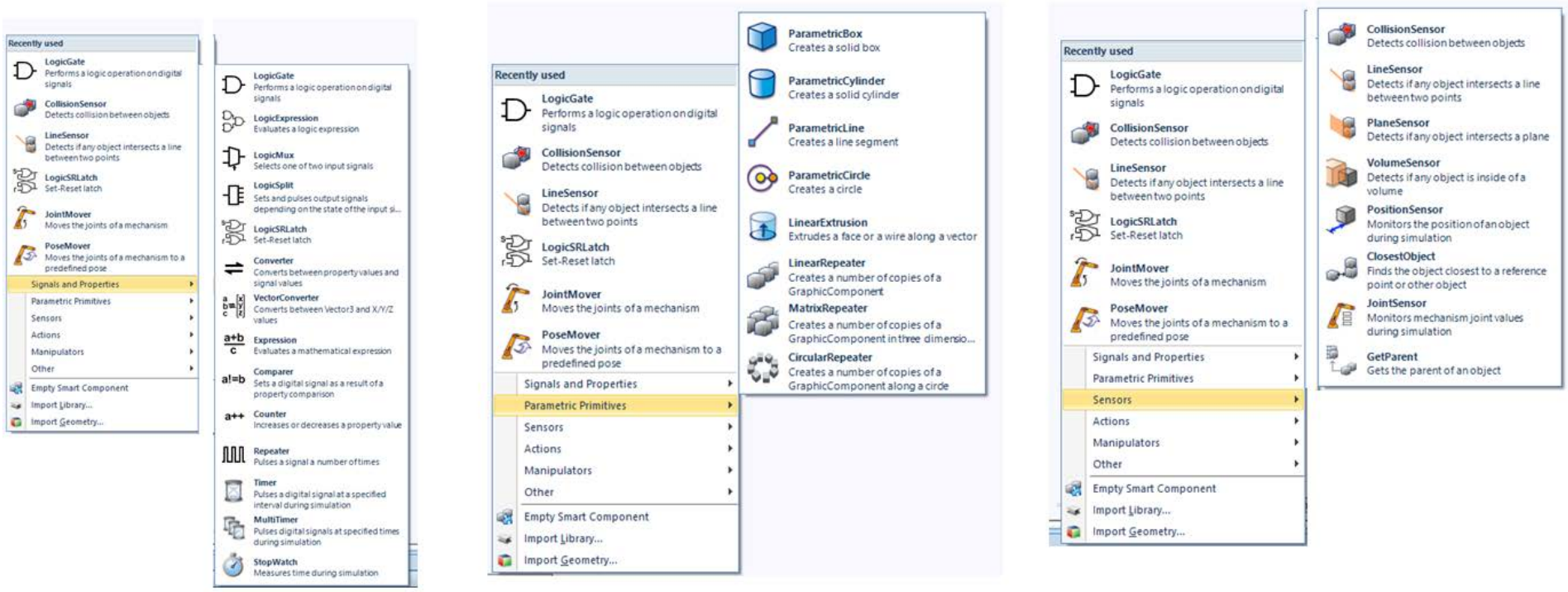


Figure 97: Menú Pestañas: "Signals and Properties", "Parametric Primitives" y "Sensors": SmartObject: RobotStudio

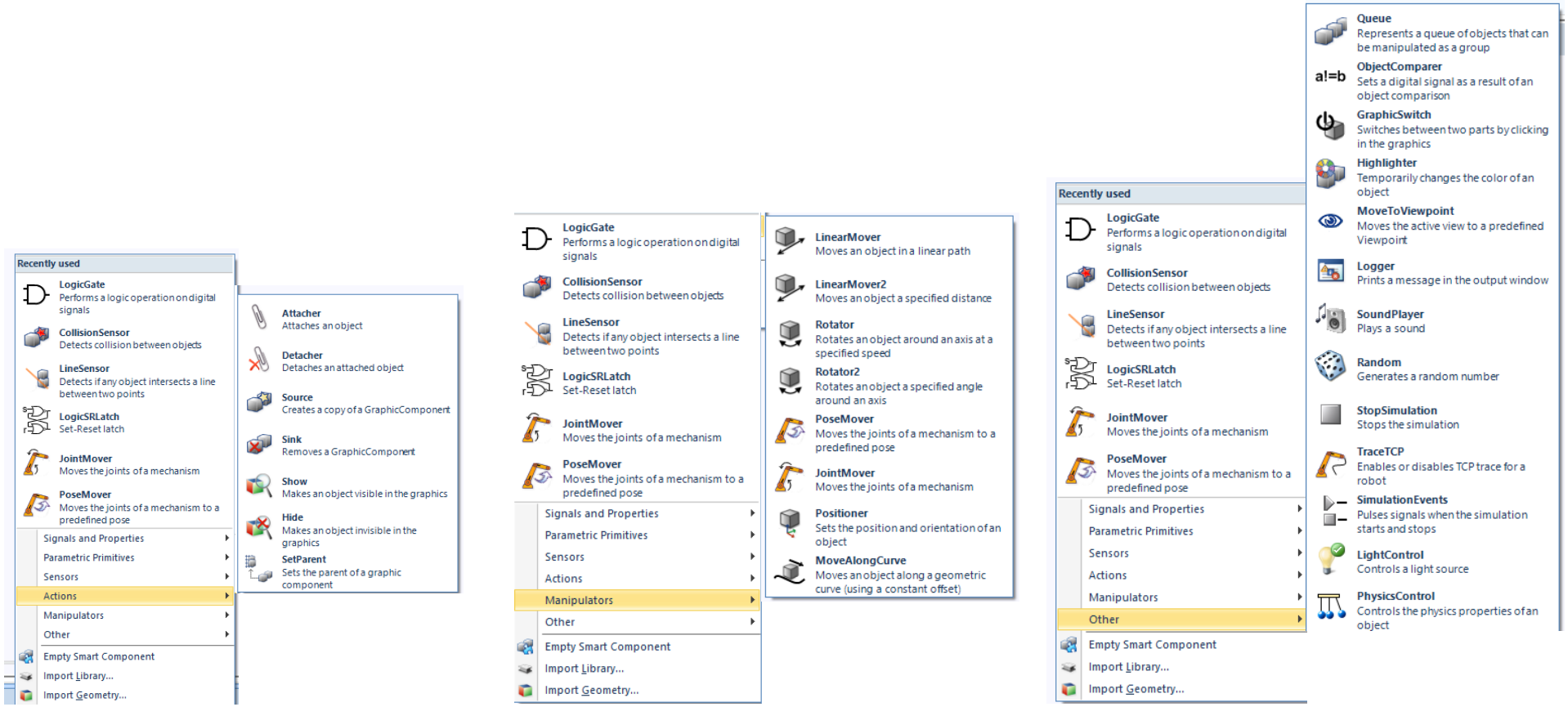


Figure 98: Menú Pestañas: "Actions", "Manipulators" y "Other": SmartObject: RobotStudio

Ficha de Ruta: "Cómo hacer una Herramienta '*Tool*"

Herramientas Simples, *sin Partes móviles*: "Asistente de Creación de Herramientas"

Para crear una Herramienta Simple, únicamente hay que seleccionar la opción de "***Create Tool***" contenidas en el **Óvalo verde** de la **Figure 94**: Menú de la Pestaña "Modeling": RobotStudio.

A) *Asistente de Creación de Herramienta:*

Permite crear fácilmente una herramienta a *Part*ir de una *Part* ya existente, o usando una *Part* simulada para representar una herramienta.

B) *Procedimiento:*

Seguid al Asistente de Creación, **Figure 99**, e id completando con los datos de herramienta:

1. Haga clic en **Crear herramienta**.
2. En el **cuadro Nombre** de herramienta, introduzca un nombre de herramienta y seleccione una de las opciones siguientes:
3. **Introduzca la Masa** de la herramienta, su **Centro de gravedad** y el **Momento de inercia** I_x, I_y, I_z, si conoce estos valores. De lo contrario, **si solo se va a simular, dejarlos por defecto**.
4. Haga clic en Siguiente.
5. En el cuadro **Nombre de TCP**, asignar un nombre para el punto central de la herramienta (TCP).
6. Introduzca la **posición del TCP respecto del sistema de coordenadas mundo**, **representa el punto de montaje de la herramienta**, por cualquiera de los métodos facilitados.
7. Haga clic en el botón de "**Óvalo verde** de **Figure 99**" para **transferir los valores al cuadro TCP**.
   Si la herramienta tiene varios TCP's,: Repita los pasos del 5 al 7 con cada TCP.
8. Haga clic en Terminado.

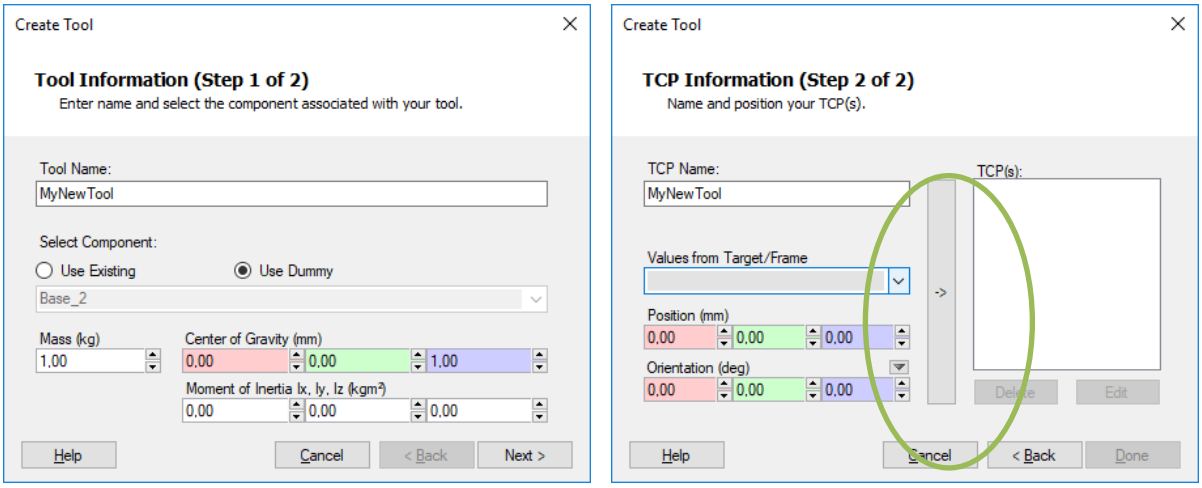


Figure 99: Menú Asistente de Creación: "Create Tool": RobotStudio

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

Herramientas Compleja, *incluye Partes móviles*: "Creación de un Mecanismo"

Para crear una Herramienta Compleja, únicamente hay que seleccionar la opción de "***Create Mechanism***" contenidas en el **Óvalo verde** de la **Figure 94: Menú de la Pestaña "Modeling": RobotStudio**, y Seguid al Asistente de Creación:

1. Haga clic en Crear Mecanismo. Se abrirá el Asistente de Creación, modelador de mecanismos, en el modo de creación.

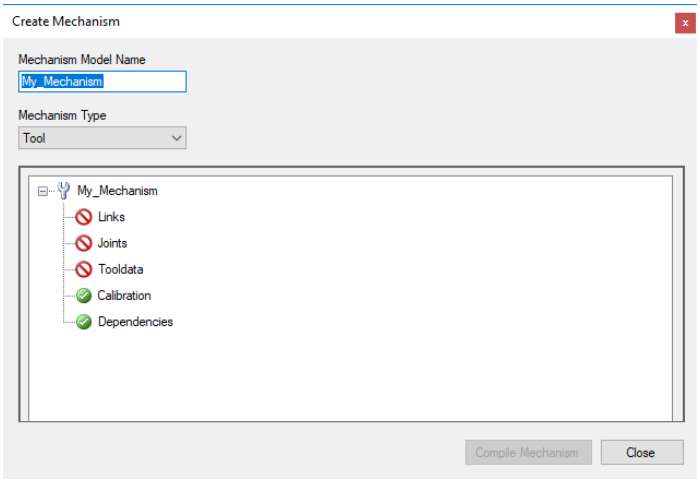


Figure 100: Menú Asistente de Creación: "Create Mechanism": RobotStudio

2. En el **recuadro Nombre**: introducir un nombre de mecanismo.
3. En la lista **Tipo de Mecanismo**: seleccione tipo Herramienta "***Tool***".
4. **Definir Eslabones (*paso obligatorio*)**: En la **estructura de árbol**: seleccionar con un clic del botón derecho en Eslabones "*Link*" y a continuación hacer clic en Añadir eslabón. se mostrará la ventana de diálogo: **Se deberá seleccionar la pestaña "Set a BaseLink", en al menos uno y únicamente en un Eslabón, que se tomará como referencia Base**. Crear eslabón:

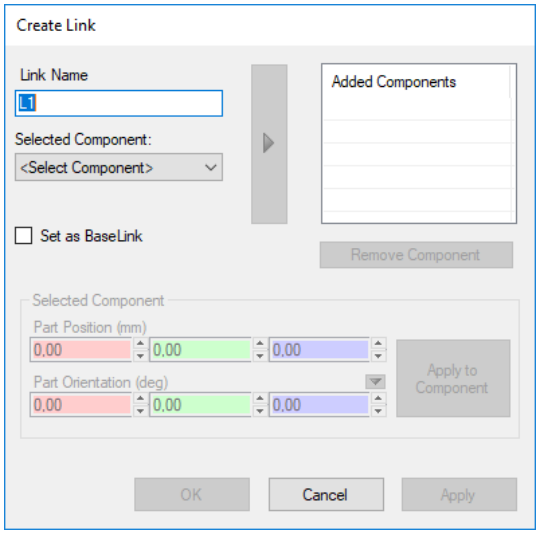


Figure 101: Menú Asistente de Creación: "Create Link": "Create Mechanism": RobotStudio

5. **Seleccionar las *Parts* del Eslabón "*Link*"**: En la lista de selección de *Parts*, seleccionar una *Part* (*se resaltará en la ventana de gráficos*) y hacer clic en el botón de flecha para añadir la pieza al cuadro de componentes del Eslabón. Atención: En la lista de *Parts* se seleccionará automáticamente una nueva pieza pero no formará *Part*e del Eslabón hasta que no forme *Part*e del cuadro de componentes del Eslabón, en concreto la pieza siguiente. Añada tantos elementos como sea necesario.
6. **Definir propiedades del Eslabón "*Link*"**: Para cada una de las *Part*es del Eslabón se deben introducir los valores de la referencia respecto de la base, NO la del Eslabón, RobotStudio realiza la transformación entre la referencia parte, la referencia mundo y la referencia base. A continuación hacer clic en Aplicar a pieza. Repita el proceso con cada pieza del Eslabón según sea necesario. Finalmente haga clic en Aceptar.
7. **Repetir los paso 5 y 6,** hasta tener todos los Eslabones que componen la *Tool*.

8. **Creación de Ejes**:
   Es estrictamente necesario que un Eslabón tenga la categoría de Base de la Herramienta, seleccionada la opción "Set a BaseLink".
   En la estructura de árbol, hacer clic con el botón derecho en Ejes "*Joint*" y a continuación hacer clic en Añadir eje para mostrar la ventana de diálogo Crear Eje "*Create Joint*". Completar la ventana de diálogo Crear Eje y a continuación haga clic en Aceptar.

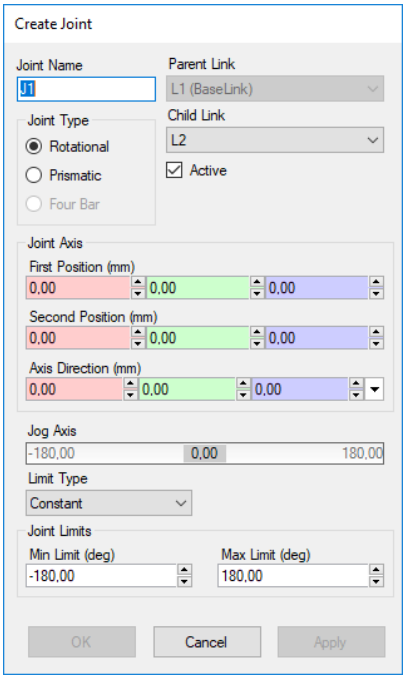


Figure 102: Menú Asistente de Creación: "Create Link": "Create Joint": RobotStudio

9. **Crear TCP's (*paso obligatorio*)**: En la estructura de árbol, hacer clic con el botón derecho en Datos de Bases de Coordenadas/Herramientas "*Create Tooldata*" y haga clic en Añadir base de coordenadas/herramienta para mostrar la ventana de diálogo Crear base de coordenadas/herramienta.

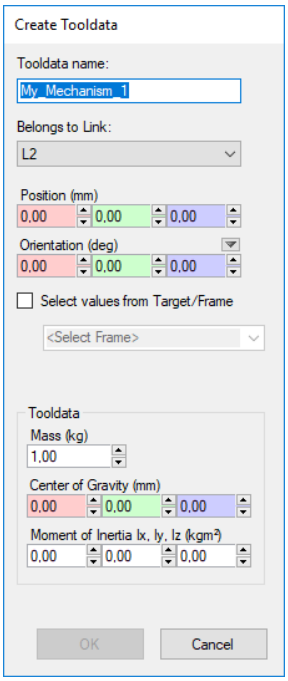


Figure 103: Menú Asistente de Creación: "Create Link": "Create Tooldata": RobotStudio

Completar la ventana de diálogo Crear Base de Coordenadas/Herramienta y a continuación hacer clic en Aceptar.

10. **Repetir los pasos 10 y 11,** hasta tener todos los TCP's que componen la *Tool*.

Sólo se pueden hacer los pasos 11 y 12 si se ha definido el paso 8:

11. **Configurar la Calibración (*paso opcional*)**: En la estructura de árbol, hacer clic con el botón derecho en Calibración y, a continuación, hacer clic en Añadir calibración para abrir la ventana de diálogo Crear calibración. Seguir el asistente.
    Complete la ventana de diálogo Crear calibración y, a continuación, haga clic en Aceptar.
12. **Configuración de Ejes Dependientes (*paso opcional*)**: En la estructura de árbol, haga clic con el botón derecho en Dependencia y, a continuación, haga clic en Añadir dependencia para abrir la ventana de diálogo Crear dependencia.
    Complete la ventana de diálogo Crear dependencia y, a continuación, haga clic en Aceptar.

13. **Crear la *Tool*:** Si, y solo si, todos los nodos son válidos (*aparece un "Check" verde*): **Compilar el Mecanismo**.

A) *Compilación de Mecanismos:*

1. **Compilar mecanismos nuevos**: se añadirá un nuevo mecanismo en la Estación, con el nombre predeterminado "Mecanismo_" seguido de un número de índice.
   Compilación de mecanismos existentes: se guarda sin ninguna pose o correlación de ejes, ni tiempos de transición.
2. Para compilar un mecanismo, realice las operaciones siguientes:
   a. Mecanismo nuevo o editado, haga clic en Compilar mecanismo.
      El mecanismo se inserta en la estación activa. Las *Part*es del eslabón son clonadas con nombres nuevos, pero los eslabones actualizan sus referencias de pieza. **Al cerrar el modelador de mecanismos, estas piezas clonadas se eliminan**.
   b. El modelador de mecanismos pasará al modo de modificación.

B) *Para completar y/o modificar el modelado de un mecanismo:*

1. Si los valores del grupo Correlación de ejes son correctos, hacer clic en Establecer.
2. **Configurar la cuadrícula Poses**: Para añadir una pose hacer clic en Añadir y a continuación completar la ventana de diálogo Crear pose.
   a. Para **añadir una pose**: hacer clic en Añadir y a continuación completar la ventana de diálogo Crear pose. Hacer clic en Aplicar, seguido de Aceptar.
   b. Para **editar una pose**: seleccionar en la cuadrícula y hacer clic en Editar y a continuación completar la ventana de diálogo Modificar pose. Hacer clic en Aceptar.
   c. Para **eliminar una pose**: seleccionar en la cuadrícula y hacer clic en Eliminar.
3. Editar los tiempos de transición.
4. Y Cerrar.

Ficha de Ruta: "Cómo hacer un Cambio de Herramienta"

En primer lugar, hay que mencionar que, **el cambio de herramienta en RobotStudio no es inmediato**, el **grado de complejidad es directamente proporcional a** lo **completa** que se encuentre la **Estación y** la **dependencia de la *FrameTool*$_i$ con los *WorkObjects*$_j$**, pudiendo llevarse unas pocas horas.

En segundo lugar, es **necesario ser cuidadoso con el procedimiento**. Se recomienda ver el a*part*ado: **Consideraciones en el Proceso de Cambio de Herramientas desde RobotStudio** en el **ANEXO II**.

Procedimiento Cambio de Herramienta:

1. ¿**Es necesario eliminar la "OldTool**"?
   - **No**: Ocultar la visualización de la Herramienta (eliminar el "*tick*" de la Opción "Visible")
   - **Sí**: Desenlazar (*"Detach"*) la Herramienta: si es simple pulsar botón derecho sobre la imagen de Llave Inglesa, si es compleja dentro del Controlador Lógico, buscar y seleccionar la Llave Inglesa.
     Se preguntará por pantalla si se desea restaurar la posición de la Herramienta
     (***Sí****: la herramienta se reubicará donde se encuentre su FrameTool;*
     No*: la herramienta permanecerá en la última ubicación espacial en la que se encontrase el Robot*)

2. **PASO OPCIONAL**: **Eliminar la "OldTool" del "tooldata"**:
   - **No**: Aconsejable no hacerlo.
   - **Sí**: **ATENCIÓN**: RobotStudio modificará sin avisar la "*FrameOldTool*" por "*Tool0*" en todos los *Paths* y RAPID_Programas en los que apareciese "*FrameOldTool*".

3. **Cargar** ("*Load*") **y Vincular** ("*Link*") **la nueva "*NewTool*"**:
   - OBLIGATORIO DESACOPLAR DE LA LIBRERÍA para poder enlazar "*Link*" la "*NewTool*"
   - CUIDADO: La "*NewTool*" se debe acoplar al Robot, no al *Link6*(Brida) del Robot. ACOPLAR NO ES ENLAZAR "*Attach*", acoplar implica cargar la información de la Herramienta y poder así hacer los cálculos de Transformación de Matrices.

4. **IMPORTANTE**: **Asignar en la pestaña *Tool* la "*NewTool*" y anclarla** en el árbol del Robot ("*System1:T_ROB1:Tooldata:*NewToolFrame*") **como "*Active Tool*".**

5. **Comprobar la Alcanzabilidad "*Reachability*" de la "*NewTool*"** para toda $Path_i$ en todos los $WorkOjects_j$ donde entre en juego.

   - **ATENCIÓN**: Si se ha hecho el "Paso 2": Para todo Objetivo ("$Target_n$"): *Tool* ="*Tool0*", necesario cambiarlo desde RAPID y/o eliminar todos los $Path_i$ y volver a definir todos los Objetivos.
   - Si la "*NewTool*" es física y estructuralmente distinta, posee distinta ubicación y orientación de TCP, será necesario Adaptar/Corregir los $Traget_n$.

6. **Sincronizar con RAPID**: seleccionar sentido de la Estación a RAPID: Actualizando todos los $Target_n$, $Path_i$, *WorkObjects* y *tooldata*.

7. **Modificar manualmente** o con la ayuda de "*Find*/*Remplace*":
   - *Find and Replace*: Modificar para toda "*OldFrame*" transformar en "*NewFrame*"
   - Repasar/comprobar todas las posibles referencias/Links a las "*OldFrame*"

8. **OPCIONAL**: Sincronizar con RAPID: de RAPID a la Estación: $Path_i$
   - Si no se desea Sincronizar: Seleccionar "*Apply*": Aplicará los cambios de código RAPID al Robot sin modificar contenido de la Estación, ejecutando desde la Pestaña de RAPID-Controlador.

9. **Ejecutar intento no tener que aplicar correcciones, "***y cruzar los dedos***"**, para NO tener que aplicar correcciones.
   Fallo Habitual: Repetir desde el Punto 5 en adelante.

Ficha de Ruta: "Cómo Configurar un Detector de Colisiones"

RobotStudio permite **detectar y registrar las colisiones que se producen entre los objetos** de la estación de diferentes formas.

Se recomienda **ver el a*part*ado**: **Consideraciones de los Detectores de Colisiones** en el **ANEXO II**.

Configuración Parámetros Generales de la Estación:

Para acceder a la configuración de parámetros generales existen dos posibles vías:

- **Desde el Menú de Opciones**: File:Options:Simulation:Collision: *enables
- **Desde la Pestaña de Simulation**: Simulation:Collision_Detection_Option: Options:Simulation:Collision: *enables

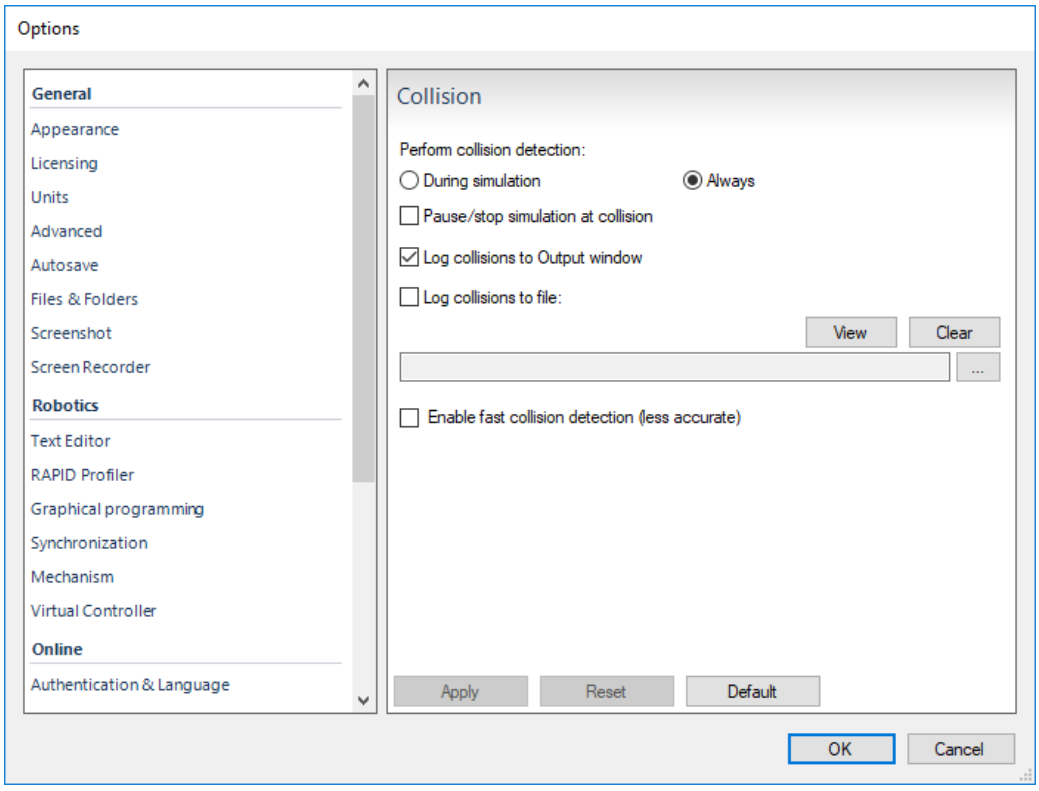


Figure 104: Configuración Parámetros Generales de la Estación

En esta Pestaña se podrá decidir:

- Si se desea que la Detección de Colisiones se resuelva exclusivamente en las Simulaciones o se dé también durante el proceso de elaboración de la Estación.
- Si se desea acceso a la información de Detección de Colisiones por pantalla emergente y/o guardar la información en algún fichero externo a RobotStudio.

Configuración Parámetros Específicos de Detección de Colisiones: Estación

Ejecutando la funcionalidad "Create Collision Set" en el directorio siguiente de RobotStudio: *Simulation:Collision:Create_Collision_Set:(click botón derecho sobre el bloque de Collision)Properties_Collision_Set:*

En ella se configuran las propiedades del bloque de forma individual, permitiendo la visualización gráfica de las colisiones, o poder registrar las colisiones en la ventana de salida o en un archivo de registro separado.

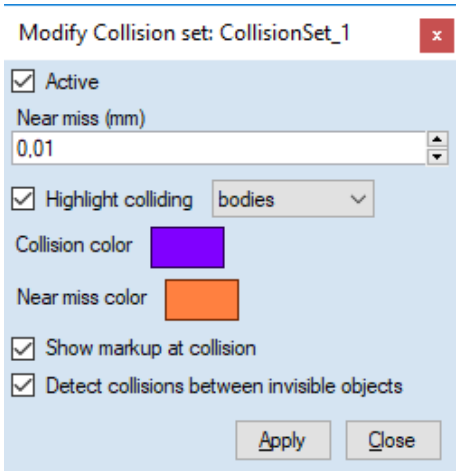


Figure 105: Configuración Parámetros Específicos de Detección de Colisiones: Estación

Propiedades Principales:

- **Posibilita apoyo gráfico.** Se pueden seleccionar colores distintos tanto para la Colisión como para Cuasi-Colisión, los objetos se colorean ante la colisión en la simulación
- **Permite la aparición de información de la Colisión por pantalla**
- **Posible definir dos grandes grupos** de "Objetos A" y "Objetos B", pero **solo puede hacer la comparación entre los grupos**, no entre objetos específicos de cada grupo.
- **Se puede Programar varios grupos distintos**, creando una jerarquía de objetos para acotar la identificación de Colisión.

Configuración Parámetros Específicos de Detección de Colisiones: SmartObject

Uso del "*CollisionSensor*" Detector Interno del Controlador de la Herramienta (*SmartObject*):

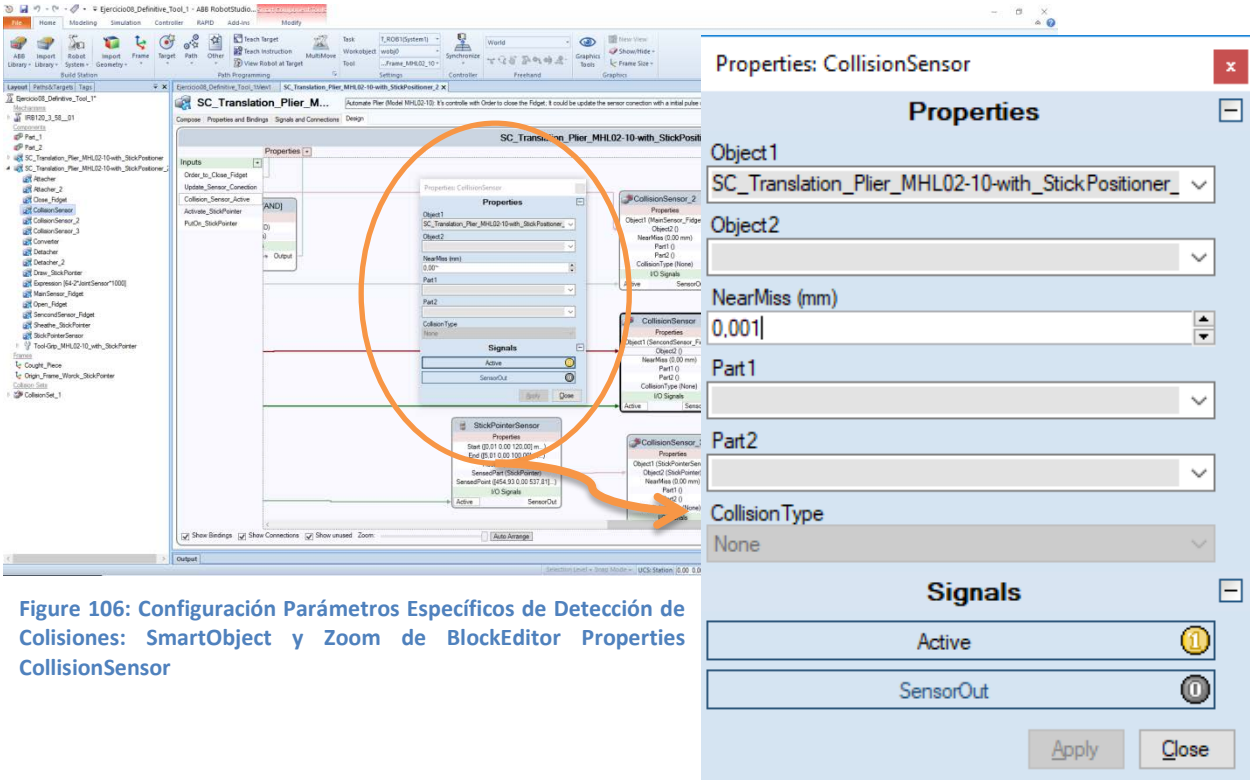


Figure 106: Configuración Parámetros Específicos de Detección de Colisiones: SmartObject y Zoom de BlockEditor Properties CollisionSensor

Es **necesario configurar** tanto las **interacciones de lógicas del SmartObject** y de la estación, como **editar/definir unas propiedades mínimas en el bloque de propiedades**, edición limitada:

- **No posibilita apoyo gráfico.** No se colorea la colisión en la simulación
- **No aparece información por pantalla**
- **No posible definir grupos** de "Objetos A" y "Objetos B", **solo puede hacer la comparación entre 2 objetos**

A cambio, **posibilita**:

- **Poder crear una variable propia en el SmartObject**, y por ende en la Estación, que **poder interconectar con el controlado y reutilizar en el código de RAPID**
- **Aprovechar la información procedente de otros bloques del SmartObject**, como por ejemplo los objetos sensados/detectados por uno o dos sensores
- **Crear funciones lógicas complejas de detección de objetos**, más fieles y próximas a las interacciones de la realidad
- Comprobación **más finita que el CollisionSet** (*CollisionSet: Se basa en la comparación por esfera, excluye los elementos terminados en punta*)

Para más observaciones del *CollisionSensor* , ver el a*part*ado: "**CollisionSensor**:".

Configuración Parámetros Específicos de Detección de Colisiones: EventManager

Simulation: "Configure: *Recuadro esquina inferior izquierda* (*sí está muy escondido*): EventManager:Add: Simulation:Collision:*Define_Parametres"

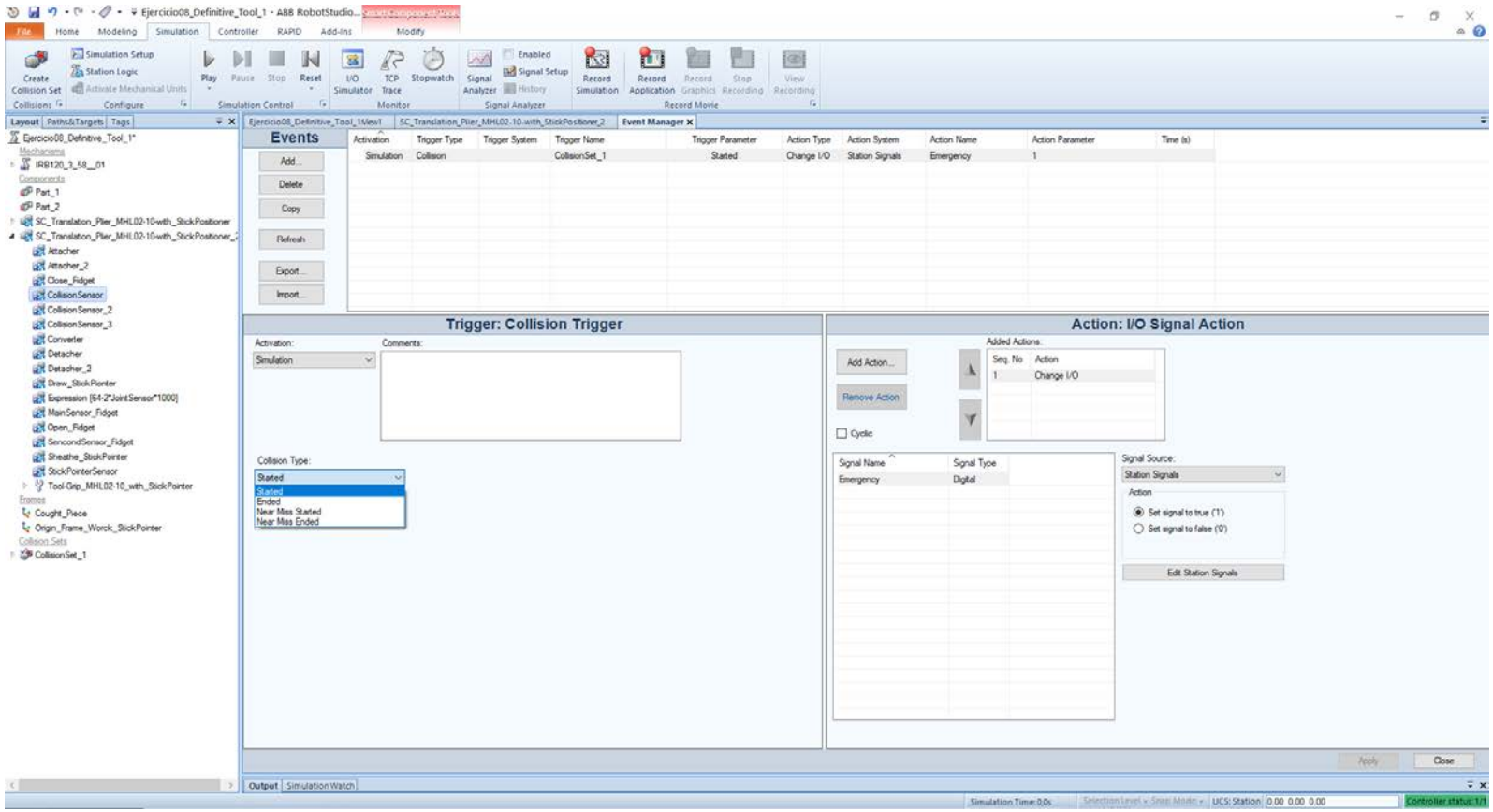


Figure 107: Configuración de Parámetros "EventManager": RobotStudio

Procedimiento limitado y obsoleto, ha sido **reemplazado por los SmartObject**. Aunque si solo se va a utilizar para esto es **relativamente cómodo y rápido**.

Edición limitada:

- **No posibilita apoyo gráfico.** No se colorea la colisión en la simulación
- **No aparece información por pantalla.**
- **No es posible definir grupos** de "Objetos A" y "Objetos B", solo puede hacer la **comparación entre 2 objetos**
- **Interacciones lógicas con el resto de la Estación limitadas a acciones binarias simples**
- Se trata de una **gestión asíncrona**, exige ser escrupuloso sincronizándolo con el resto del sistema, o se introducirán nuevos problemas como la perdida de señal o indicador de que ha existido colisión

Ficha de Ruta: "Cómo Configurar y Medir el Tiempo en Simulación"

RobotStudio utiliza de **forma predeterminada** el "Modo de Divisiones de Tiempo", no obstante es posible cambiar al Modo de Ejecución de tiempo a "Modo Ejecución Libre".

Tipo de Simulación de Tiempo

A) *Tiempo de Ejecución Libre*

Este Modo de ejecución de Tiempo "prioriza" que sea correcto el tiempo de ciclo, a costa de que el establecimiento de señales y el disparo de eventos puedan ser inexactos.

Es "posible" que el comportamiento, es decir la sincronización entre elementos, entre la Estación Simulada y la Célula Real, no sea exactamente la misma que si se ejecutasen de forma paralela e independiente todos los sistemas.

B) División de Tiempo: (*Por defecto*): "Time Slice"

Es posible utilizar divisiones de tiempo para **garantizar que la temporización de las señales y las demás interacciones entre los controladores sean exactas**.

RobotStudio sincroniza los controladores, dividiendo un segmento de tiempo en pequeños intervalos y esperando a que todos los controladores completen la división de tiempo en curso, antes de que ninguno pueda empezar con una nueva. Por tanto, los controladores se sincronizan "correctamente".

Desventaja: no es posible abrir el FlexPendant virtual y la simulación puede resultar algo lenta, se aprecia cierto "*lag*" viendo pequeños saltos en la simulación.

ATENCIÓN: Si la simulación utiliza eventos o implica a varios controladores diferentes, DEBE USARSE EL MODO DE TIEMPO VIRTUAL DIVISIÓN DE TIEMPO (TIME SLICE) para **asegurar que la temporización entre los controladores se simule correctamente**.

Modos de Configuración de Parámetros Temporales

A) *Configuración del Tiempo de Simulación:*

Para configurar los parámetros temporales hay que acceder al Menú de Opciones de RobotStudio: "*File:Options:Simulation:Simulation_Clock: *enables*":

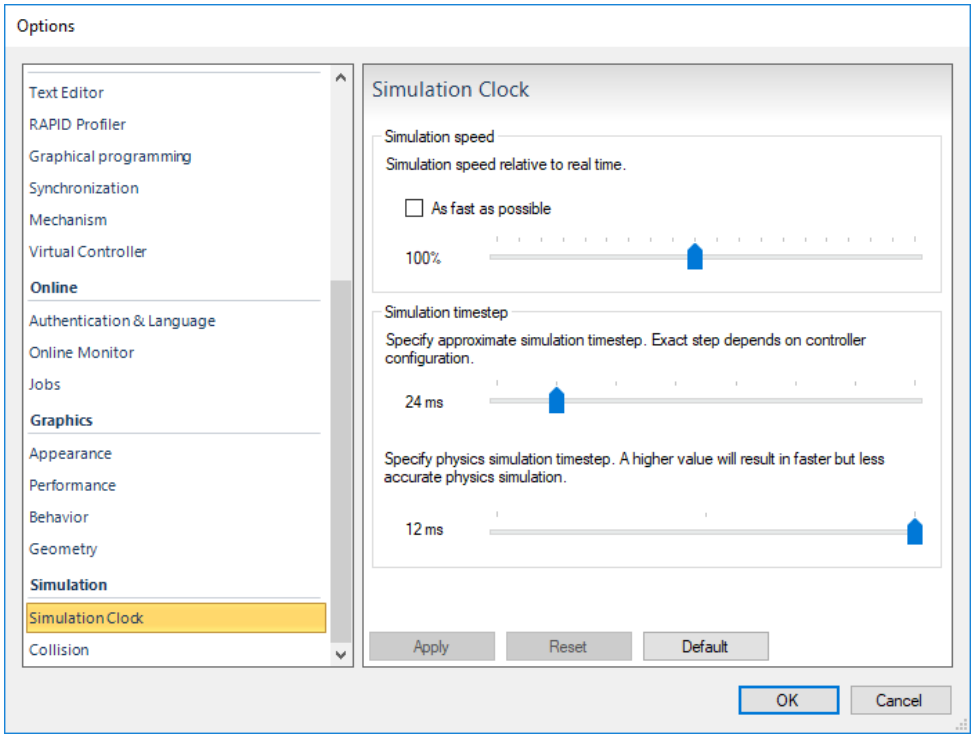


Figure 108: Configuración de Parámetros Temporales en Simulación

RobotStudio permite **configurar**, de forma limitada y preconcebida, los **parámetros de Velocidad de Simulación y el Tamaño del Paso de Simulación (*Step*)**.

Step: Se considera el periodo de actualización del programa o instante en que el programa calcula y chequea la configuración de E/S de los distintos elementos que componen la instalación. A menor tamaño de *Step* mayor frecuencia de actualización de E/S y cálculo de gestiones propias del sistema, a costa de un mayor consumo de recursos del sistema, y una simulación "supuestamente" más finita.

ATENCIÓN: Se recomienda **calcular/estudiar el paso de tiempo ("*TimeStep*"), en función de la sincronización de elementos en la Estación**. Dado que salpica directamente al código RAPID implementado para las simulaciones, siendo necesario forzar sincronismo: "*WaitTime [seg]*".

B) *Configuración del Modo de Tiempo de Simulación*

La configuración de los modos temporales desde las Pestaña de Simulation: *Pestaña_Simulation:Configure:Simulation_Setup: Virtual_time_mode:Time_Slice || Free Run*

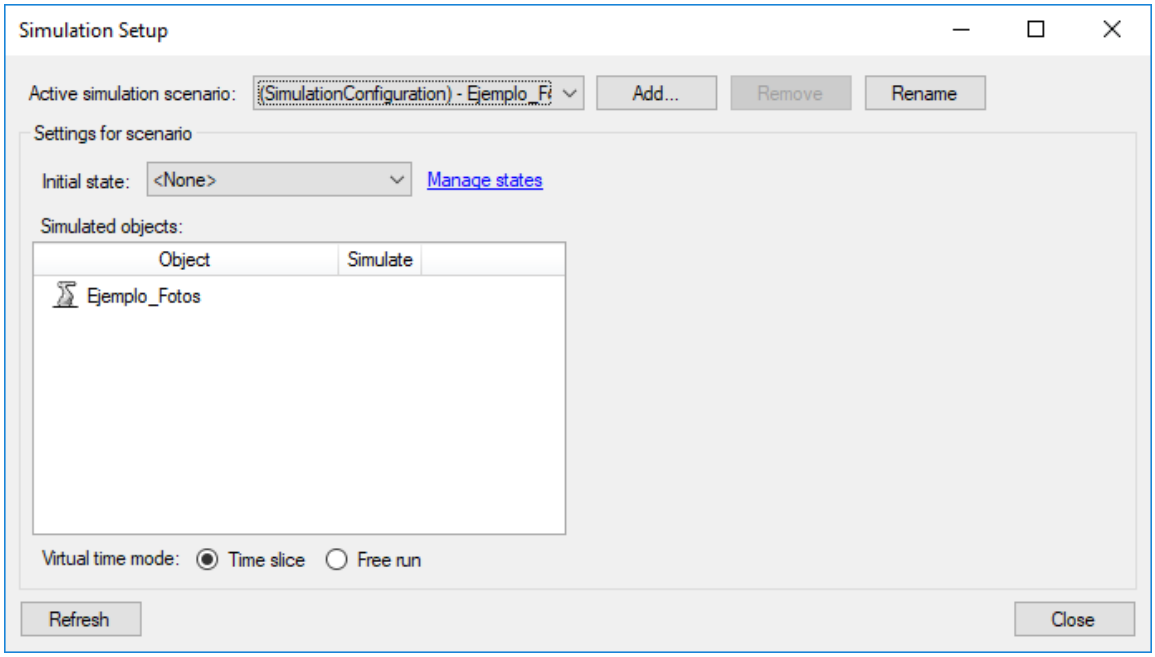


Figure 109: Configuración del Modo de Gestión de Tiempo en Simulación

Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation

La **función Cronómetro de la Pestaña Simulación** se utiliza para **medir el tiempo transcurrido entre dos puntos de disparo de una simulación/proceso,** además de la simulación de proceso en su conjunto.

Los tipos de disparadores, **disparador de inicio y disparador de fin,** que se especifican son:

- Inicio de simulación
- Paro de simulación
- Objetivo cambiado: Especificar la unidad mecánica y el objetivo.
- Valor de E/S: Especificar la unidad mecánica de origen de la que proviene la señal, el tipo de señal de E/S y el valor de la señal.

Se pueden configurar varios cronómetros para una misma simulación, así como denominar/asignar un nombre para cada cronómetro.

Se pueden controlar los cronómetros usando: SmartObjec, EventManager o directamente desde RAPID, apoyándose en el conexionado lógico de la LogicStation.

La configuración de los *Clocks*, para mediciones en Simulaciones: *Pestaña_Simulation:Monitor:Stopwatch: *to_define_parametre*

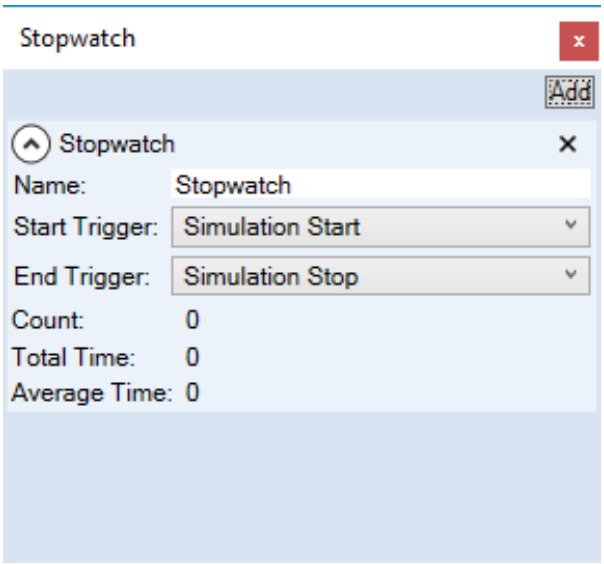


Figure 110: Control de Tiempos Específicos en Simulación: Función Cronómetro de Variables de la LogicStation

Procedimiento recomendado para poder medir/estimar los tiempos de ejecución de los propios *SmartObjects* y de los elementos que componen la Estación.

ANEXO IV

4. Limitaciones de los Sensores:

4. 1. Tipo de Limitaciones de los Sensores

A la vista de los resultados obtenidos en las simulaciones, se han encontrado algunas limitaciones que afectan a la elección y uso de los sensores en:

4. 1. 1. Tiempo de Simulación

Definir el tamaño del "*Step*", para gestionar/actualizar las E/S. Se cree que **RobotStudio** ha sido diseñado con:

- **Gestión de tiempos no continua (*discreta*)**
- **Simulación continua (*asíncrona*)**.

RobotStudio sólo es capaz de identificar la existencia de Colisión, entre el objeto a identificar y el plano que contiene al Sensor, en el instante de actualización del *Step*.

La actualización de E/S, de los elementos que componen la Estación, es inversamente proporcional al tamaño del *Step* de Simulación: *A menor Step mayor frecuencia de actualización de E/S*. O dicho de otra forma, a menor *Step* menor anchura de pulso entre actualizaciones, lo que se traduce en que siempre que la duración de la señal de colisión se encuentre activada un tiempo superior a dicho pulso será detectada por los elementos de la Estación.

4. 1. 2. Objetos a identificar

Los objetos representados en la Estación, NO SON SÓLIDOS propiamente dicho, **se encuentran definidos como objetos de tipología CAD o Geometría Matemática**. *Todo sólido representado en la pantalla de edición se encuentra compuesto exclusivamente de las caras o superficies geométricas que se visualizan.*

4. 1. 3. Posición y Orientación del Plano del Sensor

La **posición y orientación de los Planos que contienen al Sensor** es especialmente **relevante para la identificación de objetos**. Como se ha mencionado anteriormente, RobotStudio sólo es capaz de identificar la existencia de Colisión, entre el objeto a identificar y el plano que contiene al Sensor, en el instante de actualización del *Step*.

Por lo que habrá que tener en cuenta, y **evitar, que el Plano de Referencia del Sensor**, plano sobre el que se compara, **no sea**:

- **Paralelo a las superficies de sujeción**
- **Tangencial al vector de movimiento del objeto a identificar**.

*Se busca la **intersección progresiva**, para dar tiempo a actualizar los registros de E/S*.

Adicionalmente **se aconseja**:

- Las **dimensiones del Plano de Sensor sean superiores al tamaño del Objeto a identificar**.
- **Combinar planos tangenciales entre sí**, en aquellos supuestos en que la variable principal a respetar sea la velocidad de aproximación de los objetos: ***evitará el problema de detección de piezas más grandes que el plano a identificar***.

*El objetivo es **evitar que el Plano del Sensor pueda acabar contenido en el interior del Objeto**, y no detectar la existencia del Objeto que lo envuelve.*

4. 1. 4. Restricción de Velocidad

Como se ha mencionado anteriormente, RobotStudio sólo es capaz de identificar la existencia de Colisión, entre el objeto a identificar y el plano que contiene al Sensor, en el instante de actualización del *Step*. Esto se traduce en que **la velocidad de los Objetos**, ya sea el **objeto a identificar o** el **objeto que contenga el Plano del Sensor**, **será inversamente proporcional a la duración de la señal de detección**. *Es decir, a mayor velocidad relativa entre los cuerpos menor duración del pulso de la señal de detección. Si este pulso es inferior al Step, RobotStudio NO detectará la existencia de Colisión*.

La **Velocidad Relativa de los Objetos que *Part*icipan en la Detección e Identificación de Objetos** queda **condicionada por**:

- **Tiempo de Actualización de registros de E/S**, en función del TimeStep
- **Tipo de sensor escogido y posición** del plano del sensor

*Se busca dotar de **duración suficiente en la intersección de los Objetos**, para dar tiempo a actualizar los registros de E/S*.

4. 1. 5. Ubicación exacta del Objeto en el Espacio/Estación

Ninguno de los sensores facilitados por RobotStudio **garantiza el contenido informativo de la ubicación exacta del objeto respecto de la referencia mundo de la Estación, sólo informa la existencia de una intersección en algún punto del plano/s** que representan el sensor utilizado.

4. 2. Consecuencia de las Limitaciones

4. 2. 1. Intersección de Planos Paralelos

Limitación por Tamaño de Sensores

Como se ve en apartados anteriores, RobotStudio sólo es capaz de identificar la existencia de Colisión, entre el objeto a identificar y el plano que contiene al Sensor, en el instante de actualización del *Step*. Esto se traduce en que **la velocidad de los Objetos**, ya sea el **objeto a identificar o** el **objeto que contenga el Plano del Sensor**, **será inversamente proporcional a la duración de la señal de detección**. Y por ende la distancia de penetración del Plano del Sensor y parte de los dedos sobre el volumen del objeto a detectar.

Dicho esto, las siguientes imágenes muestran como el Plano del Sensor y parte de los dedos se introduce en el volumen del objeto, condición necesaria para detectar la intersección de Planos.

En el siguiente vídeo se pueden ver las **limitaciones un Sensor Paralelo ante la detección de Objetos de distinta área, de mayor y menor tamaño que el área del Sensor**: "SensorLimits_Parallel_Plane"

a) Sensor Suficientemente Grande

Si el Plano del Sensor ocupa un área mayor que la del Objeto a detectar, la **penetración se encontrará condicionada únicamente ante el tamaño del *Step***. Como se puede apreciar en las imágenes de la **Figure 111** .

Principales características serían que:

- El **Controlador de la Herramienta**, hecha con SmartObject, **detecta** la existencia de **colisión**
- **No es necesario un sensor auxiliar**: al existir y detectarse plano de corte entre el sensor principal y la pieza contenida

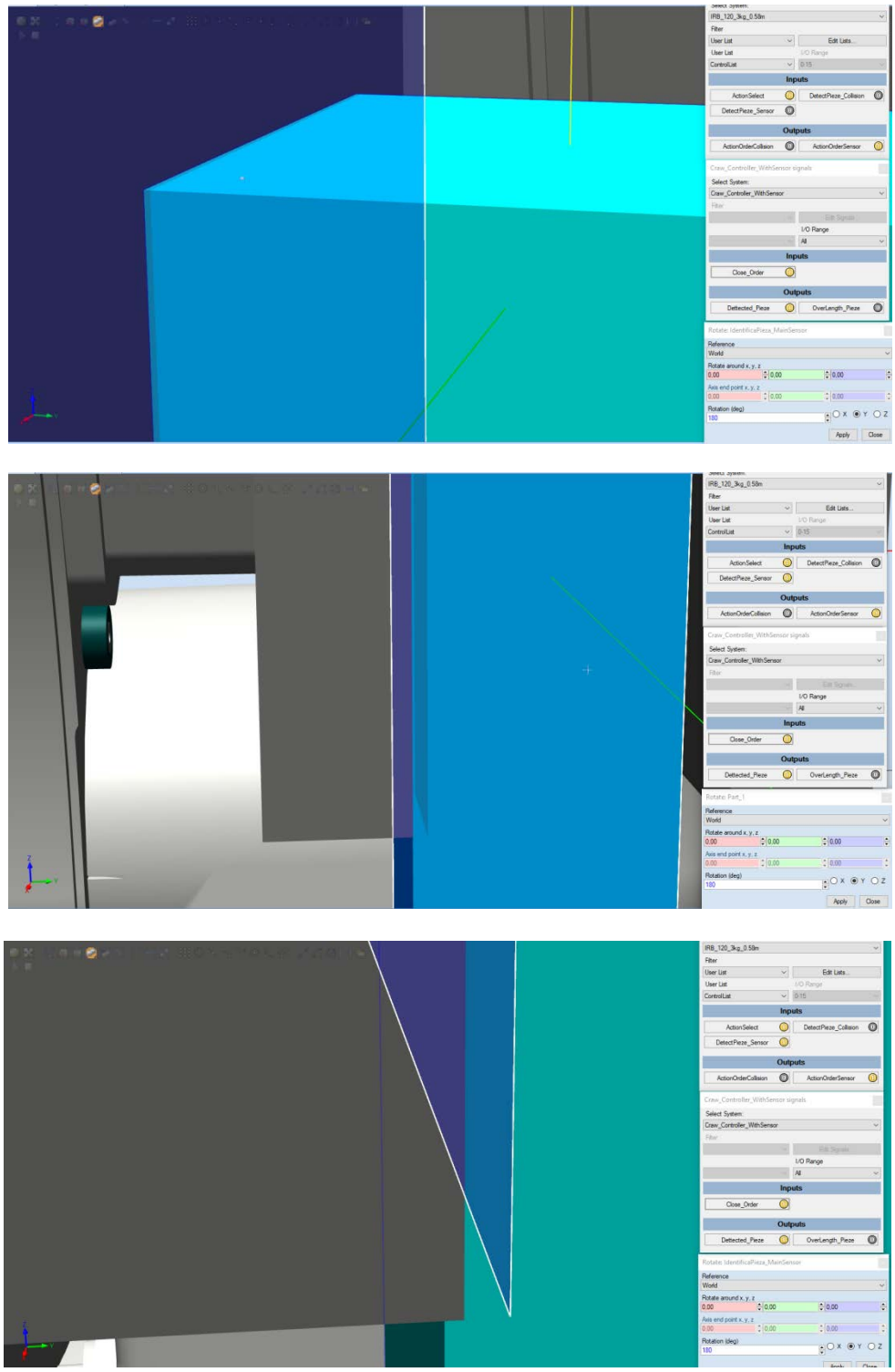


Figure 111: Limitaciones de los Sensores: Sensor Suficientemente Grande

b) Sensor Insuficientemente Grande

Si el Plano del Sensor es menor al área del Objeto a detectar y se encuentra completamente contenido por esta, la **penetración se encontrará condicionada a la existencia de un segundo sensor y al tamaño del *Step***. Como se puede apreciar en las imágenes de la **Figure 112**.

Principales características serían que:

- El **Controlador de la Herramienta**, hecha con SmartObject, **no detectará** la existencia de **colisión**, salvo que se cuente con apoyo de un Sensor auxiliar con inclinación distinta al Principal, preferiblemente Tangencial a este.
- **Absolutamente necesario el sensor auxiliar**. El sensor principal está completamente contenido en el volumen del Objeto, ubicado en el interior del objeto (*objetos definidos como conjunto de planos NO POSEE VOLUMEN SÓLIDO*), lo que provoca que el Sensor Principal no sea capaz de detectar la existencia de Colisión.

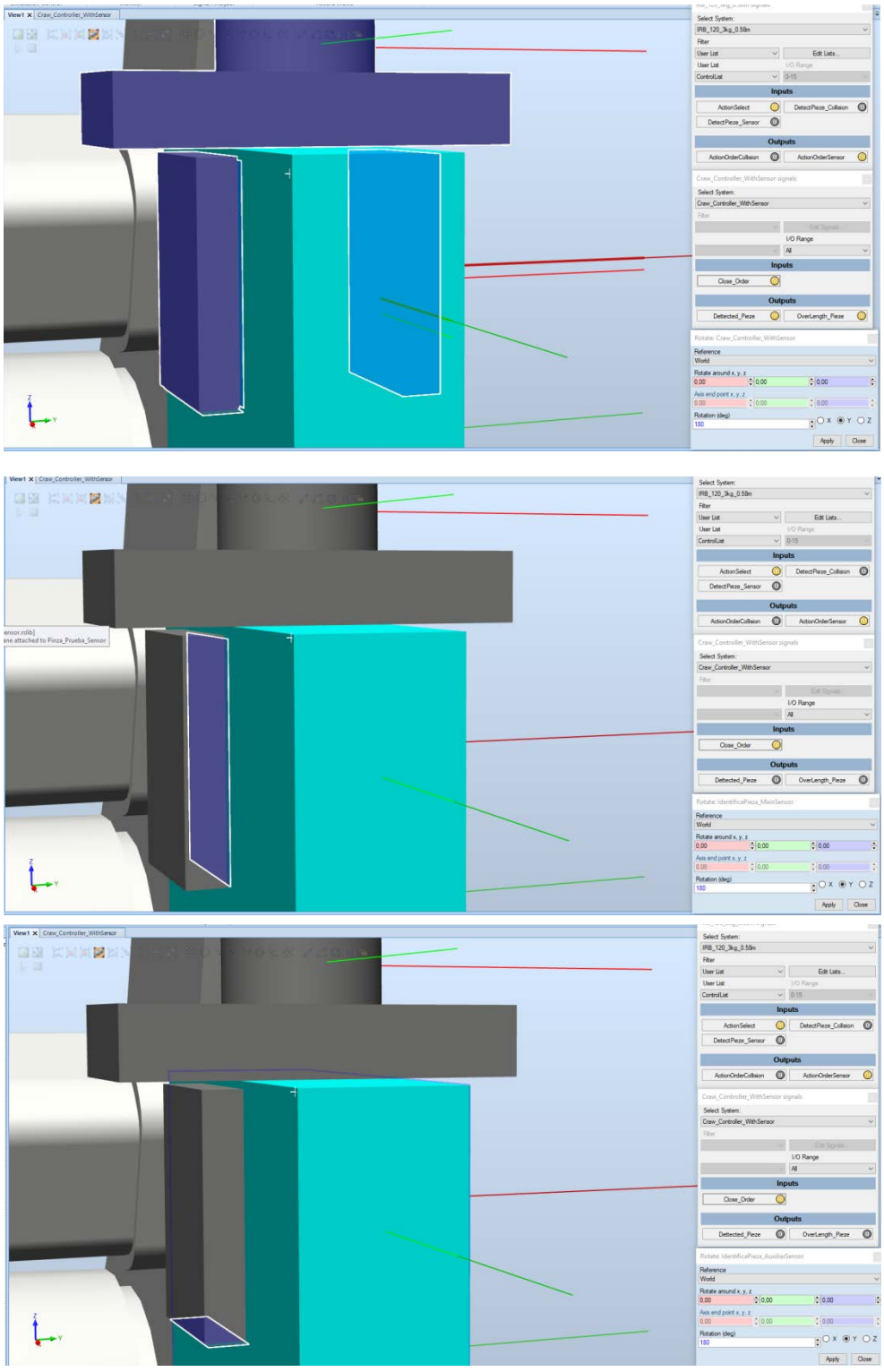


Figure 112: Limitaciones de los Sensores: Sensor Insuficientemente Grande

Captura del Problema en Simulación: Paralelismo del Plano del Sensor, y de menor tamaño, con el Objeto a Identificar.

Supuesto en que el **Plano de Sensor es paralelo**, o cuasi-paralelo, **al Plano de la cara del Objeto** a detectar **y la superficie del Plano del Sensor es de menor tamaño que el Plano a Detectar**. Es decir, que exista riesgo de que el Plano del Sensor **quede completamente contenido en el Plano de la Cara del Objeto**.

a) Consecuencia

Si el momento de intersección se da fuera del instante del "*Step*" (*que por lógica se da en la práctica totalidad de las simulaciones, el 99,99% de las ocasiones*), es decir, fuera del periodo de muestreo de las E/S, el resultado será la pérdida del flanco de activación del sensor. Ergo no se detectará la existencia de colisión, aunque el sensor se encuentre completamente contenido en el objeto a detectar.

Es decir, **la representación física del Sensor se puede llegar a encontrar en el interior del volumen del objeto que se desea detectar**.

b) Vídeos de Demostración

En el siguiente vídeo se pueden ver las **limitaciones de un Sensor Paralelo y de menor área que el Plano de la Cara del Objeto a Detectar**: "**SensorLimits_Parallel_Plane**"

Y la **solución propuesta**, un segundo Sensor Perpendicular al Sensor Principal que entra en funcionamiento sólo durante el cierre de la Pinza: "**SensorLimits_Parallel_Solution**"

Otras soluciones contempladas pasarían por **modificar la posición** (*superficie cubierta por el Sensor*) desplazando la ubicación del sensor hasta estar fuera de la perpendicularidad en el instante de detección del objeto. Aunque el problema del tamaño del sensor respecto al objeto, sólo se solucionaría dotando al Plano del Sensor de una inclinación respecto de su posición actual, la consecuencia sería que se alejaría del comportamiento de los objetos en la realidad (*no se vería en las simulaciones el efecto de agarre de objetos, quedando los dedos de la garra separados de los objetos a agarrar*).

4. 2. 2. Selección de un Detector de Colisiones "CollisionSensor" para Identificar Objetos

A) *Propiedades del CollisionSenser*

Existe una alternativa a los Sensores, pasa por el uso de un *CollisionSenser* que permite detectar colisiones, y eventos de casi colisión, **entre un objeto A y otro B** (*si no se especifica uno de los objetos, el otro se comprueba frente a toda la Estación*) e informando al sistema por medio de una interrupción de programa. Lo que provoca que existan algunas diferencias de funcionamiento, aunque **ninguno de estos sensores garantiza una ubicación exacta en el objeto, sólo informa de que existe una intersección en algún punto del plano/s** que representan el sensor utilizado.

La **comparación** la hace **por medio de la intersección entre el volumen de los objetos A y B**. Se posibilita la Activación/Desactivación del bloque en función de la Lógica implementada. **Representa un menor tiempo de cómputo**, lo que es una ventaja, que las operaciones propias de los sensores convencionales pero a costa de ser mucho más pesado el fichero final.

El sensor detecta la colisión de un objeto/pieza "*Part*" como conjunto en sí mismo, es decir detectará las colisiones entre los propios cuerpos "*Body*" que la componen en aquellos casos en que se estén tocando. Esto **genera la necesidad de crear la herramienta "*Tool*" con piezas suspendidas en el espacio y asegurar que no se encuentren superpuestas entre sí** para ninguna de las configuraciones o trayectorias posibles, ni siquiera apoyadas.

- **Este sensor INFORMA de qué pieza está en colisión** con la pieza comprobada.
- Es **posible comparar entre objetos específicos**, seleccionados de una lista, **u objetos relativos a una posición** en la que se encuentre un sensor identificando a estos (*a través de la conexión lógica-analógica, cable "morado"*).
- **No posee representación gráfica en la simulación**. No se encuentran asociados a una posición espacial.

Si se desea más información del funcionamiento de un detector de colisiones o de más consideraciones a tener en cuenta para su implementación: Ver "**Consideraciones de los Detectores de Colisiones**".

B) *Limitación del CollisionSenser*

Es **insuficiente** con **la inclusión de precisión "fine" en la orden previa de movimiento** del robot: **Ver vídeo demostración:** **Acumulative_Error_CollisionSenser.**

Para aquellos programas en los que no se recolocan los objetos solos, o no se ejecutan los comandos de creación y destrucción, es **necesario tomar consideraciones adicionales en código RAPID** como dotar al fichero RAPID de "WaitTime xxxx" para evitar problemas en la acción de soltar piezas**. Ver vídeo**: **Solution_Acumulative_Error**

4. 2. 3. Necesidad de Forzar un "Refresco" o Actualización del Estado de los Sensores

Situación Inicial de los Bloques Sensibles a Activación (*Sensores*)

Si se ha diseñado un SmartObject condicionado a la activación de una orden (*ejemplo: uso de sensores para detectar objetos*), el comportamiento que se ha encontrado en RobotStudio es el de **mantener como valores iniciales de la Simulación la Configuración de Salidas registrada en el último estado en el que se encontró**, es decir en la última activación simulada.

Dicho de otra forma, la **configuración de los valores de salida de cada bloque** posee **memoria entre simulaciones**. Son **consideradas variables persistentes**, y se mantendrá en dicha configuración **hasta que no exista una variación de las E/S del SmartObject**. Esto **se traduce en que no calculará la configuración del Estado del Instante en el que se encuentra inicialmente al arrancar una nueva Simulación.**

A) *Actualización de Estado de los Bloques Sensibles a Activación (Sensores)*

A lo anteriormente comentado hay que añadir que, una vez cumplida alguna condición de variación de Entradas, **sólo calculará el nuevo estado de configuración interna para las variables reasignadas**, para el resto de variables no actualizará la configuración de Salidas. ***Todo este "detalle" no tendría relevancia si el programa identificase la gestión de E/S por nivel, en vez de hacerlo por variación de nivel.*** Lo que provoca tener que aplicar consideraciones en la elaboración del programa que difieren del comportamiento real de una instalación.

B) *Tipos de Actualización de Estado de los Bloques*

a) Actualización Manual

Si se opta por dejar la configuración inicial a expensas del usuario que simule, es decir **activar/refrescar/configurar CADA BLOQUE SENSIBLE manualmente**, o lo que es lo mismo dejar el **bit Activo en 1 por defecto**, puede suponer un verdadero problema.

Problema especialmente costoso en tiempo, a la hora de poder pre-condicionar el Estado Inicial para todos los bloques que componen la Estación, en función de la cantidad de bloques utilizados y de la complejidad del SmartObject.

Y sí, habrá que hacer "esto" antes de CADA UNA DE LAS SIMULACIONES LANZADAS.

b) Actualización Automática o Programada

Refrescar por código desde RAPID o implementando una variable de entrada manual, que se deberá manipular una única vez y antes de lanzar las simulaciones. Aviso que requiere de **lógica cableada propia**, y elevará notablemente la complejidad de la lógica cableada final del SmartObject.

Tanto la opción de RAPID como la activación/desactivación manual pasan por implementar una variable **externa, que haga la función de interruptor on/off,** conectada directamente a la casilla "Active" de cada bloque sensible.

Consecuencia: *Aparición de verdaderos CROMENBERGS en la simulación*

4. 2. 4. Consecuencias de la Velocidad de Movimiento junto con la Frecuencia de Actualización de E/S (*Step*)

Durante la simulación, **en cada "Step", RobotStudio calcula, predice y simula la velocidad de los objetos y la posición alcanzada para el siguiente "*Step*"**.

A menor Step mayor frecuencia de actualización de E/S. O dicho de otra forma, a menor *Step* menor anchura de pulso entre actualizaciones, lo que se traduce en que siempre que la duración de la señal de colisión se encuentre activada un tiempo superior a dicho pulso será detectada por los elementos de la Estación.

Aunque no se ha podido corroborar cómo lo resuelve, se ha constatado en las simulaciones que **a la hora de intentar detectar un objeto,** el resultado dependerá de si el instante de actualización del "*Step*" coincide con el momento temporal en el que se da la intersección entre alguno de los planos de objeto con el plano/s del sensor.

Dicho instante **se encuentra íntimamente ligado a las velocidades de los objetos, sensor y objeto a detectar**, que participan en la identificación de colisión. **A mayor velocidad relativa entre los cuerpos, menor tiempo de intersección entre planos, y viceversa**.

Esto se traduce en que **la velocidad de los Objetos**, ya sea el **objeto a identificar o** el **objeto que contenga el Plano del Sensor**, **será inversamente proporcional a la duración de la señal de detección**. *Es decir, a mayor velocidad relativa entre los cuerpos menor duración del pulso de la señal de detección. Si este pulso es inferior al Step, RobotStudio NO detectará la existencia de Colisión*.

Este fenómeno puede forzar la **necesidad de configurar "*Delays*" en la lógica cableada** de detección de objetos, y/o gestor de movimiento de objetos, para **garantizar el sincronismo de E/S entre los SmartObject y el resto de la Estación**, por ejemplo un controlador IRC5 Virtual encargado de comandar un robot. *Se busca dotar de **duración suficiente en la intersección de los Objetos**, para dar tiempo a actualizar los registros de E/S*.

Todo ello se traduce en que, **será necesario estudiar la velocidad máxima y mínima de los objetos móviles**, excluidos los robots, para un correcto funcionamiento de la Estación. El estudio no es extrapolable entre Estaciones, siendo soluciones individuales para cada una de las Estaciones que se estudien.

Consecuencias: Velocidad de Aproximación de los Objetos

Considerando *TimeStep* máximo admisible 24ms, para un mayor valor del parámetro no es capaz de simular las velocidades requeridas en el análisis de este punto, como se ve en la siguiente imagen:

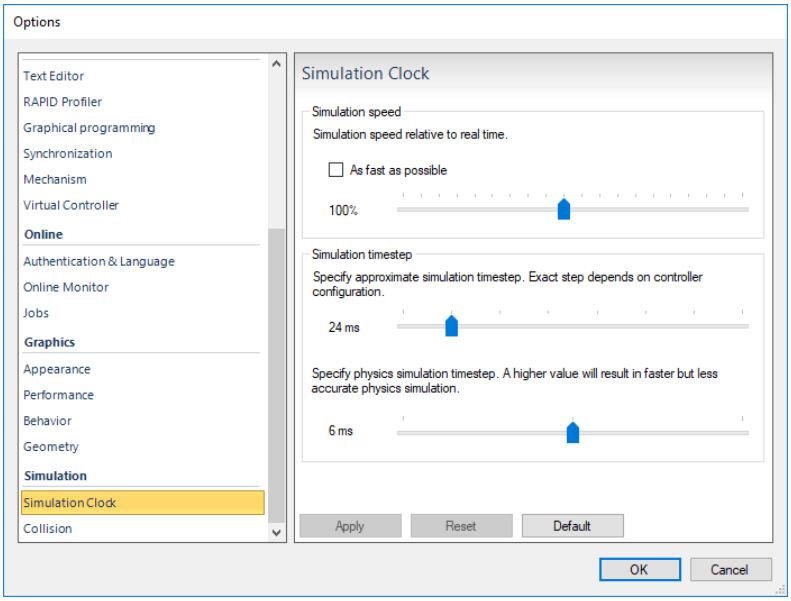


Figure 113: Step dado para comparativa de Velocidad de Aproximación entre Objetos

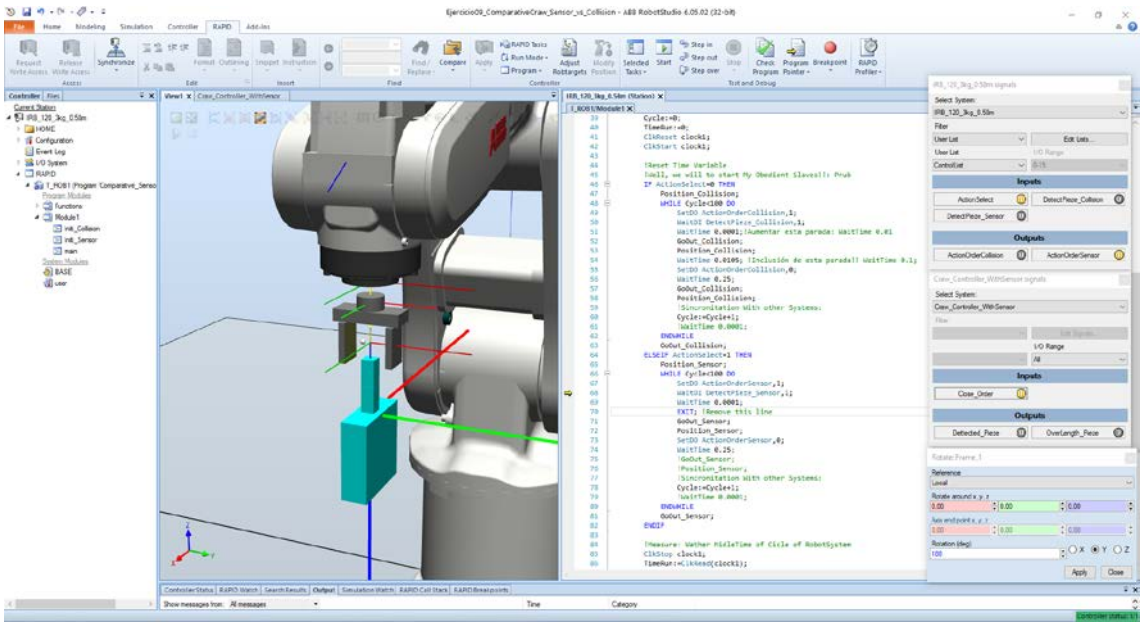


Figure 114: Estación de Prueba: Limitaciones de la Velocidad de Aproximación

Se puede apreciar dist**inta penetración en función de la Velocidad de Aproximación seleccionada y del tipo de Detector utilizado**, Sensores o *CollisionSenser*. **Resultando el más finito y con menor profundidad en la penetración el *CollisonSenser***.

A) *Resultados de la Implementación con Sensores*

a) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (18.24mm)

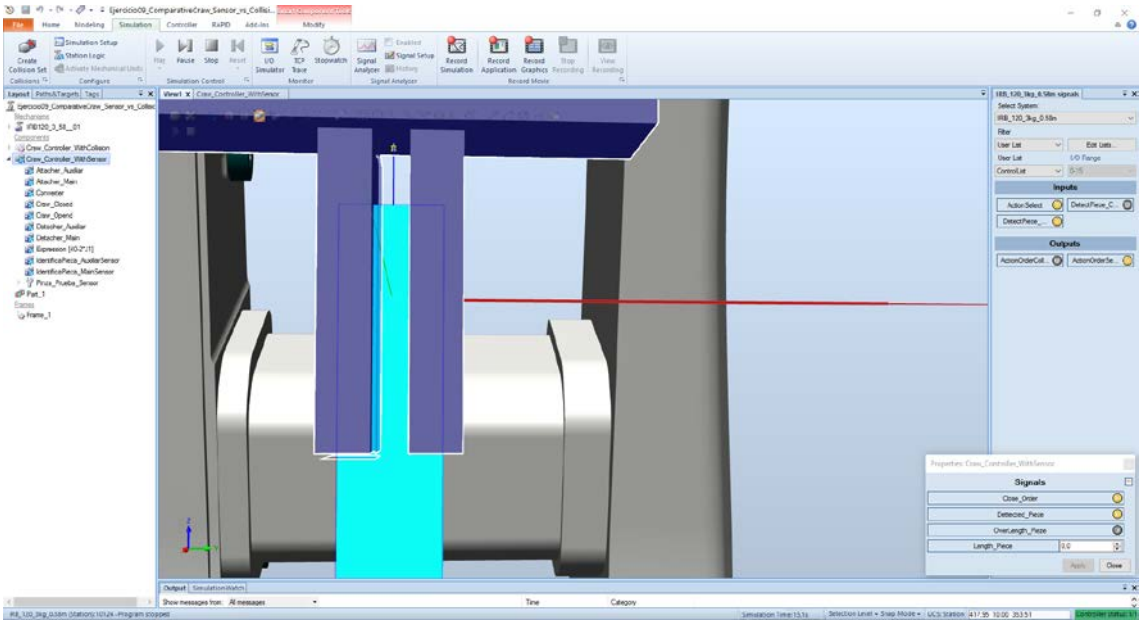


Figure 115: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18.24mm*)

b) Velocidad de Aproximación: 0.15 seg en recorrer 20 mm: Length final (15.20mm)

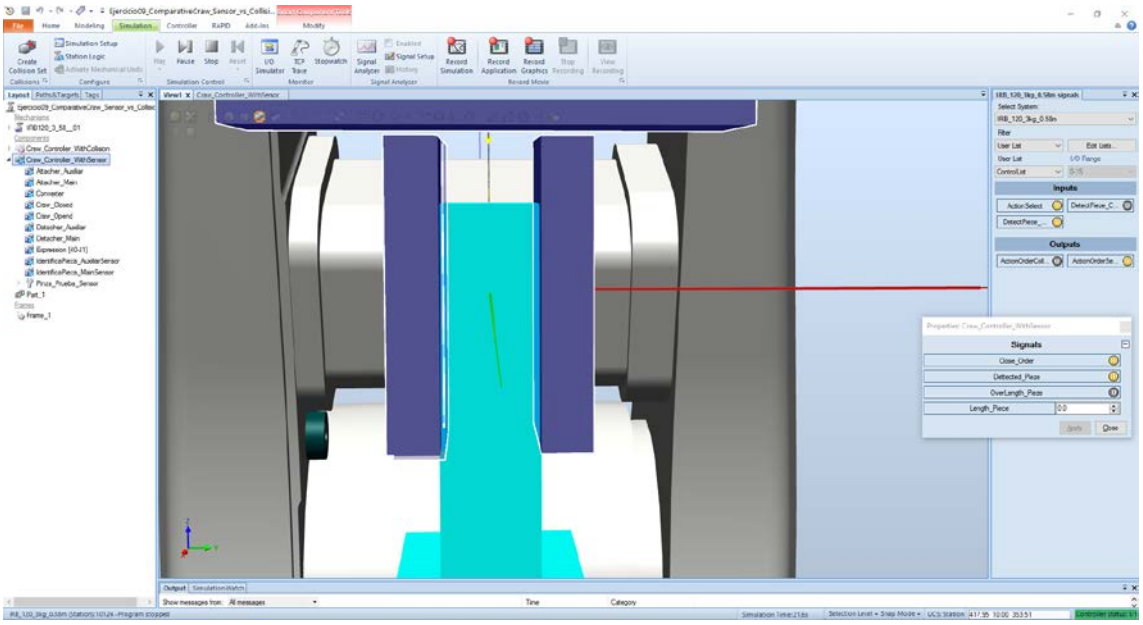


Figure 116: Velocidad de Aproximación: 0.15 seg/20 mm: Length final (*15.20mm*)

c) Velocidad de Aproximación: 0.25 seg en recorrer 20 mm: Length final (14,59mm)

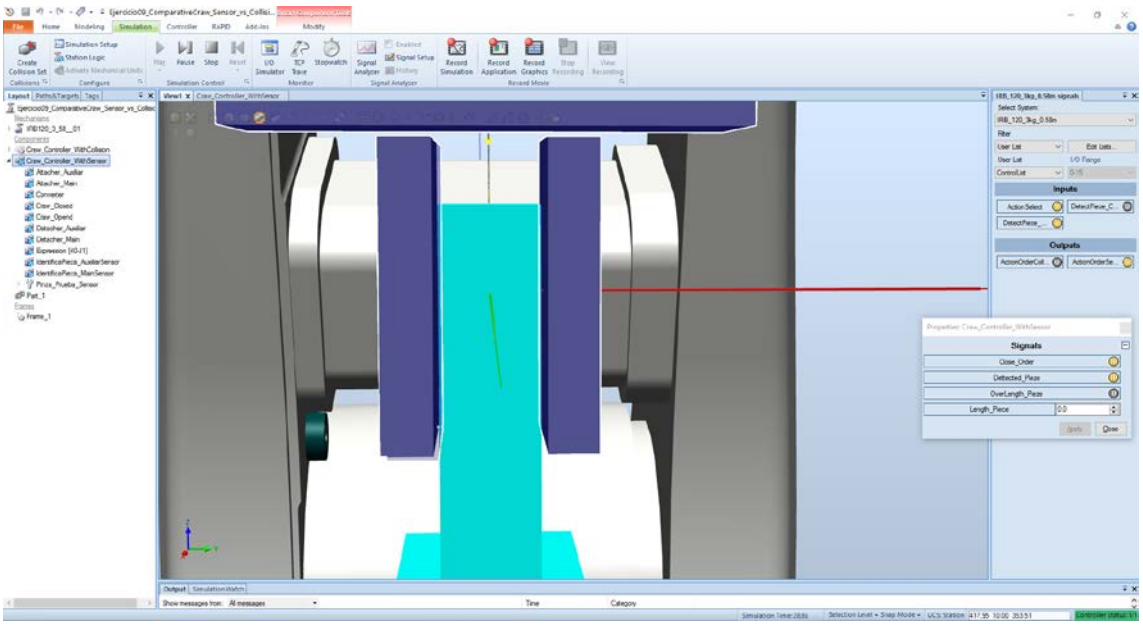


Figure 117: Velocidad de Aproximación: 0.25 seg/20 mm: Length final (*14,59mm*)

d) Velocidad de Aproximación: 0.5 seg en recorrer 20 mm: Length final (14,68mm)

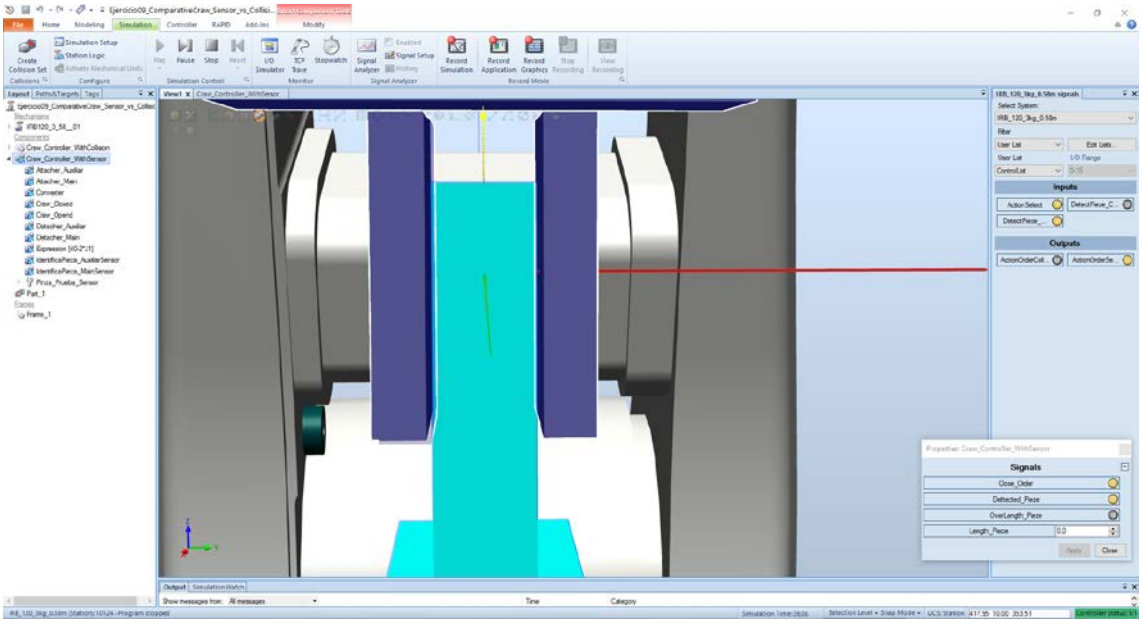


Figure 118: Velocidad de Aproximación: 0.5 seg/20 mm: Length final (*14,68mm*)

B) *Resultados de la Implementación con Sensores*

a) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (18.24mm)

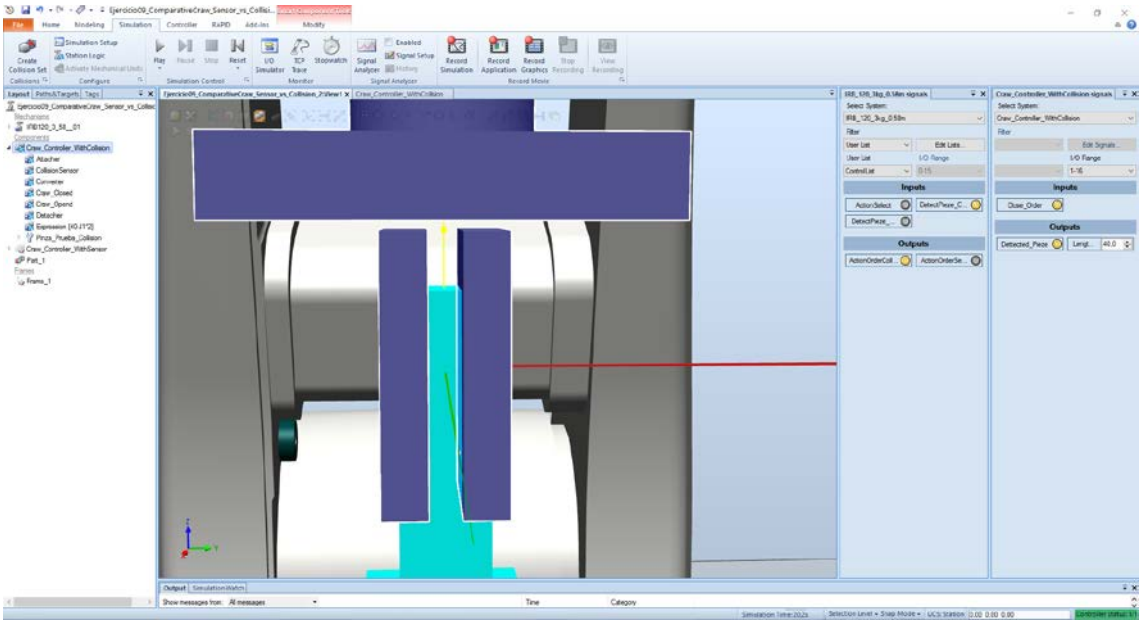


Figure 119: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18.24mm*)

b) Velocidad de Aproximación: 0.15 seg en recorrer 20 mm: Length final (15.20mm)

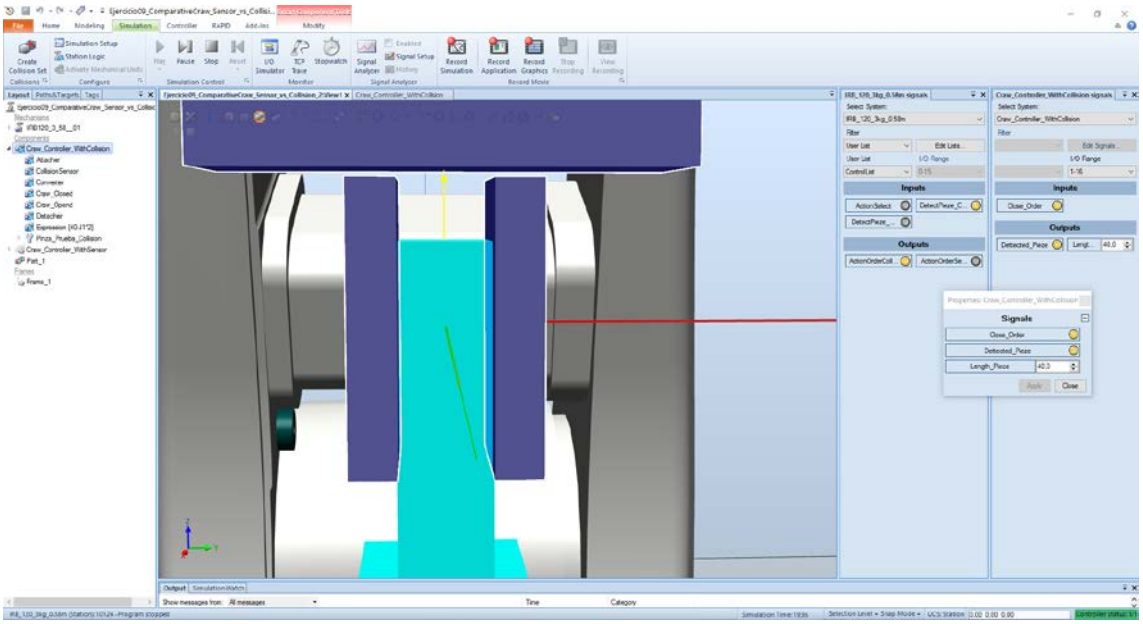


Figure 120: Velocidad de Aproximación: 0.15/20 mm: Length final (*15.20mm*)

c) Velocidad de Aproximación: 0.25 seg en recorrer 20 mm: Length final (16.42mm)

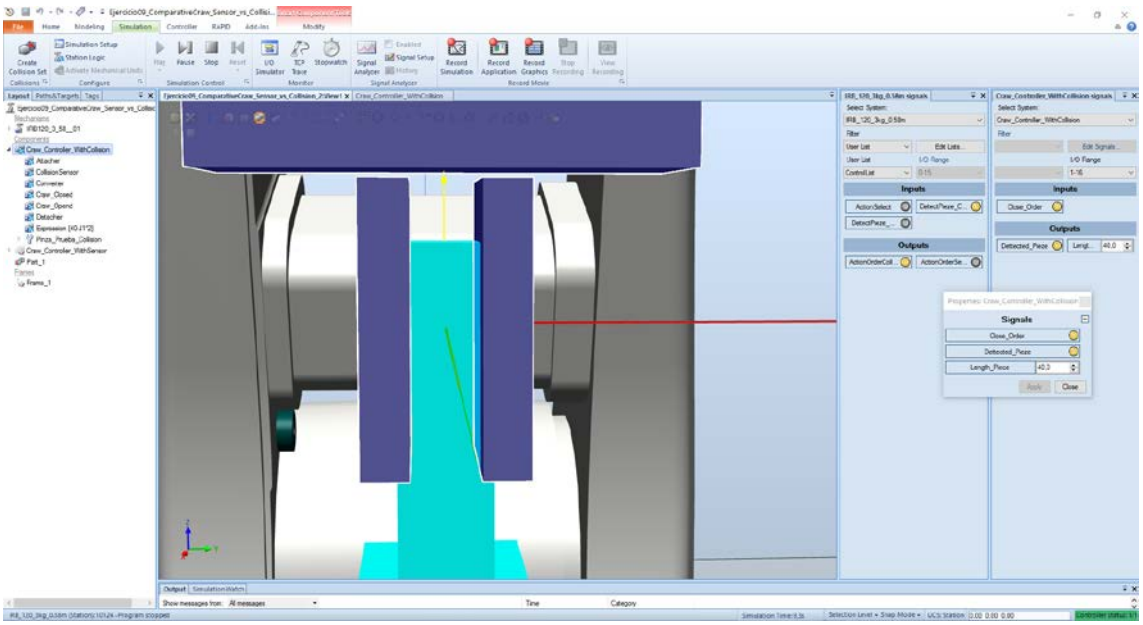


Figure 121: Velocidad de Aproximación: 0.25 seg/20 mm: Length final (*16.42mm*)

d) Velocidad de Aproximación: 0.5 seg en recorrer 20 mm: Length final (15.50mm)

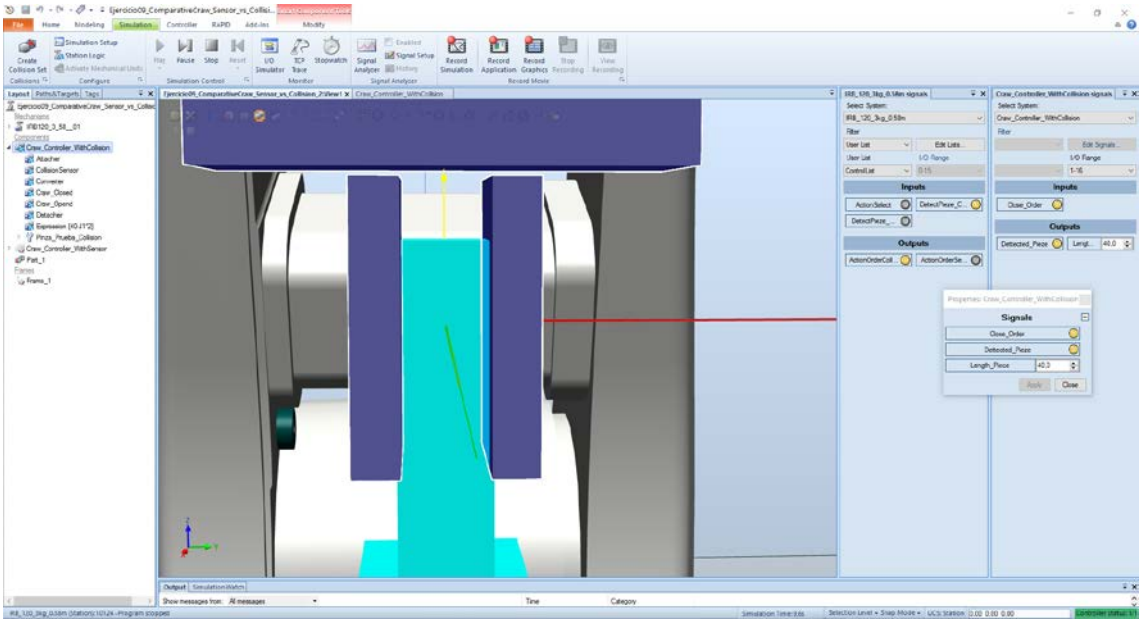


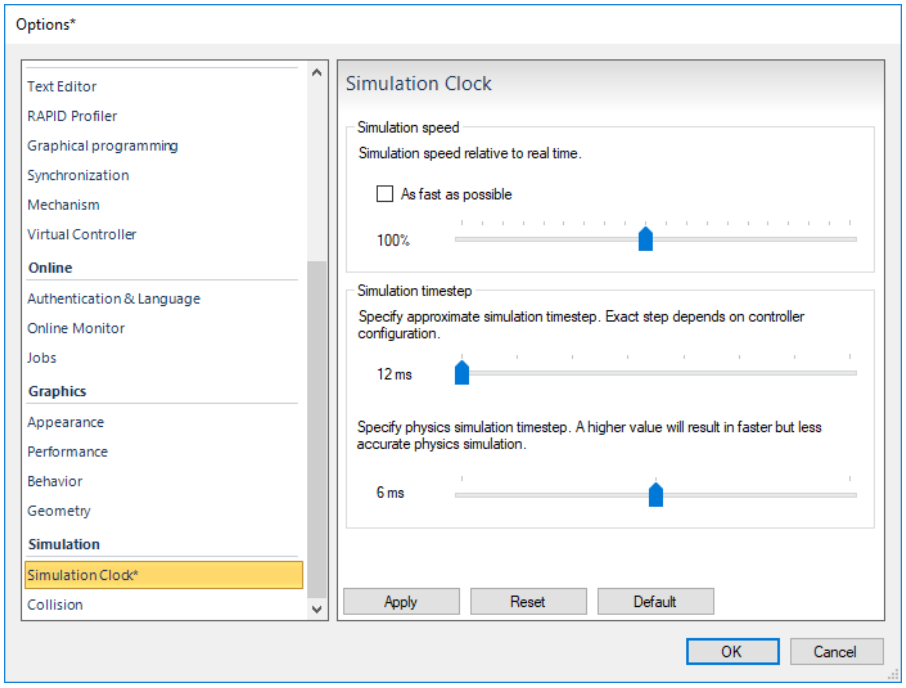
Figure 122: Velocidad de Aproximación: 0.5 seg/20 mm: Length final (*15.50mm*)

Consecuencias: Velocidad de Aproximación de los Objetos con Variación Step

Como se constata en las imágenes, solo influye el parámetro *TimeStep*, o también **denominado *Step***, la relevancia del *Physical Simulation TimeStep* no altera el contenido de la simulación, para programas de este nivel de complejidad.

A) *Resultados de la Implementación con Sensores*

a) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (15.96mm)



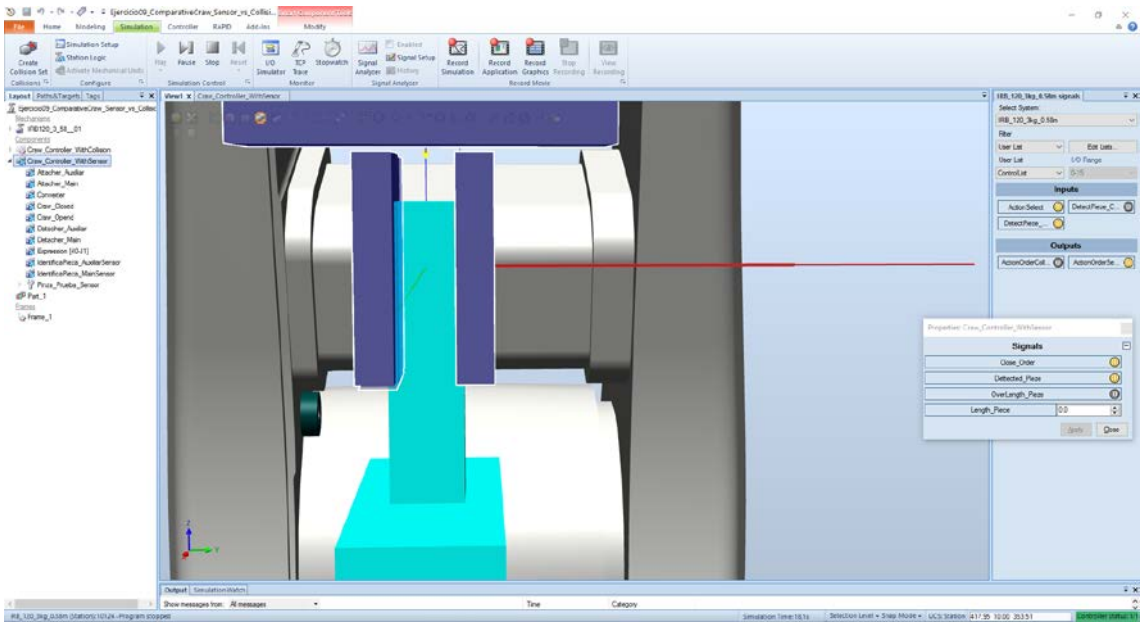
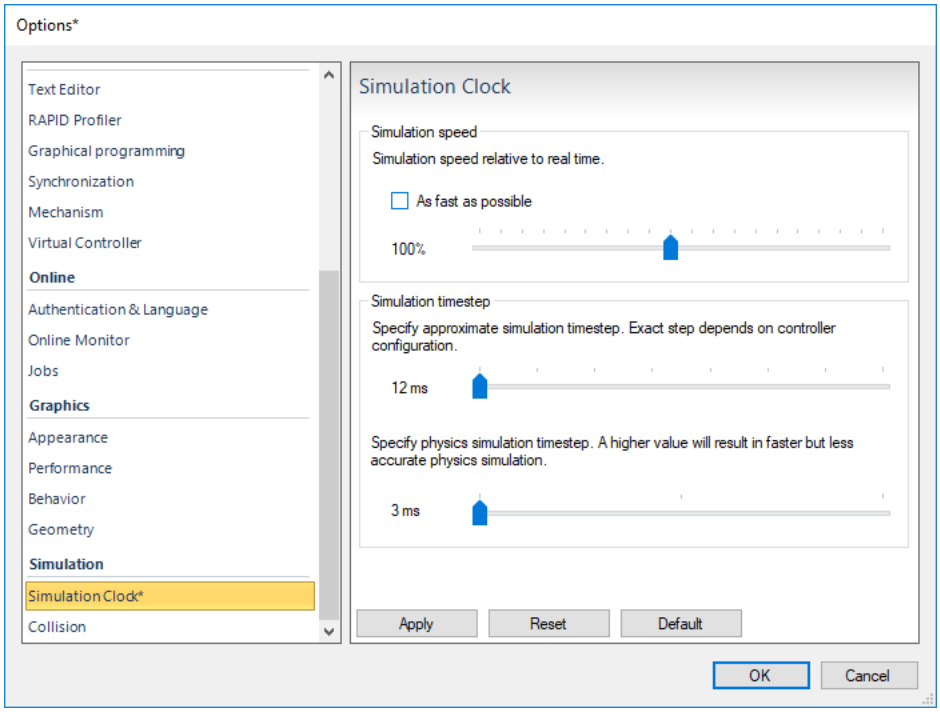


Figure 123: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*15.96mm*): TimStep 12ms (*6ms*)

b) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (15.96mm)



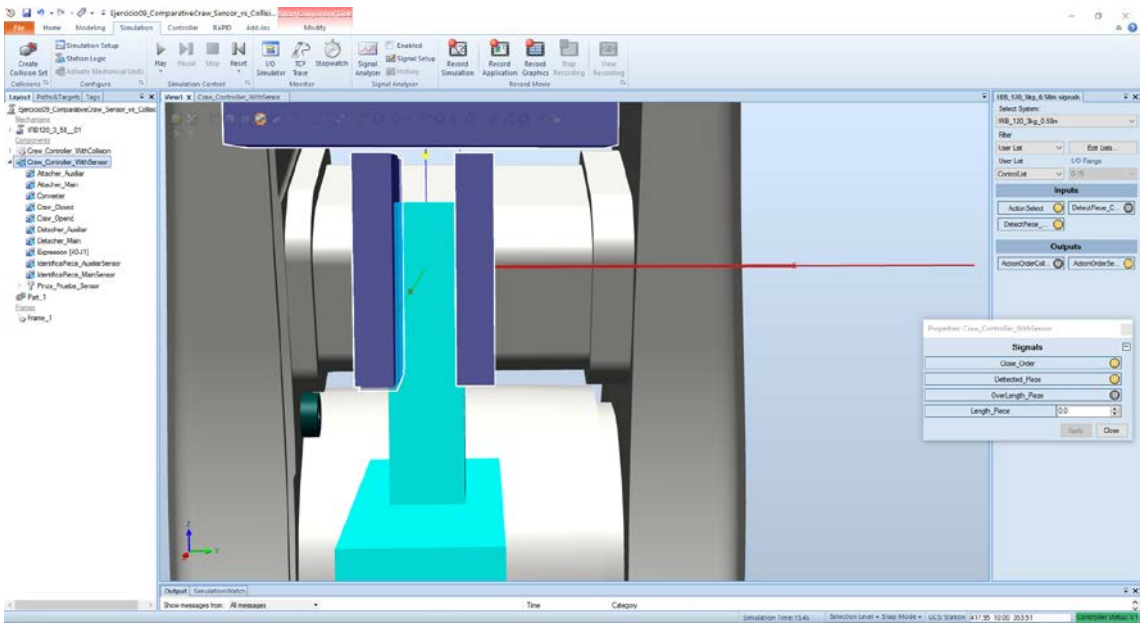
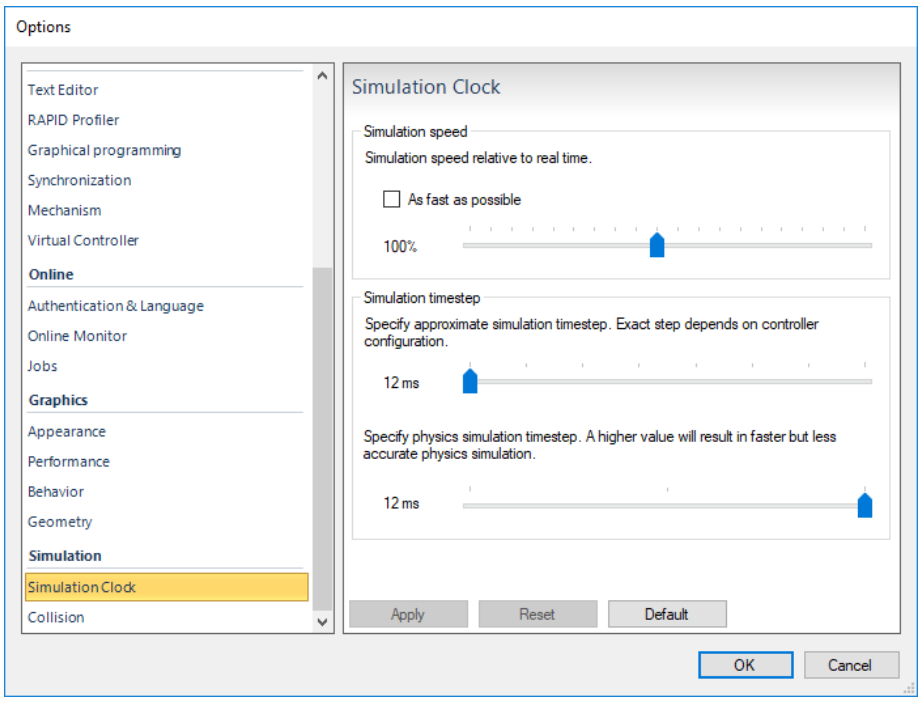


Figure 124: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*15.96mm*): TimStep 12ms (*3ms*)

c) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (15.96mm)



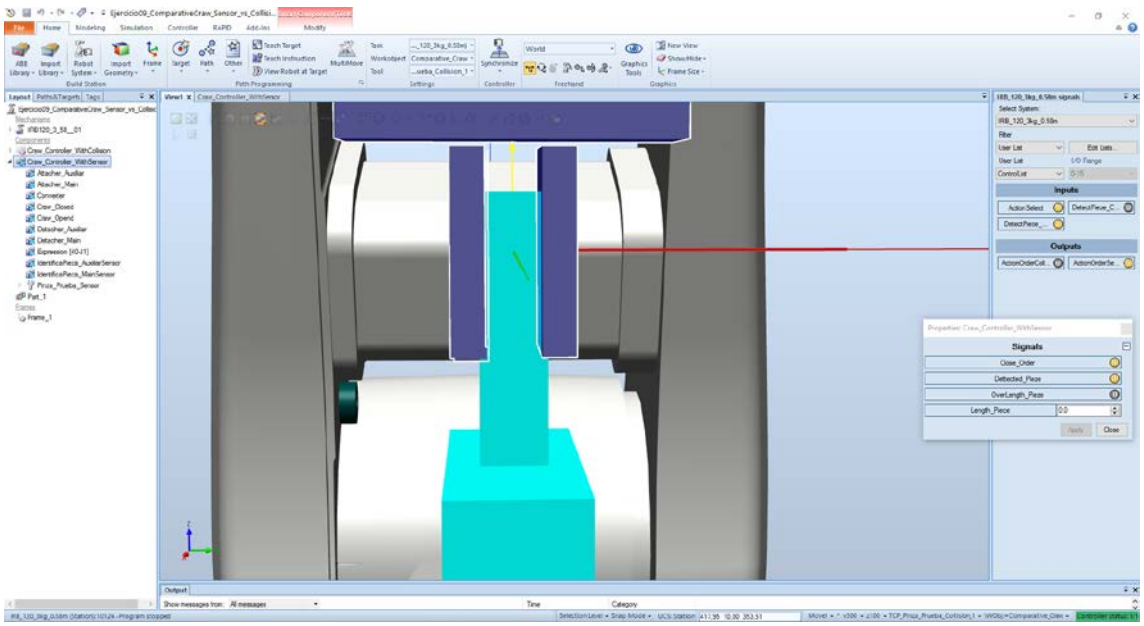
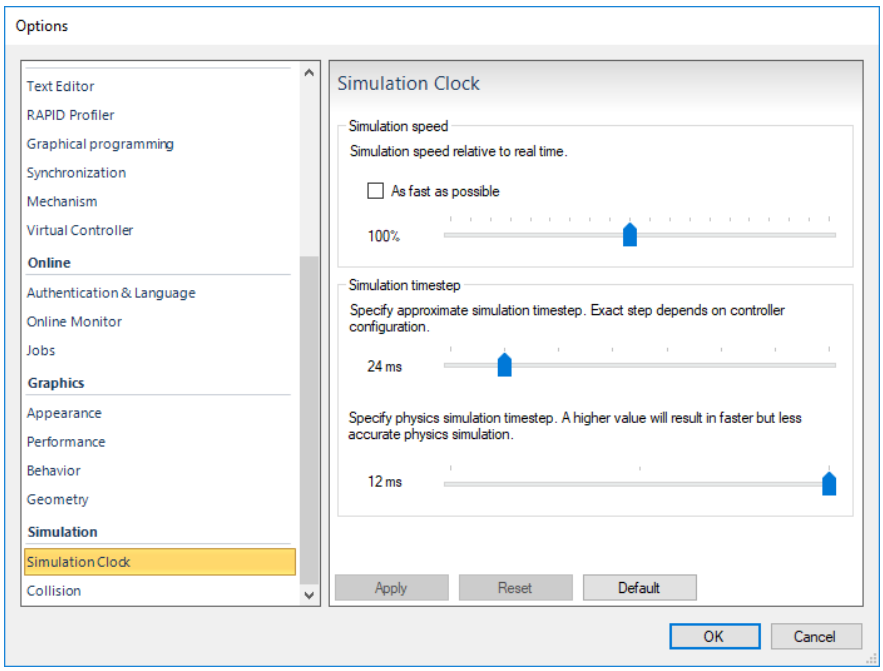


Figure 125: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*15.96mm*): TimStep 12ms (*12ms*)

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

d) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (18.24mm)



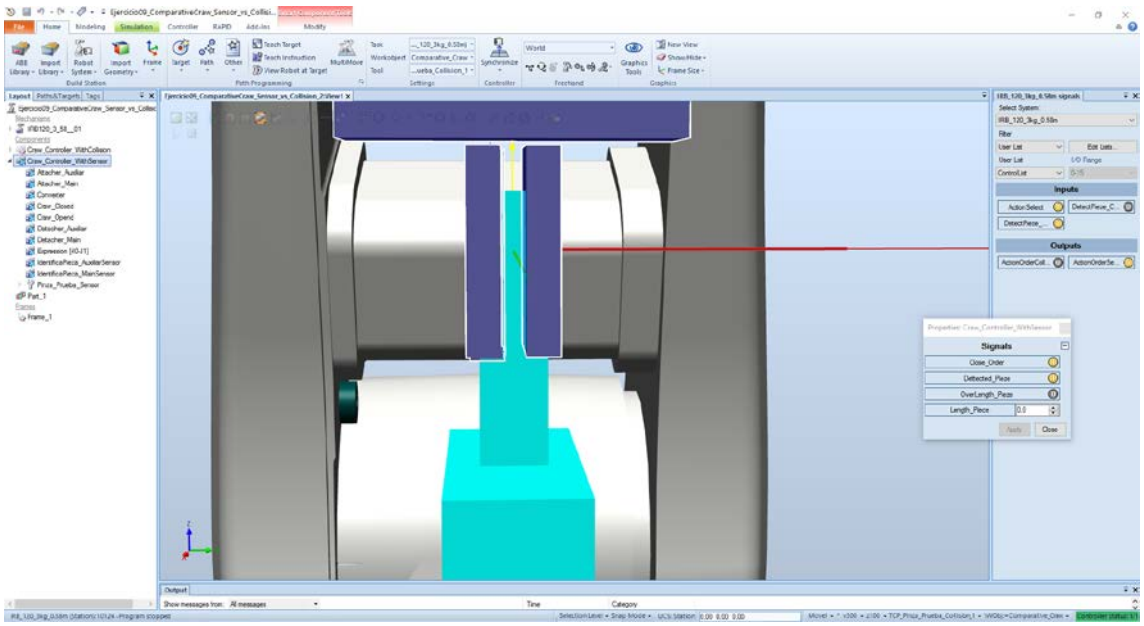
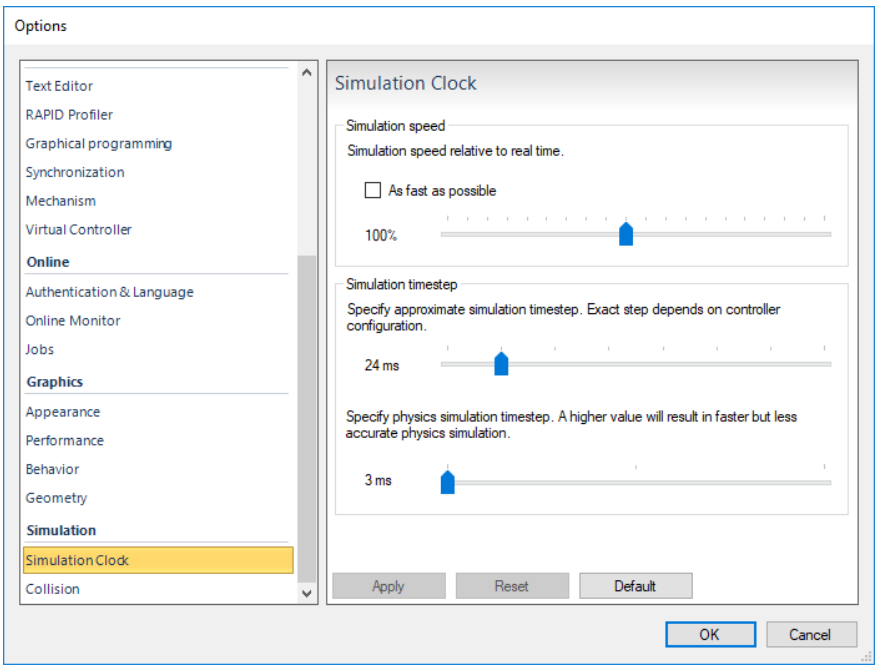


Figure 126: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18,24mm*): TimStep 24ms (*12ms*)

e) Velocidad de Aproximación: 0.1 seg en recorrer 20 mm: Length final (18.24mm)



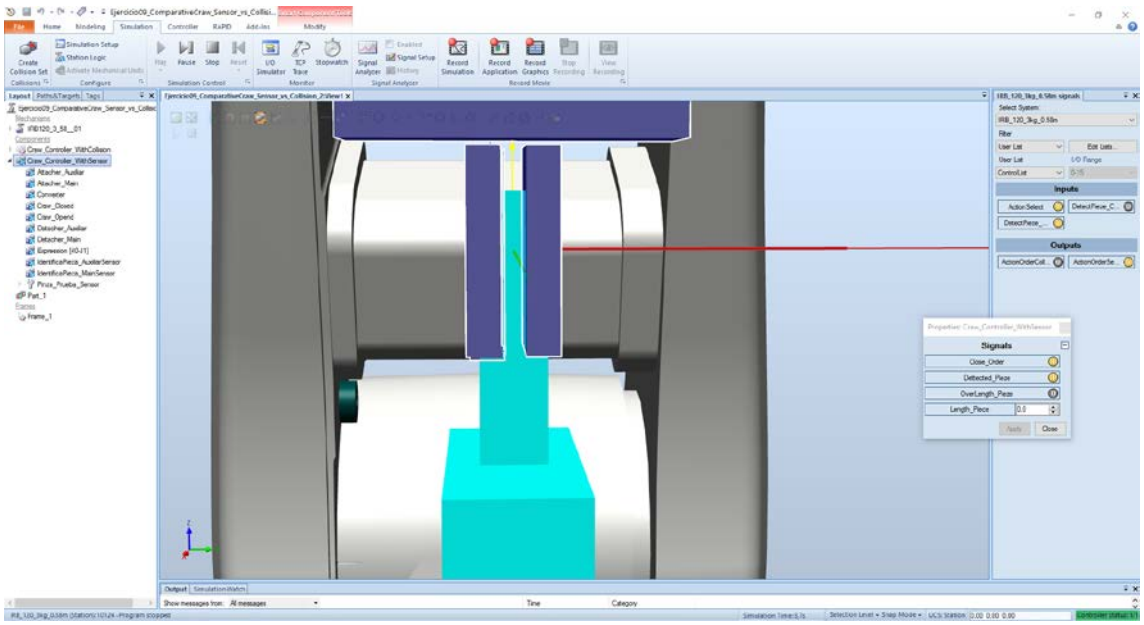


Figure 127: Velocidad de Aproximación: 0.1 seg/20 mm: Length final (*18,24mm*): TimStep 24ms (*3ms*)

ANEXO V

5. Implementación de la Herramienta Virtual: "*SR.8*"

5. 1. La Herramienta Virtual: "SR.8"

Se ha dotado al Robot de una **herramienta** lo más **versátil** posible para poder abordar un amplio abanico de **tareas**, tanto de **posicionamiento y seguimiento de trayectorias** como de manipulación **y agarre de objetos** existentes.

La herramienta es una **combinación de una Pinza Neumática** de gran apertura, de la empresa **SMC**, **con un puntero** acoplable por un mecanismo roscado con guías de precisión.

La pinza real se encuentra instalada en el Robot IRB 120 del laboratorio L0.06 del DIIS de la EINA. Modelo Real.

*Part*iendo de una **base de neumática de la empresa SMC, ver Figure 129: Cuerpo de la Pinza MHL2-10** del **Modelo MHL2_ES-10**, se han instalado **2 dispositivos que hacen la vez de "dedos"** de la garra **y** de un **orificio roscado dotado con guías mecánicas** ubicado en el centro geométrico de la base, sobre el que se puede **acoplar y desacoplar un puntero metálico** acabado en una **punta de 10µm**.

A) *Herramienta Real "Modelo MHL2_ES_10"*

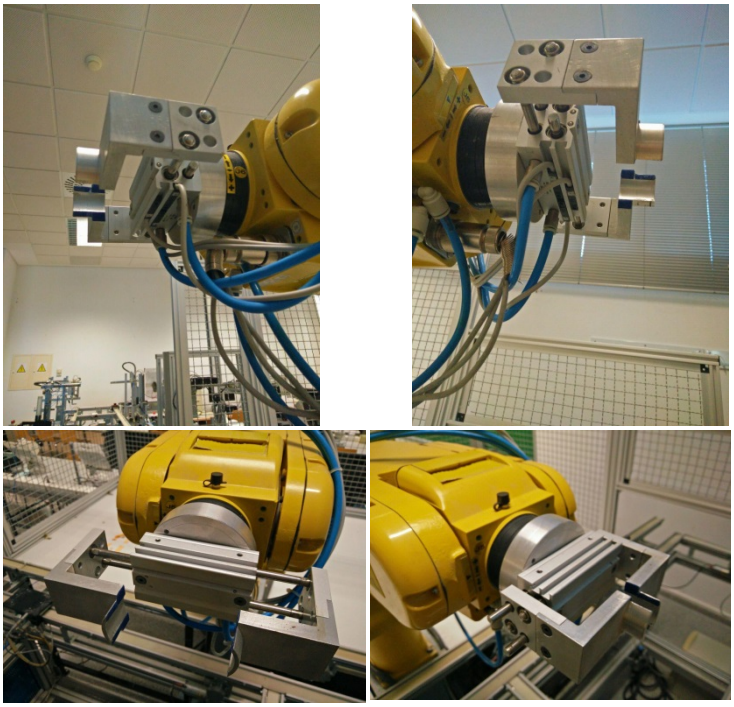


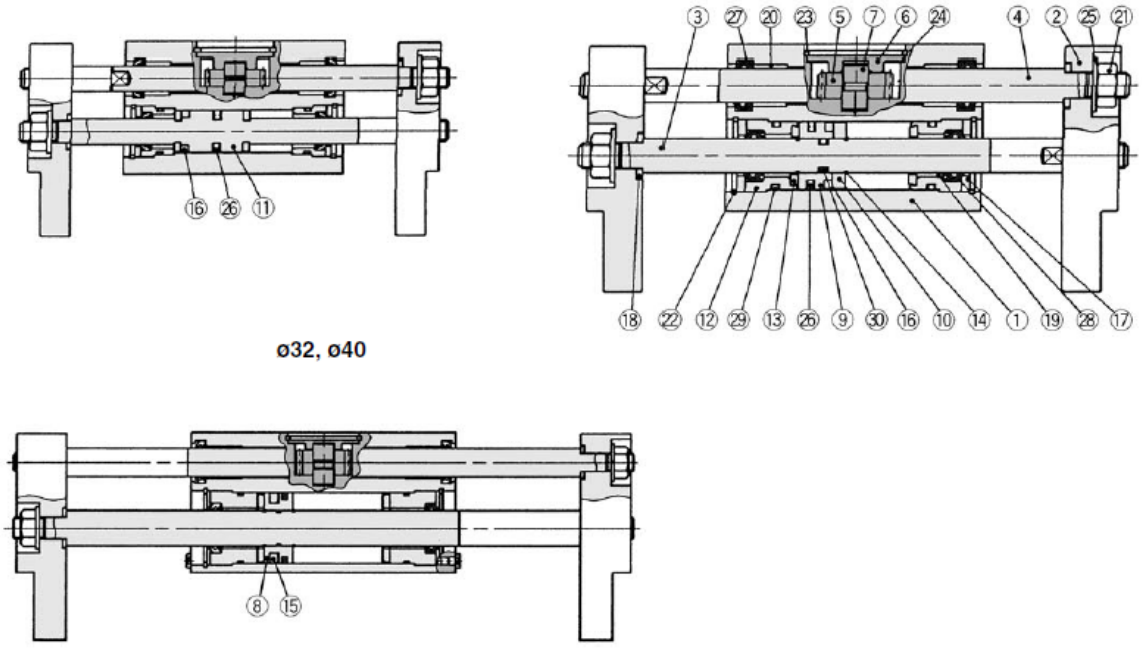
Figure 128: Herramienta Garra Real: Base neumática de la empresa SMC "*Modelo MHL2_ES-10*"

La **distribución de masas**:

- **1 kg para la pinza completa**, base de la garra y los dedos
- **100gr** para el **puntero** acoplable

Para la configuración del puntero se deberá cerrar la pinza para garantizar la verticalidad y dotar de mayor resistencia a flexión del puntero.

Es responsabilidad del usuario asegurar el correcto posicionamiento del puntero, así como garantizar las condiciones de seguridad en la manipulación del puntero y del robot.



Lista de componentes

Nº	Designación	Materiales	Observaciones
①	Cuerpo	Aleación de aluminio	Anodizado
②	Dedos	Aleación de aluminio	Anodizado
③	Vástago	Acero inoxidable	
④	Cremallera	Acero inoxidable	
⑤	Piñón	Acero al carbono	
⑥	Cubierta piñón	Acero al carbono	Niquelado electrolítico
⑦	Eje piñón	Acero inoxidable	Nitrurado
⑧	Émbolo	Latón	
⑨	Émbolo A	Latón	
⑩	Piston B	Latón	
⑪	Émbolo A	Acero inoxidable	
⑫	Tapa	Aleación de aluminio	Cromado
⑬	Amortiguador	Caucho uretano	
⑭	Clip	Acero inoxidable para muelles	
⑮	Imán	Goma sintética	

Nº	Designación	Materiales	Observaciones
⑯	Imán	Material magnético	Niquelado
⑰	Cubierta vástago B	Acero laminado frío	Niquelado electrolítico
⑱	Arandela	Acero inoxidable	Nitrurado
⑲	Casquillo	Metal lubricado	
⑳	Casquillo	Metal lubricado	
㉑	Tuerca U	Acero al carbono	Niquelado
㉒	Anillo de cierre R	Acero al carbono	Niquelado
㉓	Anillo de cierre C	Acero al carbono	Niquelado
㉔	Arandela	Acero para muelle	Revestimiento fosfato
㉕	Arandela	Acero al carbono	Niquelado

Figure 129: Cuerpo de la Pinza MHL2-10

Se han incluido las características técnicas de la pinza mecánica, así como los indicadores de limitaciones de carga máxima, ver el *DataSheet* completo de la pinza mecánica:

- Ver: **DataSheet_MHL2_ES**

5. 2. Implementación de la Herramienta Virtual

Se ha diseñado una **herramienta** lo más **versátil** posible, para poder abordar un amplio abanico de **tareas**, tanto de **posicionamiento y seguimiento de trayectorias** como de **manipulación y agarre de objetos**.

La herramienta "MHL02-10-SR8", es una **combinación de una Pinza Neumática** de gran apertura, empresa **SMC**, **con un puntero** acoplable, emulando la Herramienta real que se va a instalar en el Robot IRB 120 del laboratorio L0.06 del DIIS de la EINA.

*Part*iendo del modelo teórico de la herramienta real, se ha parametrizado y reproducido en un modelo digital, el cual poder exportar a las distintas simulaciones. *Las posibles diferencias entre la herramienta real y la herramienta simulada responden a la secuencia de fechas entre la orden de fabricación de la herramienta real y la necesidad de simular resultados con la herramienta virtual.*

RobotStudio permite **definir las características específicas de la herramienta**, **Centro Geométrico (CG) e Inercias Principales, I_x I_Y e I_z**, para cada uno de los Modos de Operación contemplados, Modo Garra y Modo Puntero.

Se ha diseñado el control lógico de la Herramienta por dos procedimientos distintos:

a. **Control lógico con Sensores** (ejemplo expuesto a continuación):
    a. Sin restricciones de diseño.
    b. El vástago posee 1 Grado de Libertad (desplazamiento traslacional, se esconde en el interior del robot virtual).
b. **Control lógico con Detector de Colisiones**:
    a. Los sólidos que componen la Herramienta NO pueden tocarse ni en situación estática ni en las trayectorias de desplazamiento.
    b. El vástago NO posee Grado de Libertad, se oculta y se visualiza en la misma posición física relativa a la base.

5. 2. 1. Propiedades Generales y Geométricas

A pesar de haberse diseñado por medio de dos procedimientos distintos, la **estructura geométrica resultante de la Herramienta para las dos configuraciones es muy similar**.

Aproximaciones en el diseño geométrico

Se han aplicado las siguientes **aproximaciones en el diseño**:

1. **Estructura Geométrica Regular**: Composición de prismas regulares, un cilindro que solo tiene relevancia en la Modalidad Puntero y varillas guía de masa despreciable.
2. Todos los Prismas, Cilindros y Varillas, se consideran **Sólidos Rígidos, Indeformables e Impenetrables**
3. A nivel de CG e Inercias: **Posee 2 Planos de Simetría en torno a Z ("xz" e "yz")**, lo que provoca que las CG_X y CG_Y sean igual a 0.
4. **Base de la Herramienta**: Centrada en el Origen con la Base de la Brida coincidente con el Centroide de la misma. Provoca que las CG_X y CG_Y sean igual a 0.
5. La **Varilla**, sólido extraíble, en la simulación guardada una **representación únicamente visual** en la modalidad garra, aunque con repercusiones en la lógica cableada.
6. **Garra de gran apertura** (*comparación con la garra real*): 64mm frente a 30mm de la real. Conveniente para facilitar la manipulación de un mayor número de objetos.

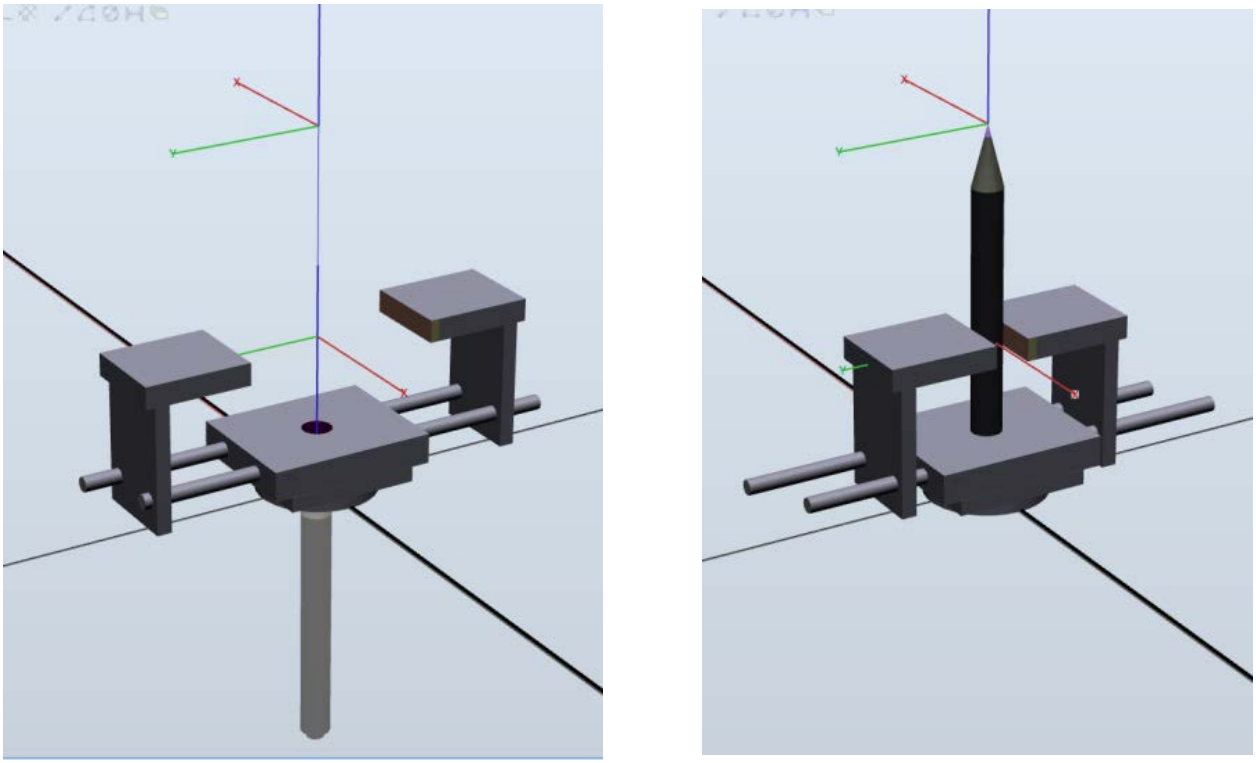


Figure 130: Diseño Geométrico de la Herramienta Virtual: SR.8 (*modalidad completa: Sticker deslizante*)

Cálculo de CG e Inercias Principales

Para calcular el CG e Inercias Principales se ha utilizado Matlab, los datos se han sacado de los *Script* implementados.

```
%Datas Piece:
%Considerar: Simétrica en X e Y
%Considerar: centrada en origen con Base de mA coincidiendo Centroide con RefO

%Masas Individuales:
mA=0.075;       %Junta de sección Circular R=44 h=5
mB=0.125;       %Base Sección Cuadrada 44x44x5 (X Y Z)
mC=0.225;       %Base Sección Cuadrada 56x44x10
mD=0.1125;      %Soporte de Dedo: 5x30x50
mE=0.175;       %Dedo: 36x30x7.5
mBarra=0.075;   %Barra: R=10 h=70
mCono=0.025;    %Cono: R=10 h=30

%Distancia respecto de la Referencia (0 0 0)
%Distancia en el Eje X: CGi-X respecto de X=0
dxA=0;
dxB=0;
dxC=0;
dxD=((0.056/2)+0.0325+(0.005/2));       %Dedos Completamente abiertos
dxE=((0.056/2)+0.0325+0.005-(0.036/2)); %Dedos Completamente abiertos
dxD_2=((0.056/2)+(0.005/2));            %Dedos Ajustados al StickPointer
dxE_2=((0.056/2)+0.005-(0.036/2));      %Dedos Ajustados al StickPointer
dxBarra=0;
dxCono=0;

%Distancia en el Eje Y:  CGi-Y respecto de Y=0:
%PARA TODA PIEZA TODOS LOS CGiY en Y=0
%Distancia en el Eje Z: CGi-Z respecto de Z=0: Para toda la pieza SOBRE el plano de
Corte de la Ref0: TODOS los CGiZ>Z=0
dzA=(0.005/2);
dzB=(0.005+0.005/2);
dzC=(0.005+0.005+0.010/2);
dzD=(0.010+(0.050/2));
dzE=(0.010+0.050+(0.0075/2));
dzBarra=(0.010+0.070/2);
dzCono=(0.010+0.070+0.030/2);
```

Figure 131: Datos Generales de CG's e Inercias Principales aplicables a los dos Modos de Simulación de Herramienta Virtual SR.8

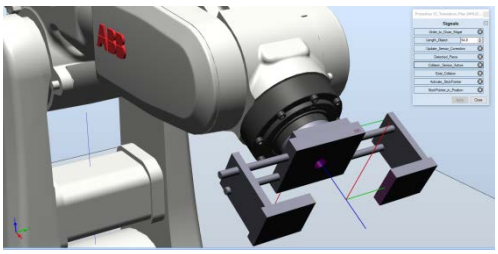
Utilizando como ejemplo la Herramienta creada con Sensores **Figure 130: Diseño Geométrico de la Herramienta Virtual: SR.8 (*modalidad completa: Sticker deslizante*)**.

La herramienta ha sido diseñada contemplando dos **modos** distintos **de funcionamiento, que el usuario puede seleccionar durante la simulación**, y sus consiguientes propiedades técnicas.

En el **cálculo del CG e Inercias Principales** se ha diferenciado entre las configuraciones:

A) *Modo Garra*

Se ha **despreciado la existencia del Puntero extraíble** y considerado la confi**guración más desfavorable con los dedos en la posición extendida**, "completamente abierta".



```
%Calculo del Centro de Gravedad: Al ser simetrico a X (XG=0) e Y (YG=0), solo existe
ZG
%Calculo de ZG: [mm]:
Za=mA*2.5;
Zb=mB*7.5;
Zc=mC*15;
Zd=mD*30;
Ze=mE*47.5;
% Garra básica suma 1 Kg
ZG_Simple=(Za+Zb+Zc+2*Zd+2*Ze)/(mA+mB+mC+2*mD+2*mE)

%%
%Inercia en x de la Herramienta: [kg/m^2]
Ax=(1/4)*mA*0.022^2;
Bx=(1/12)*mB*(0.044^2+0.005^2);
Cx=(1/12)*mC*(0.044^2+0.010^2);
D1x=(1/12)*mD*(0.050^2+0.030^2) + (0.050*0.030)*dxD^2;
E1x=(1/12)*mE*(0.0075^2+0.030^2) + (0.0075*0.030)*dxE^2;
D2x=(1/12)*mD*(0.050^2+0.030^2) - (0.050*0.030)*dxD^2;
E2x=(1/12)*mE*(0.0075^2+0.030^2) - (0.0075*0.030)*dxE^2;

Ix_Simple=Ax+Bx+Cx+D1x+E1x+D2x+E2x

%%
%Inercia en y de la Herramienta: [kg/m^2]:
%Para Todo Centro de gravedad en el Plano y=0: NO STEINER's
Ay=(1/4)*mA*0.022^2;
By=(1/12)*mB*(0.044^2+0.005^2);
Cy=(1/12)*mC*(0.056^2+0.010^2);
D1y=(1/12)*mD*(0.050^2+0.005^2);
E1y=(1/12)*mE*(0.0075^2+0.036^2);

Iy_Simple=Ay+By+Cy+D1y+E1y+D2y+E2y

%%
%Inercia en z de la Herramienta: [kg/m^2]:
%Para Tod a parte necesario: STEINER's EveryWhere
Az=(1/4)*mA*0.022^2+2*pi*0.022*dzA^2;
Bz=(1/12)*mB*(0.044^2+0.044^2)+(0.044*0.044)*dzB^2;
Cz=(1/12)*mC*(0.056^2+0.044^2)+(0.056*0.044)*dzC^2;
D1z=(1/12)*mD*(0.030^2+0.005^2)+(0.03*0.005)*dzD^2;
E1z=(1/12)*mE*(0.030^2+0.036^2)+(0.03*0.036)*dzE^2;

Iz_Simple=Az+Bz+Cz+D1z+E1z+D2z+E2z
```

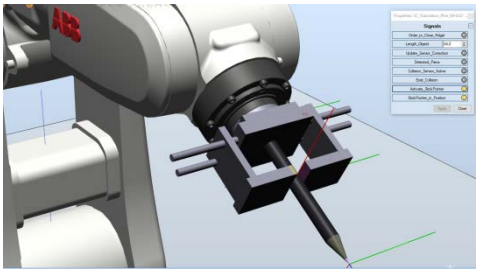
Figure 132: Script Matlab: CG's e Inercias Principales: Modo Garra: Herramienta Virtual SR.8

Masa [Kg]	C.G.=(x,y,z) [mm]			Inercias: I_x I_Y I_z [Kg/ m^2]		
1	0	0	27.8750	0.00015932	0.00017696	0.00023658

Figure 133: Propiedades Técnicas Herramienta Simulada "Modo Garra"

B) *Modo Puntero "Stick"*

La configuración cambia respecto del Modo Garra, la ubicación de los dedos pasa a ser cerrada conteniendo/agarrando el Puntero para dotarlo de mayor resistencia a flexión.



```
%Calculo del Centro de Gravedad: Al ser simetrico a X (XG=0) e Y (YG=0), solo existe
ZG
%Calculo de ZG: [mm]:
Za=mA*2.5;
Zb=mB*7.5;
Zc=mC*15;
Zd=mD*30;
Ze=mE*47.5;
% Garra básica suma 1 Kg
ZBa=mBarra*(20+70/2);
ZCo=mCono*(20+70+(1/4)*30);
% Garra + StickPointer suma 1.100 gr
ZG_Puntero=(Za+Zb+Zc+2*Zd+2*Ze+ZBa+ZCo)/(mA+mB+mC+2*mD+2*mE+mBarra+mCono)

%%
%Inercia en x de la Herramienta: [kg/m^2]
Ax=(1/4)*mA*0.022^2;
Bx=(1/12)*mB*(0.044^2+0.005^2);
Cx=(1/12)*mC*(0.044^2+0.010^2);
D1x_2=(1/12)*mD*(0.050^2+0.030^2) + (0.050*0.030)*dxD_2^2;
E1x_2=(1/12)*mE*(0.0075^2+0.030^2) + (0.0075*0.030)*dxE_2^2;
D2x_2=(1/12)*mD*(0.050^2+0.030^2) - (0.050*0.030)*dxD_2^2;
E2x_2=(1/12)*mE*(0.0075^2+0.030^2) - (0.0075*0.030)*dxE_2^2;
Barrax=(1/12)*mBarra*(3*0.005^2+0.070^2);
Conox=(3/20)*mCono*(0.005^2+4*0.030^2);

Ix_Puntero=Ax+Bx+Cx+D1x_2+E1x_2+D2x_2+E2x_2+Barrax+Conox

%%
%Inercia en y de la Herramienta: [kg/m^2]:
%Para Todo Centro de gravedad en el Plano y=0: NO STEINER's
Ay=(1/4)*mA*0.022^2;
By=(1/12)*mB*(0.044^2+0.005^2);
Cy=(1/12)*mC*(0.056^2+0.010^2);
D2y=(1/12)*mD*(0.050^2+0.005^2);
E2y=(1/12)*mE*(0.0075^2+0.036^2);
Barray=(1/12)*mBarra*(3*0.005^2+0.070^2);
Conoy=(3/20)*mCono*(0.005^2+4*0.030^2);

Iy_Puntero=Ay+By+Cy+D1y+E1y+D2y+E2y+Barray+Conoy

%%
%Inercia en z de la Herramienta: [kg/m^2]:
%Para Tod a parte necesario: STEINER's EveryWhere
Az=(1/4)*mA*0.022^2+2*pi*0.022*dzA^2;
Bz=(1/12)*mB*(0.044^2+0.044^2)+(0.044*0.044)*dzB^2;
Cz=(1/12)*mC*(0.056^2+0.044^2)+(0.056*0.044)*dzC^2;
D2z=(1/12)*mD*(0.030^2+0.005^2)+(0.03*0.005)*dzD^2;
E2z=(1/12)*mE*(0.030^2+0.036^2)+(0.03*0.036)*dzE^2;
Barraz=(1/2)*mBarra*(0.005^2)+2*pi*0.005*dzBarra^2;
Conoz=(3/10)*mCono*(0.005^2)+2*pi*0.005*dzCono^2;

Iz_Puntero=Az+Bz+Cz+D1z+E1z+D2y+E2z+Barraz+Conoz
```

Figure 134: Script Matlab: CG's e Inercias Principales: Modo Stick: Herramienta Virtual SR.8

Masa [Kg]	C.G.=(x,y,z) [mm]			Inercias: I_x I_Y I_z [Kg/ m^2]		
1.1	0	0	31.3068	0.00020401	0.00022165	0.00059966

Figure 135: Propiedades Técnicas Herramienta Simulada "Modo Stick"

OBSERVACIÓN: La influencia del vástago no supone cambios relevantes, en la implementación de la herramienta virtual para las simulaciones de RobotStudio, se podría haber despreciado la inclusión de estos cálculos a la hora de definir/crear la herramienta.

Código de Colores para Leer los Esquemas Generales Implementados

Se ha empleado un índice de colores, para la identificación de E/S, en los esquemas generales de las Herramientas implementados:

- Azul oscuro: Modo operación Garra
- Violeta: Modo StickPointer
- Naranja: Sólo de carácter informativo: Detector de Colisiones
- Verde: Informa de la distancia de apertura de la garra

5. 2. 2. Propiedades Constructivas de la Herramienta Virtual Implementada con Sensores

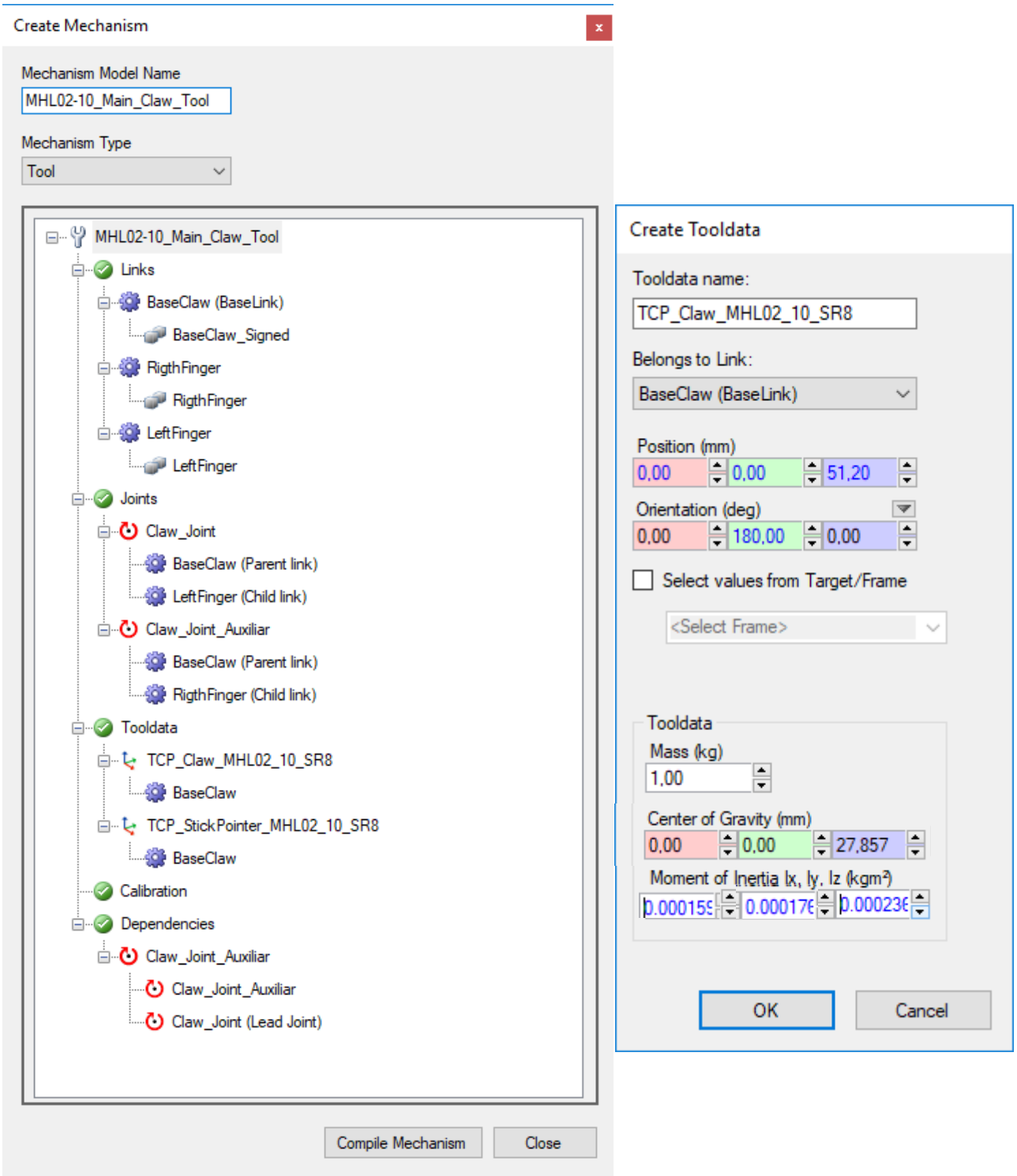


Figure 136: Propiedades Constructivas de la Herramienta Virtual Implementada con Sensores

A) *Propiedades Constructivas Principales*

a. **No sujeta a restricciones estructurales**: irrelevancia a penetración entre los sólidos que la componen.
b. No es estrictamente necesario contemplar un Grado de Libertad adicional para el vástago, se puede implementar con bloques de ocultación y visualización.
c. **Sujeta a restricciones propias de los sensores**: Ver **ANEXO IV**.
Se la ha dotado "*por si las moscas*" de un pin para forzar un refresco de los sensores, antes de ser utilizada.
d. **Necesarios 2 Sensores planos tangenciales entre sí**: uno principal y otro auxiliar en el funcionamiento en cierre de garra.
e. La **distancia de penetración del Plano del Sensor, en el sólido detectable,** es inversamente proporcional a la relación entre el "Paso de Simulación" (**20**) y la velocidad relativa de aproximación entre los sensores y el objeto a detectar.
f. **Lógica cableada**: aumenta considerablemente la complejidad y cantidad de bloques lógicos para controlar la operación de agarre seguro.
g. **Mayor tiempo de ciclo** para comprobar los estados lógicos.
h. **Posible aparición de "Lag" en simulación**, se aprecian saltos discontinuos en la simulación, aunque la ejecución del programa es correcta.
i. **Menor tamaño del fichero** ".rslib".
j. **Requiere la inclusión de precisión "fine"** en la orden previa de movimiento del robot. No necesita consideraciones adicionales en código RAPID.

B) *Esquema general*

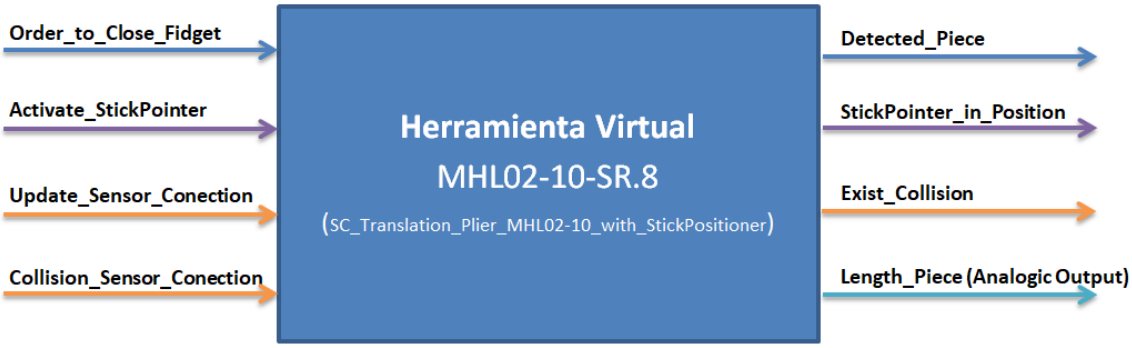


Figure 137: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con Sensores

C) *Esquema Implementado en Controlador de Sensores*

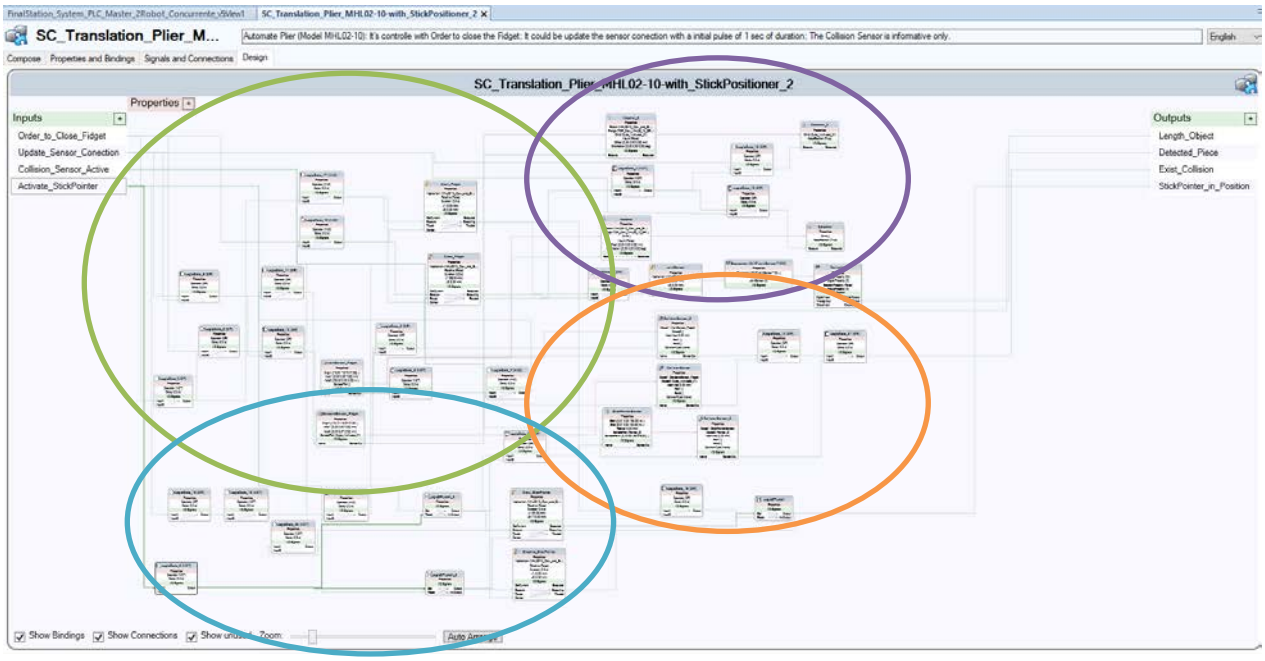


Figure 138: Esquema General: Implementación con Sensores

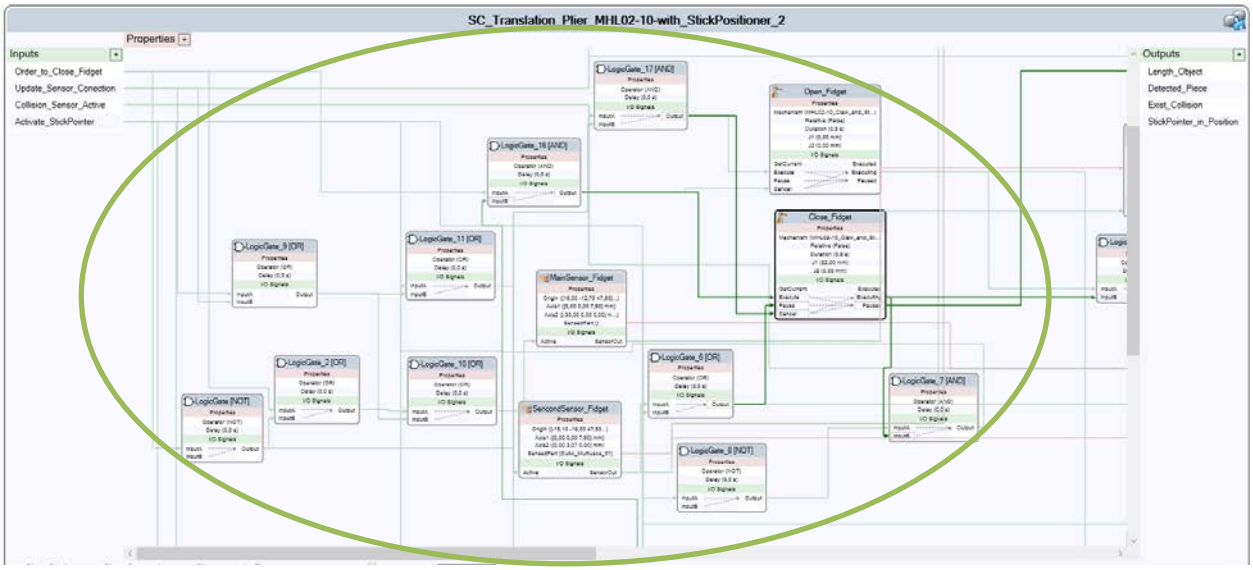


Figure 139: Esquema Específico Sensores: Modo Garra: Identificación de Objetos y Apertura/Cierre de la Garra

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

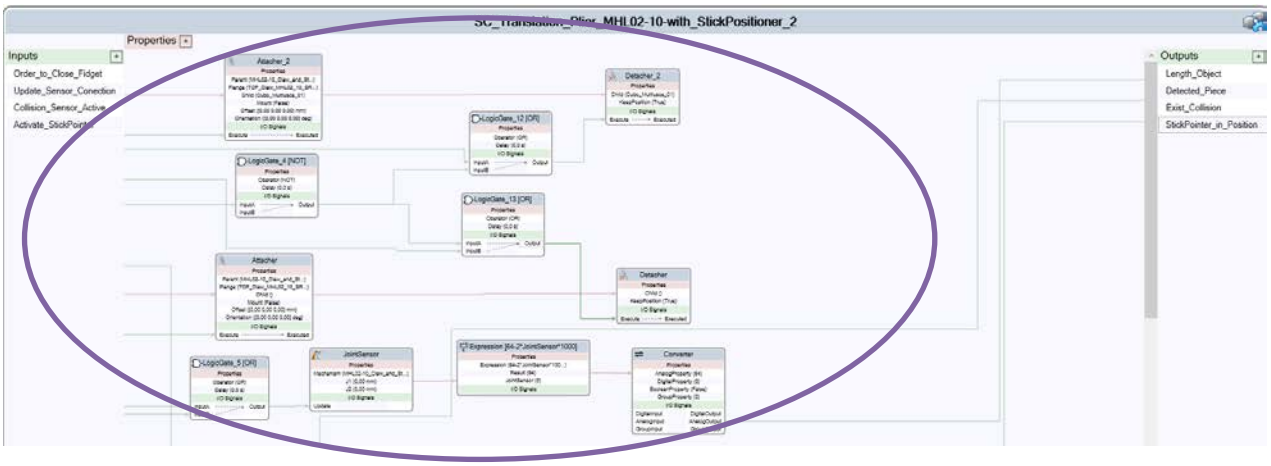


Figure 140: Esquema Específico Sensores: Modo Garra: Enlazamiento/Agarre de Objetos y Cálculo de Distancia de Apertura de Garra

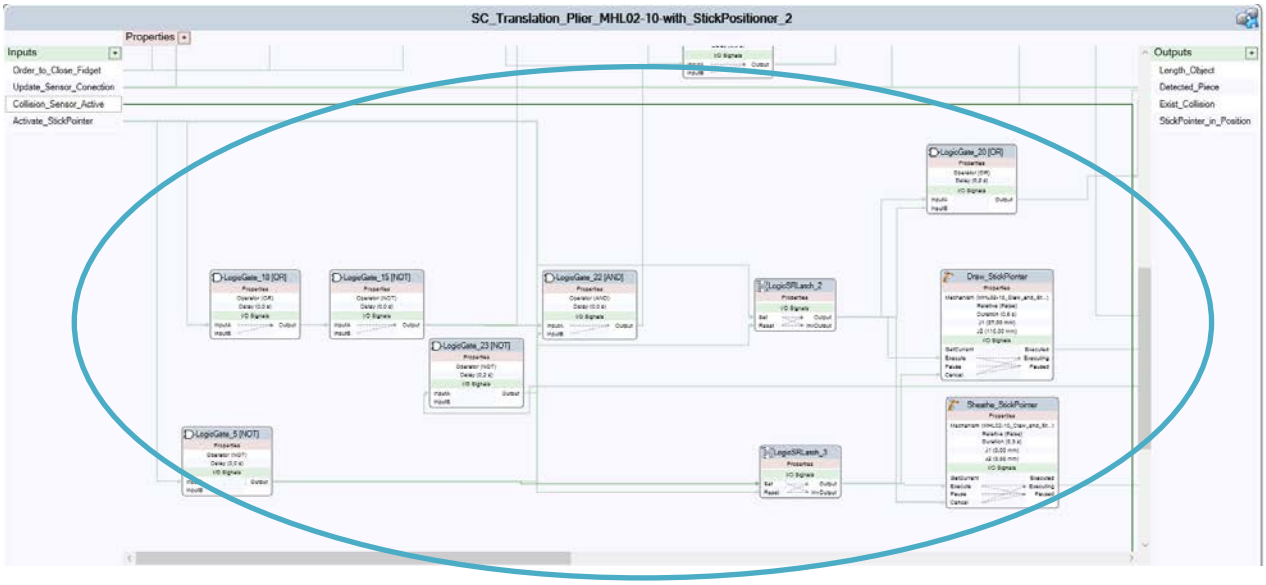


Figure 141: Esquema Específico Sensores: Control del Modo Sticker

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

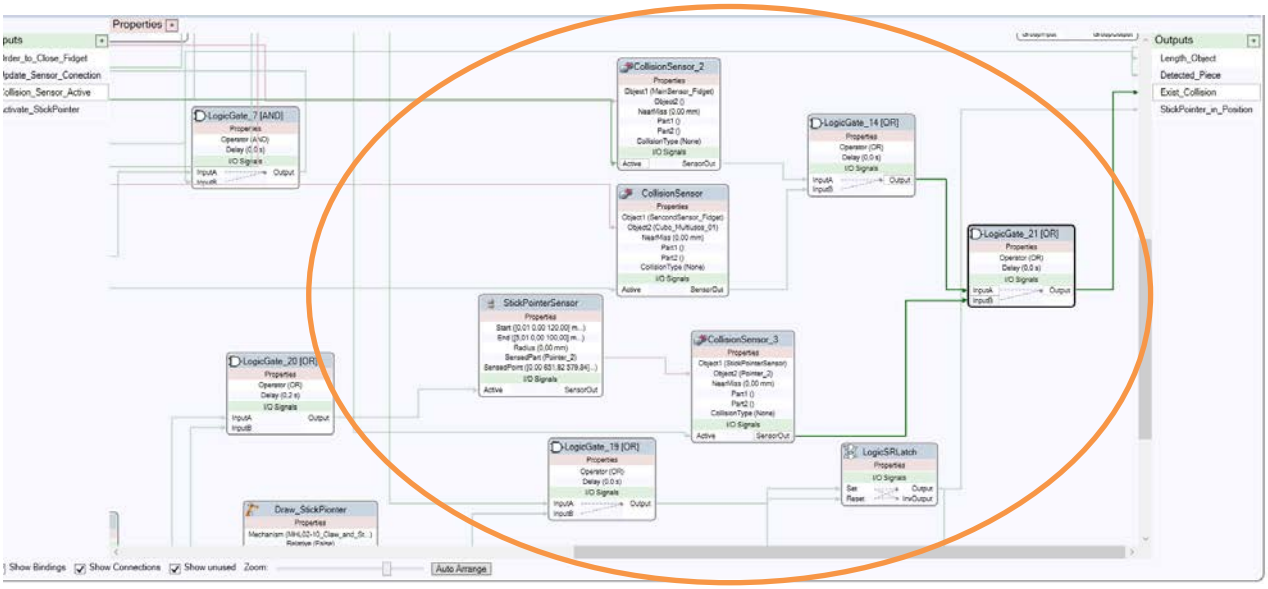


Figure 142: Esquema Específico Sensores: Detección de lo Colisiones doble Modo (*funcionalidad opcional*)

D) *Funcionamiento de la Garra: Implementada con Sensores*

Se puede ver el Funcionamiento de la Garra en el siguiente vídeo: **Tool_Sensor_Claw**.

5. 2. 3. Propiedades Constructivas de la Herramienta Virtual Implementada con Detección de Colisiones "*CollisionSenser*"

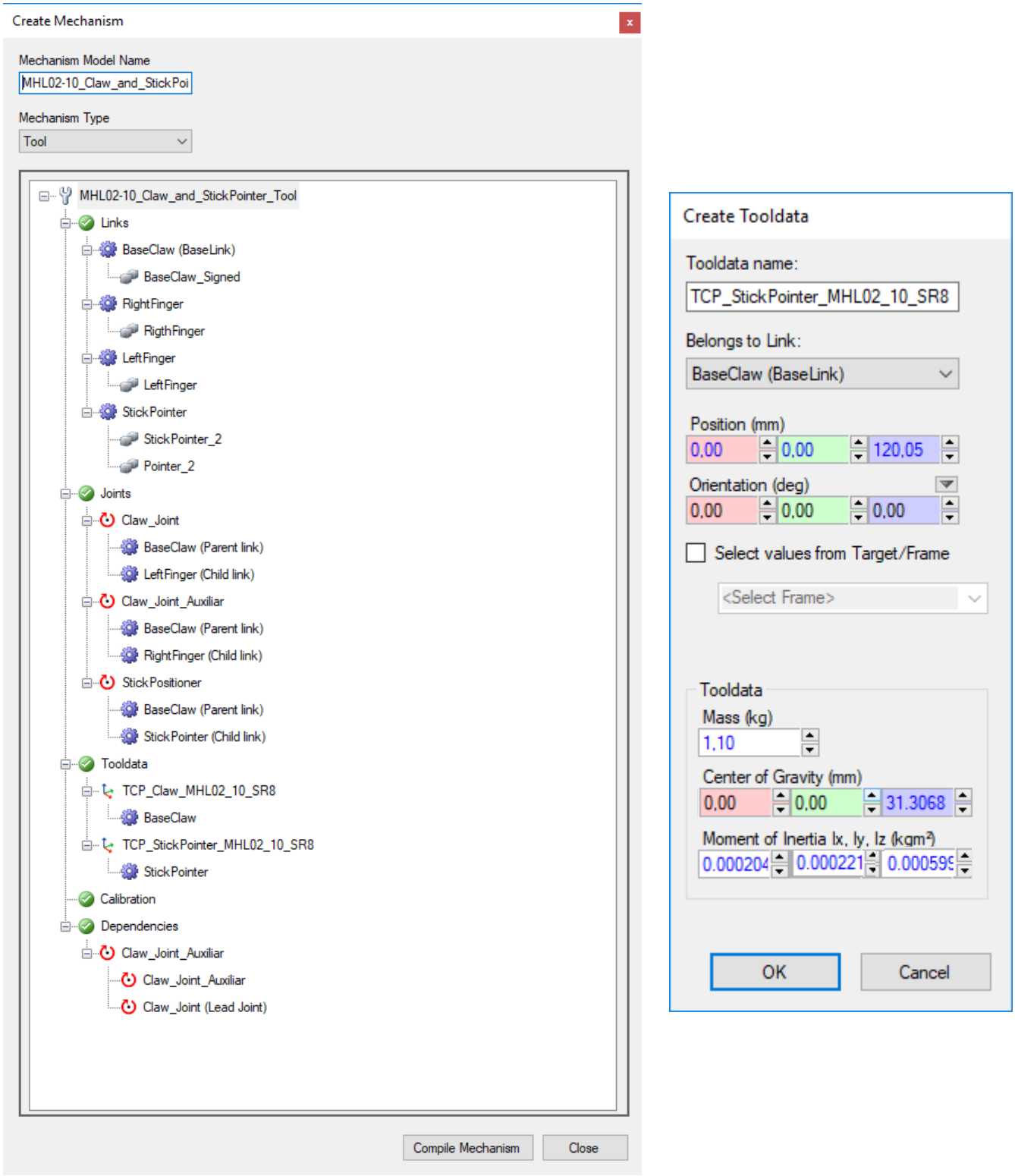


Figure 143: Propiedades Constructivas de la Herramienta Virtual Implementada con CollisionSenser

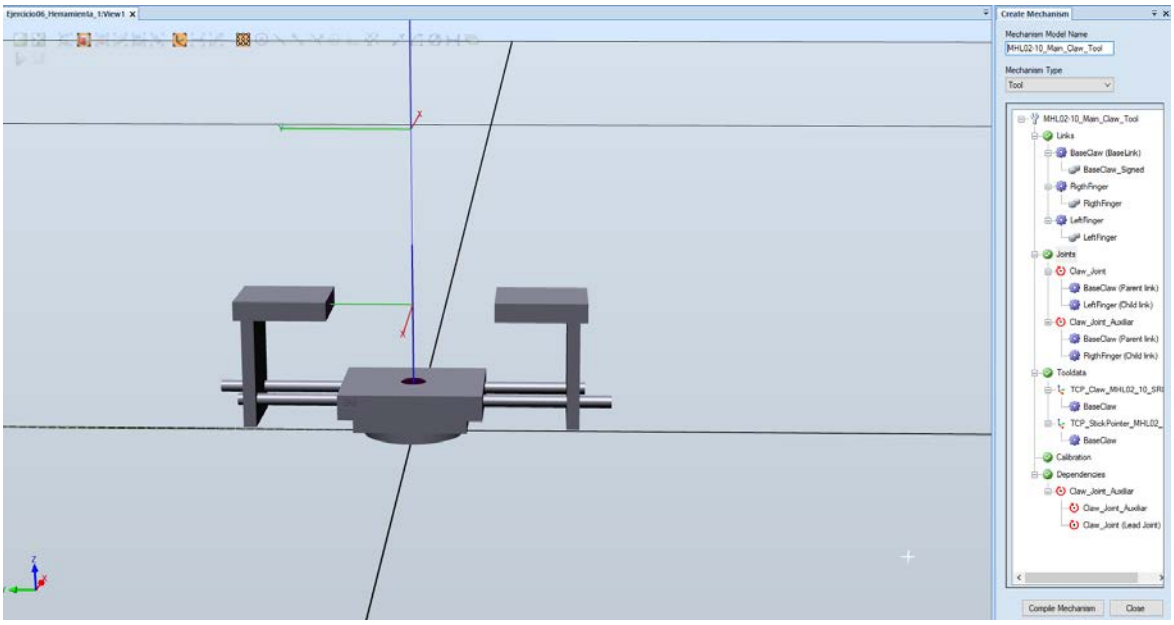


Figure 144: Diferencia Física: El Vástago NO posee GL, y por consiguiente se puede Ocultar/mostrar

A) *Propiedades Constructivas Principales*

a. **Sujeta a restricciones estructurales**: obligación de que no exista contacto, ni superposición, entre los sólidos que componen las *Part*es de la Herramienta.
b. No es necesario contemplar un Grado de Libertad adicional para el vástago, se puede implementar con bloques de ocultación y visualización, como ha sido el caso. Lo que evita la necesidad de la salida "output" validando la posición del StickPointer.
c. **No utiliza sensores convencionales**: no sujeta a restricciones propias de los sensores.
d. **Mayor robustez a penetración**: menor distancia de penetración de la herramienta sobre el sólido detectable, la penetración es más constante ante la relación de "Velocidad de Aproximación-**20**". Ver **ANEXO IV**.
e. **Menor cantidad de lógica cableada**: disminuye el número de bloques lógicos para controlar la operación de agarre seguro.
f. **Menor tiempo de ciclo** en la comprobación de los estados lógicos.
g. **No detectada la aparición de "Lag"** en simulación.
h. **Mayor tamaño del fichero** ".rslib", un orden 10 veces mayor que la modalidad de sensores.
i. Es **insuficiente** con **la inclusión de precisión "fine" en la orden previa de movimiento** del robot: **Ver vídeo demostración:** Acumulative_Error_CollisionSenser
Para aquellos programas en los que no se recolocan los objetos solos, o no se ejecutan los comandos de creación y destrucción, es **necesario tomar consideraciones adicionales en código RAPID** como dotar al fichero RAPID de "*WaitTime* xxxx" para evitar problemas en la acción de soltar piezas**. Ver vídeo**: Solution_Acumulative_Error

B) *Esquema general*

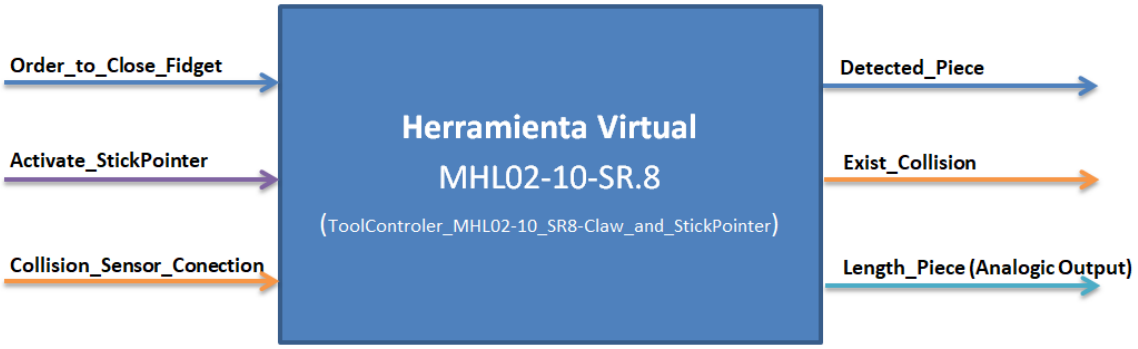


Figure 145: Esquema Genérico de las Entradas y Salidas de MHL02-10-SR8: Implementada con CollisionSensor

C) *Esquema Implementado en Controlador de CollisionSensor*

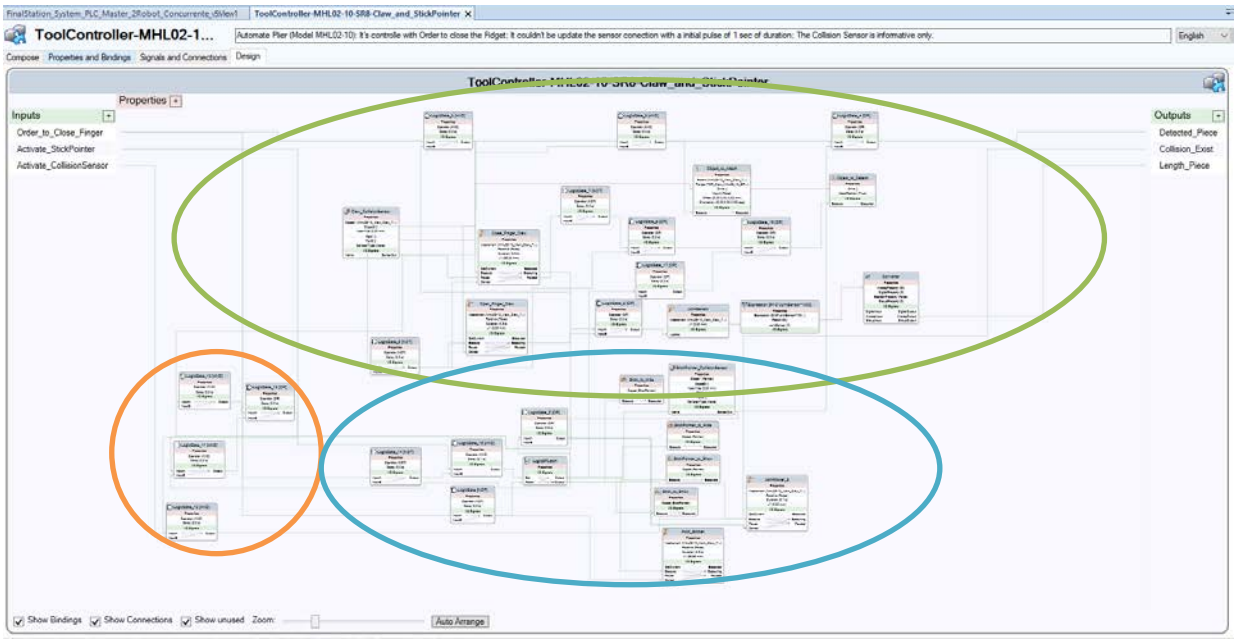


Figure 146: Esquema General: Implementación con CollisionSensor

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

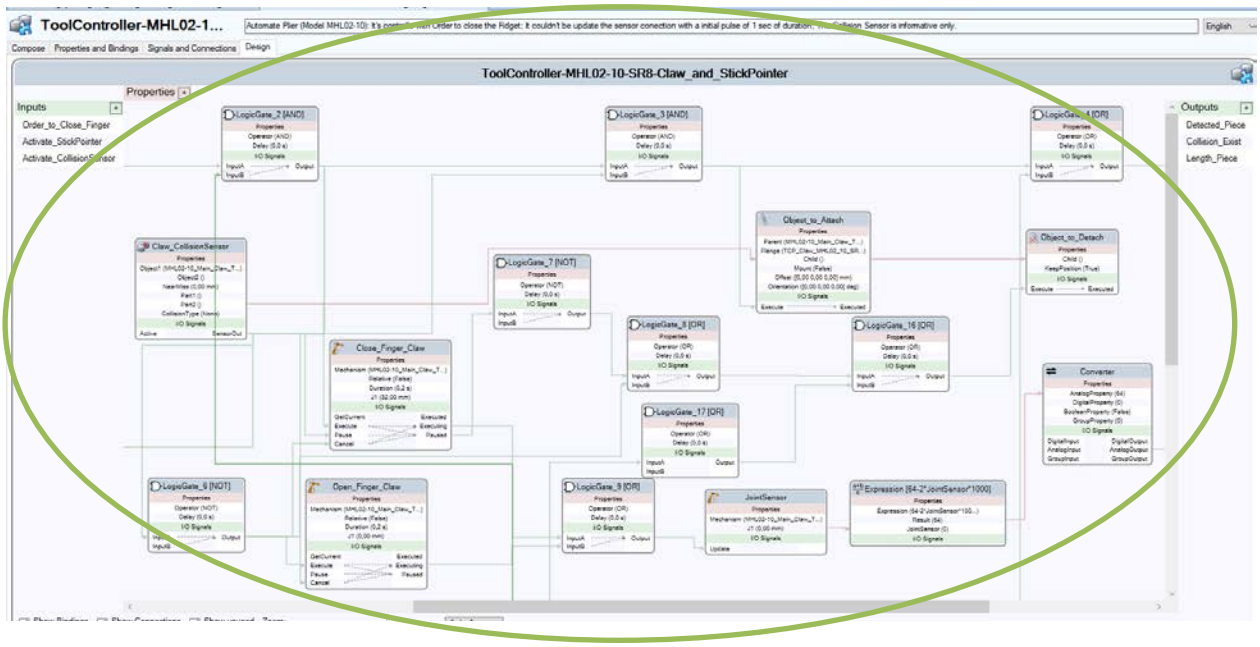


Figure 147: Esquema Específico CollisionSensor: Modo Garra: Apertura/Cierre de la Garra, Identificación de Objetos y Enlace/Desenlace de Objetos

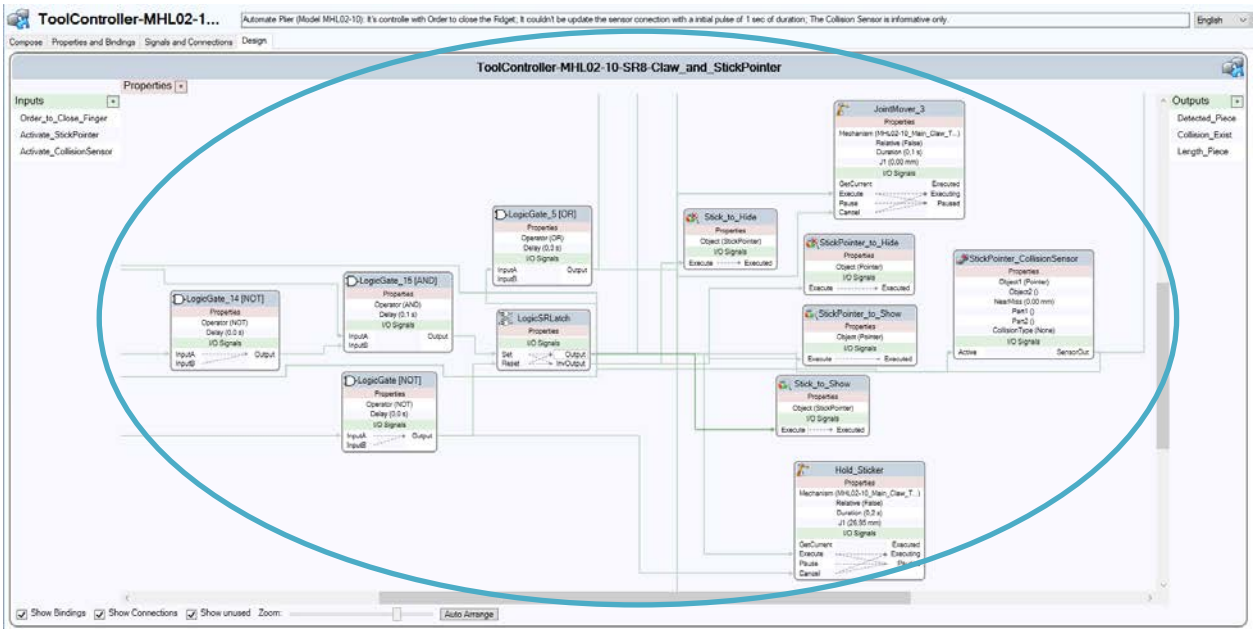


Figure 148: Esquema Específico CollisionSensor: Control del Modo Sticker

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

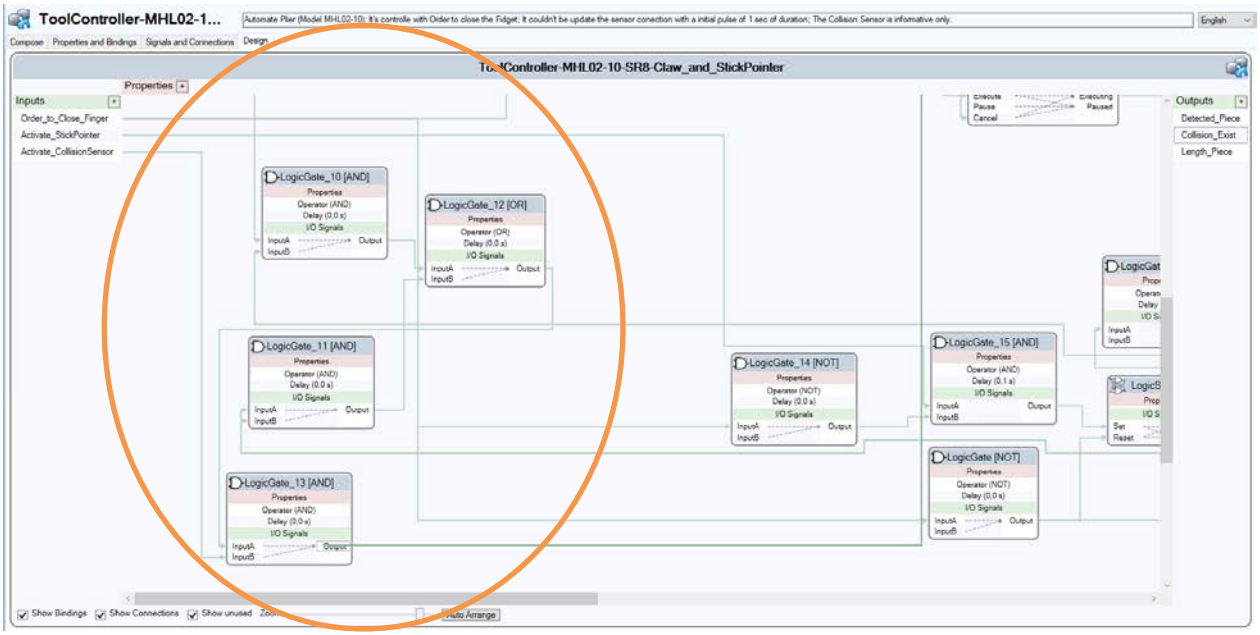


Figure 149: Esquema Específico CollisionSensor: Detección de lo Colisiones doble Modo (funcionalidad opcional)

D) *Funcionamiento de la Garra: Implementada con CollisionSensor*

Se puede ver el Funcionamiento de la Garra en el siguiente vídeo: **Tool_Collision_Claw**

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

ANEXO VI

6. Implementación del PLC 16 Estados: *Automatismo Moore*

6. 1. Simulación de PLC's

Hay que mencionar que: **RobotStudio no permite la implementación directa de PLC Maestros**, pese a ello se ha **conseguido implementar un Automatismo Tipo Moore** de dos formas distintas:

- **PLC Programable (***Punto de vista de un Programador***)**: Implementado en código estructurado, un Autómata de MOORE, con la modalidad *Switch-Case*
- **PLC de Lógica Cableada (***Tipología Industrial Típica***)**: diseñado al estilo tradicional, por medio de la implementación de "*Flip-Flop*" en un SmartObject, y sin representación gráfica sobre la Estación
- **PLC Lógica de Contactos**: al estilo tradicional, por líneas lógicas de contactos, en el *EventManager* (***DESECHADA*** *por elevado número de limitaciones y problemática en la gestión de E/S añadido a que es un sistema asíncrono; elemento relevado por los SmartObject*)

A) *Propiedades de la Estación simulada*

Partiendo de la base que la **Estación Simulada cuenta con**:

- **Sistema de suministro de piezas continuo y de distribución aleatoria**: representado por la Cinta Transportadora 1
- **Sistema de extracción segura y con ausencia de bloqueos**: representado por la Cinta Transportadora 2

El **PLC Maestro es el encargado de controlar el estado de la Estación** durante el proceso de producción. Se ha simplificado hasta conseguir un **Automatismo Tipo Moore de 16 Estados con Concurrencia**. Cada estado, representado en el **Grafo**, es una configuración absoluta de la Estación que específica qué *Part*es están en funcionamiento y qué tarea/operación se encuentra haciendo.

El PLC Maestro se encargará de coordinar 2 Robots y garantizar la concurrencia entre ambos:

- **Robot 1 "R1P$_x$C$_y$"**: responsable de proveer/posicionar las piezas sobre una zona de trabajo controlada y común y sacarlas de dichas zonas cuando ya estén mecanizadas
- **Robot 2 "R2P$_x$"**: responsable de manipular y mecanizar las piezas posicionadas previamente por el Robot 1.

B) *Automatismo Tipo Moore de 16 Estados con Concurrencia*

El **PLC** se encarga de **ordenar al Robot 1 una de las siguientes funciones**:

- Coger y posicionar las piezas de la Cinta Transportadora 1 sobre la Posición 1 de la zona de trabajo (***R111***)
- Coger y posicionar las piezas de la Cinta Transportadora 1 sobre la Posición 2 de la zona de trabajo (***R121***)
- Extraer de la Posición 1 la pieza mecanizada y depositarla en la Cinta Transportadora 2 (***R112***)
- Extraer de la Posición 2 la pieza mecanizada y depositarla en la Cinta Transportadora 2 (***R122***)

Así como **ordenarle al Robot 2 que ejecute una de las dos tareas programadas**:

- Mecanizar sobre la Posición 1 (***R12***)
- Mecanizar sobre la Posición 2 (***R22***)

Ajeno al Autómata Maestro, pero directamente dependiente de este, **se permite que el usuario seleccione diferentes Modos de Fabricación y qué Herramienta Virtual Compleja se encargará de Manipular la Posición de las Piezas**, entre las distintas posiciones programadas.

Modos de Operación Fabricación:

- **Escritura de letras**: sobre la Cara A de los "Boxies"
- ***Pick and Place***: sobre la Cara B

Se permite seleccionar una Herramienta Virtual de los 3 modelos de Garra implementadas para el Robot 1, **Robot ABB IRB 140**:

- **Figure 7: Herramienta MHL02-10-SR.8 con 2 Sensores**
- **Figure 8: Herramienta MHL02-10-SR.8 con CollisionSenser**
- **Figure 9: Herramienta Rotacional con 1 Sensor**

6. 1. 1. Programa Implementado en el Autómata: Grafo de Estados

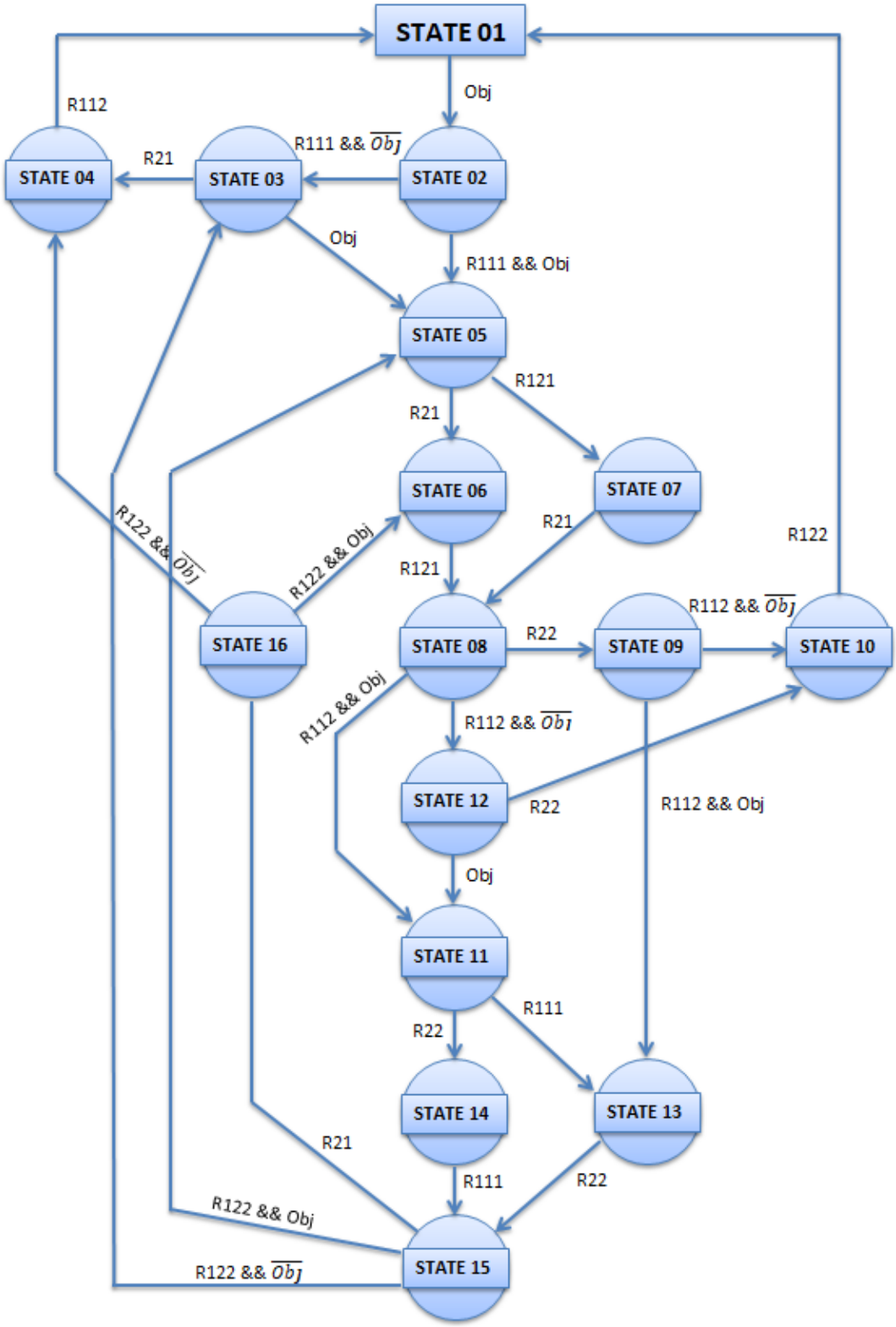


Figure 150: Grafo de Estados del PLC Master: Estación Simulada

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

Tabla 10: Programa Implementado en el Autómata: Grafo de 16 Estados con Concurrencia

ESTADO del GRAFO:	Descripción	Distribución de Posición de Trabajo	Salida del Autómata	
			Acción Robot 1	Acción Robot 2
STATE 01	Estación en Reposo	P00	-	-
STATE 02	Suministrando	P00	R111	-
STATE 03	Operando	P10	-	R21
STATE 04	Extrayendo	P20	R112	-
STATE 05	Concurrencia	P10	R121	R21
STATE 06	Suministrando	P20	R121	-
STATE 07	Operando	P11	-	R21
STATE 08	Concurrencia	P21	R112	R22
STATE 09	Extrayendo	P22	R112	-
STATE 10	Extrayendo	P02	R122	-
STATE 11	Concurrencia	P01	R111	R22
STATE 12	Operando	P01	-	R22
STATE 13	Suministrando	P02	R111	-
STATE 14	Operando	P11	-	R22
STATE 15	Concurrencia	P12	R122	R21
STATE 16	Extrayendo	P22	R122	-

Codificado la información:

- **Robot 1 "R1P_xC_y"**: **R1** (*Robot 1*) trabaja sobre $\mathbf{P_x}$ (***1**: Posición 1 ; **2**: Posición 2*) tomando como referencia $\mathbf{C_y}$ (***1**: Suministrar/Coger pieza de la Cinta Transportadora 1 ; **2**: Extraerla pieza mecanizada y depositarla en la Cinta Transportadora 2*).
- **Robot 2 "R2P_x"**: **R2** (*Robot 2*) trabaja sobre $\mathbf{P_x}$ (***1**: Posición 1 ; **2**: Posición 2*).
- **Posición "Pxy"**: **x** es la Posición 1 e **y** es la Posición 2, para ambos casos: (***0**: Posición vacía ; **1**: Posición ocupada y siendo manipulada ; **2**: Posición Ocupada y pendiente de Extracción de pieza mecanizada*).
- **Existencia de Objeto en Cinta Transportadora 1**:
  - **Obj**: Variable que indica la existencia de pieza en la Cinta Transportadora 1, para que sea transportada por el Robot 1 hasta la $\mathbf{P_x}$.
  - $\overline{\boldsymbol{Obj}}$: No hay piezas para producir.

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

6. 1. 2. PLC's con Controlador IRC-5: Implementado en RAPID

El **Controlador IRC-5 Virtual** se encarga de comandar y coordinar todos los elementos que componen la Estación Simulada, respeta la **siguiente secuencia**:

1. Inicialización de todos los componentes de la Estación
    1.1 Puesta en marcha de cintas transportadoras
    1.2 Posicionamiento los objetos "*Boxies*
2. Ejecutar el Programa Principal.

El Programa Principal, ejecutivo cíclico o "*Loop*", se ha implementado con la metodología de **2 *Swich-Case*,** uno para actualizar el Estado y otro para actualizar las salidas. El código **se resuelve en dos tiempos**:

1. **Actualiza el Estado del Autómata**, con la primera estructura "case", chequeando la variable de "Estado del Autómata" actual y las condiciones de transición asociadas al Estado actual (Configuración de Entradas)
2. **Actualiza la configuración de Salidas** en función del Estado actual/activo, con un segundo "case" con la variable de "Estado del Autómata" encadenada al primero.

La **activación de salidas** se ha implementado **por nivel**, responde al problema de la gestión de E/S de RobotStudio. *Si se presenciasen "fallos" en la comunicación de E/S, se aconseja forzar sincronismos por código RAPID (inclusión de "Delays" con duración superior al valor de un "**20**" e inferior a dos) al final del programa, perdiendo velocidad de procesamiento a cambio de garantizar la comunicación entre los elementos de la Estación.*

Se puede ver un vídeo del funcionamiento en: **Estación_con_PLC_Controlador_IRC-5_Virtual**

6. 2. Código Completo del PLC Maestro: "*MODICON: M 340*"

T_ROB1/Module1

```
1   MODULE Module1
2       !*********************************************************
3       !
4       ! Module:  Module1
5       !
6       ! Description:
7       !   PLC Master: Comanda Entorno y 2 Robots: R1 (Ejecutor de tareas) : R2 (Posicionamiento)
8       !       Concurrent System:
9       !       Nomenclatura:
10      !           Robot Positioner: Rxyz [x: nºRobot(1)// y: nº Position(1/2)// z: nº conveyor (1:input 2:output)]
11      !           Robot Worker: Rxy [x: nºRobot(2)// y: nº Position at work(1/2)]
12      !           Position Pallet: Pvw [v: State position 01// w:State position 01 //; RangeVariable(0:null//1::Ocupped//2:Ocupped and Completed)]
13      ! Author: Octavio Augusto Ansón Clemente
14      !
15      ! Version: 2.0.puf
16      !
17      !*********************************************************
18
19
20      !*********************************************************
21      !
22      ! Procedure main
23      !
24      !   This is the entry point of PLC Master program
25      !
26      !*********************************************************
27
28      !Main Program: Used bucle:
29      PROC main()
30          !Inicializacion desde HomePosition
31          Initialitation;
32          !GoHomeBoxy;
33          MoveConveyorInput;
34          MoveConveyorOutput;
35          !InitVariable:
36          Cycle:=0;
37
38          !ProductionControl: Number of ended box
39          EndedBox:=0;
40
41          WaitDI Running_SmartO_PLC,0;
42
43          !Control Time-Cycle
44          Time:=0;
45          ClkReset clock2;
46          ClkStart clock2;
47          !Reset Time Variable
48
49          !Well, we will to start My Obedient Slaves!!: Prub
50          WHILE Running_SmartO_PLC=0 DO
51              !EndedBox<100: Production Control           !Cycle<150000: Medir Tc            !ClkRead(clock2)<1500: Medir Tc
52
53              !PLC State Actualiatation:
54              TEST STATE
55              CASE 1:
56                  !Wait State: Sleep PLC: P00
57                  IF SignalPositionInputConveyor=1 THEN
58                      STATE:=2;
59                  ENDIF
60              CASE 2:
61                  !R111:P00:
62                  IF (ACKRobotSystem2=1 and SignalPositionInputConveyor=0) THEN
63                      STATE:=3;
64                  ELSEIF (ACKRobotSystem2=1 and SignalPositionInputConveyor=1) THEN
65                      STATE:=5;
66                  ENDIF
67              CASE 3:
68                  !R21:P10:
69                  IF ACKRobotSystem1=1 THEN
70                      STATE:=4;
71                  ELSEIF SignalPositionInputConveyor=1 THEN
72                      STATE:=5;
73                  ENDIF
74              CASE 4:
75                  !R112:P20:
76                  IF ACKRobotSystem2_3=1 THEN
77                      STATE:=1;
78                      EndedBox:=EndedBox+1;
79                  ENDIF
```

```
80              CASE 5:
81                  !R121:R21:P10:
82                  IF ACKRobotSystem2_2=1 THEN
83                      STATE:=7;
84                  ELSEIF ACKRobotSystem1=1 THEN
85                      STATE:=6;
86                  ENDIF
87              CASE 6:
88                  !R121:P20:
89                  IF ACKRobotSystem2_2=1 THEN
90                      STATE:=8;
91                  ENDIF
92              CASE 7:
93                  !R21:P11:
94                  IF ACKRobotSystem1=1 THEN
95                      STATE:=8;
96                  ENDIF
97              CASE 8:
98                  !R112:R22:P21:
99                  IF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=0) THEN
100                     STATE:=12;
101                     EndedBox:=EndedBox+1;
102                 ELSEIF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=1) THEN
103                     STATE:=11;
104                     EndedBox:=EndedBox+1;
105                 ELSEIF ACKRobotSystem1_2=1 THEN
106                     STATE:=9;
107                 ENDIF
108             CASE 9:
109                 !R112:P22:
110                 IF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=0) THEN
111                     STATE:=10;
112                     EndedBox:=EndedBox+1;
113                 ELSEIF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=1) THEN
114                     STATE:=13;
115                 ENDIF
116             CASE 10:
117                 !R122:P02:
118                 IF ACKRobotSystem2_4=1 THEN
119                     STATE:=1;
120                     EndedBox:=EndedBox+1;
121                 ENDIF
122             CASE 11:
123                 !R111:R22:P01:
124                 IF ACKRobotSystem2=1 THEN
125                     STATE:=14;
126                 ELSEIF ACKRobotSystem1_2=1 THEN
127                     STATE:=13;
128                 ENDIF
129             CASE 12:
130                 !R22:P01:
131                 IF SignalPositionInputConveyor=1 THEN
132                     STATE:=11;
133                 ELSEIF ACKRobotSystem1_2=1 THEN
134                     STATE:=10;
135                 ENDIF
136             CASE 13:
137                 !R111:P02:
138                 IF ACKRobotSystem2=1 THEN
139                     STATE:=15;
140                 ENDIF
```

```
141             CASE 14:
142                 !R22:P11:
143                 IF ACKRobotSystem1_2=1 THEN
144                     STATE:=15;
145                 ENDIF
146             CASE 15:
147                 !R122:R21:P12:
148                 IF ACKRobotSystem1=1 THEN
149                     STATE:=16;
150                 ELSEIF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=0) THEN
151                     STATE:=3;
152                     EndedBox:=EndedBox+1;
153                 ELSEIF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=1) THEN
154                     STATE:=5;
155                     EndedBox:=EndedBox+1;
156                 ENDIF
157             CASE 16:
158                 !R122:R21:P12:
159                 IF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=0) THEN
160                     STATE:=4;
161                     EndedBox:=EndedBox+1;
162                 ELSEIF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=1) THEN
163                     STATE:=6;
164                     EndedBox:=EndedBox+1;
165                 ENDIF
166             ENDTEST
167
168             !Output of PLC Actualitation:
169             TEST STATE
170             CASE 1:
171                 Reset OrderRobotSystem1_R21;
172                 Reset OrderRobotSystem1_R22;
173                 Reset OrderRobotSystem2_R111;
174                 Reset OrderRobotSystem2_R112;
175                 Reset OrderRobotSystem2_R121;
176                 Reset OrderRobotSystem2_R122;
177             CASE 2:
178                 !R111:P00:
179                 Reset OrderRobotSystem1_R21;
180                 Reset OrderRobotSystem1_R22;
181                 SetDO OrderRobotSystem2_R111,1;
182                 Reset OrderRobotSystem2_R112;
183                 Reset OrderRobotSystem2_R121;
184                 Reset OrderRobotSystem2_R122;
185             CASE 3:
186                 !R21:P10:
187                 SEtDO OrderRobotSystem1_R21,1;
188                 Reset OrderRobotSystem1_R22;
189                 Reset OrderRobotSystem2_R111;
190                 Reset OrderRobotSystem2_R112;
191                 Reset OrderRobotSystem2_R121;
192                 Reset OrderRobotSystem2_R122;
193             CASE 4:
194                 !R112:P10:
195                 Reset OrderRobotSystem1_R21;
196                 Reset OrderRobotSystem1_R22;
197                 Reset OrderRobotSystem2_R111;
198                 SetDO OrderRobotSystem2_R112,1;
199                 Reset OrderRobotSystem2_R121;
200                 Reset OrderRobotSystem2_R122;
```

```
201             CASE 5:
202                 !R121:R21:P10:
203                 SetDO OrderRobotSystem1_R21,1;
204                 Reset OrderRobotSystem1_R22;
205                 Reset OrderRobotSystem2_R111;
206                 Reset OrderRobotSystem2_R112;
207                 SetDO OrderRobotSystem2_R121,1;
208                 Reset OrderRobotSystem2_R122;
209             CASE 6:
210                 !R121:P20:
211                 Reset OrderRobotSystem1_R21;
212                 Reset OrderRobotSystem1_R22;
213                 Reset OrderRobotSystem2_R111;
214                 Reset OrderRobotSystem2_R112;
215                 SetDO OrderRobotSystem2_R121,1;
216                 Reset OrderRobotSystem2_R122;
217             CASE 7:
218                 !R21:P11:
219                 SetDO OrderRobotSystem1_R21,1;
220                 Reset OrderRobotSystem1_R22;
221                 Reset OrderRobotSystem2_R111;
222                 Reset OrderRobotSystem2_R112;
223                 Reset OrderRobotSystem2_R121;
224                 Reset OrderRobotSystem2_R122;
225             CASE 8:
226                 !R112:R22:P21:
227                 Reset OrderRobotSystem1_R21;
228                 SetDO OrderRobotSystem1_R22,1;
229                 Reset OrderRobotSystem2_R111;
230                 SetDO OrderRobotSystem2_R112,1;
231                 Reset OrderRobotSystem2_R121;
232                 Reset OrderRobotSystem2_R122;
233             CASE 9:
234                 !R112:P22:
235                 Reset OrderRobotSystem1_R21;
236                 Reset OrderRobotSystem1_R22;
237                 Reset OrderRobotSystem2_R111;
238                 SetDO OrderRobotSystem2_R112,1;
239                 Reset OrderRobotSystem2_R121;
240                 Reset OrderRobotSystem2_R122;
241             CASE 10:
242                 !R122:P02:
243                 Reset OrderRobotSystem1_R21;
244                 Reset OrderRobotSystem1_R22;
245                 Reset OrderRobotSystem2_R111;
246                 Reset OrderRobotSystem2_R112;
247                 Reset OrderRobotSystem2_R121;
248                 SetDO OrderRobotSystem2_R122,1;
249             CASE 11:
250                 !R111:R22:P01:
251                 Reset OrderRobotSystem1_R21;
252                 SetDO OrderRobotSystem1_R22,1;
253                 SetDO OrderRobotSystem2_R111,1;
254                 Reset OrderRobotSystem2_R112;
255                 Reset OrderRobotSystem2_R121;
256                 Reset OrderRobotSystem2_R122;
```

```
257             CASE 12:
258                 !R22:P01:
259                 Reset OrderRobotSystem1_R21;
260                 SetDO OrderRobotSystem1_R22,1;
261                 Reset OrderRobotSystem2_R111;
262                 Reset OrderRobotSystem2_R112;
263                 Reset OrderRobotSystem2_R121;
264                 Reset OrderRobotSystem2_R122;
265             CASE 13:
266                 !R111:P02:
267                 Reset OrderRobotSystem1_R21;
268                 Reset OrderRobotSystem1_R22;
269                 SetDO OrderRobotSystem2_R111,1;
270                 Reset OrderRobotSystem2_R112;
271                 Reset OrderRobotSystem2_R121;
272                 Reset OrderRobotSystem2_R122;
273             CASE 14:
274                 !R22:P11:
275                 Reset OrderRobotSystem1_R21;
276                 SetDO OrderRobotSystem1_R22,1;
277                 Reset OrderRobotSystem2_R111;
278                 Reset OrderRobotSystem2_R112;
279                 Reset OrderRobotSystem2_R121;
280                 Reset OrderRobotSystem2_R122;
281             CASE 15:
282                 !R122:R21:P12:
283                 SetDO OrderRobotSystem1_R21,1;
284                 Reset OrderRobotSystem1_R22;
285                 Reset OrderRobotSystem2_R111;
286                 Reset OrderRobotSystem2_R112;
287                 Reset OrderRobotSystem2_R121;
288                 SetDO OrderRobotSystem2_R122,1;
289             CASE 16:
290                 !R122:P22:
291                 Reset OrderRobotSystem1_R21;
292                 Reset OrderRobotSystem1_R22;
293                 Reset OrderRobotSystem2_R111;
294                 Reset OrderRobotSystem2_R112;
295                 Reset OrderRobotSystem2_R121;
296                 SetDO OrderRobotSystem2_R122,1;
297             ENDTEST
298             SetAO CurrentState,STATE;
299             !Sincronitation With other Systems:
300             Cycle:=Cycle+1;
301             !WaitTime 0.0001;
302             SetAO CurrentCycle,Cycle;
303             !Unsigned Range(0:1000000)
304         ENDWHILE
305
306         !Measure: Wather MidleTime of Cicle of RobotSystem
307         ClkStop clock2;
308         Time:=ClkRead(clock2);
309         Tc:=Time/Cycle;
310         SetAO TimeCycle,Time;
311         Time:=Time/Cycle;
312
```


Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

```
313     !Signed Range(-100;100): [Initial Solve: Tc1=0.0010039] [Tc2=0.00155835] [Tc(10000)=2.1751(t=21.8)] [Tc(30000)=0.00200903(60.3)] [Tc(1000000)=0.0019317] Look at "user" Module
314     !With time1=100seg: 53781,000 Cycles: Tc1= 0.001859393        !With time2=300seg: 154364 Cycles: Tc2= 0.00194346        !With time2=600seg: 301416 Cycles: Tc2= 0.0019906
315     !With time2=900seg: 450777 Cycles: Tc2=0.00199655        !With time2=1500seg: 746042 Cycles: Tc2=0.00201061
316
```

```
318     ENDPROC
319
320
321
322
323
324     !!Imlementation: Functions/Methodes/Procces:
325
326     LOCAL PROC Initialitation()
327         Reset PositionerBoxInConveyor;
328         Reset MoveInputConveyor;
329         Reset MoveOutputConveyor;
330         !STATE:=1; !If the Station run as first simulation activate this line...
331         Reset RecibedOrderRobotSystem1;
332         Reset RecibedOrderRobotSystem2;
333         Reset OrderRobotSystem1_R21;
334         Reset OrderRobotSystem1_R22;
335         Reset OrderRobotSystem2_R111;
336         Reset OrderRobotSystem2_R112;
337         Reset OrderRobotSystem2_R121;
338         Reset OrderRobotSystem2_R122;
339     ENDPROC
340
341     ! Entorno a las instalaciones principales:
342
343     LOCAL PROC MoveConveyorInput()
344         Reset MoveInputConveyor;
345         SetDO MoveInputConveyor,1;
346     ENDPROC
347
348     LOCAL PROC MoveConveyorOutput()
349         Reset MoveOutputConveyor;
350         SetDO MoveOutputConveyor,1;
351     ENDPROC
352
353
354     !Funciones Secundarias:
355     LOCAL PROC GoHomeBoxy()
356         SetDO PositionerBoxInConveyor,1;
357         Reset PositionerBoxInConveyor;
358     ENDPROC
359
360
361 ENDMODULE
```


Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

6. 3. Código Completo del PLC Maestro: SmartObject

B) *Esquema General del SmartObject*

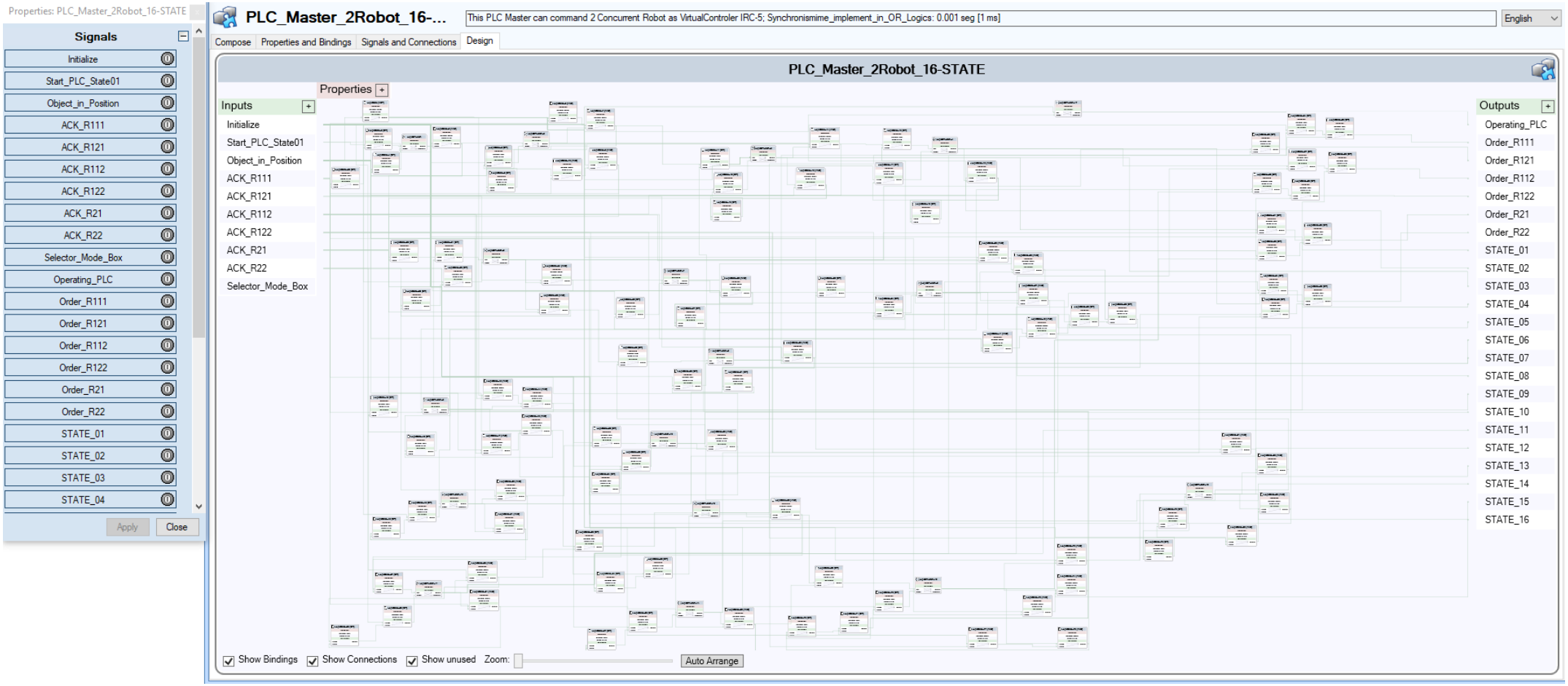


Figure 151: Esquemático del PLC Master 16 Estados: SmartObject

A) *Esquema Desglosado*

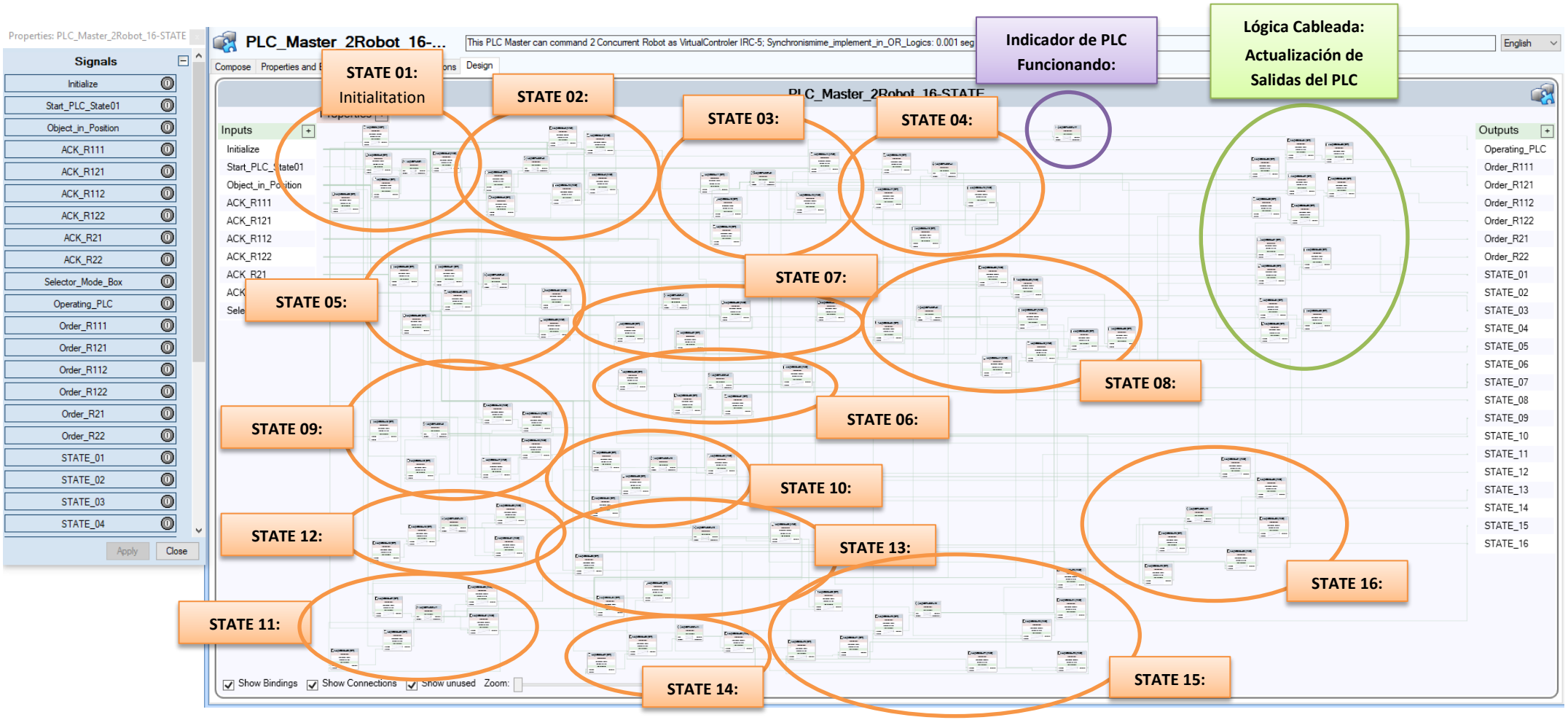


Figure 152: Identificación de las partes del Esquemático del PLC Master 16 Estados: SmartObject

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

B) *Esquemático resultante del Grafo de Estados implementado:*

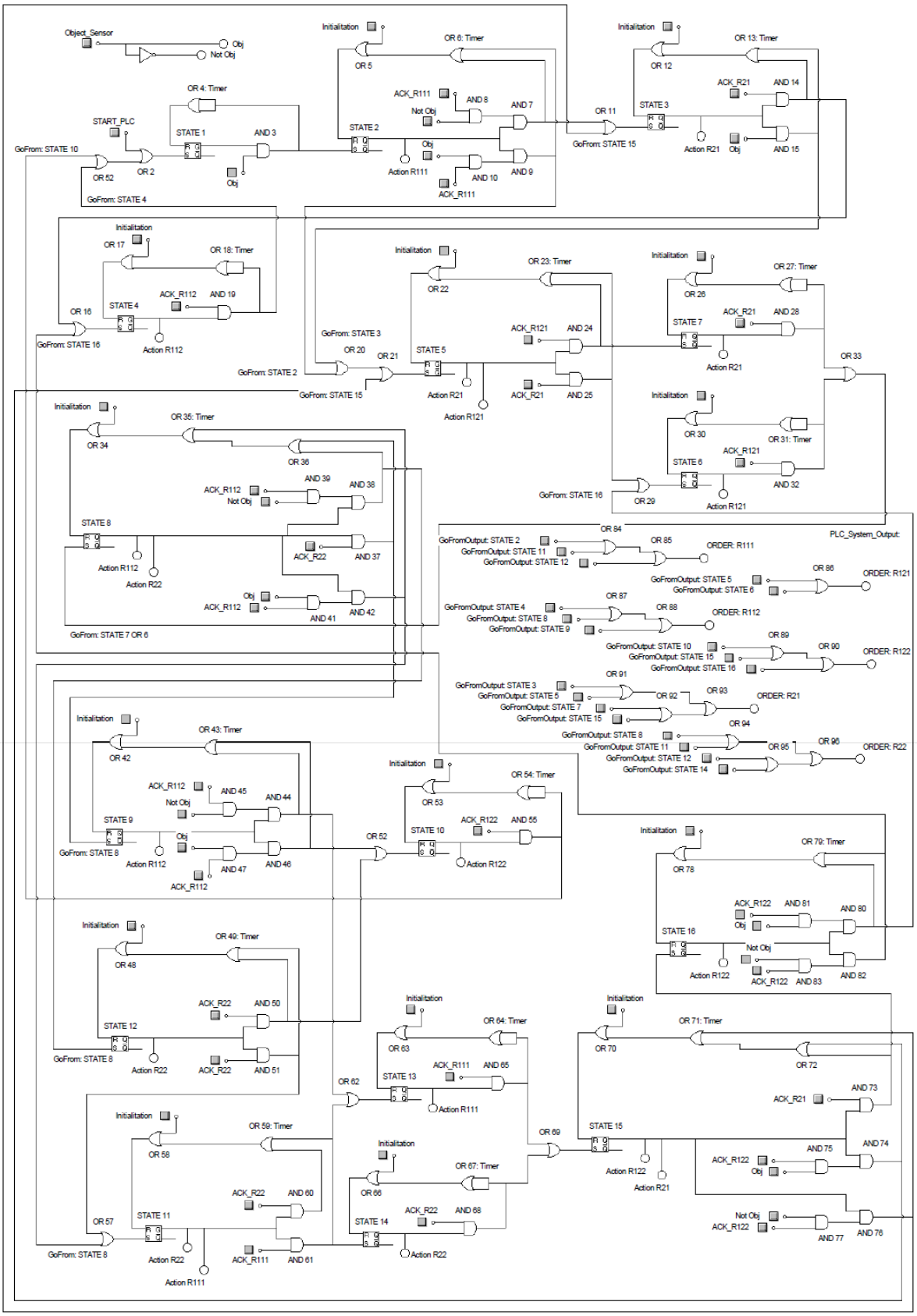


Figure 153: Esquemático del PLC Master 16 Estados: Flip-Flop del SmartObject

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

C) *Esquemático Desglosado*

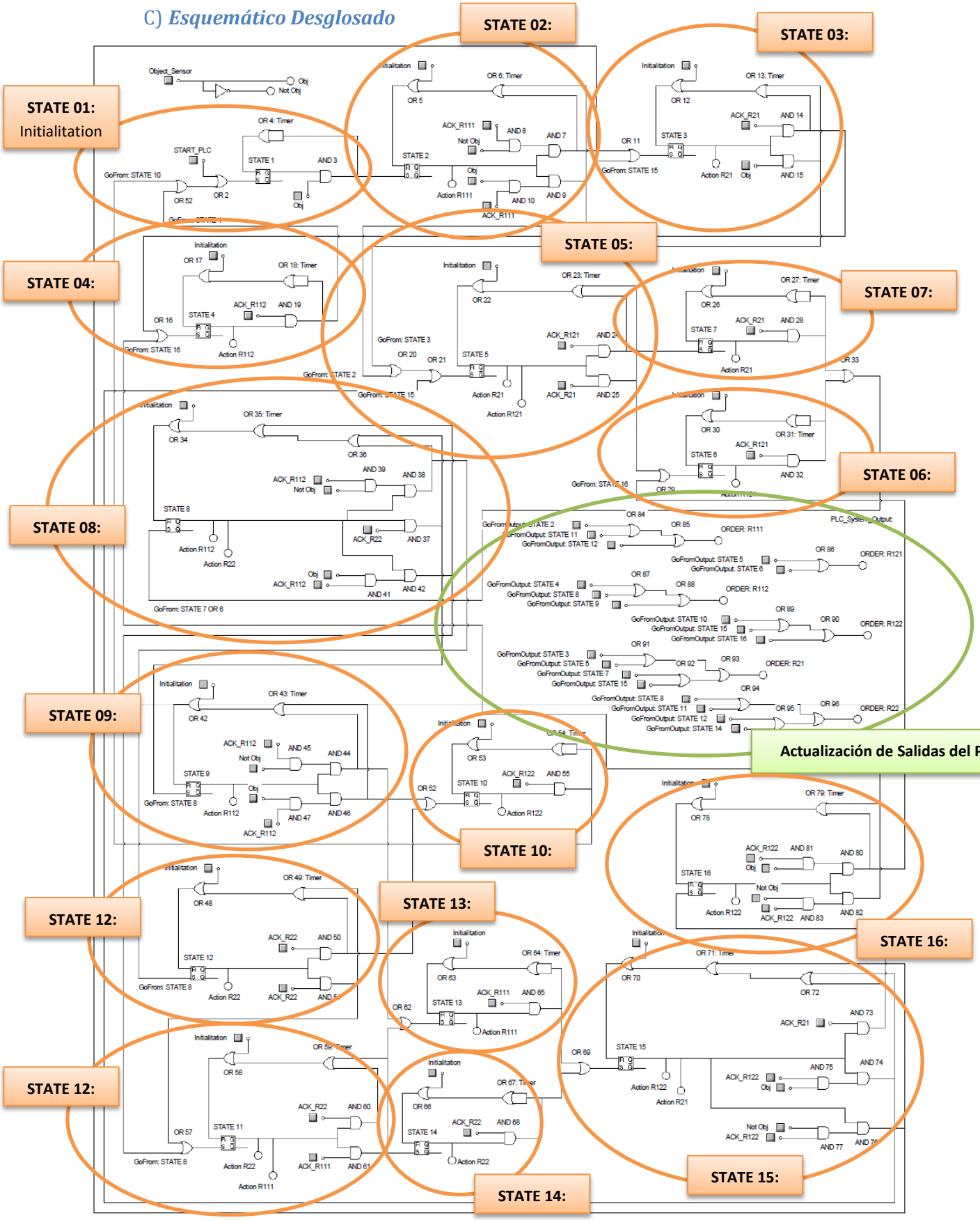


Figure 154: Identificación Partes Esquemático Flip-Flop del SmartObject

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

ANEXO VII

7. La Estación Simulada: *"Célula Flexible 8"*

7. 1. Introducción a la Estación Simulada: "*Célula Flexible 8*"

Se ha concebido la **Estación como una síntesis de resultados del análisis de cada punto descrito** en el "**ANEXO II**" para la implementación de una Estación Completa y con el fin de exponer las diferencias entre los distintos procedimientos a la hora de abordar un mismo problema.

Se ha trabajado sobre un caso genérico, una **célula de trabajo industrial compuesta por**: 2 Robots distintos encargados de la manipulación y mecanizado de piezas, 2 Cintas transportadoras ("*Conveyors*"), encargadas de proveer material y extraerlo, un Autómata Maestro ("PLC") encargado de comandar los distintos elementos y ejecutar el programa principal, y escenografía básica. Como se puede ver en la **Figure 155** (*y vídeo*) y en la **Tabla 11**.

Para ello, **se han diseñado e implementado los elementos que componen la Estación**, incluido el diseño de una herramienta virtual similar a la herramienta real del Robot ABB IRB 120 ubicado en el laboratorio DIIS L0.06, propiedad del departamento de Informática e Ingeniería de Sistemas de la EINA-Unizar.

Adicionalmente se ha previsto la interacción del usuario con la Estación, permitiéndole decidir qué elementos participan en la simulación y el modo de operación, por medio de un Display.

7. 2. Equipamiento de la "Célula Flexible 8"

La Estación consta de un PLC Maestro, 2 Robots, 2 Cintas transportadoras ("*Conveyors*"), un almacén inteligente con reposicionamiento de objetos, varios elementos manipulables ("*Boxies*") y escenografía básica.

- **PLC Maestro:** encargado de ejecutar el programa principal y lograr que todo interactúe en armonía. *Sin Representación Gráfica.*
- **Cintas transportadoras (*"Conveyors"*):** elementos encargados de abastecer y sacar de la célula de trabajo el material manipulado por los Robots.
- **Robot IRB 140** dotado con una **Herramienta compleja: (*seleccionar una entre tres opciones*):** se encarga de transportar y posicionar las piezas que van a ser manipuladas, en función de la operación que haya demandado el usuario.
- **Robot IRB 120** dotado con una **herramienta simple con doble operación**: se encarga de simular las operaciones de mecanizado y/o manipulación de los objetos:
    - "*Escritura de letras O-C-T*": respetando una secuencia específica o escribiendo letras sueltas sobre la Cara A de los "*Boxies*"
    - "*Pick and Place*": sobre la Cara B.

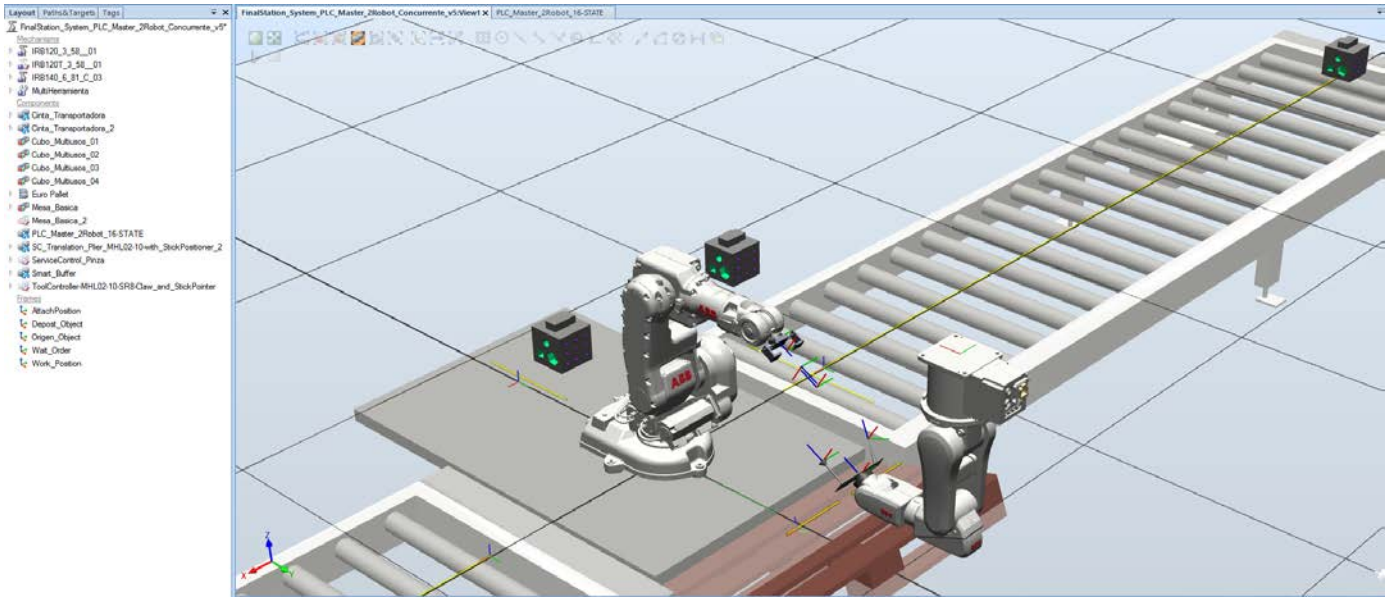


Figure 155: Simulación Célula Industrial Completa (*y Enlace a un Vídeo: Ctrl+click izquierdo*)

Resumen de los principales elementos de la Estación Virtual Final

Tabla 11: Elementos que componen la Estación Simulada: Célula 8

Elemento Principal	Complemento Auxiliar	El Usuario puede Seleccionar las siguientes Modalidades en Simulación
PLC Maestro	Programa Principal	**Ejecución por SmartObject**
		Ejecución por Controlador IRC-5 Virtual
Modo Operación	Programa Principal	**Escritura de letras: sobre "Cara A" del Cubo**
		Pick and Place: sobre "Cara B" del Cubo
Robot: IRB 120 (Figure 157)	Herramienta doble Operación (**Figure 156**)	-
	Programa Principal	-
Robot IRB 140 (Figure 158)	Herramienta Principal	**Herramienta MHL02-10-SR.8 con 2 Sensores (Figure 161)**
		Herramienta MHL02-10-SR.8 con CollisionSenser (Figure 159)
		Herramienta Rotacional con 1 Sensor (Figure 160)
	Programa Principal	-
Conveyor Aprovisionamiento (Figure 162)	Lógica Cableada	-
Conveyor Extracción (Figure 163)	Lógica Cableada	-
Almacén Inteligente	Lógica Cableada	**Posicionamiento "Boxies" n1:n4**
Cajas Manipulables "Boxies" (Figure 164)	-	-
Escenografía Varia	Mesas, Pallets,…	-

Figure 156: Herramienta Doble Operación

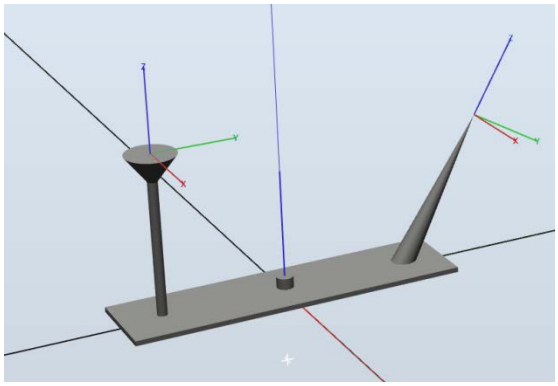


Figure 157: Robot ABB IRB 120

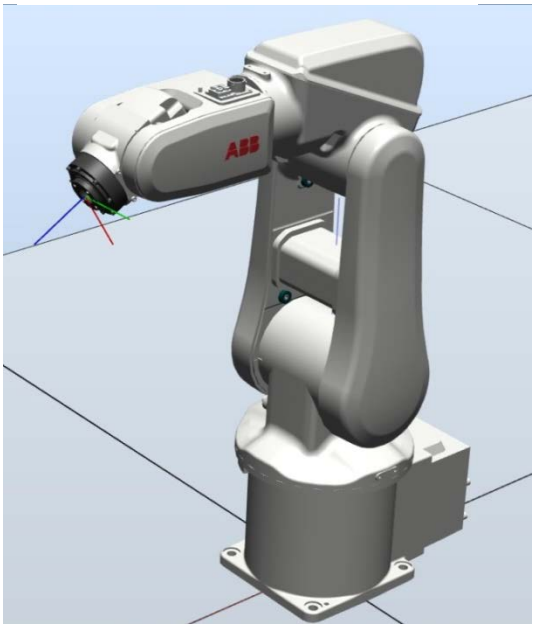


Figure 158: Robot ABB IRB 140

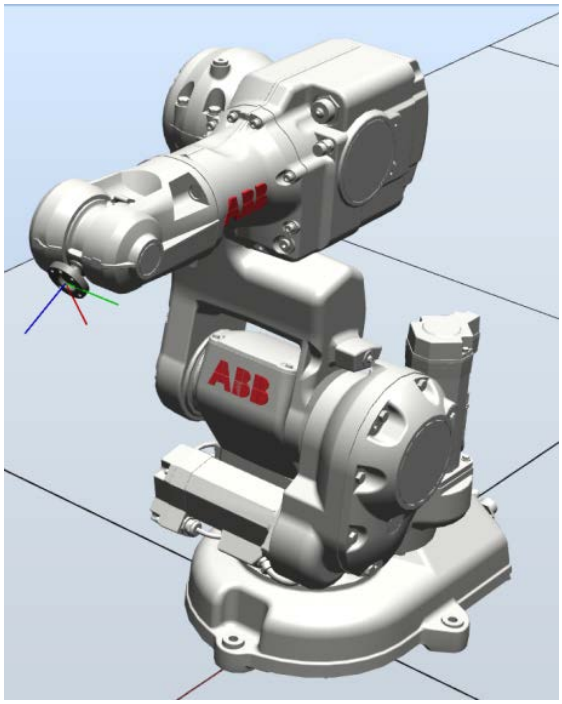


Figure 161: Herramienta MHL02-10-SR.8 con 2 Sensores

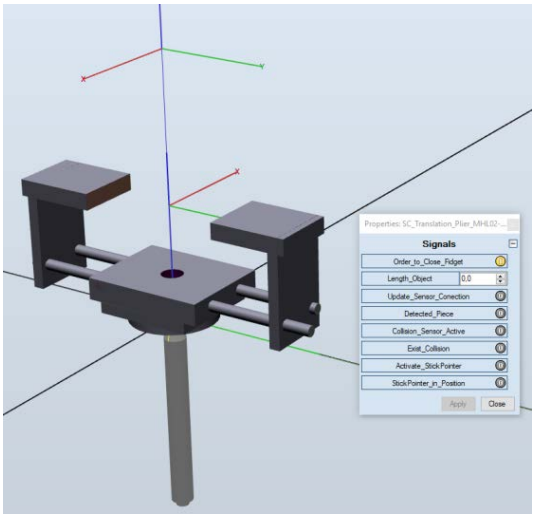


Figure 159: Herramienta MHL02-10-SR.8 con CollisionSenser

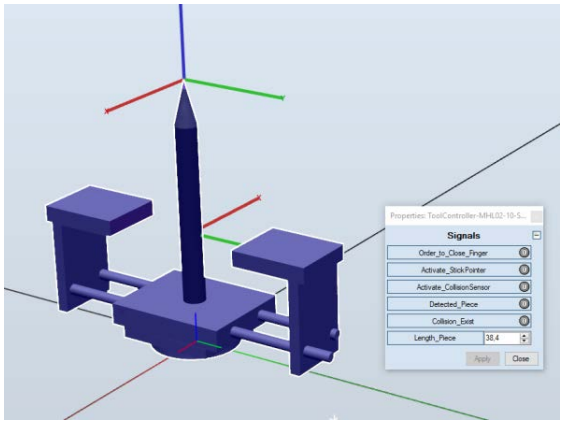
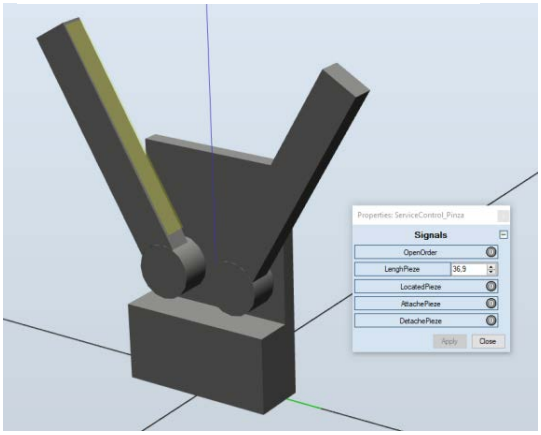


Figure 160: Herramienta Rotacional con 1



Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

Figure 162: Conveyor de Aprovisionamiento

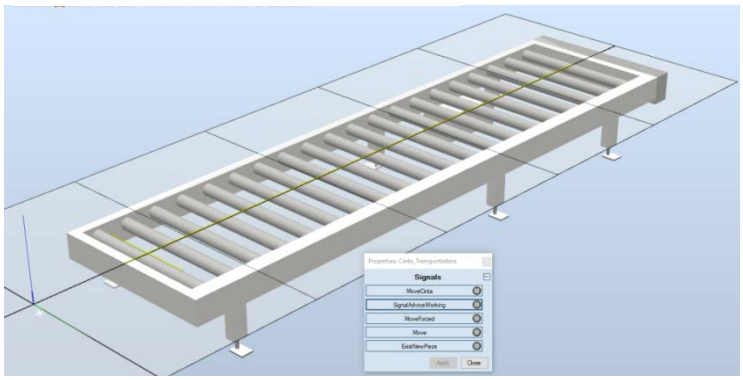


Figure 163: Conveyor de Extracción

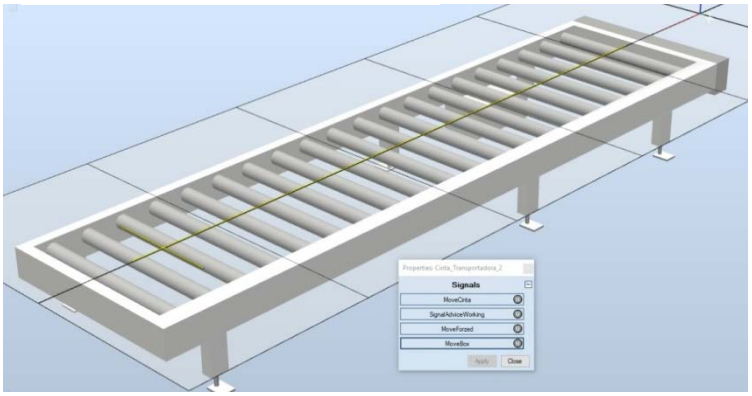
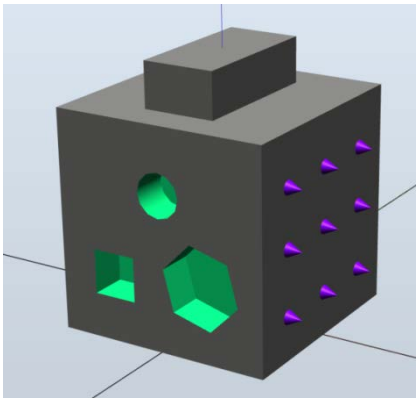


Figure 164: Cajas Manipulables "Boxies"



A) *Objeto de Trabajo de la Estación:*

El objeto sobre el que se trabaja en las simulaciones es el representado en la **Figure 164**, un Cubo multiusos compuesto de un asidero, para ser agarrado, y dotado de:

- Una cara (*Cara A*) habilitada con punteros (coloreadas en morado) sobre los que poder trazar Letras y/o Números (simulando un Display 7 segmentos)
- Una cara (*Cara B*) con distintas figuras geométricas (sombreadas en verde) para poder ser manipuladas con un "Pick and Place"

B) *Cintas Transportadoras "Convellor"*

Las Cintas Transportadoras son Objetos Inteligentes, implementados con un *SmartObjects*. Cuyas principales propiedades son:

- **Movimiento de Objetos**: Pueden mover un único objeto simultáneamente a lo largo del rail representado con un sensor lineal (*en amarillo*)
- **Identificar la Posición de Objetos**: Para informar al autómata de la existencia/inexistencia de un objeto nuevo para producir ("*Conveyor* 1": Abastecimiento)/abandonar el sistema ("*Conveyor* 2": Extracción).

Se ha configurado/programado de forma distinta cada una de las Cintas Transportadoras:

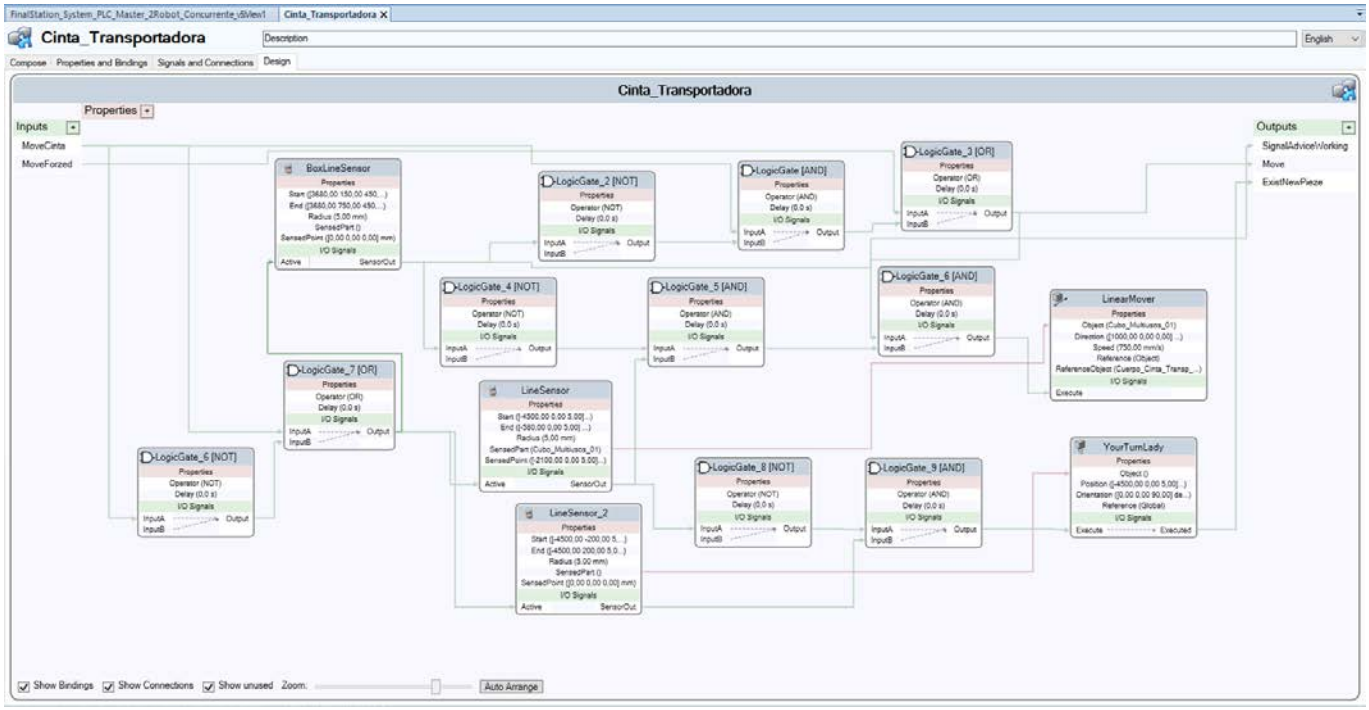


Figure 165: Implementación Lógica: *Conveyor* de Abastecimiento

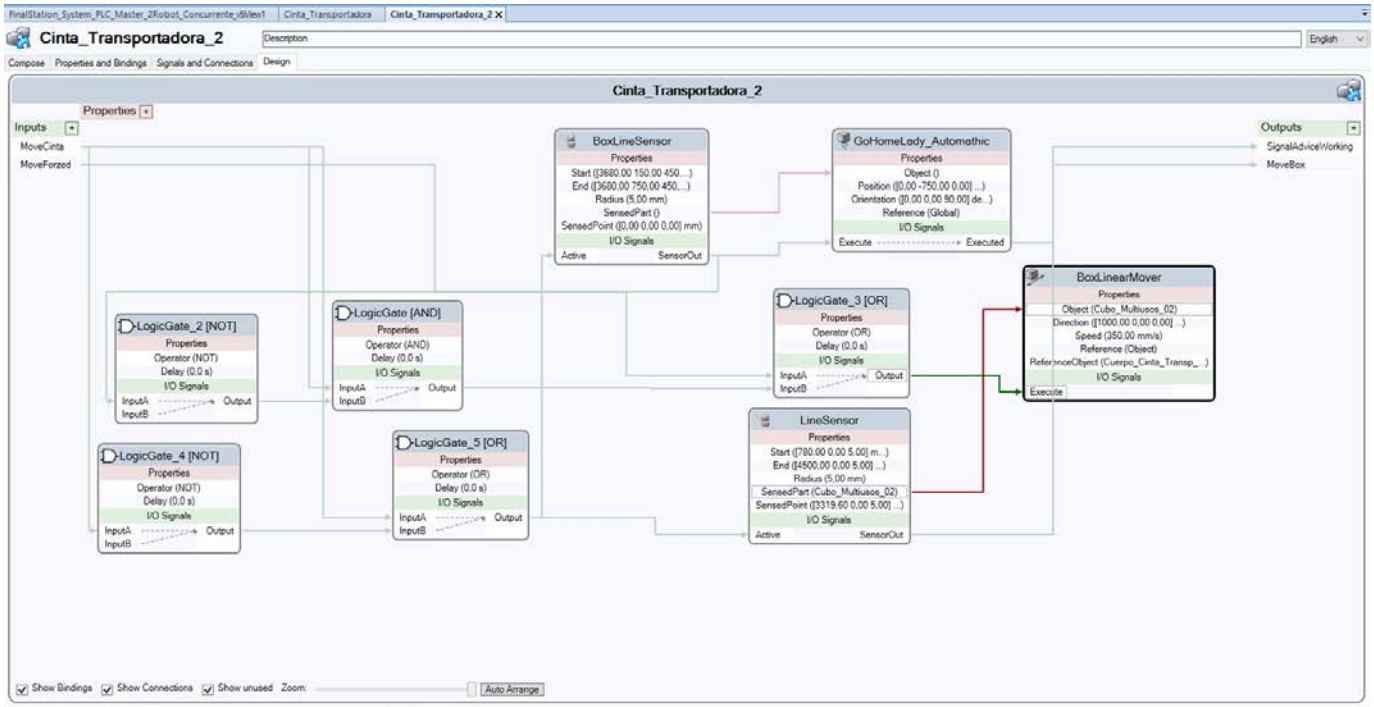


Figure 166: Implementación Lógica: Conveyor de Extracción

Titular: Octavio Augusto Ansón Clemente
Director: Antonio Romeo Tello

C) *Almacen Inteligente "SmartBuffer"*

Elemento encargado de reposicionar los "*Boxies*" en el Sistema, y sea de forma automática o permitiéndole el control al usuario, e identificar la existencia "*Boxies*" en una de las posiciones de fabricación (*PosA o PosB*).

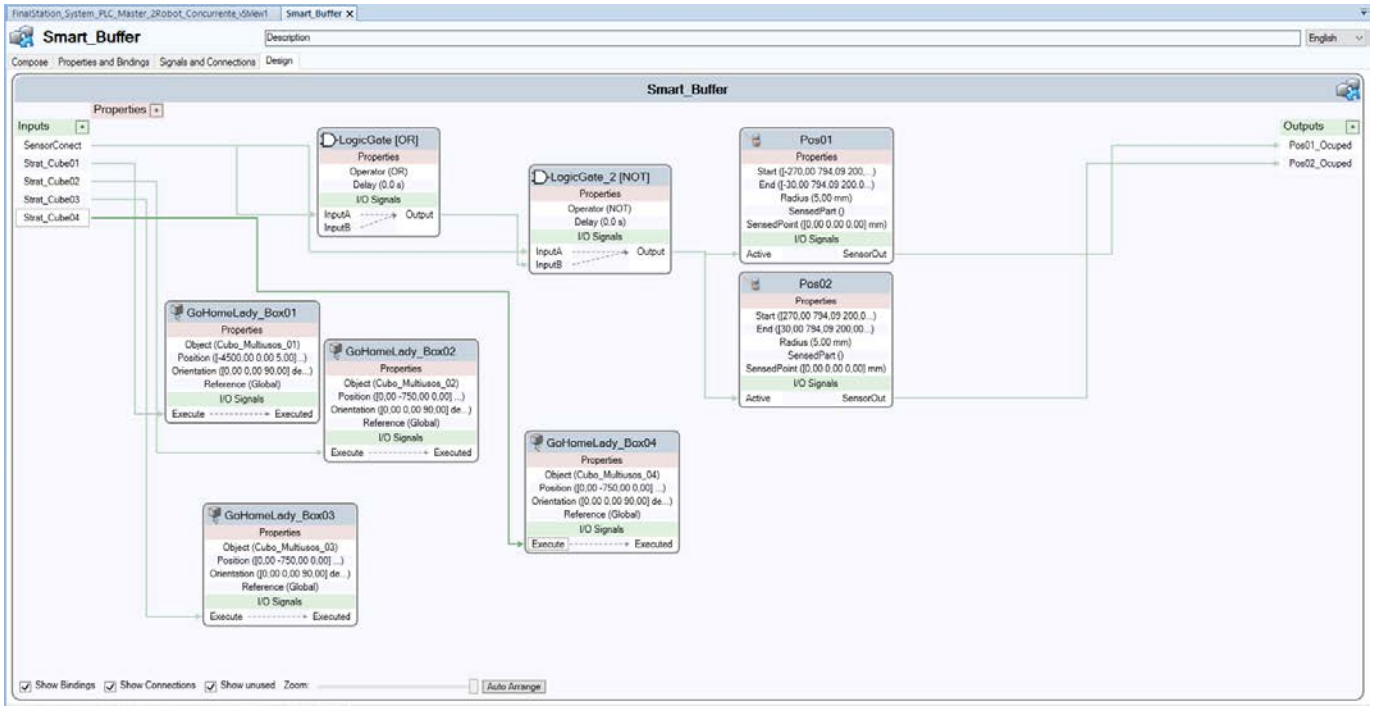


Figure 167: Implementación Lógica: SmartBuffer

D) *Herramienta Virtual "SR.8"*

Se ha diseñado el control lógico de la Herramienta por dos procedimientos distintos:

a. **Control lógico con Sensores** (ejemplo expuesto a continuación):
   a. Sin restricciones de diseño.
   b. El vástago posee 1 Grado de Libertad (desplazamiento traslacional, se esconde en el interior del robot virtual).

b. **Control lógico con Detector de Colisiones**:
   a. Los sólidos que componen la Herramienta NO pueden tocarse ni en situación estática ni en las trayectorias de desplazamiento.
   b. El vástago NO posee Grado de Libertad, se oculta y se visualiza en la misma posición física relativa a la base.

Para ver la Implementación de la Herramienta Virtual "SR.8" ver **ANEXO V**.

7. 3. Jerarquía de Control: "*Célula Flexible 8*"

Contemplando exclusivamente la Estación, **el PLC Maestro es el elemento de mayor rango**. Pudiendo ser alterado, tanto en modos de operación como en la herramienta del Robot IRB 140, por la acción directa de los usuarios a través de la activación de bits selectores.

Por debajo del PLC Maestro, se encuentran las cintas transportadoras y los Robots, seguidos estos de sus herramientas.

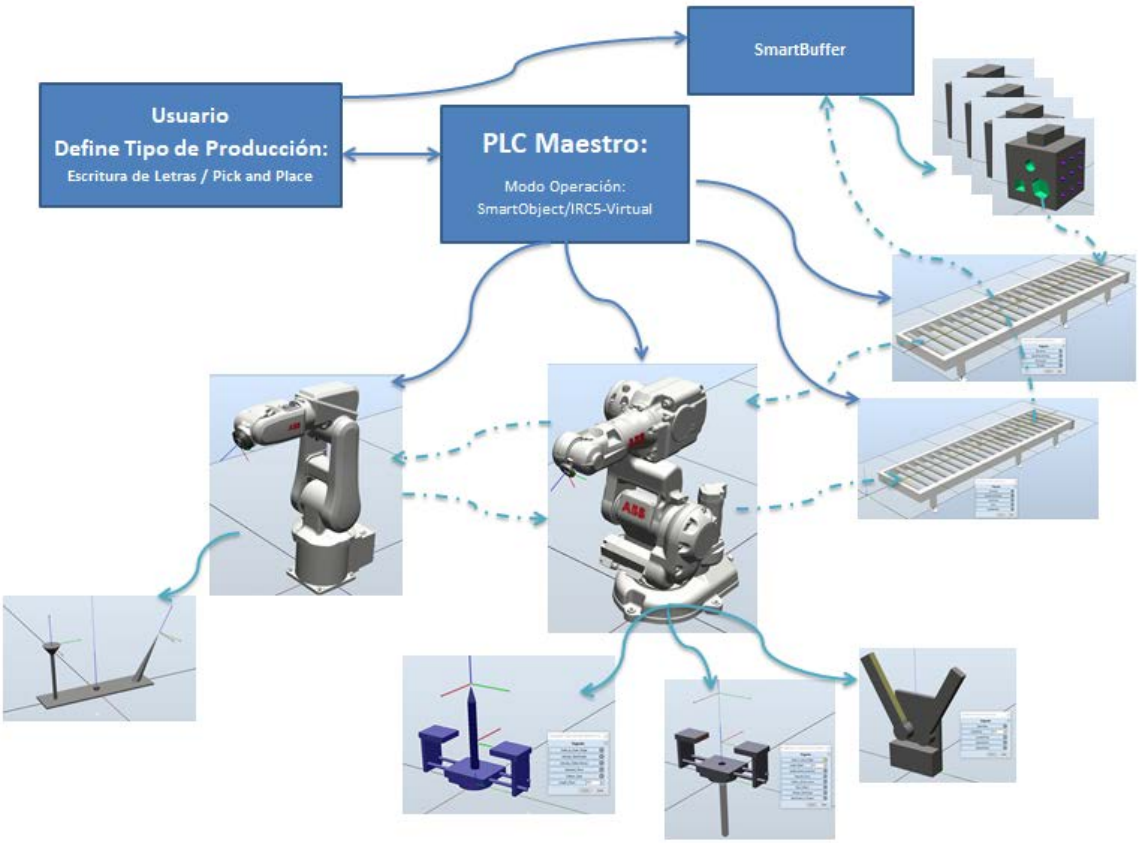


Figure 168: Jerarquía de Control y Relaciones entre componentes de la Estación Simulada

En los siguientes enlaces **se pueden ver distintos vídeos del funcionamiento de la Estación**, tanto operando con el PLC en modalidad SmartObject como en modalidad Líneas de Código con un IRC5-Virtual:

- **CompleteStation-IRC5_PLC_Master**
- **CompleteStation-SmartObject_PLC_Master**
- **CompleteStation-SmartObject_PLC_Master_PickAndPlace**

7. 3. 1. Configuración de E/S de los Controladores

A) *Configuración del Controlador Virtual IRC 5 encargado de las funciones PLC Maestro*

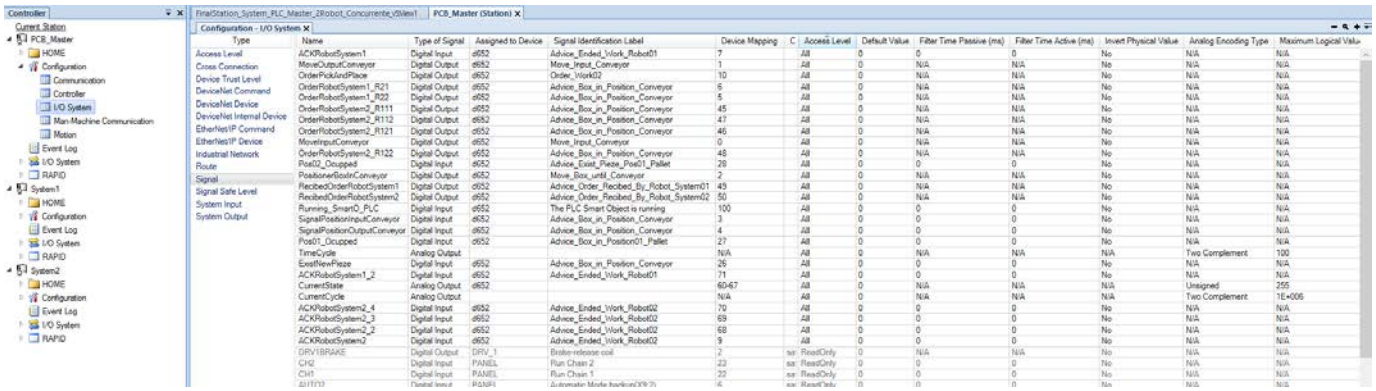


Figure 169: Configuración de E/S: Controlador PLC Master

B) *Configuración Controlador IRB 120*

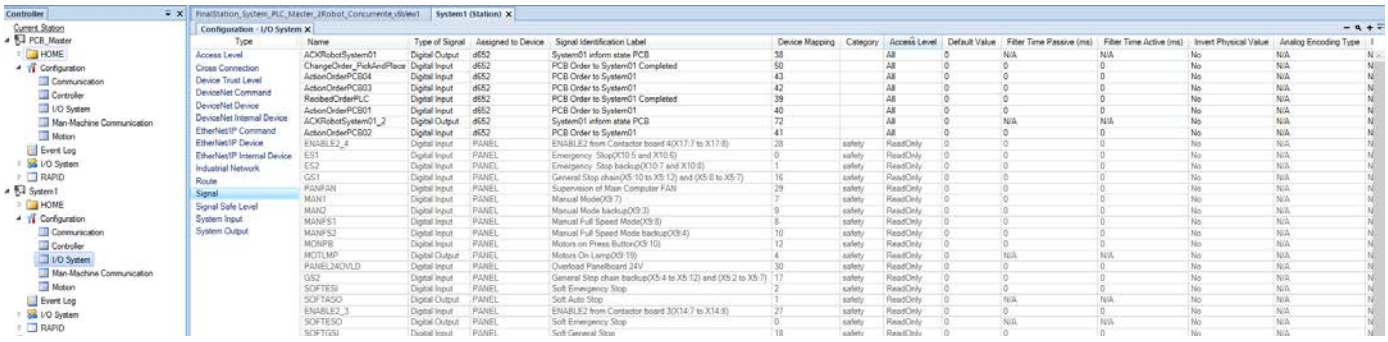


Figure 170: Configuración de E/S: Controlador Robot IRB 120 (Encargado de Mecanizar)

C) *Configuración IRB 140*

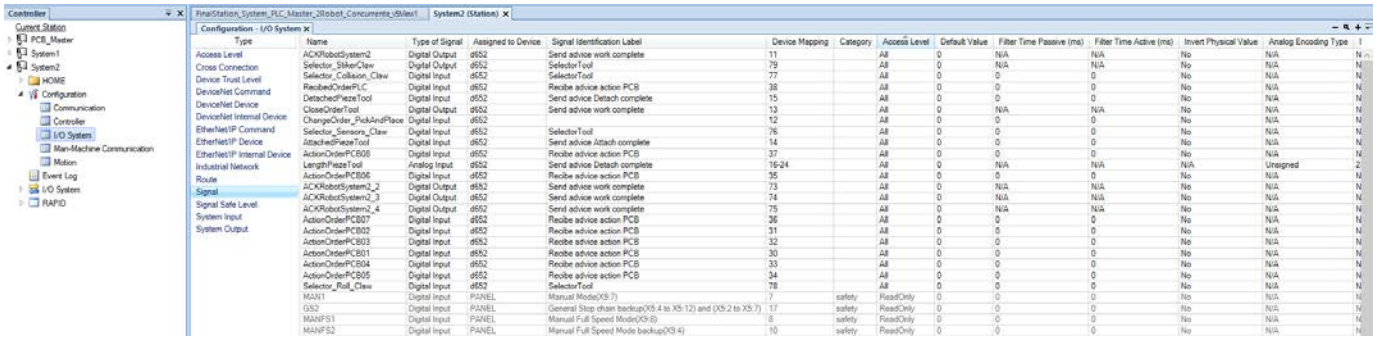


Figure 171: Configuración de E/S: Robot IRB 140 (encargado de Abastacer/Extraer)

7. 3. 2. Conexionado Lógico de la Estación

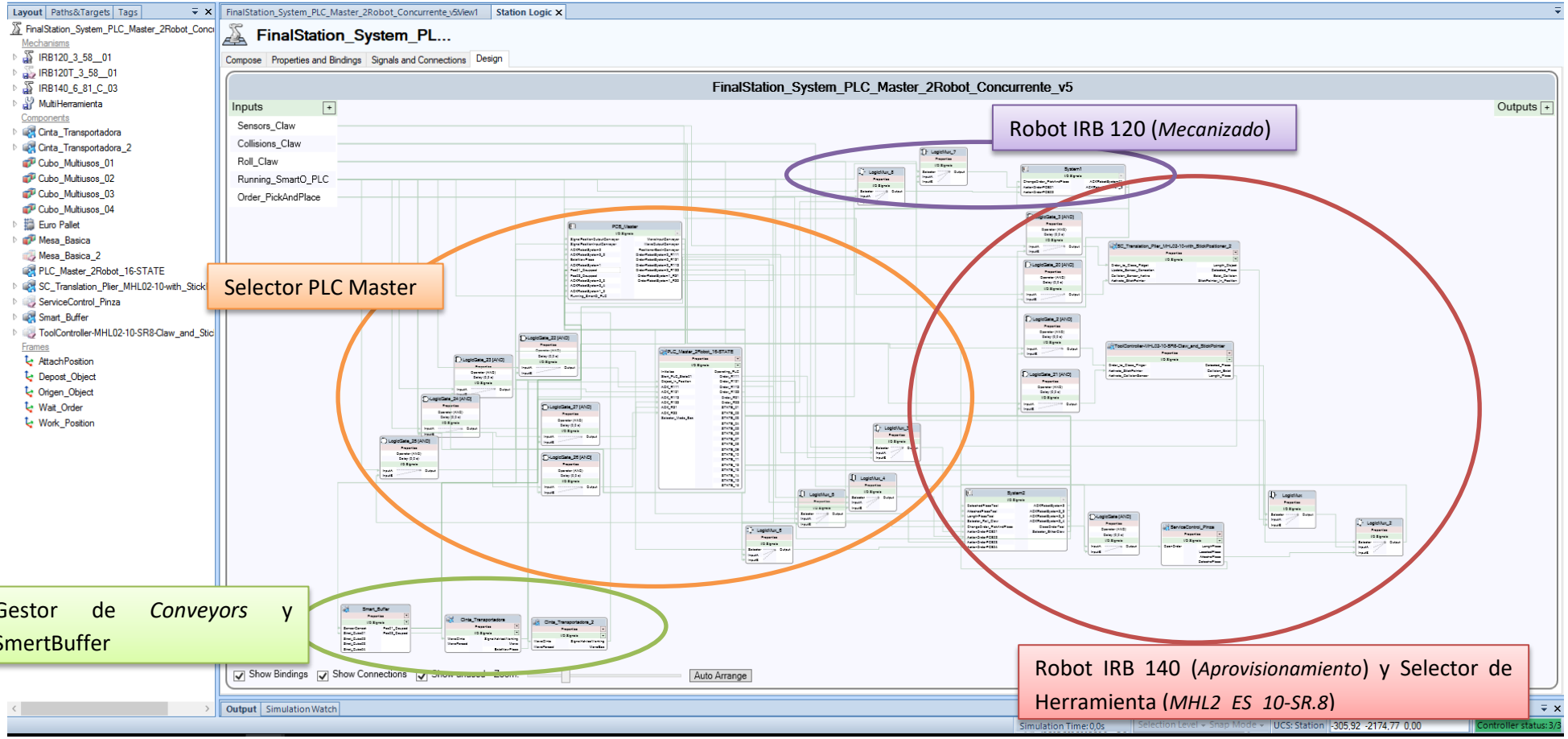


Figure 172: Cableado Lógico General de la Estación: "*Célula 8*"

Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

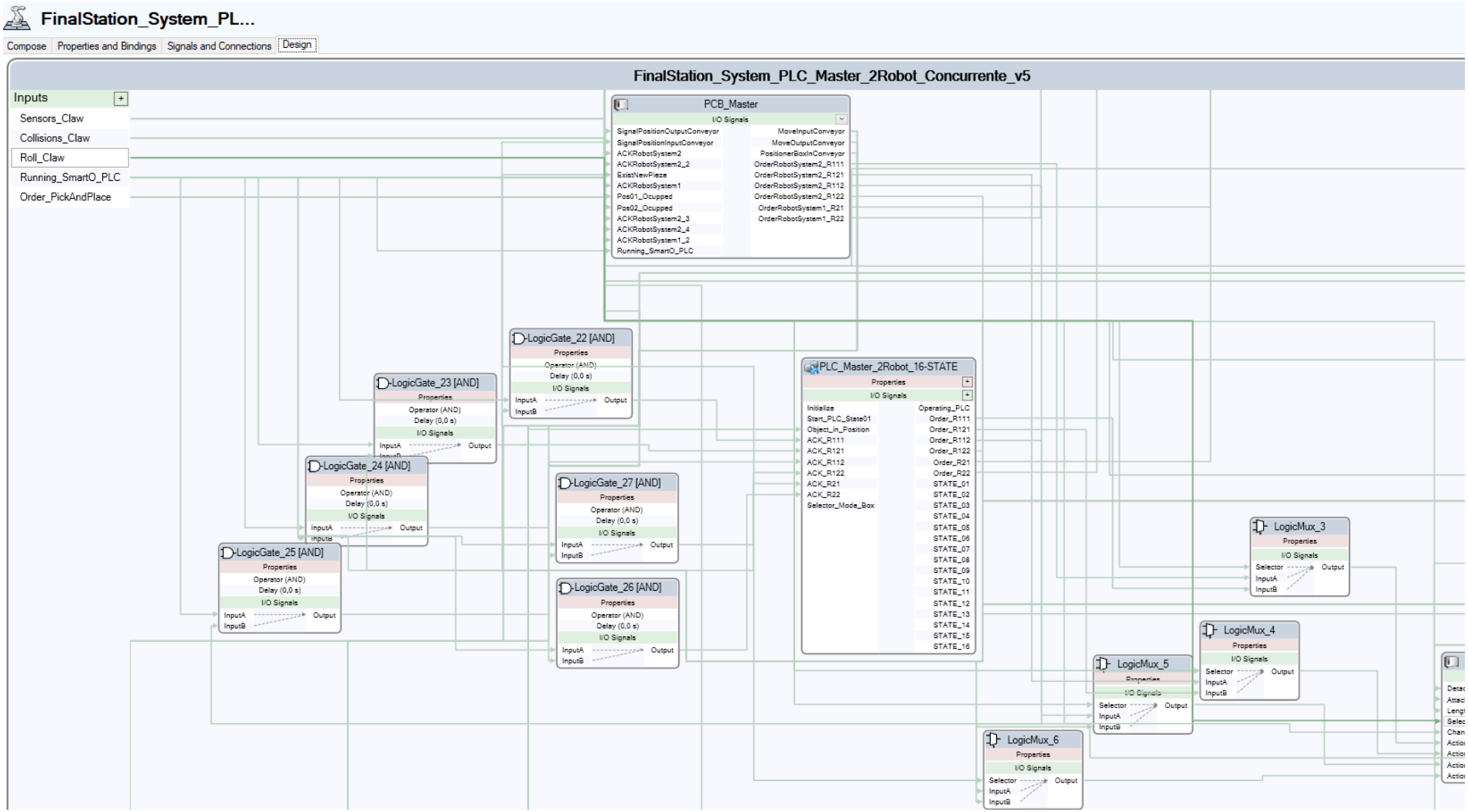


Figure 173: Cableado Lógico Específico de la Estación "Célula 8": Selector de PLC Master

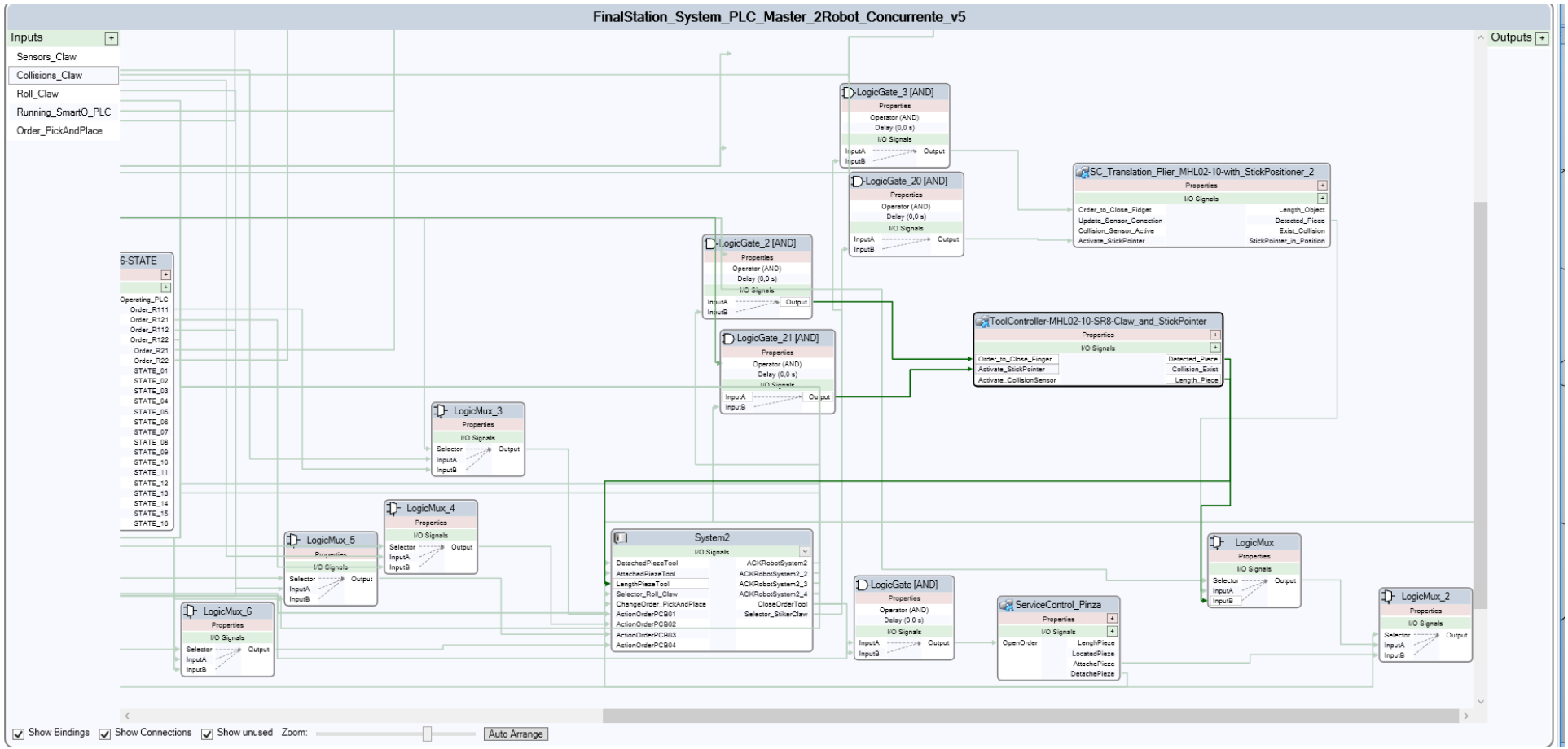


Figure 174 : Cableado Lógico Específico de la Estación "Célula 8": Robot IRB 140 (Aprovisionamiento) y Selector de Herramienta (MHL2_ES_10-SR.8)

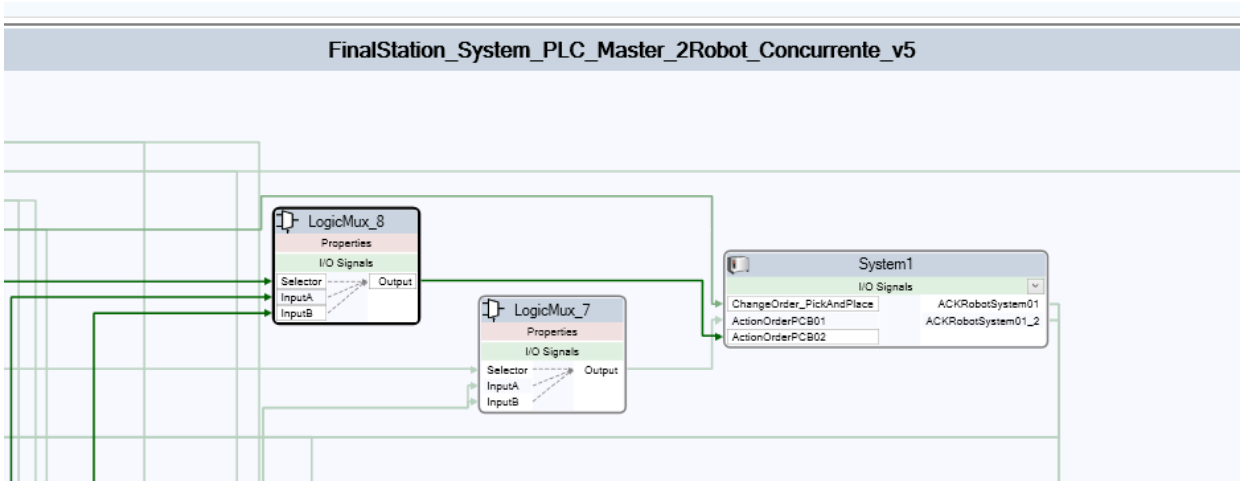


Figure 175: Cableado Lógico Específico de la Estación "Célula 8": Robot IRB 120 (Mecanizado)

7. 3. 3. Panel Selector del Usuario

El usuario puede seleccionar:

- Modo de Operación de la Célula (*"escritura de Letras" o "Pick and Place"*) por medio de la activación de bit "**Order_PickAndPlace**" (***bit a 1**: Modo Pick and Place ; **bit a 0**: Modo Escritura del Letras*)
- Modo de Ejecución del PLC Master de la Célula (*"SmartObject" o "IRC 5 Virtual"*) por medio de la activación de bit "**Running_SmartO_PLC**" (***bit a 1**: Modo SmartObject ; **bit a 0**: Modo IRC 5 Virtual*)
- Herramienta a utilizar, sólo es posible tener activo un único bit selector de: "Collisions_Claw", "Roll_Claw" o "Sensor_Claw" (*bit a 0: No seleccionada ; Bit a 1: Seleccionada*)

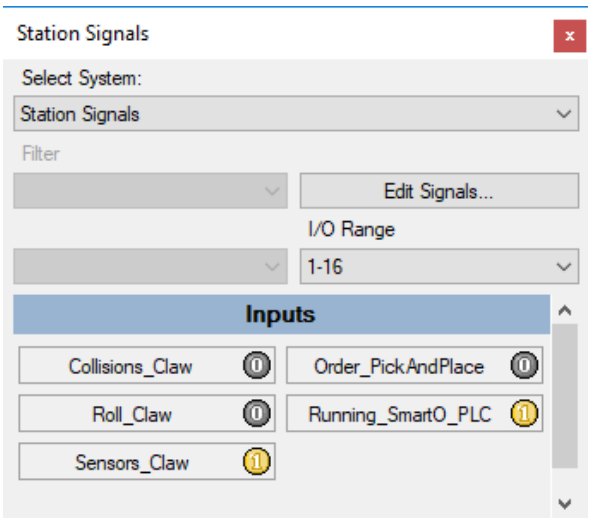


Figure 176: Panel Selector del Usuario

Si se procede a un cambio de Herramienta del Robot IRB 140, el usuario deberá ocultar la Herramienta actual (*directamente sobre el "árbol" de elementos de la Estación*) y asignar en el plano activa (*volver visible*) la nueva Herramienta.

7. 4. PLC Maestro de la Estación

Se debe mencionar que: **RobotStudio no permite la implementación directa de PLC Maestros**, pese a ello se ha **conseguido programar un Automatismo Tipo Moore de 16 Estados con Concurrencia** de dos formas distintas:

- **Tipología industrial típica**: empleo de lógica cableada por medio del uso de **Biestables**, en un SmartObject
- **Punto de vista de un Programador**: por líneas de código en un Controlador IRC-5 Virtual.

Dicho esto, el PLC Maestro implementado se encargará de ejecutar el programa principal y lograr que todos los elementos que componen la Estación cumplan su cometido e interactúen en armonía.

El PLC Maestro, para cualquiera de las implementaciones seguidas (*con SmartObject o con ICR-5 Virtual*), no requiere de *Representación Gráfica* en la Estación Virtual.

Para más información de la implementación del PLC se recomienda ver **ANEXO VI.**

7. 4. 1. Código Completo del PLC Maestro: "Controlador IRC-5 Virtual"

Se ha implementado con la metodología de líneas de código por medio de **2 Swich-Case, uno para actualizar el Estado y otro para actualizar las salidas**. La **activación de salidas** se ha implementado **por nivel**. Para más información ver **ANEXO VI.**

T_ROB1/Module1

```
MODULE Module1
    !***********************************************************
    !
    ! Module:  Module1
    !
    ! Description:
    !   PLC Master: Comanda Entorno y 2 Robots: R1 (Ejecutor de tareas) : R2 (Posicionamiento)
    !       Concurrent System:
    !       Nomenclatura:
    !           Robot Positioner: Rxyz [x: nºRobot(1)// y: nº Position(1/2)// z: nº conveyor (1:input 2:output)]
    !           Robot Worker: Rxy [x: nºRobot(2)// y: nº Position at work(1/2)]
    !           Position Pallet: Pvw [v: State position 01// w:State position 01 //; RangeVariable(0:null//1::Ocupped//2:Ocupped and Completed)]
    ! Author: Octavio Augusto Ansón Clemente
    !
    ! Version: 2.0.puf
    !
    !***********************************************************


    !***********************************************************
    !
    ! Procedure main
    !
    !   This is the entry point of PLC Master program
    !
    !***********************************************************

    !Main Program: Used bucle:
    PROC main()
        !Inicializacion desde HomePosition
        Initialitation;
        !GoHomeBoxy;
        MoveConveyorInput;
        MoveConveyorOutput;
        !InitVariable:
        Cycle:=0;

        !ProductionControl: Number of ended box
        EndedBox:=0;

        WaitDI Running_SmartO_PLC,0;

        !Control Time-Cycle
        Time:=0;
        ClkReset clock2;
        ClkStart clock2;
        !Reset Time Variable

        !Well, we will to start My Obedient Slaves!!: Prub
        WHILE Running_SmartO_PLC=0 DO
            !EndedBox<100: Production Control           !Cycle<150000: Medir Tc            !ClkRead(clock2)<1500: Medir Tc

            !PLC State Actualiatation:
            TEST STATE
            CASE 1:
                !Wait State: Sleep PLC: P00
                IF SignalPositionInputConveyor=1 THEN
                    STATE:=2;
                ENDIF
            CASE 2:
                !R111:P00:
                IF (ACKRobotSystem2=1 and SignalPositionInputConveyor=0) THEN
                    STATE:=3;
                ELSEIF (ACKRobotSystem2=1 and SignalPositionInputConveyor=1) THEN
                    STATE:=5;
                ENDIF
            CASE 3:
                !R21:P10:
                IF ACKRobotSystem1=1 THEN
                    STATE:=4;
                ELSEIF SignalPositionInputConveyor=1 THEN
                    STATE:=5;
                ENDIF
            CASE 4:
                !R112:P20:
                IF ACKRobotSystem2_3=1 THEN
                    STATE:=1;
                    EndedBox:=EndedBox+1;
                ENDIF
```

```
            CASE 5:
                !R121:R21:P10:
                IF ACKRobotSystem2_2=1 THEN
                    STATE:=7;
                ELSEIF ACKRobotSystem1=1 THEN
                    STATE:=6;
                ENDIF
            CASE 6:
                !R121:P20:
                IF ACKRobotSystem2_2=1 THEN
                    STATE:=8;
                ENDIF
            CASE 7:
                !R21:P11:
                IF ACKRobotSystem1=1 THEN
                    STATE:=8;
                ENDIF
            CASE 8:
                !R112:R22:P21:
                IF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=0) THEN
                    STATE:=12;
                    EndedBox:=EndedBox+1;
                ELSEIF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=1) THEN
                    STATE:=11;
                    EndedBox:=EndedBox+1;
                ELSEIF ACKRobotSystem1_2=1 THEN
                    STATE:=9;
                ENDIF
            CASE 9:
                !R112:P22:
                IF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=0) THEN
                    STATE:=10;
                    EndedBox:=EndedBox+1;
                ELSEIF (ACKRobotSystem2_3=1 and SignalPositionInputConveyor=1) THEN
                    STATE:=13;
                ENDIF
            CASE 10:
                !R122:P02:
                IF ACKRobotSystem2_4=1 THEN
                    STATE:=1;
                    EndedBox:=EndedBox+1;
                ENDIF
            CASE 11:
                !R111:R22:P01:
                IF ACKRobotSystem2=1 THEN
                    STATE:=14;
                ELSEIF ACKRobotSystem1_2=1 THEN
                    STATE:=13;
                ENDIF
            CASE 12:
                !R22:P01:
                IF SignalPositionInputConveyor=1 THEN
                    STATE:=11;
                ELSEIF ACKRobotSystem1_2=1 THEN
                    STATE:=10;
                ENDIF
            CASE 13:
                !R111:P02:
                IF ACKRobotSystem2=1 THEN
                    STATE:=15;
                ENDIF
```


Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

```
141             CASE 14:
142                 !R22:P11:
143                 IF ACKRobotSystem1_2=1 THEN
144                     STATE:=15;
145                 ENDIF
146             CASE 15:
147                 !R122:R21:P12:
148                 IF ACKRobotSystem1=1 THEN
149                     STATE:=16;
150                 ELSEIF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=0) THEN
151                     STATE:=3;
152                     EndedBox:=EndedBox+1;
153                 ELSEIF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=1) THEN
154                     STATE:=5;
155                     EndedBox:=EndedBox+1;
156                 ENDIF
157             CASE 16:
158                 !R122:R21:P12:
159                 IF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=0) THEN
160                     STATE:=4;
161                     EndedBox:=EndedBox+1;
162                 ELSEIF (ACKRobotSystem2_4=1 and SignalPositionInputConveyor=1) THEN
163                     STATE:=6;
164                     EndedBox:=EndedBox+1;
165                 ENDIF
166             ENDTEST
167
168             !Output of PLC Actualitation:
169             TEST STATE
170             CASE 1:
171                 Reset OrderRobotSystem1_R21;
172                 Reset OrderRobotSystem1_R22;
173                 Reset OrderRobotSystem2_R111;
174                 Reset OrderRobotSystem2_R112;
175                 Reset OrderRobotSystem2_R121;
176                 Reset OrderRobotSystem2_R122;
177             CASE 2:
178                 !R111:P00:
179                 Reset OrderRobotSystem1_R21;
180                 Reset OrderRobotSystem1_R22;
181                 SetDO OrderRobotSystem2_R111,1;
182                 Reset OrderRobotSystem2_R112;
183                 Reset OrderRobotSystem2_R121;
184                 Reset OrderRobotSystem2_R122;
185             CASE 3:
186                 !R21:P10:
187                 SEtDO OrderRobotSystem1_R21,1;
188                 Reset OrderRobotSystem1_R22;
189                 Reset OrderRobotSystem2_R111;
190                 Reset OrderRobotSystem2_R112;
191                 Reset OrderRobotSystem2_R121;
192                 Reset OrderRobotSystem2_R122;
193             CASE 4:
194                 !R112:P10:
195                 Reset OrderRobotSystem1_R21;
196                 Reset OrderRobotSystem1_R22;
197                 Reset OrderRobotSystem2_R111;
198                 SetDO OrderRobotSystem2_R112,1;
199                 Reset OrderRobotSystem2_R121;
200                 Reset OrderRobotSystem2_R122;
```

```
201             CASE 5:
202                 !R121:R21:P10:
203                 SetDO OrderRobotSystem1_R21,1;
204                 Reset OrderRobotSystem1_R22;
205                 Reset OrderRobotSystem2_R111;
206                 Reset OrderRobotSystem2_R112;
207                 SetDO OrderRobotSystem2_R121,1;
208                 Reset OrderRobotSystem2_R122;
209             CASE 6:
210                 !R121:P20:
211                 Reset OrderRobotSystem1_R21;
212                 Reset OrderRobotSystem1_R22;
213                 Reset OrderRobotSystem2_R111;
214                 Reset OrderRobotSystem2_R112;
215                 SetDO OrderRobotSystem2_R121,1;
216                 Reset OrderRobotSystem2_R122;
217             CASE 7:
218                 !R21:P11:
219                 SetDO OrderRobotSystem1_R21,1;
220                 Reset OrderRobotSystem1_R22;
221                 Reset OrderRobotSystem2_R111;
222                 Reset OrderRobotSystem2_R112;
223                 Reset OrderRobotSystem2_R121;
224                 Reset OrderRobotSystem2_R122;
225             CASE 8:
226                 !R112:R22:P21:
227                 Reset OrderRobotSystem1_R21;
228                 SetDO OrderRobotSystem1_R22,1;
229                 Reset OrderRobotSystem2_R111;
230                 SetDO OrderRobotSystem2_R112,1;
231                 Reset OrderRobotSystem2_R121;
232                 Reset OrderRobotSystem2_R122;
233             CASE 9:
234                 !R112:P22:
235                 Reset OrderRobotSystem1_R21;
236                 Reset OrderRobotSystem1_R22;
237                 Reset OrderRobotSystem2_R111;
238                 SetDO OrderRobotSystem2_R112,1;
239                 Reset OrderRobotSystem2_R121;
240                 Reset OrderRobotSystem2_R122;
241             CASE 10:
242                 !R122:P02:
243                 Reset OrderRobotSystem1_R21;
244                 Reset OrderRobotSystem1_R22;
245                 Reset OrderRobotSystem2_R111;
246                 Reset OrderRobotSystem2_R112;
247                 Reset OrderRobotSystem2_R121;
248                 SetDO OrderRobotSystem2_R122,1;
249             CASE 11:
250                 !R111:R22:P01:
251                 Reset OrderRobotSystem1_R21;
252                 SetDO OrderRobotSystem1_R22,1;
253                 SetDO OrderRobotSystem2_R111,1;
254                 Reset OrderRobotSystem2_R112;
255                 Reset OrderRobotSystem2_R121;
256                 Reset OrderRobotSystem2 R122;
```

```
257             CASE 12:
258                 !R22:P01:
259                 Reset OrderRobotSystem1_R21;
260                 SetDO OrderRobotSystem1_R22,1;
261                 Reset OrderRobotSystem2_R111;
262                 Reset OrderRobotSystem2_R112;
263                 Reset OrderRobotSystem2_R121;
264                 Reset OrderRobotSystem2_R122;
265             CASE 13:
266                 !R111:P02:
267                 Reset OrderRobotSystem1_R21;
268                 Reset OrderRobotSystem1_R22;
269                 SetDO OrderRobotSystem2_R111,1;
270                 Reset OrderRobotSystem2_R112;
271                 Reset OrderRobotSystem2_R121;
272                 Reset OrderRobotSystem2_R122;
273             CASE 14:
274                 !R22:P11:
275                 Reset OrderRobotSystem1_R21;
276                 SetDO OrderRobotSystem1_R22,1;
277                 Reset OrderRobotSystem2_R111;
278                 Reset OrderRobotSystem2_R112;
279                 Reset OrderRobotSystem2_R121;
280                 Reset OrderRobotSystem2_R122;
281             CASE 15:
282                 !R122:R21:P12:
283                 SetDO OrderRobotSystem1_R21,1;
284                 Reset OrderRobotSystem1_R22;
285                 Reset OrderRobotSystem2_R111;
286                 Reset OrderRobotSystem2_R112;
287                 Reset OrderRobotSystem2_R121;
288                 SetDO OrderRobotSystem2_R122,1;
289             CASE 16:
290                 !R122:P22:
291                 Reset OrderRobotSystem1_R21;
292                 Reset OrderRobotSystem1_R22;
293                 Reset OrderRobotSystem2_R111;
294                 Reset OrderRobotSystem2_R112;
295                 Reset OrderRobotSystem2_R121;
296                 SetDO OrderRobotSystem2_R122,1;
297             ENDTEST
298             SetAO CurrentState,STATE;
299             !Sincronitation With other Systems:
300             Cycle:=Cycle+1;
301             !WaitTime 0.0001;
302             SetAO CurrentCycle,Cycle;
303             !Unsigned Range(0:1000000)
304         ENDWHILE
305
306         !Measure: Wather MidleTime of Cicle of RobotSystem
307         ClkStop clock2;
308         Time:=ClkRead(clock2);
309         Tc:=Time/Cycle;
310         SetAO TimeCycle,Time;
311         Time:=Time/Cycle;
312
```

```
313     !Signed Range(-100;100): [Initial Solve: Tc1=0.0010039] [Tc2=0.00155835] [Tc(10000)=2.1751(t=21.8)] [Tc(30000)=0.00200903(60.3)] [Tc(1000000)=0.0019317] Look at "user" Module
314     !With time1=100seg: 53781,000 Cycles: Tc1= 0.001859393        !With time2=300seg: 154364 Cycles: Tc2= 0.00194346        !With time2=600seg: 301416 Cycles: Tc2= 0.0019906
315     !With time2=900seg: 450777 Cycles: Tc2=0.00199655        !With time2=1500seg: 746042 Cycles: Tc2=0.00201061
316
```

```
318     ENDPROC
319
320
321
322
323
324     !!Imlementation: Functions/Methodes/Procces:
325
326     LOCAL PROC Initialitation()
327         Reset PositionerBoxInConveyor;
328         Reset MoveInputConveyor;
329         Reset MoveOutputConveyor;
330         !STATE:=1; !If the Station run as first simulation activate this line...
331         Reset RecibedOrderRobotSystem1;
332         Reset RecibedOrderRobotSystem2;
333         Reset OrderRobotSystem1_R21;
334         Reset OrderRobotSystem1_R22;
335         Reset OrderRobotSystem2_R111;
336         Reset OrderRobotSystem2_R112;
337         Reset OrderRobotSystem2_R121;
338         Reset OrderRobotSystem2_R122;
339     ENDPROC
340
341     ! Entorno a las instalaciones principales:
342
343     LOCAL PROC MoveConveyorInput()
344         Reset MoveInputConveyor;
345         SetDO MoveInputConveyor,1;
346     ENDPROC
347
348     LOCAL PROC MoveConveyorOutput()
349         Reset MoveOutputConveyor;
350         SetDO MoveOutputConveyor,1;
351     ENDPROC
352
353
354     !Funciones Secundarias:
355     LOCAL PROC GoHomeBoxy()
356         SetDO PositionerBoxInConveyor,1;
357         Reset PositionerBoxInConveyor;
358     ENDPROC
359
360
361 ENDMODULE
```


Titular: Octavio Augusto Ansón Clemente

Director: Antonio Romeo Tello

7. 4. 2. Código Completo del PLC Maestro: SmartObject

A) *Esquema del SmartObject:*

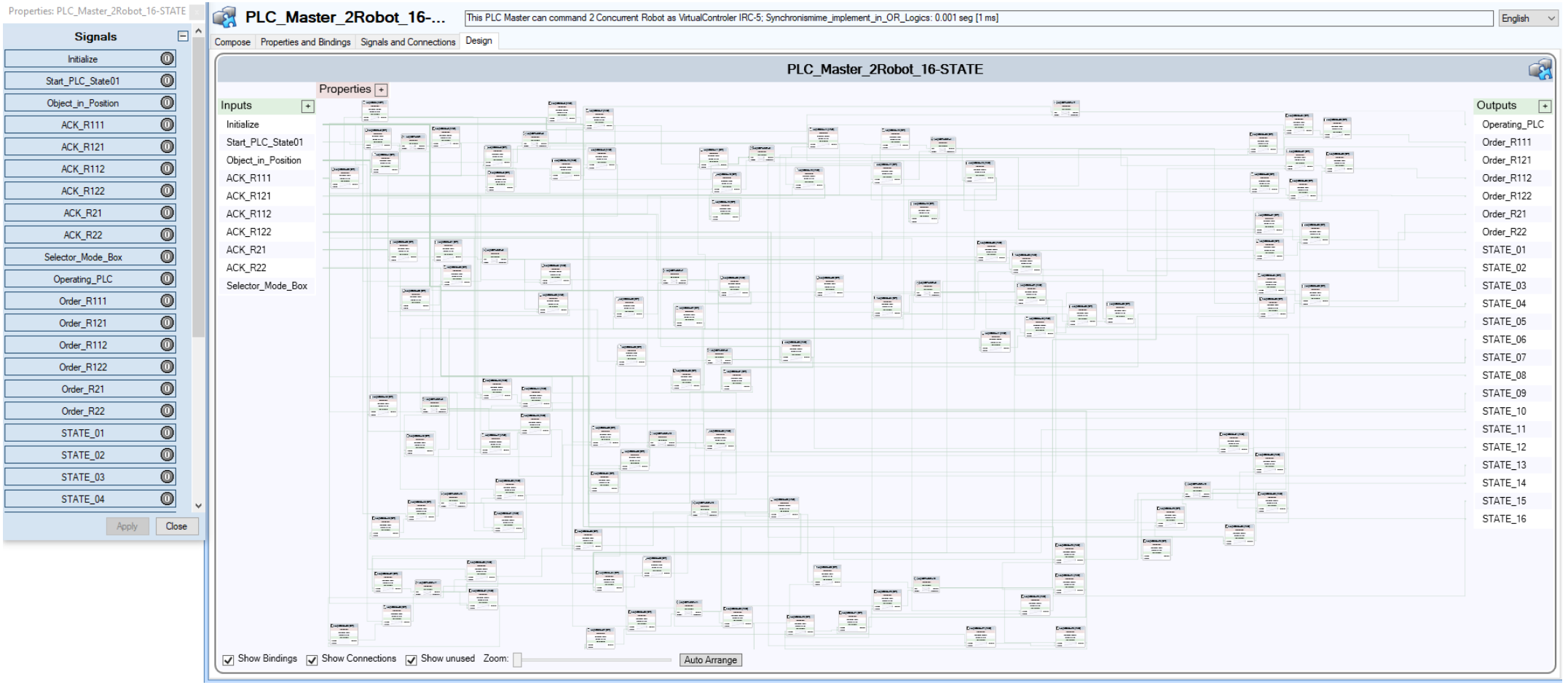


Figure 177: Esquemático del PLC Master 16 Estados: SmartObject

B) *Esquemático resultante del Grafo de Estados implementado:*

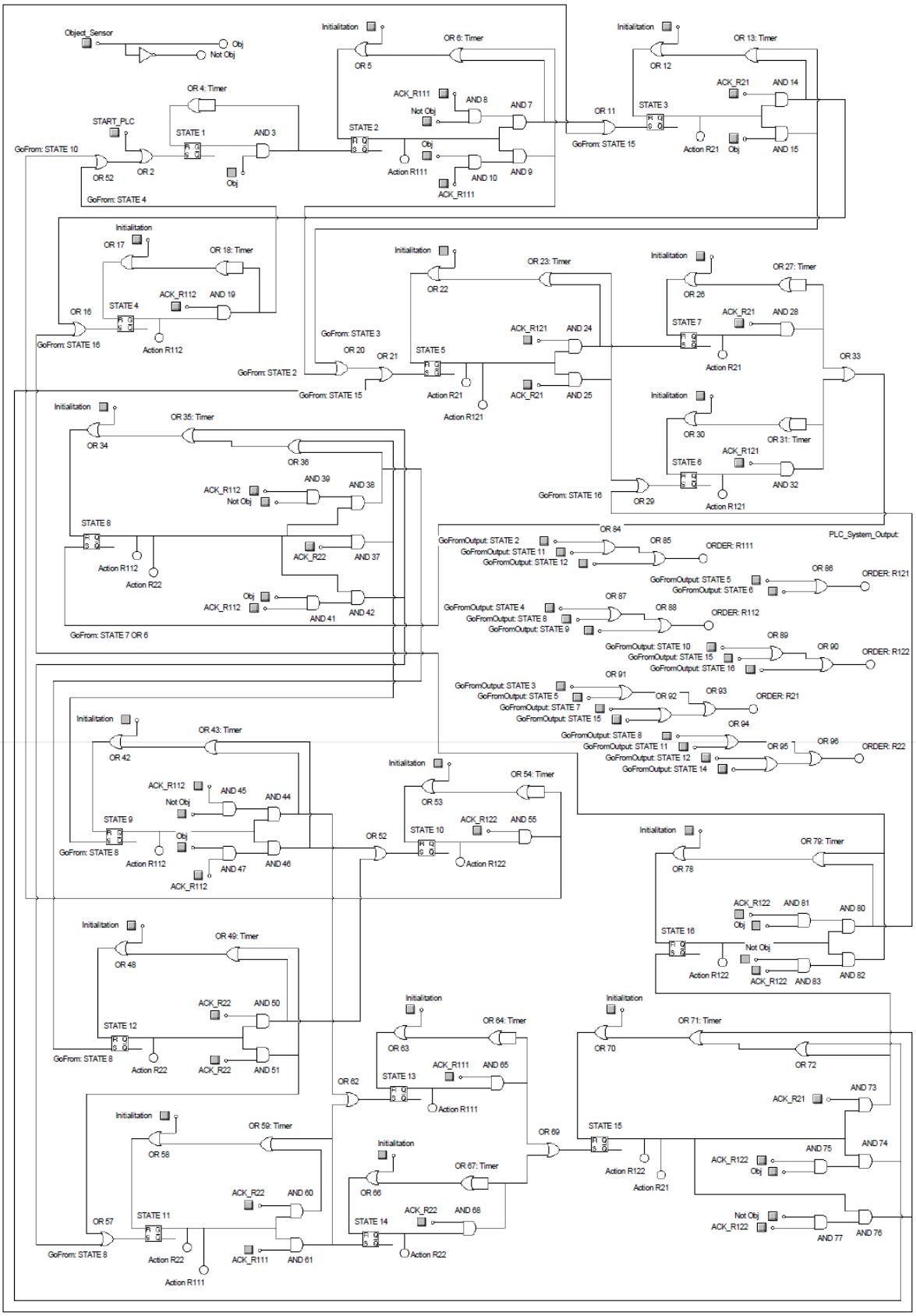


Figure 178: Esquemático del PLC Master 16 Estados: Flip-Flop del SmartObject

7. 5. Aplicaciones de los Robot en RAPID

A) *Metodología Seleccionada*

A causa de que RobotStudio favorece el uso de una estructura predefinida de 2 niveles (*referencias de usuario* junto con la referencia del objeto de trabajo *Wobjdata*), colgando así mismo los Target del WorkObject y operando con las distintas herramientas para relativizar las localizaciones. Se ha seleccionado trabajar directamente con WorkObject + Target, apoyado de función Offs (*en las aproximaciones a los Targets*).

En la imagen siguiente se puede apreciar la ubicación de los Target y las trayectorias de los Robots.

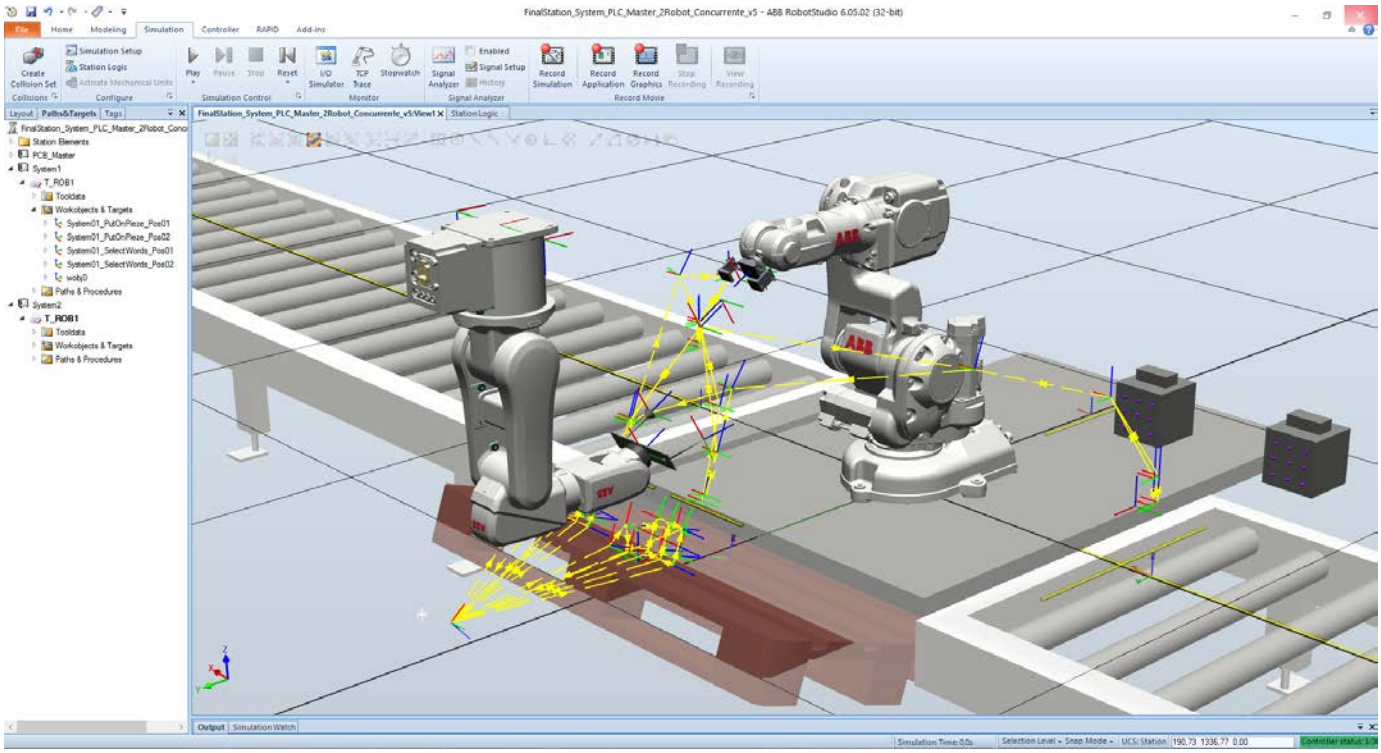


Figure 179: Trayectorias de los Robots: Estación Simulada "Célula 8"

B) *Consideraciones de los Robots*

Los Robots **han sido configurados como esclavos perfectos**, si no están obedeciendo una orden previa estarán esperando una nueva orden del PLC Maestro.

Ambos Robots **constan de dos ficheros**, el Módulo principal y el Módulo de Funciones Programadas:

- **Módulo Principal**: está formado por un programa de inicialización y un *Loop* donde se estará esperando una nueva orden, momento en el que se chequeará cual de todas las órdenes es, se ejecutará y se informar al Maestro de que la ha completado con éxito.
- **Módulo de Funciones**: Fichero con los datos específicos de todos los elementos del programa y una lista de Procesos/Ordenes programadas.

a) Observación Importante de la Simulación de Robots:

Se ha detectado que **es necesario proveer de pequeños "*Delays*", tras las órdenes de cierre de garra en las simulaciones y en la activación de Salidas del IRC-5 Virtual**, caso que difiere del comportamiento de la instalación en la realidad. Se cree que es debido a la problemática en la gestión de E/S, problemática que se ha constatado en los **ANEXO II**.

Los "*Delays*" se deben dimensionar en función del *Step* seleccionado en la configuración del Programa, siendo el $T_{Delay} > T_{Step}$. Ver **Ficha de Ruta: "Cómo Configurar y Medir el Tiempo en Simulación"**.

El código implementado en los Robots es:

C) *Robot 1: Programa Principal*

```
MODULE Module1

    !Main Program: Used bucle:
    PROC main()
        Initalitation;

        WHILE TRUE DO
            !Wait Order PLC: Perfect Slave: "Yes My Lord!":
            !Wait until exist input variation:
            WaitUntil(ActionOrderPCB01=1 OR ActionOrderPCB02=1 OR ActionOrderPCB03=1 OR ActionOrderPCB04=1 OR ActionOrderPCB05=1 OR ActionOrderPCB06=1 OR ActionOrderPCB07=1 OR ActionOrderPCB08=1);
            !Actualitation STATE variable with Input_Information:
            IF (ActionOrderPCB01=1 AND ChangeOrder_PickAndPlace=0) THEN
                STATE:=1;
            ELSEIF (ActionOrderPCB02=1 AND ChangeOrder_PickAndPlace=0) THEN
                STATE:=2;
            ELSEIF (ActionOrderPCB03=1 AND ChangeOrder_PickAndPlace=0) THEN
                STATE:=3;
            ELSEIF (ActionOrderPCB04=1 AND ChangeOrder_PickAndPlace=0) THEN
                STATE:=4;
            ELSEIF (ActionOrderPCB01=1 AND ChangeOrder_PickAndPlace=1) THEN
                STATE:=5;
                !ActionOrderPCB05=1
            ELSEIF (ActionOrderPCB02=1 AND ChangeOrder_PickAndPlace=1) THEN
                STATE:=6;
                !ActionOrderPCB06=1
            ELSEIF (ActionOrderPCB03=1 AND ChangeOrder_PickAndPlace=1) THEN
                STATE:=7;
                !ActionOrderPCB07=1
            ELSEIF (ActionOrderPCB04=1 AND ChangeOrder_PickAndPlace=1) THEN
                STATE:=8;
                !ActionOrderPCB08=1
            ENDIF

            !Robot Programs:
            TEST STATE
            CASE 1:
                IF Selector_Roll_Claw=1 THEN
                    BIC_Pos01Letters_RollClaw;
                ELSE
                    BIC_Pos01Letters_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2,1;
                WaitDI ActionOrderPCB01,0;

            CASE 2:
                IF Selector_Roll_Claw=1 THEN
                    BIC_Pos02Letters_RollClaw;
                ELSE
                    BIC_Pos02Letters_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2_2,1;
                WaitDI ActionOrderPCB02,0;

            CASE 3:
                IF Selector_Roll_Claw=1 THEN
                    BOC_Pos01Letters_RollClaw;
                ELSE
                    BOC_Pos01Letters_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2_3,1;
                WaitDI ActionOrderPCB03,0;

            CASE 4:
                IF Selector_Roll_Claw=1 THEN
                    BOC_Pos02Letters_RollClaw;
                ELSE
                    BOC_Pos02Letters_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2_4,1;
                WaitDI ActionOrderPCB04,0;

            CASE 5:
                IF Selector_Roll_Claw=1 THEN
                    BIC_Pos01PAndP_RollClaw;
                ELSE
                    BIC_Pos01PAndP_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2,1;
                WaitDI ActionOrderPCB01,0;

            CASE 6:
                IF Selector_Roll_Claw=1 THEN
                    BIC_Pos02PAndP_RollClaw;
                ELSE
                    BIC_Pos02PAndP_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2_2,1;
                WaitDI ActionOrderPCB02,0;

            CASE 7:
                IF Selector_Roll_Claw=1 THEN
                    BOC_Pos01PAndP_RollClaw;
                ELSE
                    BOC_Pos01PAndP_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2_3,1;
                WaitDI ActionOrderPCB03,0;

            CASE 8:
                IF Selector_Roll_Claw=1 THEN
                    BOC_Pos02PAndP_RollClaw;
                ELSE
                    BOC_Pos02PAndP_MHL02Claw;
                ENDIF

                SETDO ACKRobotSystem2_4,1;
                WaitDI ActionOrderPCB04,0;

            ENDTEST
            !Dont Wait Refuse of PLC: Perfect Slave: "The Work are completed My Lord!":
            !Sincronitation with PLC_Master
            Reset ACKRobotSystem2;
            Reset ACKRobotSystem2_2;
            Reset ACKRobotSystem2_3;
            Reset ACKRobotSystem2_4;
        ENDWHILE
    ENDPROC

    !!Imlementation: Functions/Methodes/Procces:


    !Initialitation Variables:
    LOCAL PROC InitialitationTool_RollClaw()
        SetDO CloseOrderTool,0;
        WaitDI AttachedPiezeTool,0;

        WaitTime 0.0001;
        SetDO CloseOrderTool,1;
        WaitTime 1;
        SetDO CloseOrderTool,0;
        WaitAI LengthPiezeTool\GT,30;
    ENDPROC

    LOCAL PROC InitialitationTool_MHL02()
        SetDO CloseOrderTool,0;
        !WaitDI AttachedPiezeTool,0;

        WaitTime 0.0001;
        SetDO Selector_StikerClaw,1;
        WaitTime 0.5;
        SetDO Selector_StikerClaw,0;
        WaitTime 0.001;

        SetDO CloseOrderTool,1;
        WaitTime 1;
        SetDO CloseOrderTool,0;
        WaitAI LengthPiezeTool\GT,62;
    ENDPROC

    LOCAL PROC Initalitation()
        !Initialitation:
        Reset ACKRobotSystem2;
        Reset ACKRobotSystem2_2;
        Reset ACKRobotSystem2_3;
        Reset ACKRobotSystem2_4;
        !HomePosition
        IF Selector_Roll_Claw=1 THEN
            MoveJ HomeRelax,v1000,z100,Pinza_RollFidgets_ToolData\WObj:=wobj0;
            InitialitationTool_RollClaw;
        ELSE
            MoveJ HomeRelax,v1000,z100,TCP_Claw_MHL02_10_SR8\WObj:=wobj0;
            InitialitationTool_MHL02;
        ENDIF

        !CheckTool:
        WaitTime 0.1;
    ENDPROC


ENDMODULE
```

Figure 180: Código del Robot 1: IRB 120 (Abastacemiento): Estación Simulada

D) *Robot 2: Programa Principal*

```
MODULE Module1
    !Main Program: Used bucle:
    PROC main()
        !Inicialization in HomePosition
        Initialitation;
        !Wait Order PLC: Perfect Slave: "Yes My Lord!":
        WHILE TRUE DO
            !Wait until exist input variation:
            WaitUntil(ActionOrderPCB01=1 OR ActionOrderPCB02=1 OR ActionOrderPCB03=1 OR ActionOrderPCB04=1);
            !Actualitation STATE variable with Input_Information:
            IF (ActionOrderPCB01=1 AND ChangeOrder_PickAndPlace=0) THEN
                STATE:=1;
            ELSEIF (ActionOrderPCB02=1 AND ChangeOrder_PickAndPlace=0) THEN
                STATE:=3;
            ELSEIF (ActionOrderPCB01=1 AND ChangeOrder_PickAndPlace=1) THEN
                STATE:=2;
                !ActionOrderPCB03=1
            ELSEIF (ActionOrderPCB02=1 AND ChangeOrder_PickAndPlace=1) THEN
                STATE:=4;
                !ActionOrderPCB04=1
            ENDIF

            !Robot Programs: Slave Work!
            TEST STATE
            CASE 1:
                !Wait Order PLC: Draw Letters in the Box Position01
                Pos01DrawLetter_O;
                Pos01DrawLetter_C;
                Pos01DrawLetter_T;
                SetDO ACKRobotSystem01,1;
                WaitDI ActionOrderPCB01,0;
            CASE 2:
                !Wait Order PLC: Pick And Place in the Box Position01
                Pos01PickAndPlaces;
                SetDO ACKRobotSystem01,1;
                WaitDI ActionOrderPCB01,0;
            CASE 3:
                !Wait Order PLC: Draw Letters in the Box Position02
                Pos02DrawLetter_O;
                Pos02DrawLetter_C;
                Pos02DrawLetter_T;
                SetDO ACKRobotSystem01_2,1;
                WaitDI ActionOrderPCB02,0;
            CASE 4:
                !Wait Order PLC: Pick And Place in the Box Position02
                Pos02PickAndPlaces;
                SetDO ACKRobotSystem01_2,1;
                WaitDI ActionOrderPCB02,0;
            ENDTEST

            !Sincronitation with PLC_Master
            Reset ACKRobotSystem01;
            Reset ACKRobotSystem01_2;
        ENDWHILE
    ENDPROC

    LOCAL PROC Initialitation()
        Reset ACKRobotSystem01;
        Reset ACKRobotSystem01_2;
        !HomePosition
        MoveJ HomeIRB120,v1000,z100,TCP2_Soldador_MultiHerramienta\WObj:=wobj0;
        WaitTime 0.1;
    ENDPROC


ENDMODULE
```

Figure 181: Código del Robot 2: IRB 120 (Mecanizado): Estación Simulada