Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

MASTER EN INFORMÁTICA INDUSTRIAL

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO FIN DE MASTER

Simulación y Diagnóstico de una Instalación Industrial mediante Factory I/O y OPC

Autor:

Daniel García Fernández

Tutor UVA:

Gregorio Sainz Palmero
Departamento de Ingeniería de Sistemas y Automática

Tutor Empresa:

Germán de Cruz Quintanilla
Bosch Rexroth, S.L.

Valladolid, Julio de 2017.

RESUMEN Y PALABRAS CLAVES

En el presente proyecto se ha modelado y simulado una instalación real de fabricación y logística la cual está actualmente en funcionamiento. Se ha empleado un simulador de instalaciones industriales, Factory I/O 2.1.3 adaptando la instalación real a una instalación simulada en función de las características que el software permite, usando mesas de transporte, mesas giratorias, elevadores y transfers bidireccionales. Para el control de esta instalación se ha realizado un programa de PLC con Siemens Step 7 (modelo CPU 315-2 PN/DP) capaz de simular algún modo de funcionamiento de la instalación, pero con un código más sencillo. Se ha creado un servidor OPC capaz del acceder al autómata simulado y conectarlo con un cliente en Phyton, lenguaje en que se programará un interfaz gráfico con las librerías de Qt. Para acabar la simulación de la instalación se creará un registro de eventos y se podrá realizar un análisis de estos eventos.

Palabras claves:

- Simulación
- Automatismos
- Comunicaciones
- Data Analisys
- OPC

In this project, a real manufacturing and logistics installation has been modelled and simulated which is currently active. An industrial simulator, Factory I / O 2.1.3 has been used, adapting the actual installation to a simulated installation according to the characteristics of the software, using conveyors, turntables, elevators and bi-directional transfers. For the control of this installation a PLC program with Siemens Step 7 (model CPU 315-2 PN / DP) could simulate some mode of operation, but with a simpler code. An OPC server has been created to access to the simulated model and connect it to a client in Phyton. A graphic interface will be programmed with the Qt libraries. To finish the simulation of the installation, an event log will be created and an analysis of these events can be performed.

Keywords:

- Simulation
- Automatisms
- Communications
- Data Analysis
- OPC

ÍNDICE

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

En la actualidad cada vez tiene más importancia realizar un correcto análisis y un estudio completo antes de proceder a realizar la puesta en servicio de una instalación.

Con este proyecto se pretende realizar un análisis previo a la puesta en servicio de la instalación, con lo que podemos valorar distintos modos de funcionamiento, obtener información que es de interés para su análisis, simular tiempos de ciclos, etc., algo crítico para que pueda entregarse una instalación, se pueden ver los diferentes modos de funcionamiento y el cliente puede realizar ciertas observaciones a la hora de esta implementación.

Siendo de gran interés y utilidad el poder realizar todos estos análisis sin necesidad de contar con un modelo físico además del ahorro de espacio físico y tiempo que esto conlleva.

Por otro lado, cada día es más habitual el ir adelantado la producción a la industria 4.0, para ello con el presente proyecto se pretenden utilizar diversas técnicas actualmente disponibles y en creciente uso aplicadas al análisis de la instalación, tanto aspectos físicos, como a nivel de software y posibles eventos programados, complementado con el uso de un control remoto, en forma de HMI adaptable a dispositivos móviles.

1.2. OBJETIVOS

El objetivo principal de este proyecto parte de la necesidad de la realización de una instalación industrial, a partir de la cual se ha desarrollado una solución en base a los conocimientos y técnicas recibidos en el master relacionados con el modelado y la simulación de una planta real, orientándolo hacia una aplicación industrial 4.0 con control remoto.

A partir de esta idea principal, este proyecto tiene unos objetivos específicos, que son:

- A partir de las especificaciones de la instalación real, crear un modelo físico de simulación representado mediante los componentes mecánicos de la instalación real en el software de simulación.
- Permitir el control automático de la instalación en simulación mediante el desarrollo un programa de autómata simplificado capaz de simular alguno de los funcionamientos de la instalación real.
- Capturar la información de la instalación y sus sistemas para su tratamiento, con posibilidad de tratamiento remoto.
- Disponer de un HMI (Human Machine Interface) que permita representar el estado de la simulación, los eventos de la simulación, y capaz de establecer modos de funcionamiento de la instalación.
- Efectuar un análisis de los diferentes eventos que surjan durante la simulación de cara al mantenimiento de la propia instalación.

1.3. ESTRUCTURA DEL PROYECTO

En este apartado se detalla la estructura del presente proyecto, indicando el contenido de cada capítulo y las aportaciones más importantes realizadas.

Capítulo 2: Propuesta de Instalación y Simulación

En este capítulo se hace una propuesta de forma lógica del modelo para la instalación y se explicará cual es el procedimiento seguido en la simulación.

Capítulo 3: Aspectos Teóricos del Modelo

En este capítulo se hará una introducción a las diferentes técnicas y consideraciones usadas para la implementación, así como los aspectos teóricos básicos del modelo.

Capítulo 4: Implementación del Modelo

Se explicará cuáles son las herramientas que se han usado para la implementación del modelo, así como la forma y el proceso seguido para realizar esta implementación. También se comentarán los resultados obtenidos y el análisis de éstos.

Capítulo 5: Conclusiones

En este capítulo se muestran las conclusiones que se pueden extraer una vez realizado todo el proceso de implementación y análisis, así como unas posibles mejoras que se pueden realizar en un futuro.

Anexos

Se mostrará cual ha sido la distribución temporal del trabajo, una ayuda proporcionada por Bosch Rexroth de cara a la instalación, así como se ha

tenido que configurar todo los programas de software y unos ejemplos de código usado en la implementación.

2. PROPUESTA DE INSTALACIÓN Y SIMULACIÓN

En este capítulo se hace una propuesta de forma lógica del modelo para la instalación y se explicará cual es el procedimiento seguido en la simulación. Se comentan cual son las restricciones que hay que tener en cuenta para crear el modelo de simulación, y cuál es nuestra propuesta de modelo y de simulación.

2.1. CUADERNO DE CARGAS Y RESTRICCIONES

El objetivo de la instalación será el transporte de piezas desde una zona de entrada a una zona de salida. Además, será necesario que durante diversos puntos intermedios se acumen piezas por si hubiera algún problema en la instalación que nos suministra piezas poder obtener piezas de este stock intermedio.

Además, habrá que tratar el problema de que la entrada de piezas será en un nivel 0, y salida de piezas en un nivel 1, de altura superior, por lo que será necesario un elevador que eleve la pieza.

Será restricción el uso de mesas de rodillos para transportar las piezas y no de cintas transportadoras, debido al peso y característica metálica de la pieza.

Además, el cliente nos especifica que las piezas llegan con un intervalo de 30 segundos y es necesario que podamos suministrar una pieza con un tiempo máximo de 45 segundos, por este motivo será de especial utilidad el ir rellenando de piezas las zonas de cúmulo para poder contrarrestar la diferencia de tiempo entre la llegada y la salida de piezas.

Existen unas restricciones mecánicas de la instalación, el elevador no podrá ir siempre a máxima velocidad para evitar vibraciones y asegurarnos el lugar de parada, aparte, será necesario que el nivel 0 sea una zona cerrada con mecanismos anti-intrusión para evitar atrapamientos.

Aparte también nos da unas pautas para implementar el control y mantenimiento de la instalación. De forma remota nos piden que se represente el estado actual de la instalación, así como que podamos rearmarla.

Debe ser posible el seleccionar acumular o desacumular en cada uno de los transfers de acúmulo, así como poder ver el estado de cada uno de los elementos funcionales pudiendo ver el estado de los sensores y memorias, y pudiendo ciclar las memorias de presencia y de tránsito.

De cara a un mantenimiento preventivo de la instalación, el cliente pide que se haga un análisis de los eventos/defectos que surjan durante el funcionamiento dándonos libertad a escoger la técnica que consideremos más adecuada siempre y cuando se permita efectuar una relación entre estos eventos. Para ello será necesario programar unos eventos que tienen lugar cuando hay memoria presencia, pero ésta no es detectada por los sensores o cuando el tránsito de un elemento funcional a otro está activo durante más de un determinado tiempo.

Todos estos eventos serán visualizados en un interfaz gráfico HMI de cara a un mantenimiento, debiendo ser posible el ejecutar la aplicación gráfica y dispositivos portátiles e independientes de la instalación.

2.2. PROPUESTA DE INSTALACIÓN Y MODELO

Una vez recibidas todas las propuestas del cliente, se realiza una propuesta de instalación (Bosch Rexroth S.L. 2016) que cumpla con todos los requisitos comentados. Para ello contaremos con cuatro zonas las cuales se proceden a describir.

En primer lugar, tendremos una zona donde se produce la entrada de piezas (como en nuestro caso es un modelo simulado esta entrada de piezas se simulará con un creador de piezas) y tras pasar por cuatro mesas llega a un transfer capaz de desplazar la pieza en dos direcciones, una de ellas permite que la pieza llega a la salida y la otra de ellas nos permite que la pieza sea acumulada en caso de que la línea esté saturada o se haya dado orden de acumular. El sinóptico correspondiente a esta zona es el siguiente:

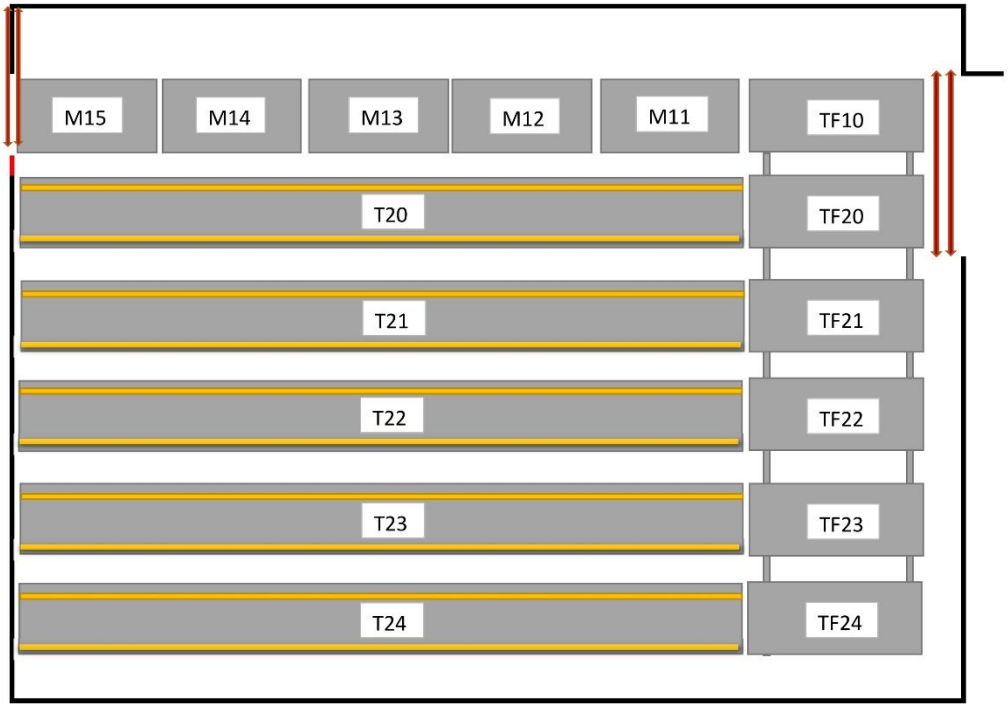


Figura 1 – Sinóptico Propuesta Zona Entrada

Tras pasar por esta zona la pieza llegaría a la zona del elevador, en esta zona existe una mesa en la que podemos guardar una pieza para su posterior liberación (M09). Una vez la pieza ha llegado al elevador ésta asciende a un nivel superior y pasa a otra zona nueva. Estas dos zonas comentadas formarán parte de una zona cerrada con una célula anti-intrusión para evitar

aplastamientos debajo del elevador. Si se decide entrar en la zona, unas barreras fotodetectoras enviarán una señal y se parará la instalación.

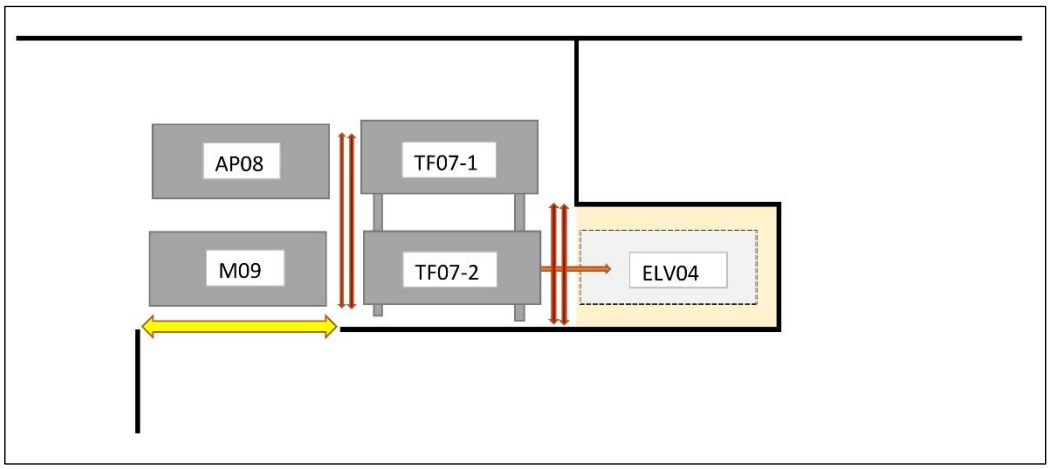


Figura 2 – Sinóptico Propuesta Zona Elevador Nivel 0

Una vez que la pieza ha llegado al nivel superior, ésta pasa por una mesa giratoria que la hará llegar a la zona de salida. La elevación contará con una pequeña velocidad para asegurarnos un correcto funcionamiento. En la parte baja del elevador existirá un cofre eléctrico en el que tendremos un led de visualización de defectos, un botón de rearme y la opción de funcionar con el elevador en modo automático o en modo manual.

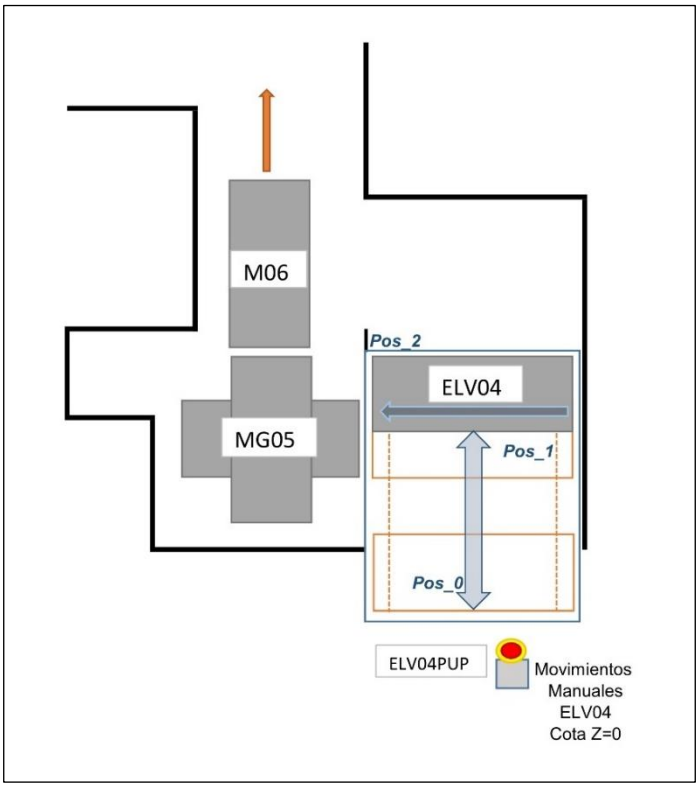


Figura 3 – Sinóptico Propuesta Zona Elevador Nivel 1

Por último, tendremos la zona de salida, en esta zona existen dos mesas-transfer de acumulo donde se pueden almacenar piezas conforme a lo dispuesto en el pliego de condiciones. Para simular la salida de piezas se usará un destructor de las mismas.

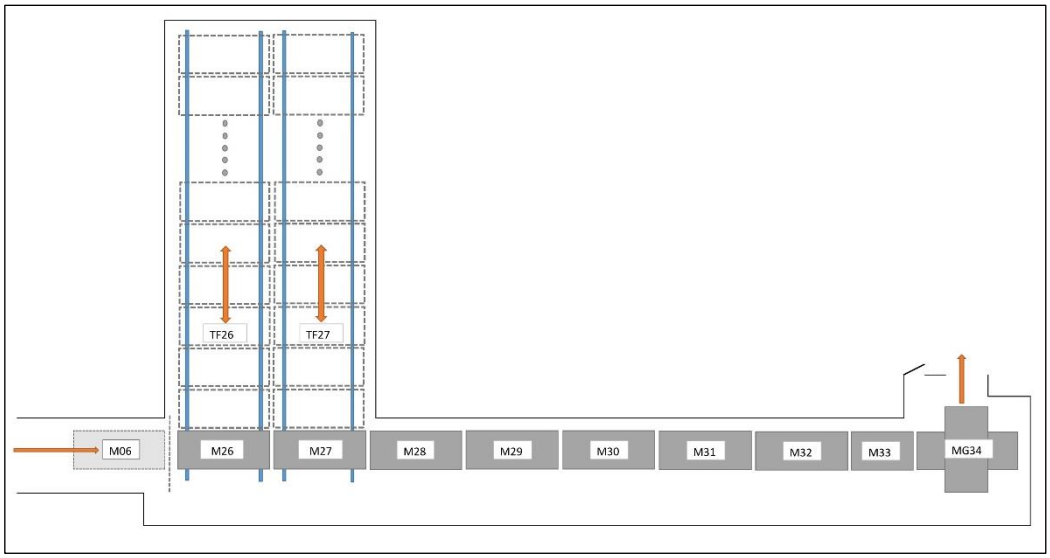


Figura 4 – Sinóptico Propuesta Zona Salida

Con todas estas descripciones ya tendríamos una idea de modelo el cual podríamos llevar a la implementación.

2.3. DESCRIPCIÓN DEL PROCESO DE SIMULACIÓN

Para realizar la simulación y los posteriores apartados de este proyecto será necesario tener instalado y configurado, como veremos más adelante el siguiente software:

- STEP7 con PLCSim (Siemens 2017)
- Factory I/O (Factory I/O s.f.)
- NetToPLCsim (NetToPLCSim s.f.)
- KEPServerEX6 (KepServerEX s.f.)
- Anaconda – Spyder (Anaconda s.f.)
    - PyQt (Qt para Phyton) (PyQt s.f.)
    - Diversas librerías para Phyton

El diagrama general que representa el modelo es el siguiente:

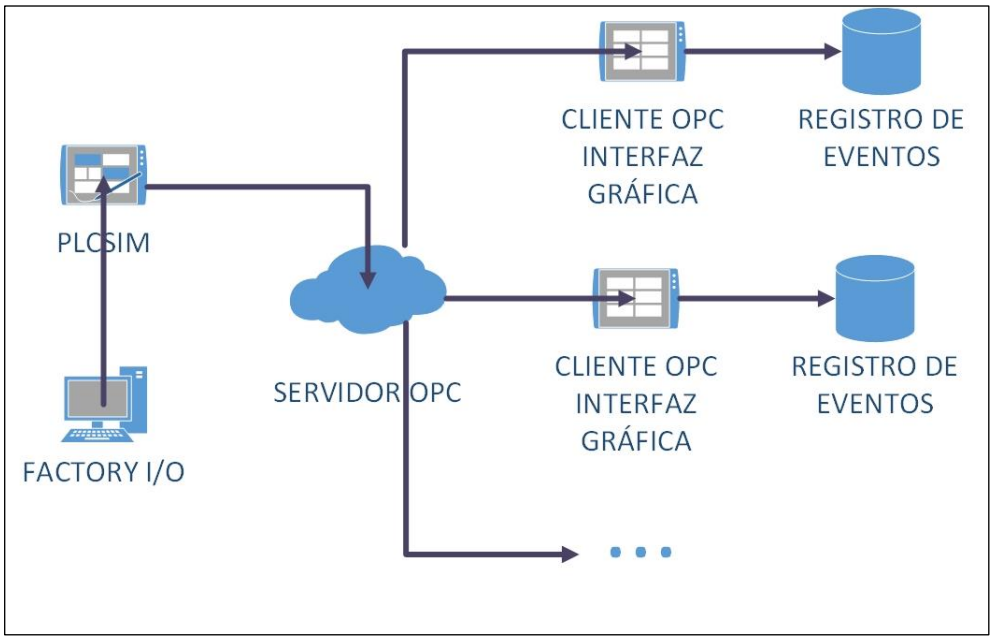


Figura 5 – Diagrama de Comunicación General

Vemos como el modelo físico simulado con el programa Factory I/O únicamente se comunica con el autómata simulado, en este caso simulando una instalación real en la que se configuran y cablean unas determinadas entradas y salidas. En una instalación real se configuraría directamente un servidor OPC sobre el autómata, pero al ser simulado mediante PLCSim es necesario el uso de más Software como vemos en la siguiente figura.

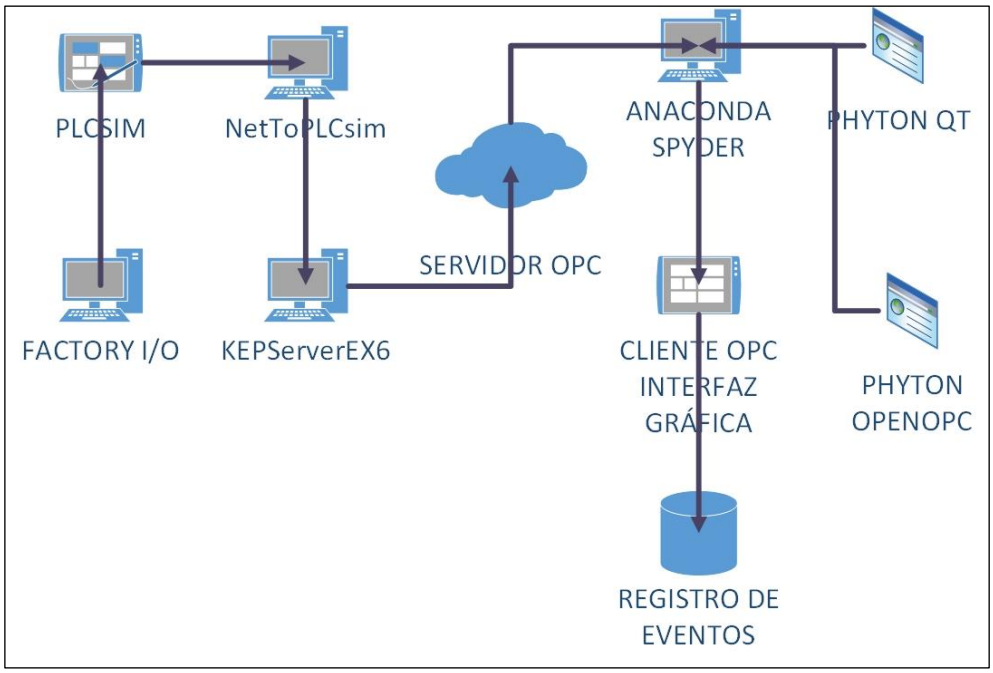


Figura 6 – Diagrama de Comunicación Software

Vemos como es necesario un programa adicional llamado NetToPLCSim para que nuestro software KEPServerEX6, que crea el servidor OPC, pueda acceder a todos los TAGS de PLCSim.

Una vez creado el servidor OPC podemos tener tantos clientes como queramos. En nuestro caso se creará un ejecutable adaptando a ser ejecutado en una Tablet o cualquier dispositivo portátil. Para programar este cliente OPC se usará un entorno de programación en Phyton mediante el software Anaconda y Spyder. Para realizar la comunicación OPC emplearemos la librería denominada OpenOPC y para realizar el interfaz gráfico emplearemos la librería Pyqt (famosas Qt gráficas en lenguaje Phyton).

En el apartado de Anexos podremos ver cuál es la forma correcta de configurar todo este software para poder simular este modelo creado.

Por último, se mostrará cual es el flujo de datos con el que atender todo el proceso de comunicaciones entre los diferentes procesos y los diferentes programas software que se usan para el desarrollo de este trabajo.

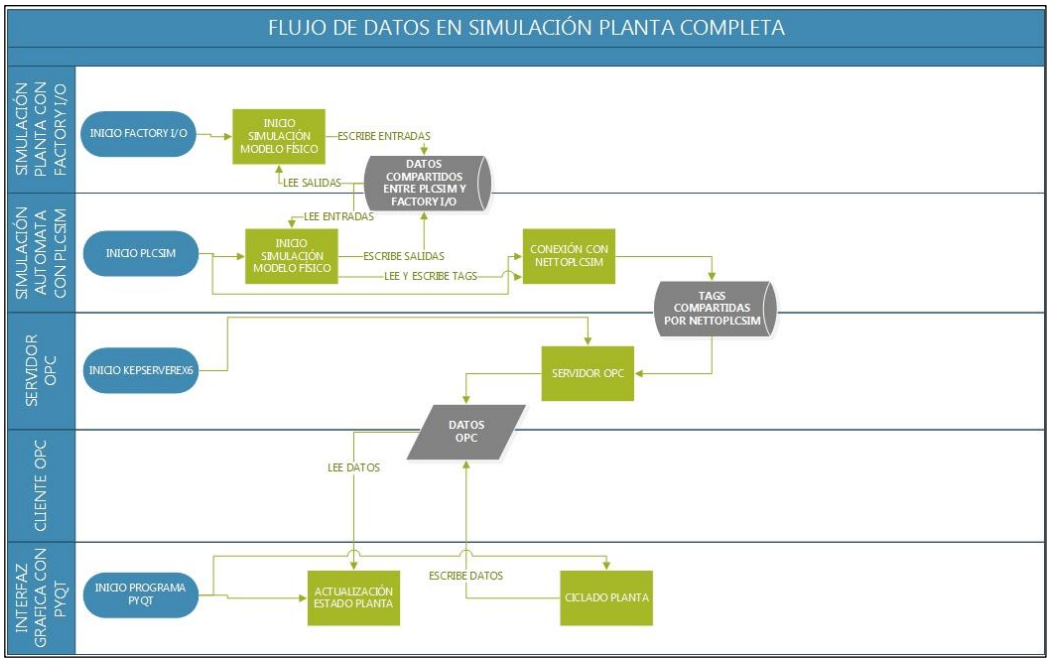


Figura 7 – Flujo de Datos Simulación

Vemos como existen tres zonas de variables compartidas, la primera de ellas no accesible al usuario que es la que comparten el programa PLCSIM y

FACTORY I/O, la segunda de ellas, tampoco accesible por el usuario, es la que comparten PLCSIM y KEPServerEX6, y la última zona de datos compartida, que sí accesible por el usuario es la creada por el servidor OPC.

Esta interconexión de programas está debidamente comentada en los Anexos y está basada en una referencia bibliográfica (González 2015).

3. ASPECTOS TEÓRICOS DEL MODELO Y CONOCIMIENTOS

En este capítulo se desarrollan los aspectos teóricos en los que nos hemos basado en la implementación del modelo. Se comentarán los aspectos teóricos relacionados con el modelo físico (mesas transportadoras, sensorización, ...), como el estándar de programación ladder para PLCs, estándar OPC de comunicación y aspectos teóricos en el análisis de datos.

3.1. ASPECTOS TEÓRICOS DEL MODELO FÍSICO

Para la implementación todas las mesas transportadoras usadas son mesas de rodillos (MIGUELETBLOG s.f.), un dispositivo mecánico que usa rodillos metálicos para facilitar el manejo y traslado de una gran cantidad de materiales, tales como cajas, llantas, palés, paquetes, etc. Siempre y cuando tengan un fondo regular.

A diferencia de otros sistemas de transporte más antiguos, un transportador de rodillos es un sistema modular que permite la combinación de segmentos con rodillos de giro libre para los puntos de operación manual de mercancía,

segmentos de rodillos accionados por gravedad, y segmentos con rodillos motorizados. Las ventajas de este tipo de transporte son:

- Proporciona un espacio extra para almacenar materiales en tránsito.
- Los objetos pueden desplazarse por los rodillos sin frenarse por completo, evitando así el formar un tapón.

Y sus principales usos industriales reales son:

- Secciones de transferencia entre bandas transportadoras.
- Para almacenaje o acumulación de cajas vacías o llenas.
- Para transferir contenedores vacíos a estaciones de trabajo o contenedores llenos al Área de Despacho.

Un ejemplo real de una cinta de rodillos motorizada es el siguiente.

Figura 8 – Cinta de Rodillos Motorizada

Otra alternativa industrial es la cinta de banda. A diferencia de la anterior esta está formada por una banda continua que se mueve entre dos tambores. La principal diferencia entra ambas bandas radica en la carga que pueden soportar, siendo mucho mayor en el transportador de rodillos que en la cinta de banda, en embargo, la cinta de banda permite el uso de piezas con base irregular aumentando la flexibilidad de la instalación industrial.

Figura 9 – Transportador de Banda

En cuanto a la sonorización prioritariamente se han usado sensores fotoeléctricos (Resnick_halliday s.f.) para la detección de presencia pieza. El principio de funcionamiento de este tipo de sensor es el siguiente: son sensores cuyos elementos de emisión y recepción están yuxtapuestos en el mismo conjunto óptico. Los rayos emitidos por el transmisor se reflejan en la superficie del objeto detectado y retornan al elemento receptor.

Figura 10 – Sensor Fotoeléctrico

Otra alternativa al uso de este tipo de sensores es el uso de sensores retroreflectivos, en esto el haz de luz rebota en el reflector y llegando al emisor si no hay un objeto entre medias. El principal inconveniente de este tipo de sensores es que necesitas instalar tanto el emisor como el reflector, y la distancia es configurable moviendo la distancia entre estos dos elementos, a diferencia de los sensores fotoeléctricos donde la distancia es configurable electrónicamente.

Si en lugar de querer detectar la presencia de pieza queremos determinar la ubicación de un elemento funcional usaremos sensores de proximidad capacitivos. Su principio de funcionamiento es el siguiente: los sensores capacitivos reaccionan ante metales y no metales que al aproximarse a la superficie activa sobrepasan una determinada capacidad. La distancia de conexión respecto a un determinado material es tanto mayor cuanto más elevada sea su constante dieléctrica.

Figura 11 – Sensor de Proximidad Capacitivo

Otra alternativa es el uso de sensores inductivos los cuales reaccionan ante la presencia de algún material metálico, estos tienen la ventaja de suelen

ser más económicos pero el inconveniente de que empeoran el rendimiento en ambientes con materiales polvorientos o granulados.

En todo el proceso de sensorización se ha intentado respetar la sonorización que se ha implementado en la instalación real.

3.2. ASPECTOS TEÓRICOS LADDER IEC 61131-3

Para la programación del autómata programable que controlará la instalación se ha seguido es estándar IEC-61131 (Standar s.f.) (John s.f.). La finalidad de esta norma es el definir los lenguajes de programación de uso más corriente, las reglas sintácticas y semánticas, el juego de instrucciones fundamental, los ensayos y los medios de ampliación y adaptación de los equipos.

En primer lugar, la normal nos comenta que existen unos elementos comunes independientes del lenguaje de programación usado. Estos elementos son:

- Tipos de datos (booleanos, enteros, reales, byte, palabra, etc.) y variables (Asignan direcciones del hardware: E/S, memoria y datos. Locales o globales).
- Modelo software (programas, tareas, recursos).
- Modelo de comunicación (variables globales, bloques funcionales, etc.).
- Unidades de organización del programa (programa, funciones, bloques funcionales).
- Elementos de configuración (recursos, tareas, variables globales, vías de acceso).

Para finalizar la norma específica cuales son los diferentes lenguajes de programación, diferenciando entre lenguajes gráficos (diagrama de escalera

o ladder y diagrama de bloques funcionales) y lenguajes literales (lista de instrucciones y texto estructurado).

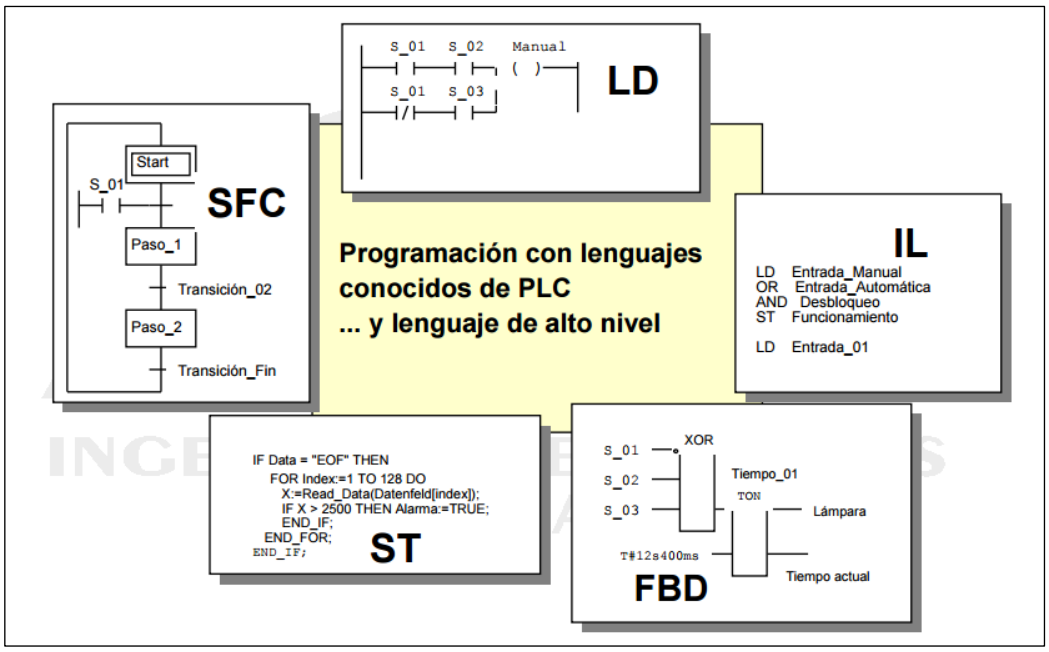


Figura 12 – Lenguajes de Programación PLCs

Como para el presente proyecto se usará el lenguaje ladder (Ingeniería s.f.) será el que se desarrolle a continuación en función a la norma IEC 61131. Este lenguaje es gráfico y deriva del lenguaje de relés. La principal ventaja de este lenguaje es que los símbolos básicos están normalizados según el estándar IEC y son empleados por todos los fabricantes.

Con este tipo de diagramas se describe normalmente la operación eléctrica de distintos tipos de máquinas, y puede utilizarse para sintetizar un sistema de control y, con las herramientas de software adecuadas, realizar la programación del PLC. Los principales elementos de este lenguaje de programación son:

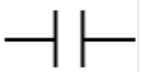
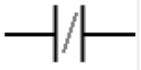
Elementos básicos en LADDER		
Símbolo	Nombre	Descripción
	Contacto NA	Se activa cuando hay un uno lógico en el elemento que representa, esto es, una entrada (para captar información del proceso a controlar), una variable interna o un bit de sistema.
	Contacto NC	Su función es similar al contacto NA anterior, pero en este caso se activa cuando hay un cero lógico, cosa que deberá de tenerse muy en cuenta a la hora de su utilización.
	Bobina NA	Se activa cuando la combinación que hay a su entrada (izquierda) da un uno lógico. Su activación equivale a decir que tiene un uno lógico. Suele representar elementos de salida, aunque a veces puede hacer el papel de variable interna.
	Bobina NC	Se activa cuando la combinación que hay a su entrada (izquierda) da un cero lógico. Su activación equivale a decir que tiene un cero lógico. Su comportamiento es complementario al de la bobina NA.
	Bobina SET	Una vez activa (puesta a 1) no se puede desactivar (puesta a 0) si no es por su correspondiente bobina en RESET. Sirve para memorizar bits y usada junto con la bobina RESET dan una enorme potencia en la programación.
	Bobina SET	Permite desactivar una bobina SET previamente activada.
	Bobina JUMP	Permite saltarse instrucciones del programa e ir directamente a la etiqueta que se desee. Sirve para realizar subprogramas.

Figura 13 – Elementos Lenguaje Programación Ladder

Si se consulta la fuente bibliográfica se puede ver con más detalle todos estos elementos.

3.3. ESTÁNDAR OPC

Para la comunicación de datos entre el autómata simulado (en su cusa se podría extender a una instalación real) y el software donde se desarrolla el HMI y se realiza el análisis de datos se realiza mediante el estándar OPC (Foundation s.f.) (OLE Process Control).

OPC (S21sec s.f.), es un mecanismo estándar de comunicación, que interconecta en forma libre, numerosas fuentes de datos donde se incluyen dispositivos de planta en la fábrica. Su arquitectura, de comunicación abierta, se concentra en el acceso a datos y no en el tipo de datos.

La arquitectura OPC es un modelo Cliente-Servidor donde el Servidor OPC proporciona una interfaz al objeto OPC y lo controla. Una aplicación cliente OPC se comunica a un servidor OPC a través de un cliente OPC específico por medio de una interfaz de automatización. El servidor OPC lleva a cabo la interfaz cliente, y opcionalmente lleva a cabo la interfaz de automatización.

En la figura se muestran diversos lenguajes de implementación de la aplicación, pero en nuestro caso usaremos Phyton.

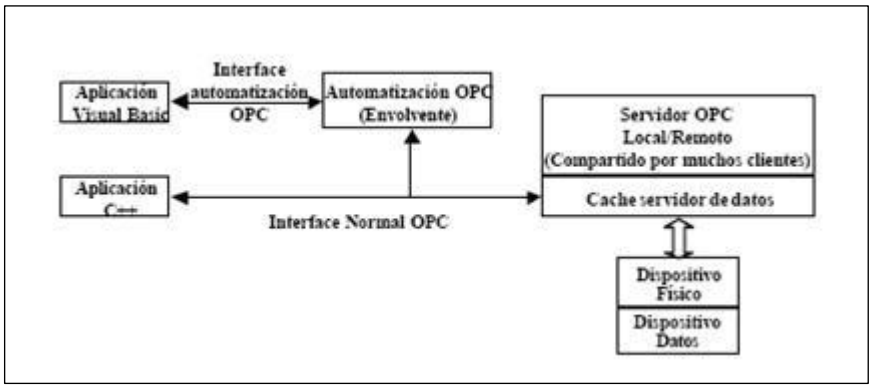


Figura 14 – Arquitectura OPC

Una aplicación cliente OPC, puede conectarse por medio de una red, a varios servidores OPC proporcionados por uno o más fabricantes. De esta forma no existe restricción por cuanto a tener un Software Cliente para un Software Servidor, lo que es un problema de interoperabilidad que hoy en día se aprecia con sistemas del tipo propietario, aumentando la flexibilidad de la aplicación.

Sistemas de control supervisorio como lo son SCADA o DCS pueden comunicarse con un Servidor OPC y proveer a este, información de los dispositivos de campo asociados. De esta forma, aplicaciones cliente OPC de otros fabricantes tendrán acceso a estos datos por medio del servidor como vemos en la siguiente figura.

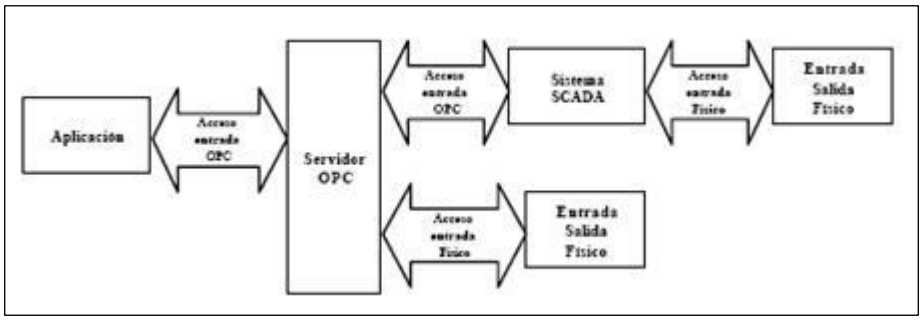


Figura 15 – Comunicación Aplicaciones-OPC

A nivel de resumen, a alto nivel, un servidor OPC está compuesto por:

- Servidor: Mantiene la información sobre si mismo, y unifica los Datos dentro de un Grupo.
- Grupo: Dota de un mecanismo que contiene en forma lógica los ítems. Se clasifican en público o Local.
- Ítem: Es un valor, una condición y permanece o varía en el tiempo. Es una dirección específica de los datos y no la fuente de datos.

Todo lo comentado pertenece al estándar OPC-DA (Data Access) pero existen unas extensiones de este que aumentan la funcionalidad, las cuales no han sido usadas en la realización del presente proyecto debido a que no son necesarias estas funcionalidades:

- Servidor de Alarmas, Condiciones y Eventos OPC: Provee de Interfaces, donde Clientes OPC son notificados de Sucesos.
- Servidor de Acceso a Datos Históricos OPC (OPC HDA): Provee de una interfaz Cliente OPC de Acceso a Datos Históricos, que facilita el uso de aplicaciones de acceso a datos.
- Intercambio de datos OPC (OPC DX): Define un conjunto de interfaces que permiten el intercambio de datos, así como la comunicación "server to server" entre dispositivos y controladores conectados a

Ethernet, que utilizan distintos protocolos. OPC-DX permite a los servidores OPC-DA intercambiar directamente datos sin la exigencia de un cliente OPC intermedio.

- Acceso de datos XML (OPC XML DA): Permite a las aplicaciones cliente ser escritas en Java, Perl, Python, y otros idiomas que soporta SOAP. SOAP y XML Web Services utiliza Protocolo de transferencia de hipertexto (HTTP) y los mecanismos de transporte y proporcionar una plataforma neutral que es más adecuado para el tráfico con base en Internet, en comparación con tecnologías como DCOM.
- Arquitectura unificada OPC (OPC UA): OPC UA integra la funcionalidad de las anteriores especificaciones (OPC DA, OPC-HDA, OPC A & E, OPC-DX, etc).

3.4. BLOXPLOTS Y CLUSTERING JERÁRQUICO

Una vez se ha creado un registro de datos, se procederá a analizar los eventos. Para ello, y de forma independiente en cada elemento funcional, se calculan el tiempo en que cada evento ha estado activo (ya que en el registro aplacen la hora de aparición y desalación), tiendo un conjunto de datos (en unidades de tiempo) sobre los que se puede realizar un análisis de datos tipo boxplot.

Para representar los datos se mostrará en forma de tabla la media y la desviación de los datos en función de los diversos tratamientos. Pero también será habitual en el informe representarlo de forma gráfica mediante boxplots (Wikipedia s.f.) o diagramas de cajas, que son unos gráficos que están basados en cuartiles y mediante los cuales se visualiza la distribución de un conjunto de datos. Está compuesto por un rectángulo, la "caja", y dos brazos, los "bigotes". Estos diagramas se crean en base a la siguiente figura.

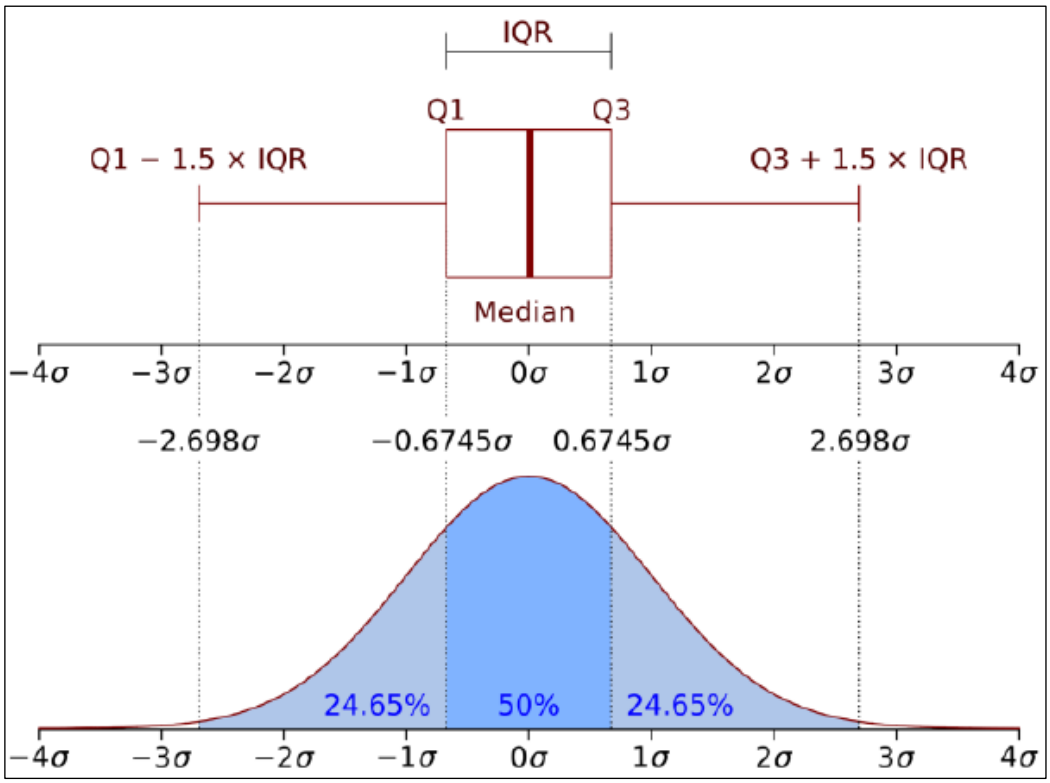


Figura 16 – Metodología Boxplot

Para calcular los parámetros Q1 y Q3 (KULLABS s.f.) se seguirán dos procesos de cálculo diferentes. La forma diferente de calcular estos valores queda reflejada en la siguiente figura.

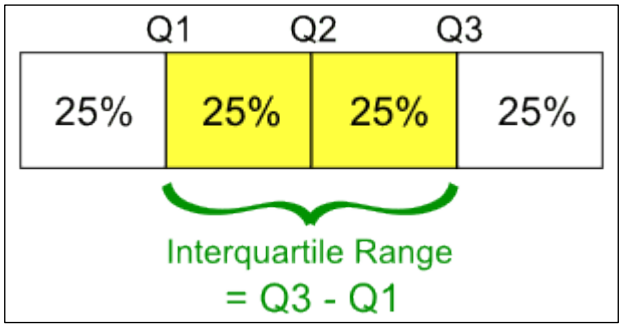


Figura 17 – Valores Q1 y Q3 en Boxplot

El cuartil Q1 es el punto donde el 25% de las observaciones quedan por debajo de él, y el Q3 donde el 25% de las muestras quedan por encima de él. El cuartil Q2 es la mediana de los datos.

Posteriormente se realizará un agrupamiento o clustering jerárquico (Jover s.f.) de datos cuyo objetivo será el Formar k agrupaciones de observaciones maximizando tanto la similitud intra-grupos como la diferencia intergrupos.

La distancia entre las observaciones permitirá realizar grupos homogéneos pero heterogéneos entre sí.

En nuestro caso de estudio como trabajaremos con medidas reales de tiempo, no será necesario eliminar valores fueras de rango, ya que todas las medidas se consideran correctas, y no será necesario realizar una normalización de ellos ya que todos están representados por la misma unidad de medida.

Para saber si existe similitud se trabaja con la distancia euclídea:

$$dist(X,Y) = \sqrt{\sum_{i=1}^{d}(X_i - Y_i)^2}$$

Ecuación 1 – Distancia Euclídea

El clustering jerárquico es útil cuando queremos obtener distintos grupos con características similares de un mismo conjunto de datos. Utiliza un algoritmo de iteración no supervisado jerárquico y aglomerativo (parten de tantos grupos como observaciones y van fusionando los grupos más similares formando clústeres cada vez más grandes hasta formar un único clúster).

Para representar este tipo de algoritmo se hará uso de los dendrogramas (Fuente s.f.) en el cual la similitud entre dos objetivos viene dada por la "altura" del nodo común más cercano.

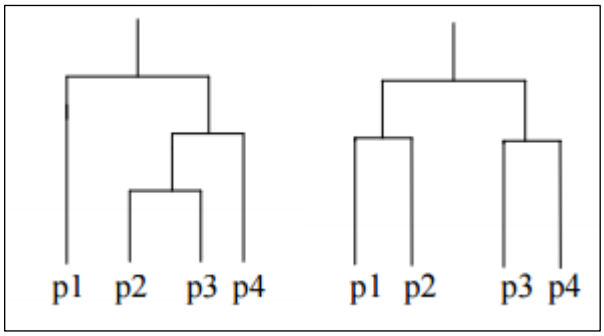


Figura 18 – Ejemplo de Dendograma

Para un conjunto de datos más grande tenemos algo como lo de la siguiente figura, en la que observamos como dos puntos muy diferentes tienen un nodo común muy alejado.

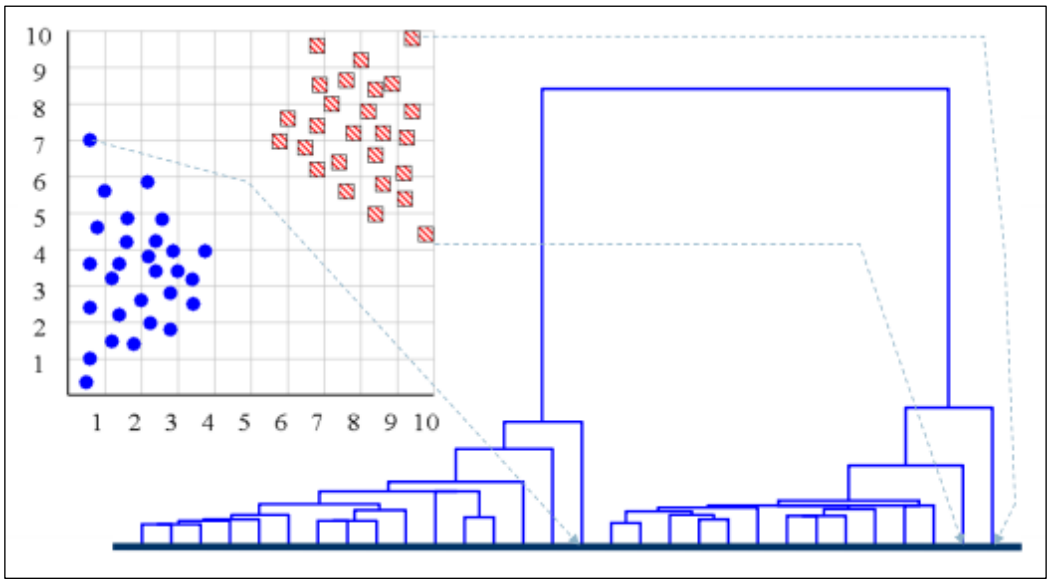


Figura 19 – Ejemplo Dendograma con Representación

4. IMPLEMENTACIÓN DEL MODELO

En este capítulo se explicará cuáles son las herramientas que se han usado para la implementación del modelo, así como la forma y el proceso seguido para realizar esta implementación. También se comentarán los resultados obtenidos y el análisis de éstos.

4.1. HERRAMIENTAS PARA LA IMPLEMENTACIÓN

4.1.1. IMPLEMENTACIÓN DEL MODELO FÍSICO

Para la implementación del modelo físico se ha usado el software Factory I/O (Factory I/O s.f.) que nos permite la simulación de instalaciones industriales. Es un software muy completo ya que cuenta con un gran interfaz gráfico además de poder establecer conexión con autómatas reales de diversas marcas como con la simulación software de los autómatas que lo permitan.

Las principales características de este software son:

- Innovación en 3D para el aprendizaje de PLC: Cuenta con 20 escenarios industriales ya creados con lo que el usuario tendrá que diseñar un programa de autómata para hacerlo funcionar.
- Librería con más de 80 componentes industriales: Se incluyen sensores, transportadores, ascensores, estaciones y muchos otros, la mayoría de ellos orientados al transporte de piezas y logística.
- Posibilidad de crear nuestros propios escenarios: Todos los componentes se pueden ubicar en cualquier situación además de que la posición y el giro es definido por el usuario.
- Todos los componentes se pueden configurar con señales analógicas o digitales.
- FACTORY I/O utiliza drivers para interaccionar con PLC, SoftPLC, Modbus y muchas otras tecnologías: Cada edición incluye un paquete de drivers para una tecnología específica (por ejemplo, Allen-Bradley Edition, Siemens Edition…).
- Diagnóstico de acerías: Desarrolla estrategias para el diagnóstico de averías provocando fácilmente fallos del tipo Contacto Abierto o Cortocircuitado en sensores y actuadores.
- Fácil integración: Integre fácilmente FACTORY I/O con el equipo de aprendizaje disponible. Enchufe y ponga en funcionamiento con Siemens y Allen-Bradley PLC (Ethernet).

Otro Software disponible para el modelado de instalaciones industriales puede ser FlexSim o Promoverá. Una breve comparativa entre estos programas software queda resumida en la siguiente tabla (Pedersen s.f.):

ASPECTO A COMPARAR	FACTORY I/O	FLEXSIM	PROMODEL
Variedad de librerías	Más de 80 componentes industriales	Cientos de componentes	Cientos de componentes
Orientado a qué Tipo de Industria	Orientado a la educación	Fabricación, minería, logística, petrolera, etc.	Aeroespacial, defensa, salud, farmacéutica, servicios, etc.
Método de Simulación	Conexión con PLC real o simulado	Tags Internos con funciones	Modelo gráfico interno
Complejidad de Uso	Baja	Alta	Alta
Precio de Licencia	253€/año 1 mes gratis	Gratis para estudiantes	35€ para estudiantes
Comparativa Gráfica			

Tabla 1 – Comparativa Herramientas Implementación Modelo Físico

IMPLEMENTACIÓN DEL MODELO FÍSICO

Nuestra instalación simulada con el software Factory I/O es muy similar a la instalación real, cuenta con dos niveles comunicados a través de un elevador. Este primer nivel simulado es el siguiente:

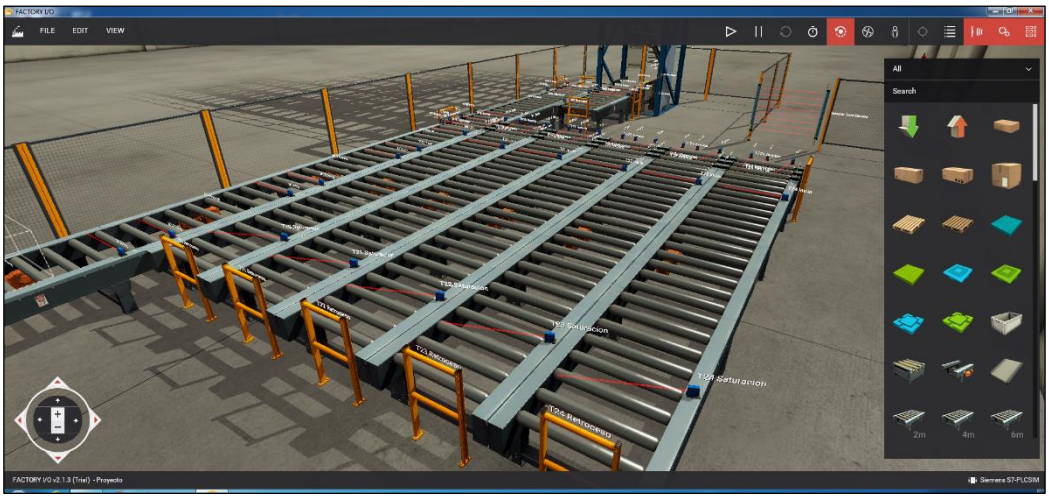


Figura 20 – Nivel 0 Simulación Instalación Factory I/O

Y el segundo nivel es:

Figura 21 – Nivel 1 Simulación Instalación Factory I/O

Vemos como existen todos los elementos listados anteriormente. Para ver cómo están implementados cada uno de los elementos, se procede a explicar cómo son las mesas de tránsito, los transferes de tránsito, los transferes de acúmulo, las mesas giratorias, y el elevador.

MESAS DE TRÁNSITO

Por ejemplo, para la mesa 31 tenemos la siguiente distribución:

Figura 22 – Distribución Mesas de Transito

Esta distribución es igual para los siguientes elementos: M15, M14, M13, M12, AP08, M09, M06, M28, M29, M30, M31, M32 y M33.

A cada mesa además se le han añadido dos sensores, uno al inicio de la mesa que se correspondería con el sensor de desfasaje, y el otro al final de la mesa que se correspondería al sensor de presencia. Estas mesas se pueden configurar para que se muevan en una sola dirección, para que se muevan en dos direcciones, en avance y retroceso, o para que mediante una señal analógica podamos simular un variador de velocidad. En nuestro caso todas las mesas se habrán configurado para que se muevan solo en avance.

TRANSFER DE TRANSITO

Por ejemplo, para el transfer TF21 tenemos la siguiente distribución:

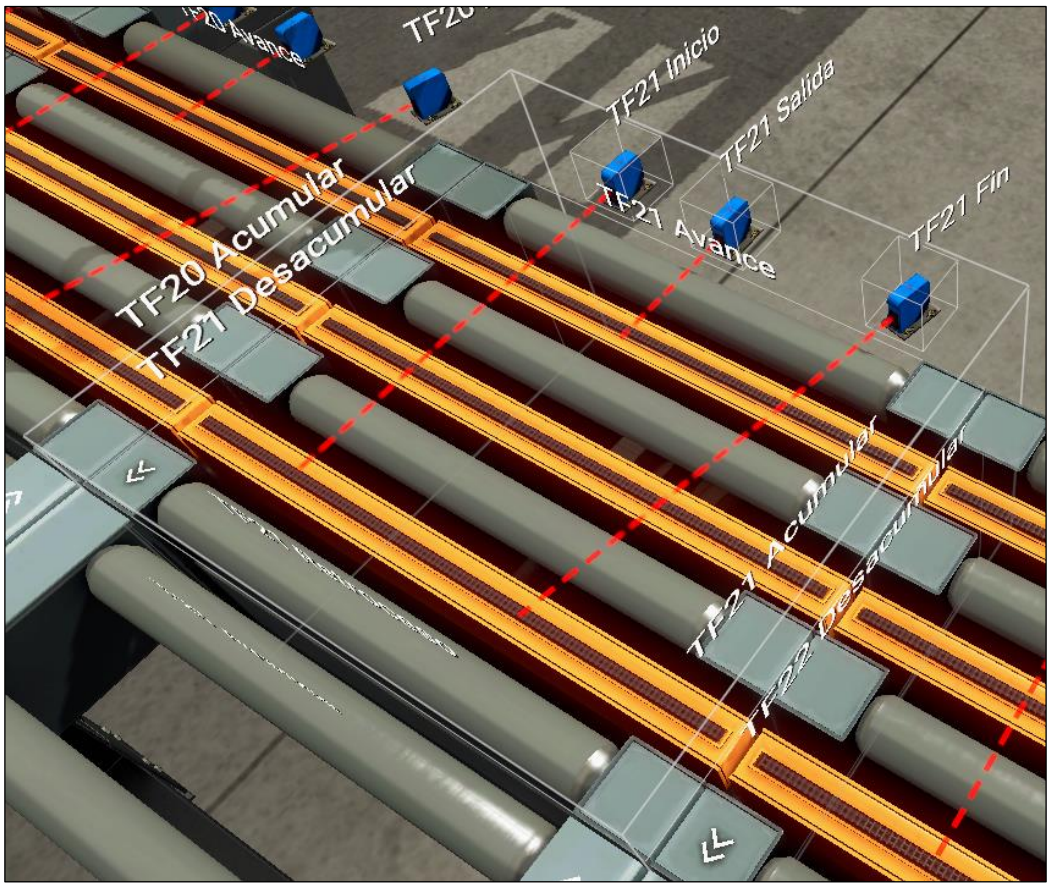


Figura 23 – Distribución Transfers de Transito

Esta distribución es igual para los siguientes elementos: TF10, TF20, TF21, TF22, TF23, TF24, TF0-1, TF07-2, M26 y M27.

Cada transfer de este tipo cuenta con tres sensores, ubicados en diferentes posiciones en función del sentido de transito del transfer. Existe un sensor de desfasaje (inicio) y otro de presencia (fin) al igual que en las mesas, y además cuentan con un sensor de salida el cual es usado cuando las piezas llegan desde la dirección opuesta (por ejemplo, para el TF21 el sensor de salida se usa para final el tránsito desde T21 a TF21).

Estos transfers se pueden mover de 4 formas, en avance de rodillos, en retroceso de rodillos, en avance de bicadena o en retroceso de bicadena, aunque en función del posicionamiento de cada transfer solo se usarán hasta tres de estos movimientos.

TRANSFER DE ACÚMULO

Por ejemplo, para el transfer T21 tenemos la siguiente distribución:

Figura 24 – Distribución Transfers de Acúmulo

Esta distribución es igual para los siguientes elementos: T20, T21, T22, T23, T24, TF26 y TF27.

Este tipo de transfers permiten acumular entre 4 y 5 piezas, y su distribución es muy similar al de las mesas de tránsito. Cuentan con dos sensores, uno de inicio y otro de saturación, pero a diferencia de las mesas, éstos permiten moverse en avance y en retroceso.

MESAS GIRATORIAS

Por ejemplo, para el transfer MG25 tenemos la siguiente distribución:

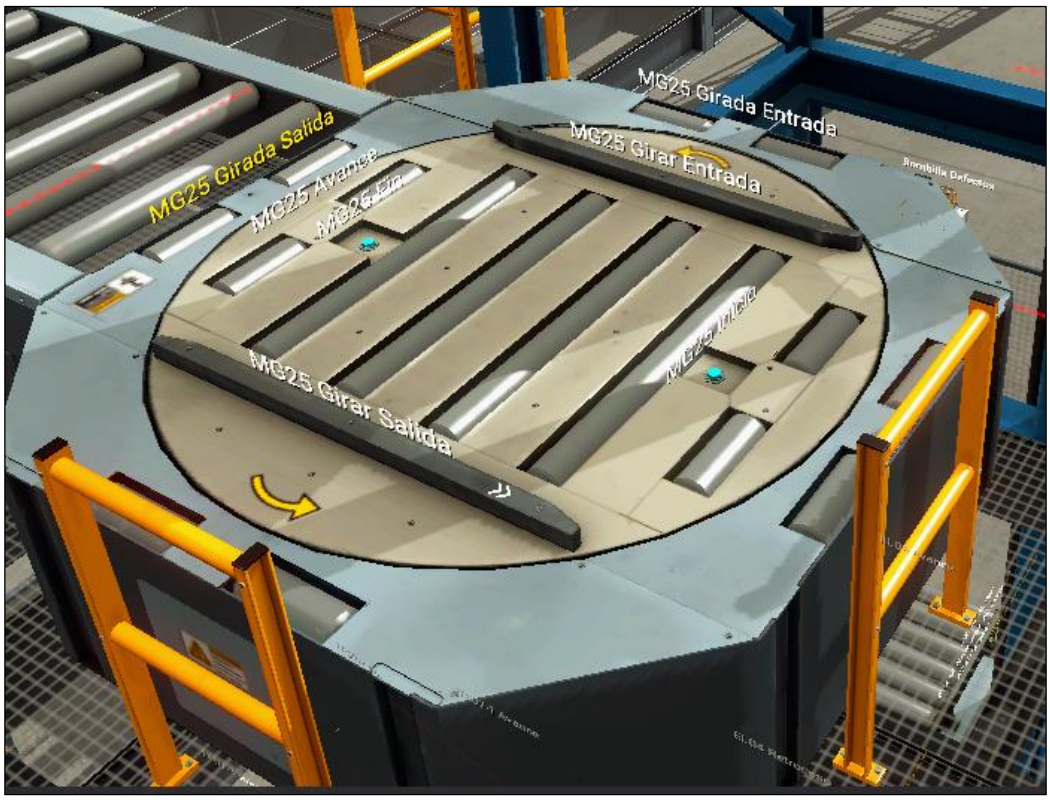


Figura 25 – Distribución Mesas Giratorias

Esta distribución es igual para los siguientes elementos: MG25 y MG34.

Estas mesas cuentan con cuatro sensores, los que hemos comentado hasta ahora, el sensor de inicio o desfasaje y el sensor de fin o presencia, y además cuentan con dos sensores, el sensor de mesa girada para entrada de pieza y el sensor de mesa girada para salida de pieza.

En estas mesas los rodillos se pueden mover en las dos direcciones, pero para nuestro modelo solo será necesario que se muevan en avance.

ELEVADOR

Por ejemplo, para el transfer EL04 tenemos la siguiente distribución:

Figura 26 – Distribución Elevador

Nuestra instalación únicamente cuenta con un elevador que tiene como mnemónico EL04.

En cuanto a sensores tenemos los siguientes: La zona de rodillos cuenta con dos sensores, uno de desfasaje y otro de presencia. La zona de elevación cuatro sensores, uno de posicionamiento abajo, otro de posicionamiento arriba, otro de pequeña velocidad arriba y otro de pequeña velocidad abajo.

En cuanto a los actuadores en este caso es necesario configurar 5 salidas, el avance de rodillos, el retroceso de rodillos, el descenso, el ascenso y el bit de pequeña velocidad.

OTROS ELEMENTOS FUNCIONALES

En este punto se describirán tres elementos funcionales, pero que no afectan al normal funcionamiento de la instalación, aunque forman parte del modelo para hacerlo más fiel al modelo real.

COFRE DE MOVIMIENTOS MANUALES

Figura 27 – Cofre de Movimientos Manuales

En este cofre se han implementado los controles más comunes de una instalación, la parada de emergencia, la visualización de defectos, el rearme

y un modo de funcionamiento manual para el elevador, simulando un cofre de la instalación.

La parada de emergencia al igual que una real devuelve siempre un bit positivo excepto cuando está pulsada, el led de visualización de defectos parpadea cuando existe algún defecto en la instalación, también visible mediante una bombilla de "llamada a operario", y un botón de rearme para poder rearmar la instalación después de haberse producido el defecto.

En cuanto a los movimientos manuales cuenta con un selector de manual/automático para el EL04, y un botón de avance y otro de retroceso, parpadeando cuando el elevador pueda subir o bajar, y fijos cuando el elevador esté arriba o abajo.

ZONA CERRADA

Mediante el uso de una barrera impedimos la entrada a la zona cerrada. La barrera usada en la siguiente:

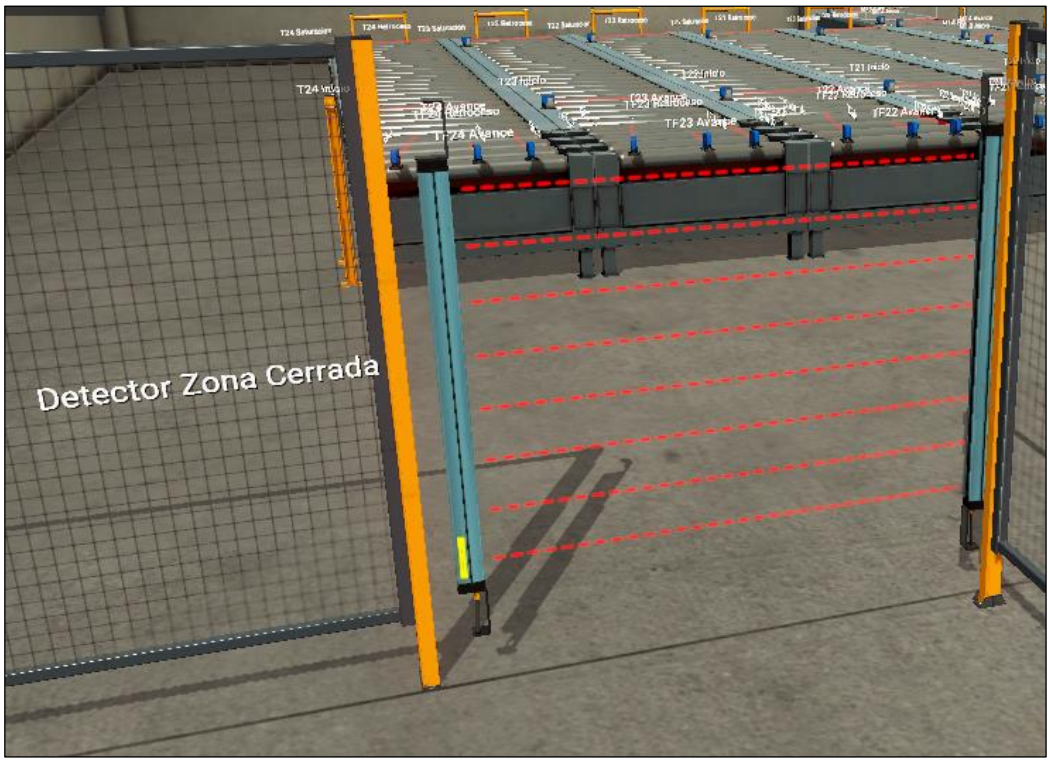


Figura 28 – Barrera Zona Cerrada

Y esta zona afecta a todo el nivel inferior como vemos en la siguiente figura:

Figura 29 – Zona Cerrada

Se observa como todo el nivel inferior está vallado impidiendo la entrada a la zona, a excepción de la zona con la barrera fotorreceptora.

CREACIÓN Y DESTRUCCIÓN DE PIEZAS

Existen dos elementos que permiten simular la entrada de piezas en la instalación y la salida de piezas. Estos elementos son controlados por el autómata y permiten activarlo o desactivarlo.

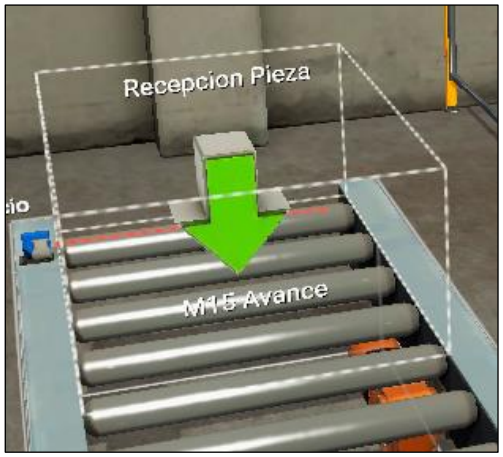


Figura 30 – Creación de Piezas

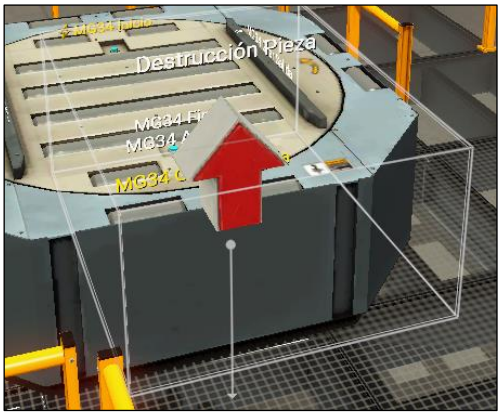


Figura 31 – Destrucción de Piezas

La recepción de piezas se hará sobre la M15 y la destrucción de piezas se hará a la salida de MG34.

OTROS ELEMENTOS NO FUNCIONALES

En este apartado se describirán otros elementos que no afectan al normal funcionamiento de la instalación, pero que tienen que ver con la mecánica o con la distribución de la instalación real.

ESCALERAS

Como el modelo real cuenta con unas escaleras para acceder al nivel superior, el programa de simulación Factory I/O cuenta con una vista en persona, la cual no le permite atravesar vayas ni subir de nivel sin escaleras, se han situado unas escaleras como vemos en la siguiente figura:

Figura 32 – Escaleras y Nivel Superior

TOPES y VALLAS DE SEGURIDAD

En cuanto a los topes de seguridad al final de todos los transfers de acúmulo, y en los laterales de los transfers de tránsito se han instalado unas vallas pequeñas que simularían un posible tope físico para evitar que la pieza caiga y transite en direcciones no correctas en función del flujo de piezas.

Figura 33 – Topes de Seguridad Simulados

Por otro lado, en el nivel superior y en las escaleras se han instalado unas vallas de seguridad sobre la plataforma para evitar la simulación de posibles caídas cuando está seleccionada la vista de personas. Por ejemplo, en una zona de la plataforma y escaleras podemos ver estas vallas de seguridad.

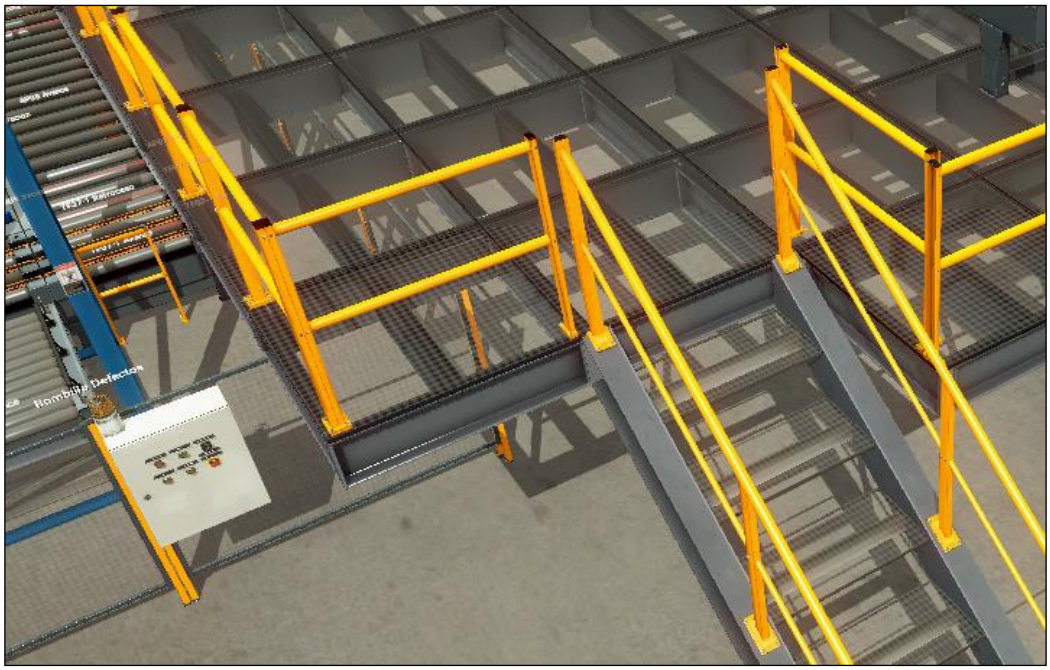


Figura 34 – Vallas en Plataforma y Escaleras

PIEZAS

Y por último y más importante, para realizar la simulación del modelo no podemos incluir los trineos con los que cuenta la instalación real, ya que la librería del software Factory I/O no cuenta con este módulo. A cambio se han seleccionado unos pallets de madera sobre los que irán montados diferentes cajas de cartón y con los que se puede simular el tránsito de piezas.

En la siguiente figura podemos ver los diferentes tipos de cajas usadas para simular la instalación.

Figura 35 – Cajas y Pallets en Simulación

4.1.2. IMPLEMENTACIÓN DEL MODELO DE AUTÓMATA

Para la implementación del modelo autómata se ha escogido un simulador de Autómatas Siemens PLCSim v5.4 SP5 (Siemens 2017) ya que es la única marca que cuenta con un simulador de programa autómata para Windows compatible con Factory I/O.

Para la creación del programa se usará el software Siemens STEP7 V5.5 SP4. Este software cuenta con las siguientes características:

- Capacidad de usar lenguajes de programación KOP (diagrama de contactos). AWL (lista de instrucción) y FBS (diagrama de funciones).
- Capacidad de ver online el programa autómata ejecutado.

- Posibilidad de configurar el hardware con diferentes dispositivos y diferentes módulos de entrada salida.
- Comunicación directa con PLCSim, software que usaremos para la simulación del modelo autómata y que cuenta con las siguientes características:
    - Posibilidad de establecer diferentes tipos de conexión (MPI, TCP/IP, Profinet).
    - Ejecución a tiempo real (dependiendo de las características del sistema operativo).
    - Posibilidad de acceder a las variables de simulación con otros programas software.

Otras marcas que comercializan software de desarrollo de aplicaciones de PLC son: Schneider y Rockwell. Como resumen de comparación entre estos tres programas de software mostramos la siguiente tabla:

ASPECTO A COMPARAR	SIEMENS	ROCKWELL	SCHNEIDER
Cuota de Mercado	28.8%	19.1%	8.3%
Integración con Soluciones de Seguridad	Si	Si	No integrado
Compatibilidad Comunicaciones BUS	Profibus, Profinet	ControlNet, DeviceNet, Ethernet/IP	Modbus, Modbus TCP
Software de Simulación	Si	No	No

Tabla 2 – Comparativa Herramientas Implementación Modelo Autómata

Aparte de estas herramientas software necesitamos una más, debido a que Siemens no ofrece software propio para acceder a las variables o marcas del autómata simulado. Para ello usaremos el software Nettoplcsim v1.2.1.0 (NetToPLCSim s.f.).

NetToPLCSim extiende el Software de Simulación PLC de Siemens "PLCSIM" por una interfaz de red TCP/IP, por lo que puede probar el modelo HMI/SCADA junto con PLCSIM sin hardware real. El resultado es que los datos están accesibles en una red TCP/IP simulada.

Las características de este software son:

- Soporta hasta 100 clientes simultáneos
- Es posible acceder a DBs, contadores, temporizadores, marcas entradas y salidas del autómata simulado.
- Soporta varias instancias de PLCSim diferentes de forma simultánea.
- Mejora el rendimiento de la transmisión de los datos.
- Soporta funciones propias de la CPU simulada tales como arranque/paro, gestión de tiempos de sistemas, acceso al estado de la CPU.

IMPLEMENTACIÓN DEL MODELO DE AUTÓMATA

MODELO HARDWARE

La configuración Hardware elegida para la elaboración de este trabajo es la siguiente:

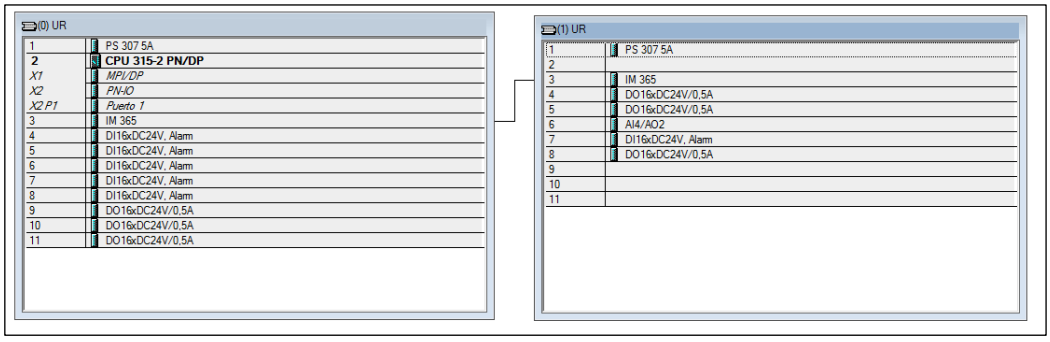


Figura 36 – Configuración Hardware STEP7

Vemos como la CPU elegida es la 315-2 PN/DP, debido a que necesitamos una CPU con tecnología de conexión TCP/IP. Posteriormente se han elegido tantos módulos de entradas salidas como entradas salidas necesita nuestro modelo físico, necesitando en este caso dos bastidores para poder conectar todos los módulos de entrada salida.

MODELO SOFTWARE

Para la configuración del software que controlará el modelo se ha usado el lenguaje de programación ladder, usando una FC de siemens para cada elemento funcional de la instalación, siendo todas estas FC llamadas desde la ejecución cíclica principal, es decir, el OB1.

A la hora de programar el FC de isla/zona se han tenido en cuenta las siguientes consideraciones:

- Si no se recibe la señal TRUE de la seta de emergencia se para la instalación completa hasta que no se vuelve a rearmar.
- Si se detecta una intrusión en la zona cerrada se para la instalación completa hasta que no se vuelve a rearmar.
- Si en las ordenes OPC (comentadas posteriormente en el apartado 8) se recibe tanto la señal de apilado como la de desapilado para un mismo transfer se para la instalación completa hasta que no se vuelve a rearmar.

En cuanto de forma generalizada para todos los elementos funcionales:

- La presencia de una mesa se monta una vez que la pieza ha llegado al sensor de fin o presencia, momento en el que se desmonta la presencia de la mesa anterior, es decir la presencia de la mesa anterior no se desmonta una vez el pallet haya abandonado la mesa, sino una vez nos hayamos asegurado de que ha llegado al siguiente destino.
- Los tránsitos se inician cuando hay presencia en la estación de origen y no hay presencia en la estación de destino.
- Si el tránsito está montado durante un determinado periodo de tiempo, se para la estación afectada hasta que no se rearme.
- Si en una estación hay presencia y no está presente el sensor de presencia, o viceversa, y aún sin haber ningún tránsito, se para la estación afectada hasta que no se rearme.

Para los transfers de tránsito, se tienen en cuenta estas consideraciones:

- Si hay un tránsito que afecta a un determinado transfer, todos los demás tránsitos que puedan afectar a ese transfer están bloqueados.

Para las mesas giratorias:

- Si no tienen presencia, se giran en posición entrada, y si tienen presencia, giran a posición salida.
- Si se envía la señal de girar durante un determinado periodo de tiempo, se para la estación afectada hasta que no se rearme.

Y para el elevador:

- Si no tiene presencia el elevador baja, y con presencia sube.
- La pequeña velocidad solo afecta al ascenso, que es cuando va con pieza y es más crítico. Con movimientos manuales siempre funciona a pequeña velocidad.
- Si se envía la señal de ascenso o descenso durante un determinado periodo de tiempo, se para la estación afectada hasta que no se rearme.

CONEXIÓN PLCSIM CON FACTORY I/O

Para poder conectar el modelo físico simulado con el programa autómata en simulación es necesario tener configuradas tantas cartas de entradas salidas de autómata sean necesarias como entradas salidas tenga el modelo físico. Una vez tenemos estas cartas de entradas salidas, es necesario que las direcciones de entradas desde la primera a la última estén contiguas, al igual que para las salidas.

CONEXIÓN PLCSIM CON SERVIDOR OPC

Como siemens no tiene un software de simulación que permita crear un servidor OPC desde un modelo simulado es necesario tener software adicional, denominado NetToPLCSim, pero hay que tener en cuenta que si

tenemos PLCSIM funcionando con este software no podemos establecer conexión online con STEP7 ni viceversa.

La forma de configurar este programa está debidamente explicada en el anexo 1.

Una vez establecida esta conexión, y abierto un cliente OPC que demande datos, este programa debe de mostrarnos las demandas de ítems que realizamos como vemos en la siguiente figura.

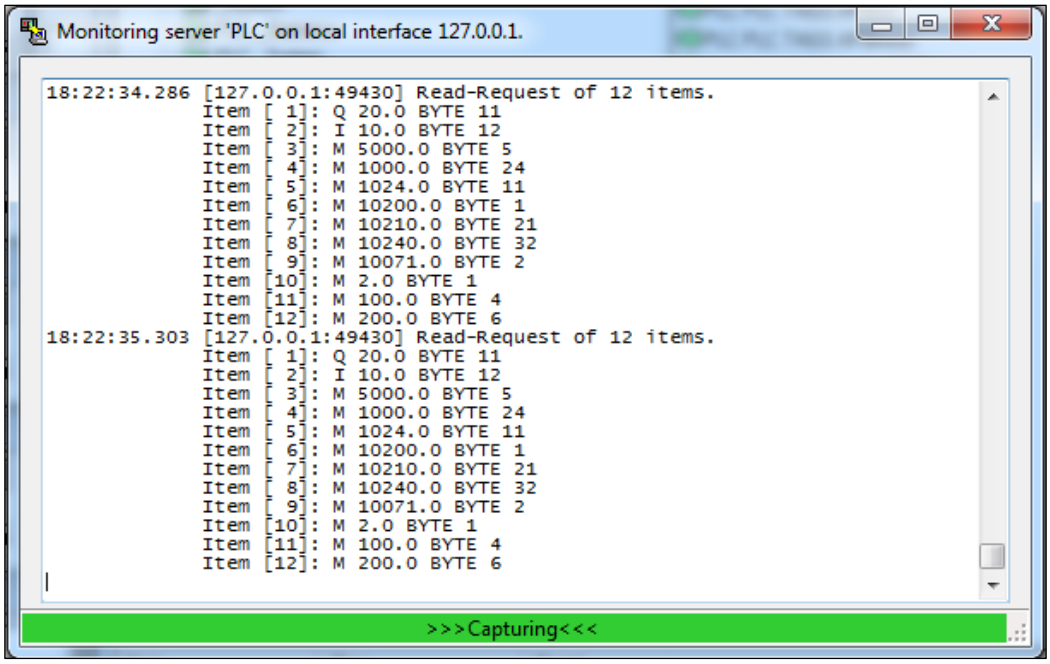


Figura 37 – Demanda de Datos A Través de NetToPLCSim

4.1.3. IMPLEMENTACIÓN DEL SERVIDOR/CLIENTE OPC

Una vez que tenemos los datos accesibles con el uso ya comentado de NetToPLCSim podremos crear nuestro servidor OPC. Para crear este servidor usaremos el software KEPServerEX 6.1.601.0 (KepServerEX s.f.) que directamente accede a estos datos configurándolo correctamente. Las características más destacadas de este software son:

- Capacidad de crear una única fuente de datos a partir de diferentes plataformas.

- Compatible con VMware e Hyper-V para el despliegue en redes públicas y privadas.
- Se integra con las aplicaciones de IT para el acceso a los datos de la planta en cualquier momento y desde cualquier lugar.
- Seguridad mejorada a través de SSL y TLS para comunicaciones seguras, autenticadas y cifradas a través de varias topologías de red.
- Múltiples opciones de redundancia para asegurar resiliencia, confiabilidad y tiempos en aplicaciones críticas.
- Interfaz simplificada para instalación, configuración, mantenimiento y resolución de problemas sencillos.
- Optimiza las comunicaciones y reduce la carga de la red y del dispositivo a través del acondicionamiento y la reducción de datos, el equilibrio de carga personalizado y la optimización de comunicaciones específicas del protocolo.
- Aísla las comunicaciones de dispositivos y aplicaciones para la resolución de problemas, ofreciendo diagnósticos OPC en tiempo real e históricos de eventos OPC y diagnósticos de comunicaciones para capturar las tramas de protocolo transferidas entre el servidor y cualquier dispositivo.

Otra alternativa al uso de KEPServerEX es Matrikon OPC Server, del cual no se realizará un estudio comparativo debido a que la conexión con NetToPLCSim es mucho más compleja.

El cliente OPC junto con el interfaz gráfico que veremos más adelante se programa en Phyton. Se ha elegido este lenguaje de programación debido a la potencia que tiene (librerías, complementos, lenguaje orientado a objetos), y debido a que también es compatible con las Qt, herramienta que se usará para el interfaz gráfico.

Como interfaz de programación se usará Anaconda con Phyton 2.7 v4.3.1 32bits (Anaconda s.f.). Anaconda es un software muy extenso que tiene una línea de comandos con la que podemos instalar directamente una librería, y

usaremos esta funcionalidad para instalar la herramienta OpenOPC 1.3.1 32bits Phyton 2.7 (OpenOPC s.f.). Esta herramienta tiene las siguientes características:

- Facilidad de uso: Debido a que la biblioteca OpenOPC implementa un número mínimo de funciones Python, es fácil de aprender y fácil de recordar. En su forma más simple, puede leer y escribir elementos OPC tan fácilmente como cualquier variable en un programa Python.
- Soporte multiplataforma: Soporta su implementación independientemente del sistema operativo usado.
- Estilo de programación funcional: OpenOPC permite encadenar las llamadas OPC en un estilo de programación elegante y funcional. Por ejemplo, puede leer los valores de todos los elementos que coincidan con un patrón de comodín utilizando una sola línea de código Python.
- Diseñado para lenguajes dinámicos: La mayoría de los toolkits de OPC están diseñados para usar con lenguajes estáticos del sistema (como C ++ o C#), proporcionando una correlación cercana a los métodos Win32 COM. OpenOPC descarta este engorroso modelo y en su lugar intenta aprovechar las características de lenguaje dinámicas proporcionadas por Python. OpenOPC es también uno de los pocos toolkits OPC-DA disponibles para cualquier lenguaje dinámico.

Otras alternativas a OpenOPC pueden ser python-opcua y PyOPC. Se muestra una tabla comparativa entre estas herramientas.

ASPECTO A COMPARAR	OPENOPC	PHYTON-OPCUA	PYOPC
Topología OPC Usada	OPC-DA	OPC-UA	OPC-DA
Forma de Implementación	Se basa en Phyton	Se convierte desde C++	Se basa en Windows
Compatibilidad	Cualquier Sistema Operativo	Cualquier Sistema Operativo	Solo Windows
Facilidad de Implementación	Fácil, solo permite operaciones simples	Complejo con muchas funcionalidades	Complejo con muchas funcionalidades

Tabla 3 – Comparativa Herramientas Cliente OPC

IMPLEMENTACIÓN DEL SERVIDOR/CLIENTE OPC

En base a los datos accesibles servidos por el programa NetToPLCSIm como veíamos en el apartado anterior podemos crear un servidor OPC con el programa KEPServerEX6, con el cual además podemos acceder a los tags de bits de memoria, de entrada, de salida, y de DBs de nuestro programa OPC. Para saber cómo configurar este programa podemos leer el anexo 1. En la siguiente figura vemos un diagrama de red del cliente servidor OPC.

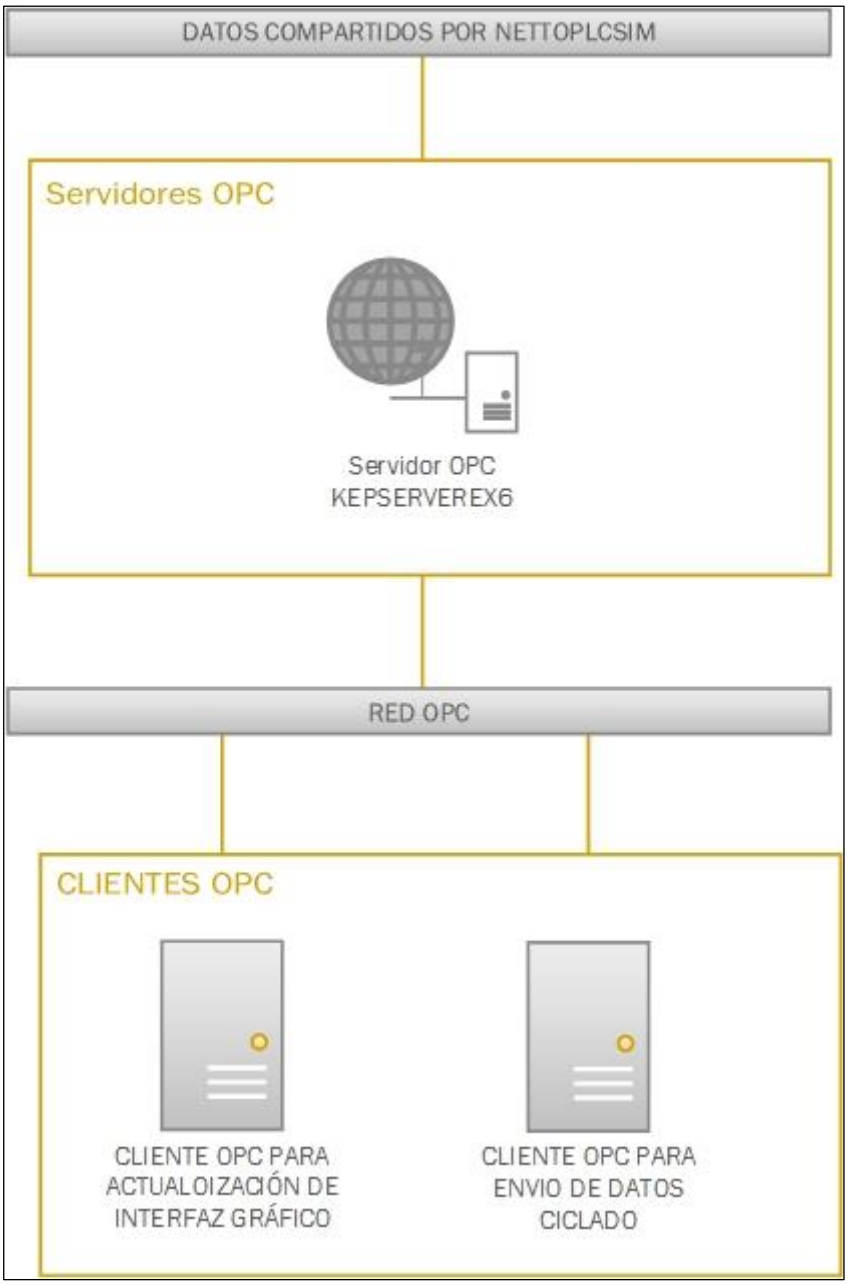


Figura 38 – Red de Datos OPC

Vemos como existe una red inicial en la que están accesible todos los datos del autómata, tanto para lectura como para escritura, conectado al autómata mediante el programa NetToPLCSIm, a esta red (no accesible mediante un cliente OPC) puede conectarse el software KEPServerEX6, el cual también podría acceder de forma directa a un autómata real conectado a una red TCP/IP.

En cuanto al cliente OPC, será programado en Python con la librería OpenOPC, la cual enseñamos a configurar en el anexo 1. Una vez instalada

esta librería la conexión con el servidor OPC es muy sencilla, únicamente tenemos que hacer uso de las siguientes funciones:

- opc=OpenOPC.client(): Con esta función creamos una variable que hemos llamado opc (en realidad es una clase con sus correspondientes atributos y funciones) donde estarán incluidas todas las configuraciones necesarias para establecer la conexión.
- opc.connect('Kepware.KEPServerEX.V6'): A través de su función conect, y al ser una red que se encuentra en el mismo equipo, no necesitamos configurar un puerto y una IP, en este caso configuramos en mnemónico que tiene el servidor OPC en el equipo.
- itemsOPC = opc.read(opc.list('PLC.PLC.TAGS')): A través de su función read podemos leer tag a tag, especificando el nombre del tag (en ese caso el argumento de la función sería 'PLC.PLC.TAGS.ITEM'), o como en nuestro caso leemos todos los tags en cada ejecución asociándolos a una variable tipo struct, que guarda el nombre del tag, el valor, la calidad, la fecha de lectura, etc. Esto es posible ya que todos nuestros tags pertenecen a un grupo denominado TAGS y podemos leer un grupo completo.
- opc.write(('PLC.PLC.TAGS.Desapilar1T24', False)): A través de su función write podemos escribir en un tag especifico, en este ejemplo escribiremos el valor False en el tag 'Desapilar1T24'
- opc.close(): A través de esta función cerramos la conexión con el servidor OPC de forma definitiva.

Nuestro cliente OPC está programado mediante hilos en Python, de tal forma que diseñamos un hilo periódico que se ejecuta con un periodo predeterminado.

La función que ejecuta el hilo queda resumida de la siguiente forma:

```
#Hilo en el que programamos el cliente OPC
def clienteOPC(obj):
    global itemsOPC
    Establecemos conexión con el servidor
    opc=OpenOPC.client()
    opc.connect('Kepware.KEPServerEX.V6')
    finalizarClienteOPC es true cuando se desea finalizar el hilo
    while not obj.finalizarClienteOPC:
        Efectuamos la lectura de datos
        itemsOPC = opc.read(opc.list('PLC.PLC.TAGS'))
        Efectuamos la escritura de datos (más líneas en el código real)
        if((obj.ui.OPCRearme.isChecked()==True) and ((itemsOPC[238])[1]==False)):
            opc.write(('PLC.PLC.TAGS.RearmeOPC', True))
        Hacemos el hilo periódico
        time.sleep(periodo)
    clienteOPCAcabado se pone a true para que la clase que llama a esta función
sepa que el hilo periódico ha acabado su ejecución
    obj.clienteOPCAcabado=True
    return
```

Figura 39 – Ejemplo Código Hilo Cliente OPC

Y para llamar a esta función desde el programa principal usamos el siguiente código:

```
def startTheThread(self):
    self.threadOPC = threading.Thread(target=clienteOPC, args=(self,))
    self.threadOPC.start()
```

Figura 41 – Ejemplo Código Llamada Hilo

De forma equivalente para finalizar el hilo:

```
def startTheThread(self):
    self.threadOPC = threading.Thread(target=clienteOPC, args=(self,))
    self.threadOPC.start()
```

Figura 40 – Ejemplo Código Finalización Hilo

4.1.4. IMPLEMENTACIÓN DEL INTERFAZ GRÁFICO HMI

Como Phyton no tiene in interfaz gráfico propio, como puede tener Java, se ha usado la librería PyQt4 v4.11.4 para Phyton 2.7 Qt4.8.7 32bits (PyQt s.f.). En esta librería se diseña de forma gráfica las pantallas, y posteriormente se transforma este archivo en código Phyton pudiendo acceder a él, gracias al uso de las clases de la programación orientada a objetos desde el programa principal. Las características de PyQt son:

- Es convertible entre lenguajes C++, C#/.NET Languages (Mono), Python: PyQt y PySide, Ada, Pascal, Perl, PHP y Ruby.
- Posee también una GUI integrada y diseñador de formularios.
- Herramienta para proyectos y administración.
- Ayuda sensible al contexto integrada.
- Depurador visual.
- Soporte para el desarrollo de aplicaciones en diferentes sistemas operativos (Windows, Linux, Android, etc.) con diferentes resoluciones.

Como comparativa se han buscado otras herramientas con las que programar interfaces gráficos en Phyton, y se ha efectuado una comparativa entre ellas.

ASPECTO A COMPARAR	PyQt	wxPython	PyGTK
Complejidad de Uso	Medio con conocimientos programación	Medio con conocimientos programación	Fácil
Tipo Aplicaciones Desarrolladas	Aplicaciones propias del usuario	Aplicaciones nativas	Aplicaciones propias del usuario
Dispositivos Soportados	Ordenadores, móviles, tablets	Ordenadores	Ordenadores
Precio	295$	Desarrollo gratuito Soporte 250$	Gratuito sin soporte

Tabla 4 – Comparativa Herramientas Desarrollo Interfaz Gráfico

Aparte de esta herramienta se han usado otras herramientas las cuales comentaré brevemente ya que su uso es esporádico.

Como crearemos un registro de eventos, éstos serán guardados en un archivo Excel, y para facilitar la lectura y escritura de estos ficheros se usará la librería openpyxl (OpenPyXL s.f.).

También se realizarán dos análisis de estos eventos, unos en boxplots, con el uso de la librería pandas.DataFrame; y un análisis de datos jerárquico con el uso de la librería scipy.cluster.hierarchy ya incluidas en la instalación de Phyton.

IMPLEMENTACIÓN DEL INTERFAZ GRÁFICO HMI

Se ha implementado un modelo gráfico HMI para que un supuesto personal de mantenimiento pueda ciclar la instalación de forma remota. A través del cliente OPC comentado anteriormente se reciben todas las variables del autómata sobre las cuales podemos leer y escribir (no se puede escribir sobre las entradas y salida, solo sobre las marcas). De esta forma se ha diseñado un interfaz gráfico como el que vemos en la siguiente figura:

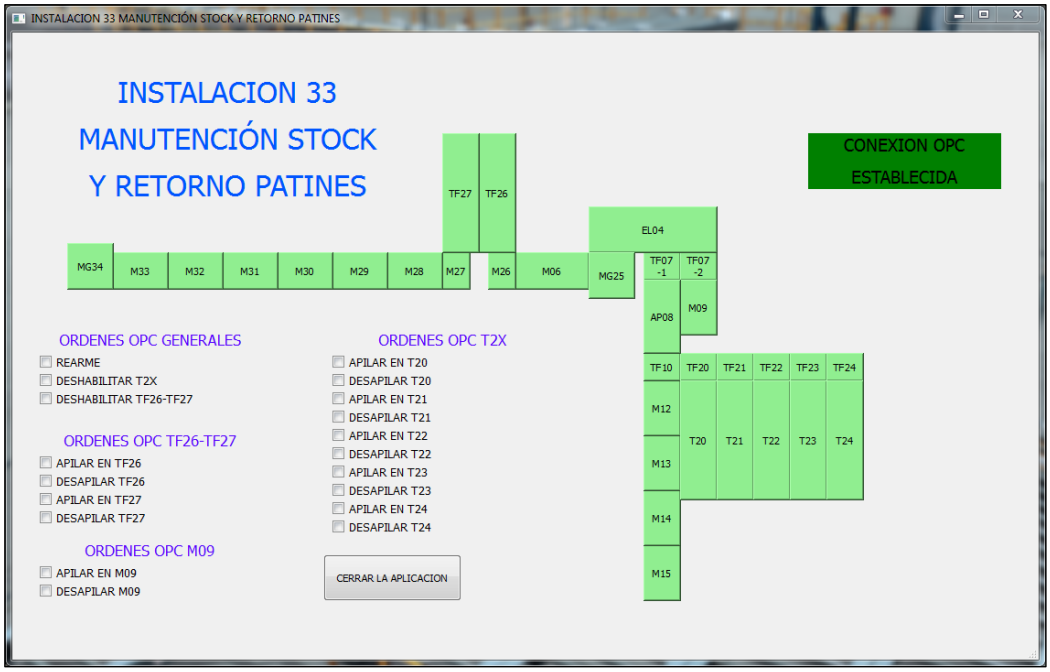


Figura 42 – Interfaz Grafico HMI

En esta pantalla tenemos acceso a todos los elementos funcionales de la instalación los cuales podremos ciclar de forma independiente. También se muestra un mensaje si la instalación está conectada con el HMI a través de OPC o si no está activa la conexión. Sobre cada elemento funcional se ha implementado un color, verde si no está en defecto y rojo si hay defecto.

Aparte de esto podemos enviar unas ordenes OPC generales, como pueden ser la orden de apilado/desapilado en los transferes de cumulo, o la orden de rearme para poder rearmar la instalación después de haber subsanado la incidencia.

Si pulsamos sobre uno de los elementos funcionales se abre una nueva ventana sobre la que podemos ciclar dicho elemento además de ver en qué estado se encuentra, por ejemplo, para la mesa 15 tenemos esta ventana:

Figura 43 – Ventana Grafica Ciclado M15

Dentro de la interfaz gráfica también tendremos una ventana sobre la que podemos ver el histórico de la instalación en forma de tabla (QtWiki s.f.), ya que cada vez que surge un evento o defecto este es guardado en un archivo Excel (R s.f.), donde se escribe el día y la hora (PUAROT s.f.) (Pherkad s.f.) en la que aparece y desaparece el evento, el elemento funcional afectado y el tipo de evento o defecto que es.

Sobre la pantalla implementada podemos ver el histórico de defectos ordenados de forma temporal y podremos escribir un comentario el cual quedará también escrito sobre el archivo Excel. Por otro lado, hay un resumen de cada mesa funcional, donde aparecen el número total de defectos, el tiempo total de todos estos defectos, y el tiempo mínimo y máximo.

Para la implementación de este registro nos hemos ayudado de semáforos los cuales controlan el acceso a las zonas críticas de lectura y escritura de fichero (NOE.JIM s.f.).

Sobre todos estos datos podremos efectuar un filtrado de tal forma que solo aparezcan los defectos asociados a un elemento funcional en concreto y también entre un determinado intervalo temporal.

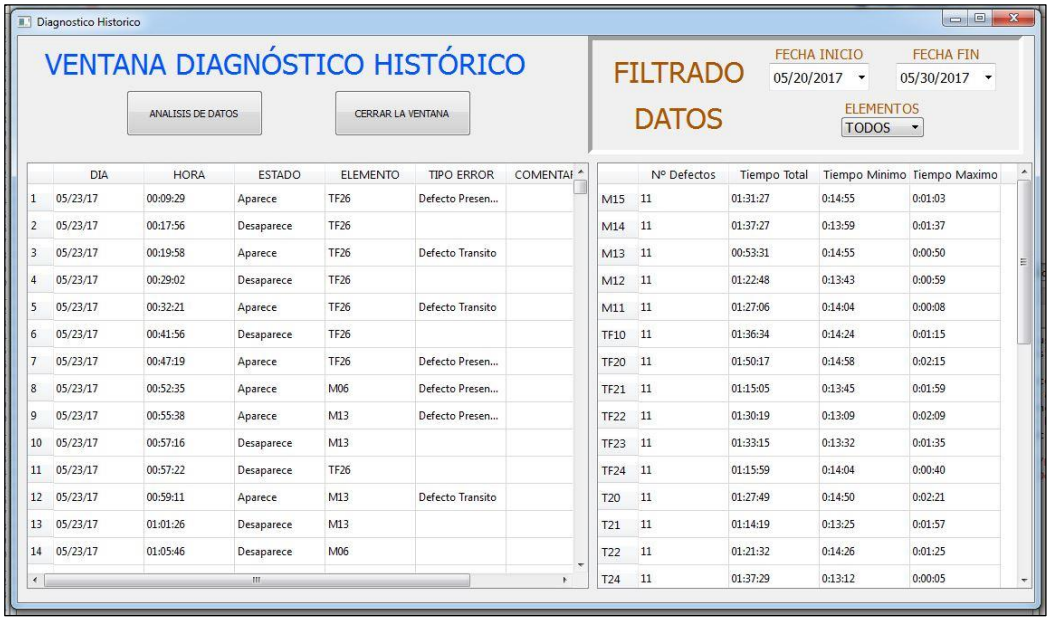


Figura 44 – Visualización Histórico de Datos

Por último y de cara a un estudio de los eventos se ha creado una pantalla como la siguiente:

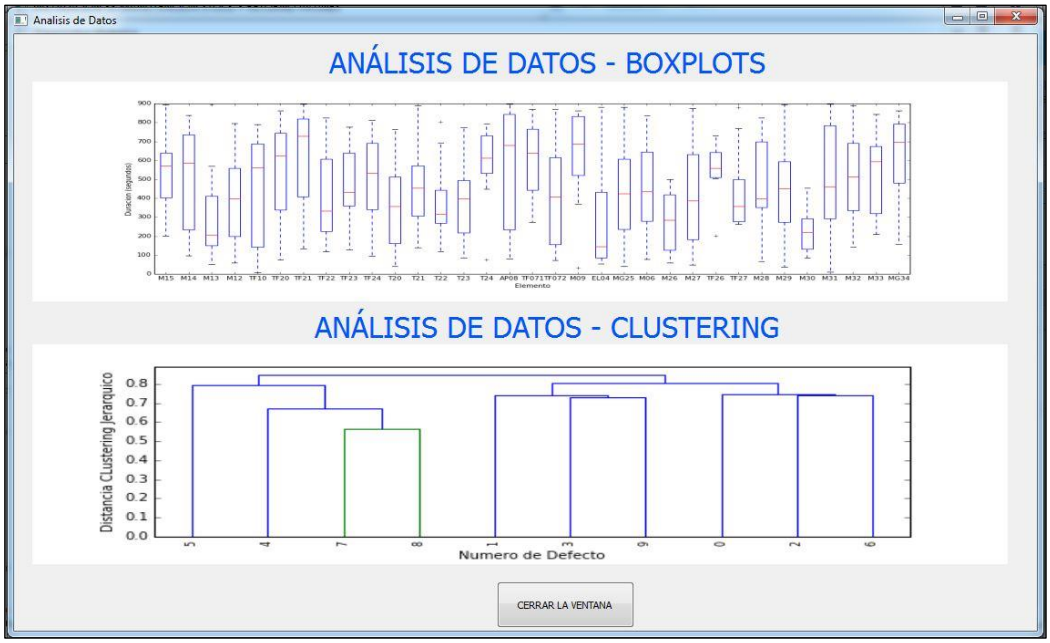


Figura 45 – Pantalla Análisis de Eventos

Sobre esta pantalla se han mostrado para cada elemento funcional los boxplots (the pandas development team s.f.) de los tiempos de duración de

los defectos. De esta forma se puede apreciar cual es el tiempo medio de cada elemento funcional y la variación de ese tiempo.

Por otra parte, se ha realizado un clustering jerárquico (luca s.f.) tal y como se ha comentado en capítulos anteriores, para poder ver la relación entre los diferentes defectos, mostrándose estos resultados sobre el interfaz gráfico (DavidBoddie s.f.).

4.2. SIMULACIÓN DEL MODELO

Para explicar la simulación del modelo nos basaremos en diferentes capturas de pantalla y tablas de datos, si bien, con el CD adjunto se pueden ver diversos videos de funcionamiento para comprender mucho mejor el proceso.

Comenzaré explicando cómo ha sido la simulación del modelo físico. En las dos siguientes figuras podemos ver como es el funcionamiento normal de la instalación antes de saturar, tanto para el nivel 0 como para el nivel 1.

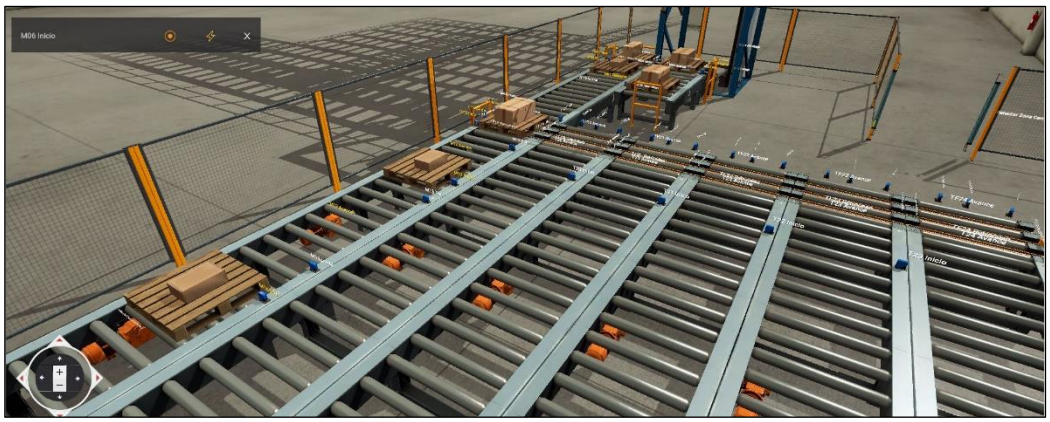


Figura 46 – Simulación Condiciones Normales Nivel 0 Modelo Físico

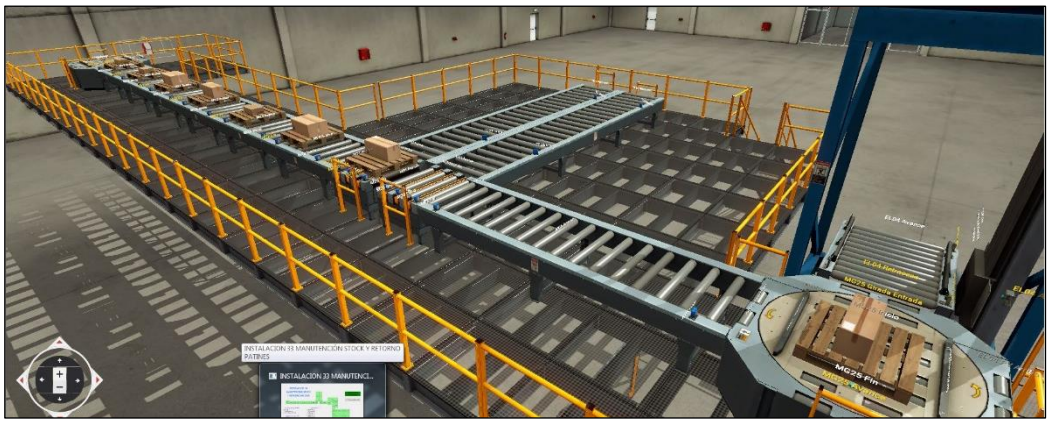


Figura 47 – Simulación Condiciones Normales Nivel 1 Modelo Físico

En estas condiciones se puede observar como las mesas de transito están con presencia de pieza, a diferencia de las mesas de cúmulo las cuales no tienen presencia debido a que la instalación no ha recibido el suficiente número de piezas como para saturar.

En la siguiente figura podemos ver como es el proceso de llenado del nivel 0, donde la zona de cúmulo es la más grande de todas y con mayor capacidad de piezas.

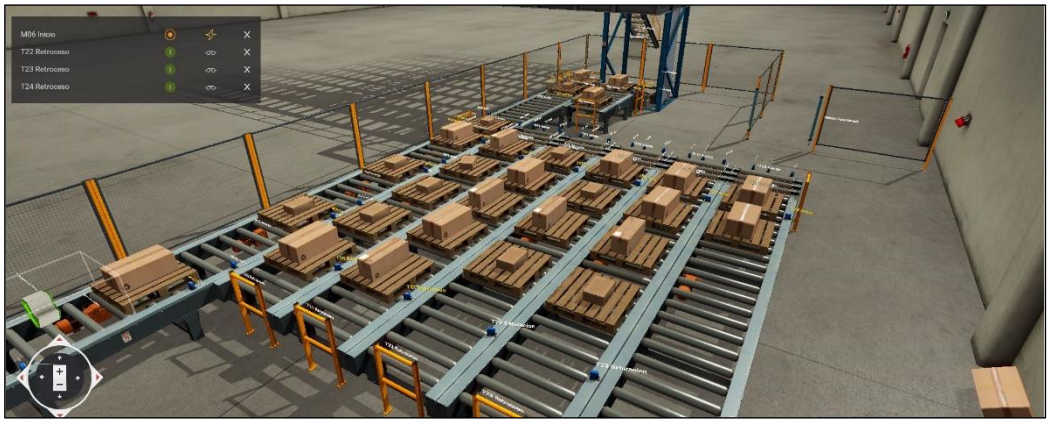


Figura 48 – Simulación Condiciones Proceso de Llenado Nivel 0 Modelo Físico

Por último, a diferencia del primer caso, existe otro caso en donde la instalación a simular no ha podido desalojar el suficiente número de piezas y la instalación ha saturado todas las mesas de tránsito y de cúmulo en ambos niveles como vemos en las dos siguientes figuras

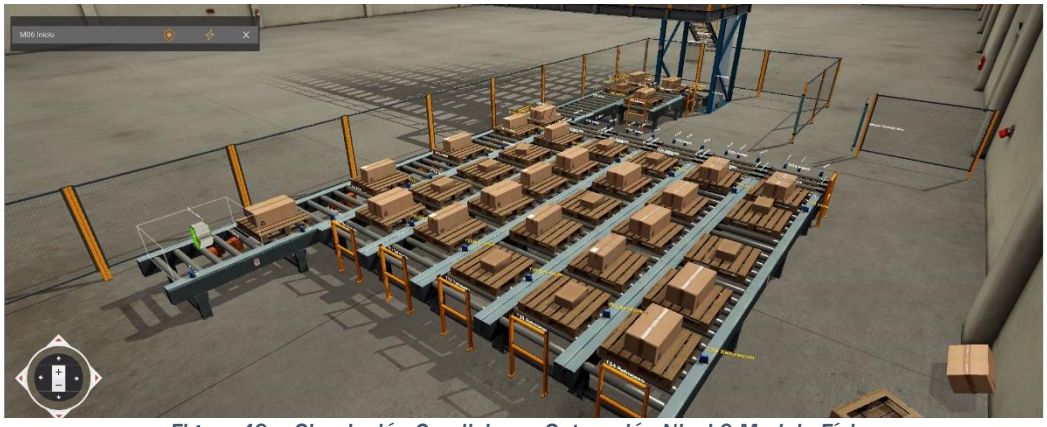


Figura 49 – Simulación Condiciones Saturación Nivel 0 Modelo Físico

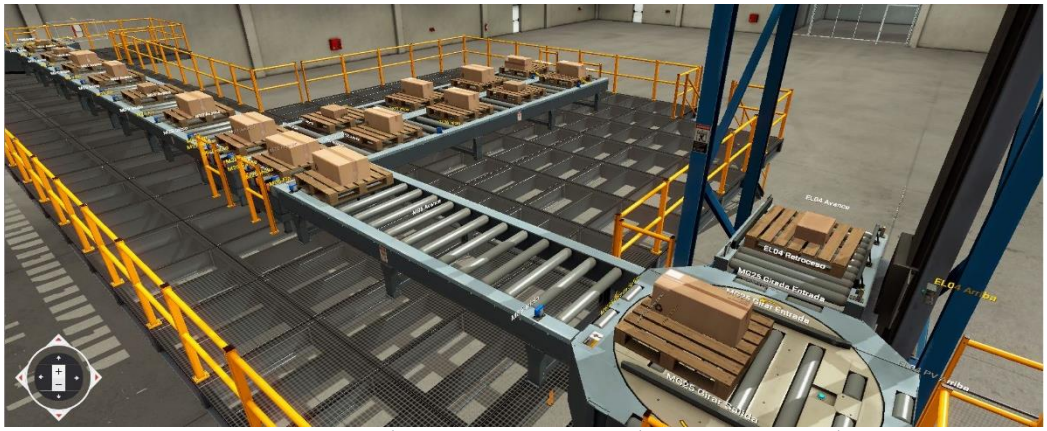


Figura 50 – Simulación Condiciones Saturación Nivel 1 Modelo Físico

Una vez descrita la simulación del modelo físico se procede a describir la simulación del interfaz gráfico. En primer lugar, veremos cómo es el aspecto en el caso de que no haya sido posible establecer la conexión OPC con el autómata.

Figura 51 – Interfaz Gráfico sin Conexión OPC

Vemos como aparece un mensaje de que no hay conexión OPC, aparte de aparecer todos los elementos funcionales en rojo (en rojo significa que no está funcionando de forma correcta).

Sin embargo, si el funcionamiento es el adecuado y la conexión OPC se ha establecido de forma correcta, veremos un mensaje indicando la conexión OPC, y todos los elementos funcionales de la instalación estarán en verde (verde indica un correcto funcionamiento).

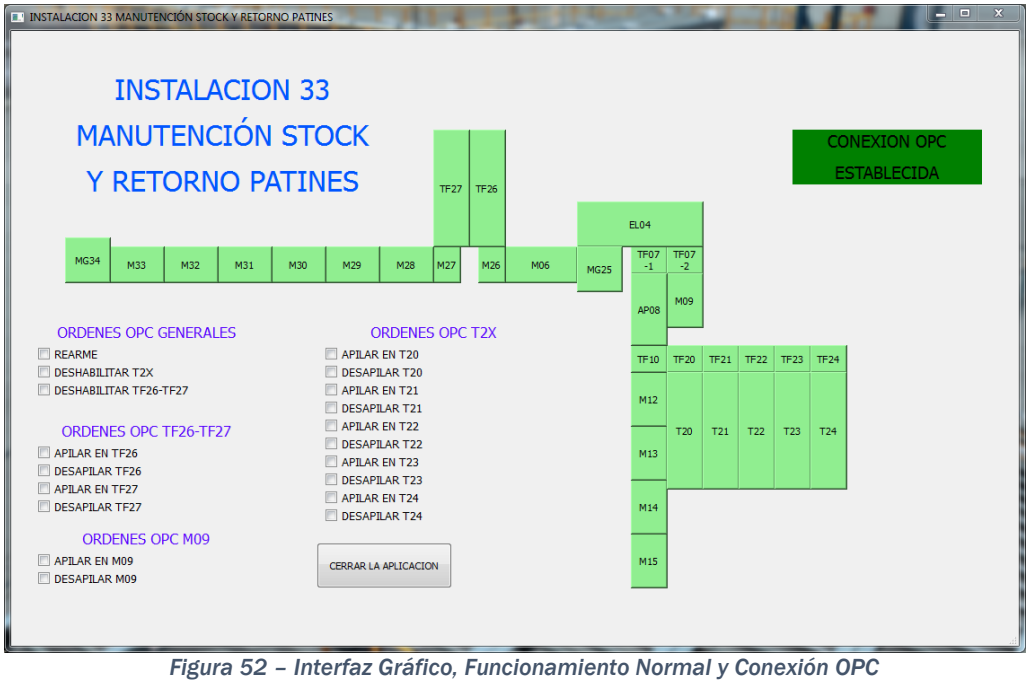


Figura 52 – Interfaz Gráfico, Funcionamiento Normal y Conexión OPC

Tal y como se ha comentado en capítulos anteriores disponemos de unas órdenes generales que podamos enviar desde la pantalla principal, sin embargo, si queremos enviar órdenes relativas a un elemento funcional en concreto debemos de pulsar sobre dicho elemento funcional. A continuación, se muestra de forma general como es esta nueva ventana para el ciclado de cada elemento funcional, para mesa de tránsito, mesas de cúmulo, mesas giratorias, elevadores y transfers.

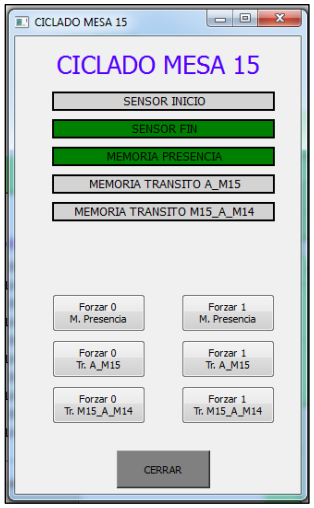


Figura 53 – Interfaz Gráfico Mesa Transito M15

Figura 54 – Interfaz Gráfico Mesa Giratoria MG25

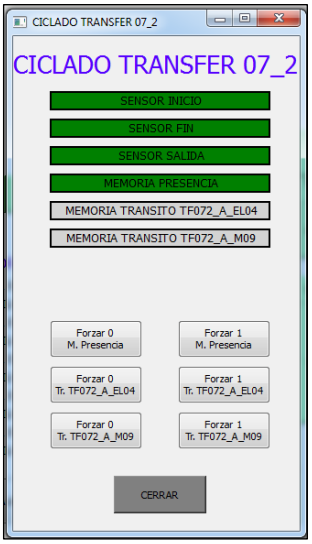


Figura 55 – Interfaz Gráfico Transfer 2 Destinos TF07-2

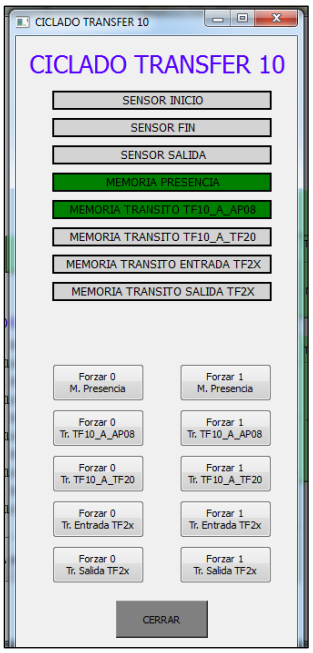


Figura 56 – Interfaz Gráfico Transfer 4 Destinos TF10

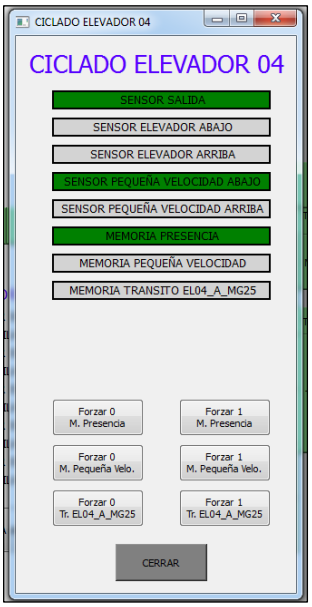


Figura 57 – Interfaz Gráfico Elevador EL04

Para acabar con la descripción del interfaz gráfico veremos cómo es la representación de eventos o defectos en el interfaz.

Si se producen uno de los tres defectos generales ya comentados (defecto emergencia, defecto intrusión o defecto recepción de datos por OPC) se muestra con un mensaje en la ventana principal, pero si el defecto es asociado a un elemento funcional (defecto presencia, transito, giro, elevación), el elemento funcional afectado quedará marcado de color rojo y deberemos de pulsar sobre él para ver cuál es el defecto y poder corregirlo. Por ejemplo, si tenemos defectos generales y simultáneamente defectos en elementos funcionales, obtenemos una ventana semejante a la que vemos a continuación.

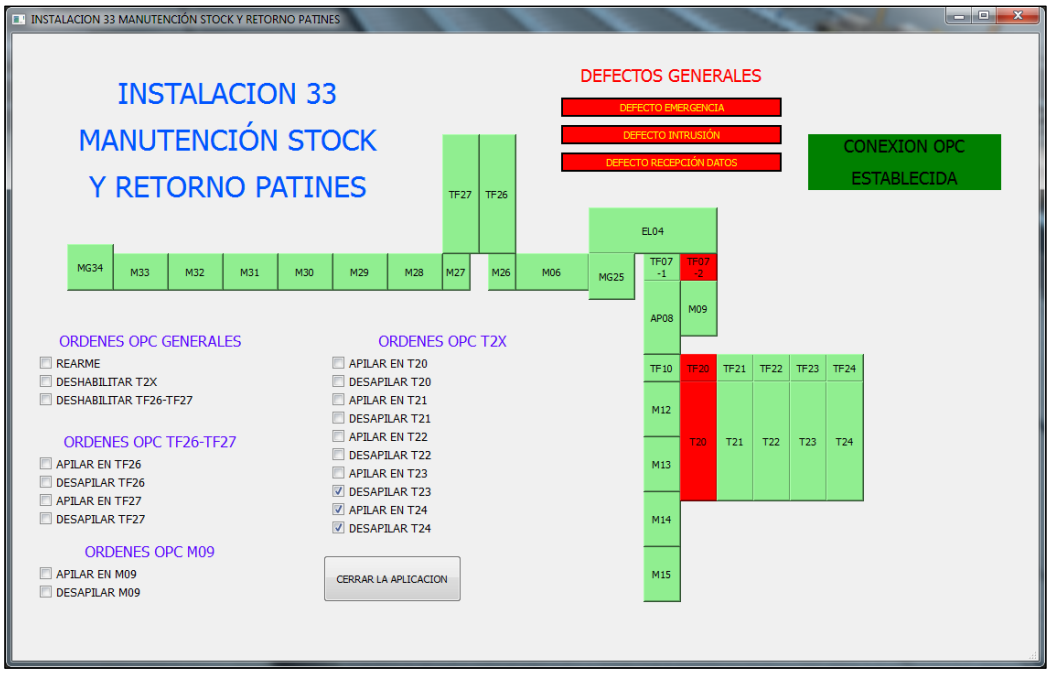


Figura 58 – Representación Defectos Interfaz Gráfica, Ventana Principal

Y si queremos ver, por ejemplo, el estado de TF20, que como se ve en la figura superior obtenemos la siguiente ventana.

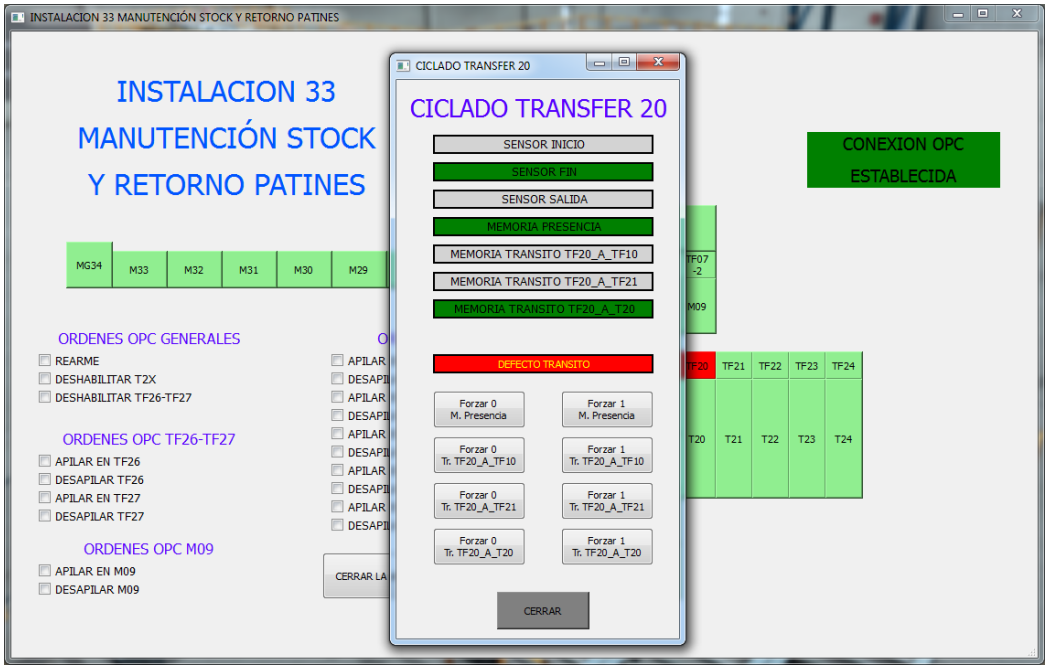


Figura 59 – Representación Defectos Interfaz Gráfica, Elemento Funcional

4.3. ANÁLISIS DE EVENTOS

Dentro de la interfaz gráfica tendremos una funcionalidad que será el análisis de eventos. Este análisis está formado por la lectura del registro de eventos sobre la cual se puede hacer un filtrado por fecha y por elemento funcional, un análisis boxplot de los defectos para cada elemento funcional y un clustering para saber la relación existente entre eventos.

Para una ejecución normal de la instalación y durante un periodo breve de tiempo tendremos un registro en tabla Excel como el siguiente.

DÍA	HORA	ESTADO	ELEMENTO	TIPO FUNCIONAL	COMENTARIO
06/18/17	12:11:32	Aparece	M15	Defecto Transito	
06/18/17	12:11:32	Aparece	EL04	Defecto Presencia	
06/18/17	12:11:32	Desaparece	Intrusion		
06/18/17	12:11:32	Desaparece	RecepcionOPC		
06/18/17	12:11:39	Desaparece	M15		
06/18/17	12:11:39	Desaparece	EL04		
06/18/17	12:11:39	Desaparece	Emergencia		
06/18/17	12:13:18	Aparece	M27	Defecto Transito	
06/18/17	12:13:47	Desaparece	M27		
06/18/17	12:14:32	Aparece	M28	Defecto Transito	
06/18/17	12:14:35	Aparece	MG25	Defecto Transito	
06/18/17	12:15:07	Desaparece	MG25		
06/18/17	12:15:07	Desaparece	M28		
06/18/17	12:15:18	Aparece	M28	Defecto Transito	
06/18/17	12:15:31	Desaparece	M28		
06/18/17	12:16:09	Aparece	MG25	Defecto Transito	

Tabla 5 – Registro de Simulación Eventos Excel

Y en el interfaz gráfico se mostrará una ventana con estos datos como vemos en la siguiente figura:

Figura 60 – Registro de Eventos en Interfaz Gráfico

Sin embargo, para poder hacer un análisis completo de la instalación se necesitarían muchos más registros, algo que se puede hacer siempre y cuando se mantenga la simulación durante horas, algo que no hemos podido hacer debido a la restricción de las licencias de evaluación del software.

Para ello se han creado 11 eventos para cada elemento funcional de tal forma que la hora y día de aparición/desaparición se ha calculado de forma aleatoria. De esta forma hemos conseguida la creación de más de 300 eventos, 11 para cada elemento funcional.

La parte inicial se este registro creado, ya que en nuestro interfaz gráfico solo podemos ver hasta 14 registros, es la siguiente:

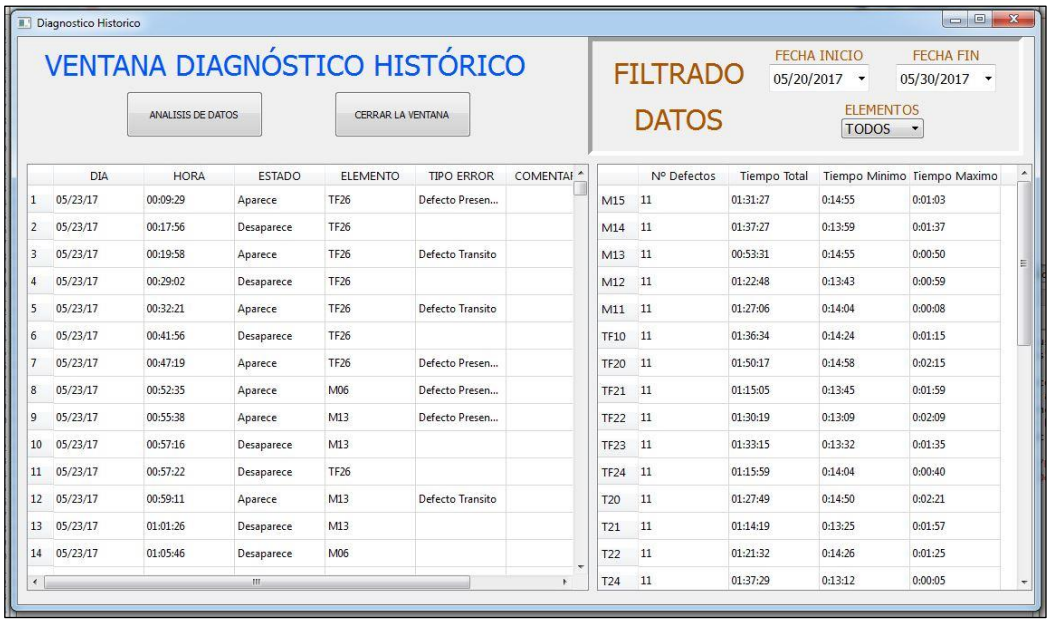


Figura 61 – Registro de Eventos Simulado

Y sobre este registro se pueden hacer dos filtrados, uno de ellos consiste en especificar la fecha de comienzo y de fin, como vemos a continuación.

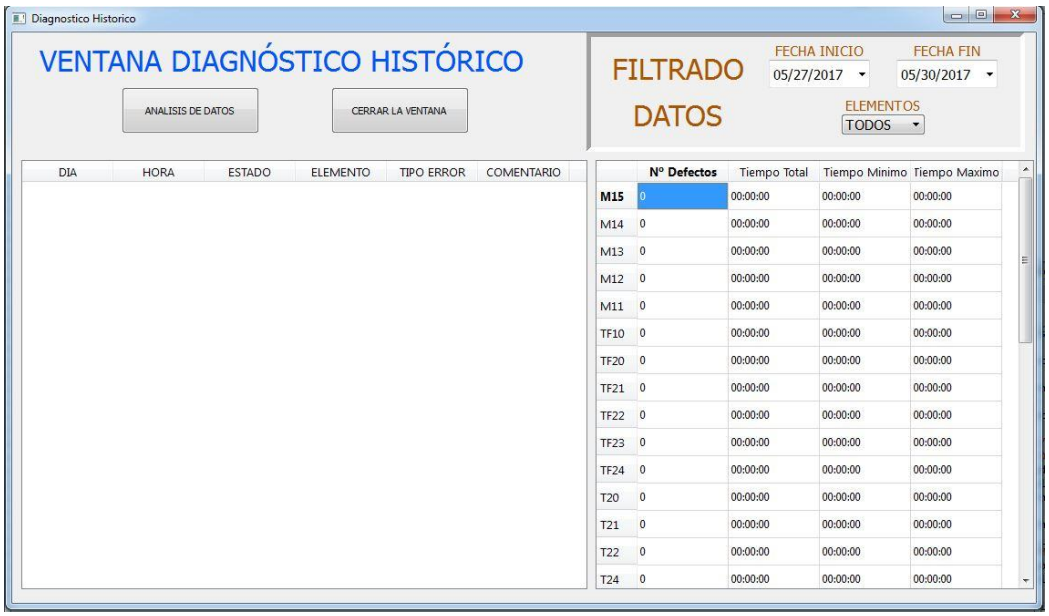


Figura 62 – Filtrado Eventos por Fecha

Vemos como no existen eventos entre las fechas filtradas por lo que la tabla estará vacía.

También se puede filtrar por elemento funcional, mostrando únicamente los eventos asociados a ese elemento funcional. Como ejemplo:

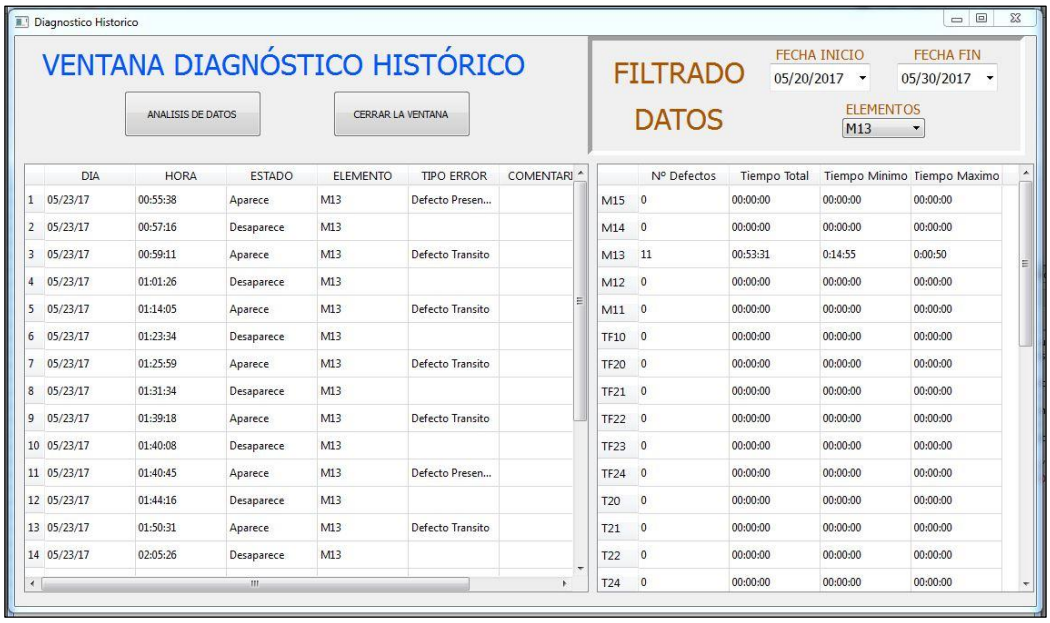


Figura 63 – Filtrado Eventos por Elemento Funcional

En este caso se ha escogido como elemento funcional la M13 y vemos como en ambas ventanas se ha aplicado el filtrado.

Por último, se hace la representación gráfica del análisis boxplot y clustering.

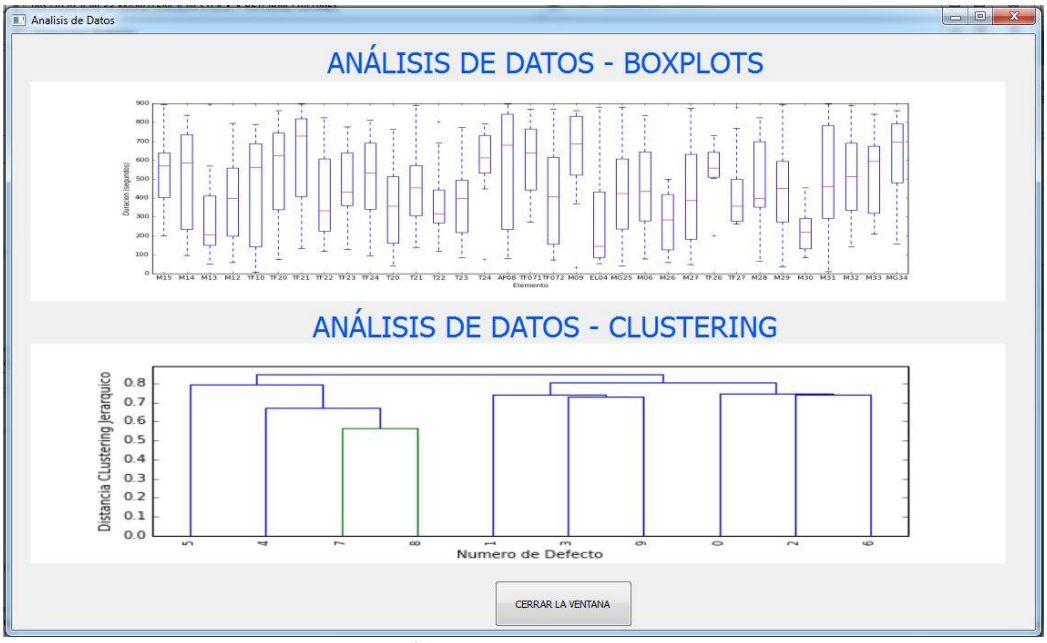


Figura 64 – Análisis de Eventos, Boxplot y Clustering

En primer lugar, analizamos el conjunto de boxplots. Vemos como es completamente diferente para cada elemento funcional, debido a la naturaleza de los numero aleatorios creados. Este análisis sería útil para comparar los tiempos de duración de los defectos en la instalación.

Posteriormente se hace un clustering jerárquico de datos. En este caso se pueden agrupar los datos en un número determinado de clústeres, debido a la naturaleza de este tipo de clustering.

Con este método podríamos ver las interrelaciones temporales que existen entre los diferentes defectos, por ejemplo, es destacado en la figura, que el séptimo y el octavo defecto están muy relacionados.

5. CONCLUSIONES

En este capítulo se muestran las conclusiones que se pueden extraer una vez realizado todo el proceso de implementación y análisis, así como unas posibles mejoras que se pueden realizar en un futuro.

A continuación, se presentan las conclusiones más importantes que se extraen del presente proyecto:

- Se ha conseguido realizar una copia digital de una instalación real, lo que nos permite realizar modificaciones sobre la misma, muy útil cuando se quieren realizar modificaciones en la misma o añadir modos de funcionamiento.
- Este modelo de instalación se ha desarrollado en base a conocimientos adquiridos en el master relacionados con las asignaturas de:
    - Programación de Autómatas y SCADA
    - Desarrollo de Aplicaciones Distribuidas Industriales

  - Pre-Procesamiento de la Información y Modelado basado en Datos
  - Simulación de Procesos Industriales
  - Programación Orientada a Objetos y Bases de Datos en Entornos Industriales
  - Fundamentos de Ingeniería del Software
  - Sistemas de Interacción Hombre-Máquina
- Los resultados de la simulación han sido buenos, ya que reflejan el modo de funcionamiento de la instalación deseado, si bien, en la simulación del modelo no se ha podido representar fielmente la instalación real debido a las limitaciones de software (parte de estas limitaciones por el uso de licencias de evaluación).
- El programa Factory I/O es una herramienta idónea para hacer una introducción a la simulación y modelado de una instalación industrial, aprovechando una de sus características principales, que es la conexión automática con PLCSim.
  - Software sencillo de usar, pero con escasez de librerías, por lo que no se puede simular muchos de los procesos industriales.
  - Gracias a este software un cliente podría aceptar o sugerir cambios en una instalación antes de su puesta en marcha.
- El estándar de comunicación OPC nos permite la posibilidad de establecer conexiones remotas de envío y recepción de información de forma sencilla.
  - A pesar de facilitar la conexión para establecer la conexión con determinados programas software, en nuestro caso para PLCSim es necesario el uso de software adicional.
- Phyton es un lenguaje de programación muy completo para el que existen numerosas librerías y complementos para poder incluir en los códigos de programa.
- Con Qt podemos diseñar de forma rápida y eficaz un interfaz gráfico muy logrado con el que simular un HMI para el control distribuido (con el uso de OPC) de la instalación.

- Las técnicas de clustering son técnicas modernas de agrupamiento de datos que nos permite establecer las relaciones entre diferentes observaciones.
    - Son técnicas las cuales tienen mucho futuro de cara a realizar un mantenimiento preventivo.
- Para realizar este proyecto se usan determinados programas software, los cuales consumen muchos recursos de ordenador, siendo importante el contar con equipos potentes y con capacidad de computación para realizar un modelado y simulación completa de una instalación industrial.

5.1. LECCIONES APRENDIDAS

Este proyecto me ha servido para fianza y adquirir conocimientos nuevos, tantos personales como técnicos.

- He comprendido la importancia de la creación de un modelo de cara a realizar un análisis, en este caso, sobre una instalación real.
- Me ha sido de gran interés el poder estudiar e implementar un control remoto, mediante OPC, debido a que cada día es más frecuente encontrar este tipo de control distribuido.
- A nivel personal he realizado un análisis completo de una instalación industrial, de forma análoga a como realizan los equipos de ingeniería, gracias a lo cual voy reforzando las competencias profesionales.
- Gracias a los conocimientos adquiridos durante mi periodo de prácticas en Bosch Rexroth he asimilado la importancia de los interfaces gráficos para el mantenimiento de las instalaciones.
- De gran utilidad para mi será, el haber aprendido el cómo formalizar y mostrar todos los conocimientos y técnicas usados en el desarrollo de este proyecto.

5.2. POSIBLES MEJORAS

Se proponen una serie de mejoras que se pueden realizar sobre el presente proyecto para mejorar las funcionalidades implementadas o bien para implementar nuevas funcionalidades.

En relación al modelado de la instalación física.

- Corregir el posicionamiento de los sensores, siempre y cuando se use un ordenador de alto rendimiento, con un procesador potente y suficiente memoria RAM, para evitar retrasos en la detección de piezas y evitar que éstas se atasquen en la simulación.
- Introducción de diversidad en las piezas de tal forma que cada diversidad pueda tener un destino diferente.
- Simular fallos en los sensores y actuadores, para ver cómo reacciona el modelo ante estos fallos.
- En zonas críticas, introducir redundancia en los sistemas.
- Implementar un cofre de movimientos manuales para cada elemento funcional.

En relación al programa autómata:

- Implementar la funcionalidad completa de la instalación real.
- Usar OBs y FBs de datos para diagnóstico y detección de errores.
- Implementar un modo de funcionamiento manual completo y no solo para elevador.
- Mejor detección de defectos en función de los sensores.

En relación al interfaz gráfico:

- Extensión del interfaz a dispositivos móviles con sistema operativo Android.
- Uso de la herramienta Qt para la auto-traducción de textos.
- Mejora del rendimiento de la aplicación.

En relación al análisis de eventos:

- Posibilidad de borrar registros del fichero Excel.

- **Posibilidad de selección del método de clustering.**
- **Mejora del rendimiento, evitando tiempos muertos, en la lectura y análisis del registro en Excel.**

6. BIBLIOGRAFÍA

Anaconda. *Anaconda Support.* s.f. https://www.continuum.io/ (último acceso: 20 de 06 de 2017).

Bosch Rexroth S.L. «Análisis Funcional Detallado.» 30 de 08 de 2016.

DavidBoddie. *Show an image using a label.* s.f. https://wiki.python.org/moin/PyQt/Show%20an%20image%20using%20a%20label (último acceso: 20 de 06 de 2017).

Factory I/O. *Formación PLC de Proxima Generación con Factory I/O.* s.f. https://factoryio.com/es/ (último acceso: 20 de 06 de 2017).

Foundation, OPC. *OPC Standard.* s.f. https://opcfoundation.org/about/what-is-opc/ (último acceso: 20 de 06 de 2017).

Fuente, Laura de la. «Análisis Cluster.» s.f. http://www.estadistica.net/Master-Econometria/Analisis_Cluster.pdf (último acceso: 20 de 06 de 2017).

González, Sergio Chico. «DESARROLLO DE UN PUESTO DE ENVASADO AUTOMATIZADO, BASADO EN ACCIONAMIENTOS

ELECTRONEUMÁTICOS, Y PUESTA EN MARCHA VIRTUAL DEL MISMO.» 30-35, 80-85. Universidad Politécnica de Madrid, 2015.

Ingeniería, Facultad de. *Diagrama de contactos (Ladder) .* Universidad Publica de la Plata. s.f. http://www.educacionurbana.com/apuntes/ladder.pdf (último acceso: 20 de 06 de 2017).

John, Michael Tiegelkmap & Karl-Heinz. *IEC-61131-3: Programming Industrial Automation Sistems.* Universidad de Oviedo - Departamento de Ingeniería de Sistema y Automática. s.f. http://www.dee.ufrj.br/controle_automatico/cursos/IEC61131-3_Programming_Industrial_Automation_Systems.pdf (último acceso: 20 de 06 de 2017).

Jones, David Beazley & Brian K. *Phyton Cookbook.* 3ª. O'Reilly, 2013.

Jover, Maria Pujol. «Clustering jerárquico.» s.f. https://miriadax.net/documents/54270034/56027201/M3-Clustering-jerarquico.pdf (último acceso: 20 de 06 de 2017).

KepServerEX. *KEPServerEX - Connects disparate devices and applications, from plant control systems to enterprise information systems.* s.f. https://www.kepware.com/en-us/products/kepserverex/ (último acceso: 20 de 06 de 2017).

KULLABS. *Note on Quartile.* s.f. https://www.kullabs.com/classes/subjects/units/lessons/notes/note-detail/2462 (último acceso: 20 de 06 de 2017).

luca. *Hierarchical clustering of time series in Python scipy/numpy/pandas?* s.f. https://stackoverflow.com/questions/34940808/hierarchical-clustering-of-time-series-in-python-scipy-numpy-pandas (último acceso: 20 de 06 de 2017).

MIGUELETBLOG. *El transportador de rodillos y su importancia en la industria.* s.f. http://www.embalajesterra.com/blog/transportador-de-rodillos-industria/ (último acceso: 20 de 06 de 2017).

NetToPLCSim. *NetToPLCSim - A Network Interface to PLCSim.* s.f. http://nettoplcsim.sourceforge.net/nettoplcsim-s7o-en.html (último acceso: 20 de 06 de 2017).

NOE.JIM. *Semáforos en Python.* s.f. https://prognoejim.wordpress.com/2015/10/13/semaforos-en-python/ (último acceso: 20 de 06 de 2017).

OpenOPC. *OpenOPC for Phyton.* s.f. http://openopc.sourceforge.net/about.html (último acceso: 20 de 06 de 2017).

OpenPyXL. *A Python library to read/write Excel 2010 xlsx/xlsm files.* s.f. https://openpyxl.readthedocs.io/en/default/ (último acceso: 20 de 06 de 2017).

Pedersen, Jesper M. «PLCS.net.» s.f. https://www.plcs.net/downloads/index.php?action=downloadfile&filename=PLC_Comparison_chart_2007_v5.xls&directory=Misc (último acceso: 20 de 06 de 2017).

Pherkad. *Operaciones con fechas y horas. Calendarios.* s.f. http://python-para-impacientes.blogspot.com.es/2014/02/operaciones-con-fechas-y-horas.html (último acceso: 20 de 06 de 2017).

Physicist, Mad. *How to install PyQt4 in anaconda?* s.f. https://stackoverflow.com/questions/21637922/how-to-install-pyqt4-in-anaconda (último acceso: 20 de 06 de 2017).

PUAROT. *Sumar y restar horas en Python.* s.f. https://foro.elhacker.net/scripting/sumar_y_restar_horas_en_python-t453487.0.html;msg2074263 (último acceso: 20 de 06 de 2017).

PyQt. *The Phton Qt Wiki.* s.f. https://wiki.python.org/moin/FrontPage (último acceso: 20 de 06 de 2017).

QtWiki. *How to Use QTableWidget.* s.f. https://wiki.qt.io/How_to_Use_QTableWidget (último acceso: 20 de 06 de 2017).

R, Aarón Díaz. *Leer documentos de Excel con Python.* s.f. http://ironsistem.com/tutoriales/python/leer-documentos-de-excel-con-python/ (último acceso: 20 de 06 de 2017).

Resnick_halliday. *Transductores y Sensores en la Automatización Industrial.* s.f. http://www.monografias.com/trabajos31/transductores-sensores/transductores-sensores.shtml (último acceso: 20 de 06 de 2016).

S21sec . *OPC: ESTANDAR EN LAS REDES INDUSTRIALES Y BUSES DE CAMPO.* s.f. https://www.s21sec.com/es/blog/2009/02/opc-estandar-en-las-redes-industriales-y-buses-de-campo/ (último acceso: 20 de 06 de 2017).

Siemens. *Simatic Step7 V5.X.* 20 de 06 de 2017. http://w3.siemens.com/mcms/simatic-controller-software/en/step7/pages/default.aspx.

Standar, IEC. *IEC 61131-3:2013* . s.f. https://webstore.iec.ch/publication/4552 (último acceso: 20 de 06 de 2017).

the pandas development team. *Plotting with matplotlib.* s.f. http://www.jeffreytratner.com/example-pandas-docs/series-str-2013-10-6/visualization.html (último acceso: 20 de 06 de 2017).

Wikipedia. *Box plot.* s.f. https://en.wikipedia.org/wiki/Box_plot (último acceso: 20 de 06 de 2017).

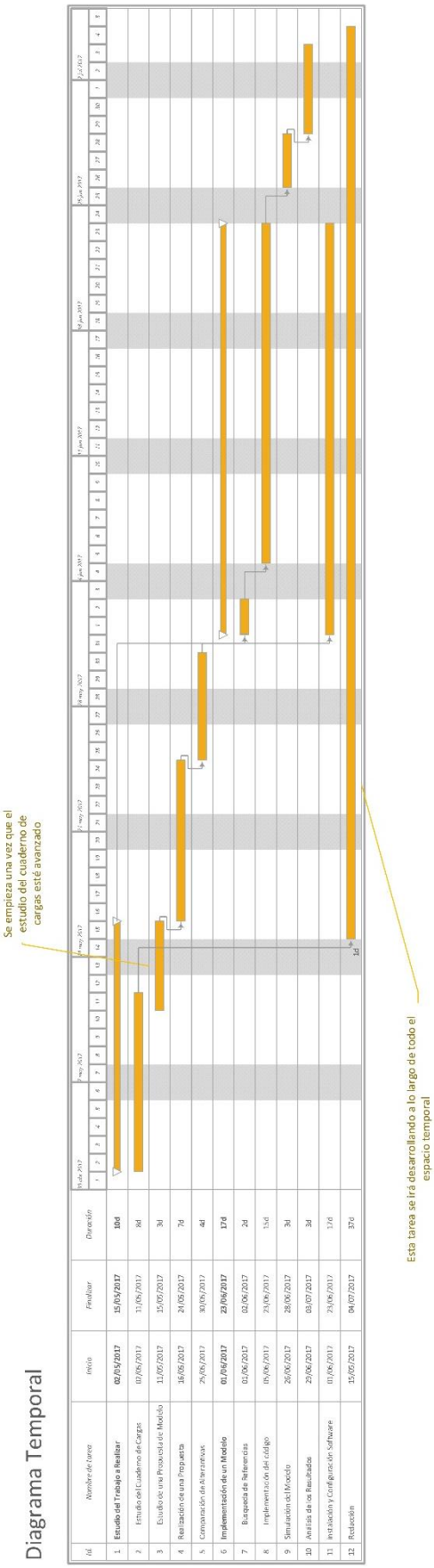
7. ANEXOS

Se mostrará cual ha sido la distribución temporal del trabajo, una ayuda proporcionada por Bosch Rexroth de cara a la instalación, así como se ha tenido que configurar todo los programas de software y unos ejemplos de código usado en la implementación.

7.1. ANEXO 1: DISTRIBUCIÓN TEMPORAL DEL TRABAJO

A forma de resumen se hace un diagrama de Gantt de como se ha distribuido el tiempo para la realización de este proyecto. De especial importancia son las tareas dedicadas a la realización de una propuesta y análisis de resultados, aunque la tarea que más tiempo ha llevado ha sido la de implementación del modelo.

Se parte de la idea que de media cada día se han dedicado unas 3h a la realización de este proyecto.



Id	Nombre de tarea	Inicio	Finalizar	Duración
1	**Estudio del Trabajo a Realizar**	**02/05/2017**	**15/05/2017**	**10d**
2	Estudio del Cuaderno de Cargas	02/05/2017	11/05/2017	8d
3	Estudio de una Propuesta de Modelo	11/05/2017	15/05/2017	3d
4	Realización de una Propuesta	16/05/2017	24/05/2017	7d
5	Comparación de Alternativas	25/05/2017	30/05/2017	4d
6	**Implementación de un Modelo**	**01/06/2017**	**23/06/2017**	**17d**
7	Busqueda de Referencias	01/06/2017	02/06/2017	2d
8	Implementación del código	05/06/2017	23/06/2017	13d
9	Simulación del Modelo	26/06/2017	28/06/2017	3d
10	Análisis de los Resultados	29/06/2017	03/07/2017	3d
11	Instalación y Configuración Software	01/06/2017	23/06/2017	17d
12	Redacción	15/05/2017	04/07/2017	37d

Figura 65 – Diagrama Temporal

7.2. ANEXO 2: PRESENTACIÓN DE LA INSTALACIÓN REAL

La instalación real (Bosch Rexroth S.L. 2016) de la que se pretende crear un modelo está implementada en una instalación real. El objetivo de esta instalación es el de efectuar una manutención, transportando trineos sobre el que van montados las estructuras de los coches entre diferentes instalaciones. Esta instalación está implementada en 5 zonas como se aprecia en la siguiente figura.

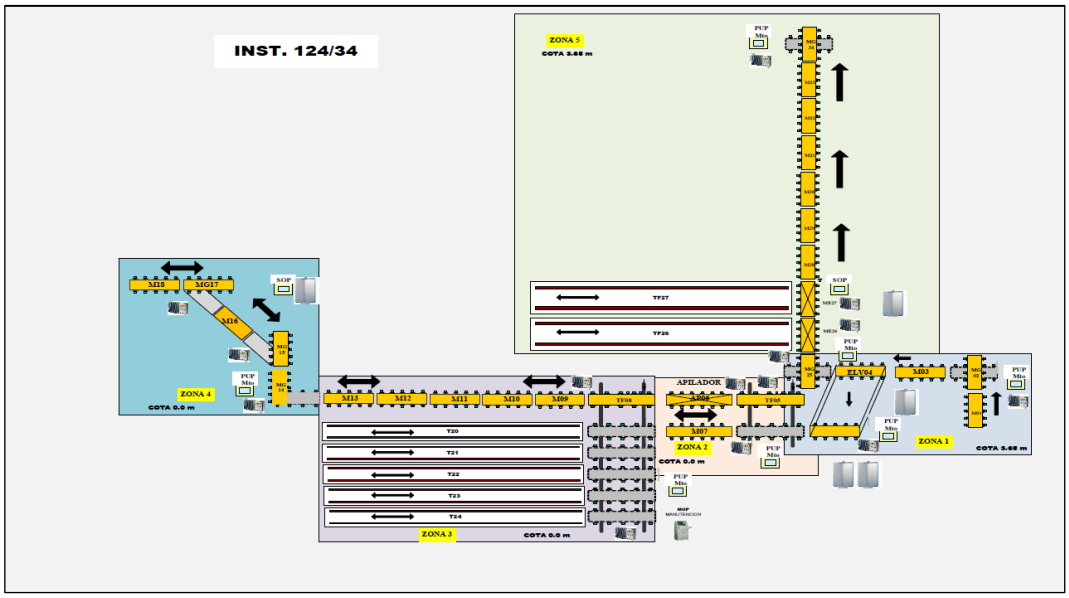


Figura 66 – Descripción de la Instalación Real

Se procede a describir las diferentes zonas y cómo se han adaptado cada una de ellas para la simulación, teniendo en cuenta que todas las zonas se unificarán en una única zona en el modelo simulado.

Tendremos en cuenta que el flujo de la instalación simulada será: La pieza se crea en la zona 3, pasa a la zona 2, posteriormente a la zona 1 y acaba en la zona 5, en la que se simula que la pieza pasaría a otra instalación. La zona 4 no se ha simulado y directamente la pieza se creará en la primera mesa de la zona 3.

7.2.1. DESCRIPCIÓN DE LA ZONA 3

El sinóptico de la zona 3 es el siguiente:

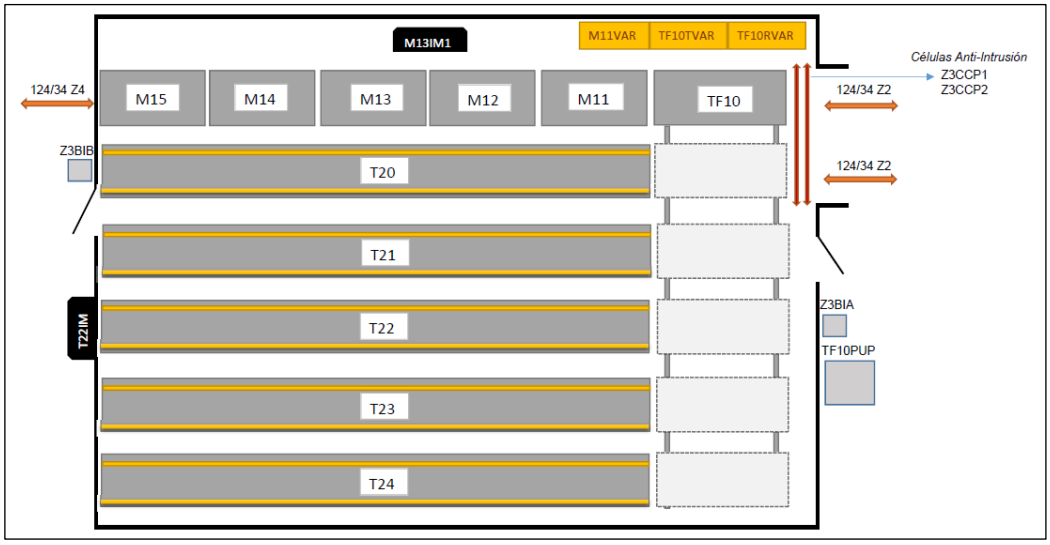


Figura 67 – Descripción de la Zona 3

Esta zona cuenta con los siguientes elementos:

- **M11**: Mesa de Paso.
- **M12**: Mesa de Paso.
- **M13**: Mesa de Paso.
- **M14**: Mesa de Paso.
- **M15**: Mesa de Paso.
- **TF10**: Transfer con tres direcciones de movimiento. Este transfer en la instalación real es capaz de realizar movimientos lineales para traspasar la pieza a alguno de los stocks de cadena siguientes. En realidad, este transfer se ha adaptado añadiendo los TF20, TF21, TF22, TF23 y TF24 para simular el funcionamiento de la instalación.
- **T20**: Stock de Cadena para almacenamiento de patines.
- **T21**: Stock de Cadena para almacenamiento de patines.
- **T22**: Stock de Cadena para almacenamiento de patines.
- **T23**: Stock de Cadena para almacenamiento de patines.
- **T24**: Stock de Cadena para almacenamiento de patines.

7.2.2. DESCRIPCIÓN DE LA ZONA 2

El sinóptico de la zona 2 es el siguiente:

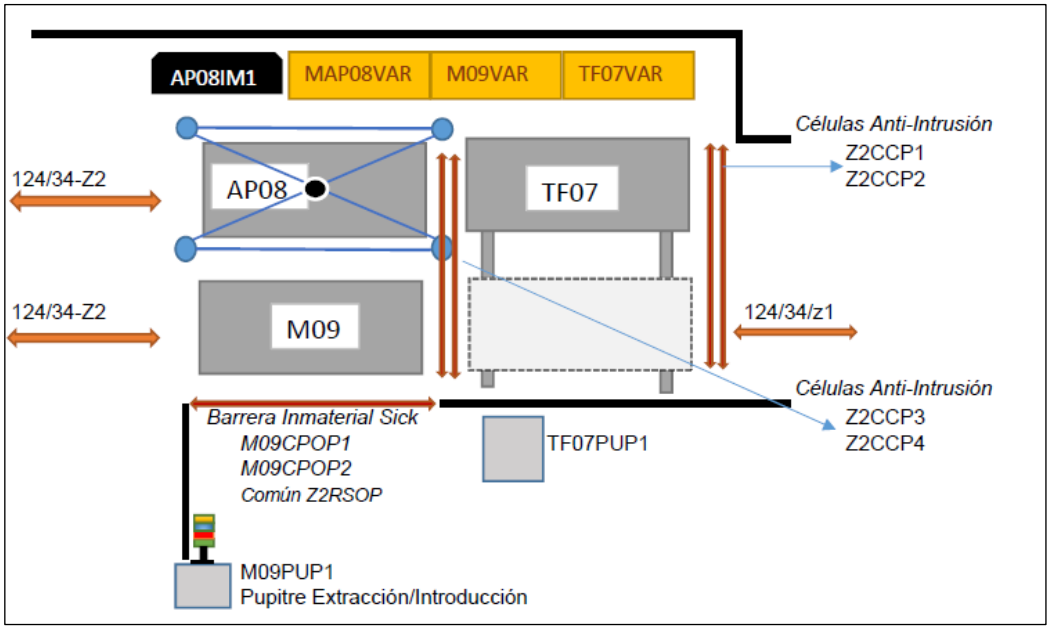


Figura 68 – Descripción de la Zona 2

Esta zona cuenta con los siguientes elementos:

- M09: Mesa de Introducción extracción de patines capaz de acumular un patín de reserva.
- TF07: Transfer de dos sentidos de movimiento. En simulación se ha sustituido por el TF07-1 y TF07-2 para realizar una simulación de funcionamiento
- AP08: Apilador/Desapilador de pilas de hasta 5 patines. En realidad, el software que usaremos no tiene un elemento capaz de simularlo, por lo que se sustituirá por una mesa de paso.

7.2.3. DESCRIPCIÓN DE LA ZONA 1

El sinóptico de la zona 1 es el siguiente:

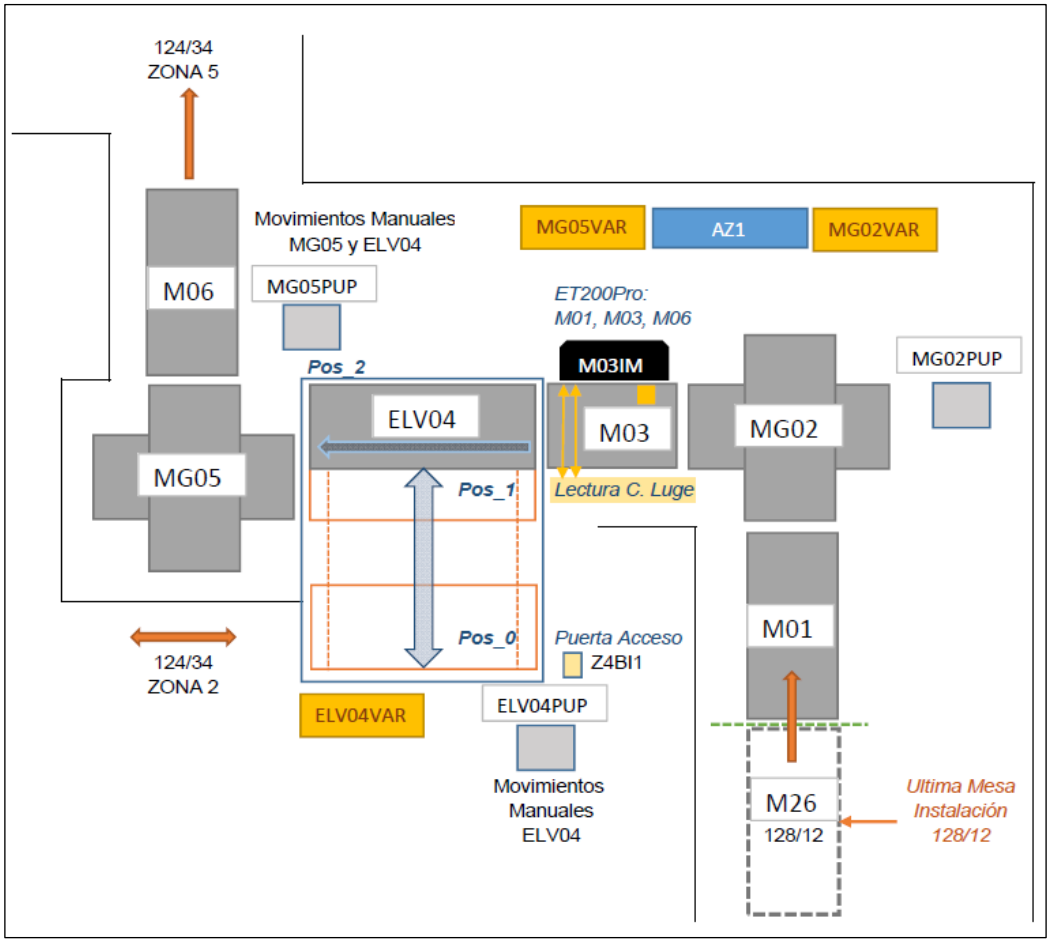


Figura 69 – Descripción de la Zona 1

Esta zona cuenta con los siguientes elementos:

- **M01**: Esta mesa no se incluirá en la simulación
- **M03**: Esta mesa no se incluirá en la simulación
- **M06**: Mesa de rodillos.
- **MG02**: Esta mesa no se incluirá en la simulación.
- **MG25**: Mesa pivotante de dos posiciones
- **ELV04**: Elevador de Tres Posiciones, pero con dos posiciones programadas.

7.2.4. DESCRIPCIÓN DE LA ZONA 5

El sinóptico de la zona 5 es el siguiente:

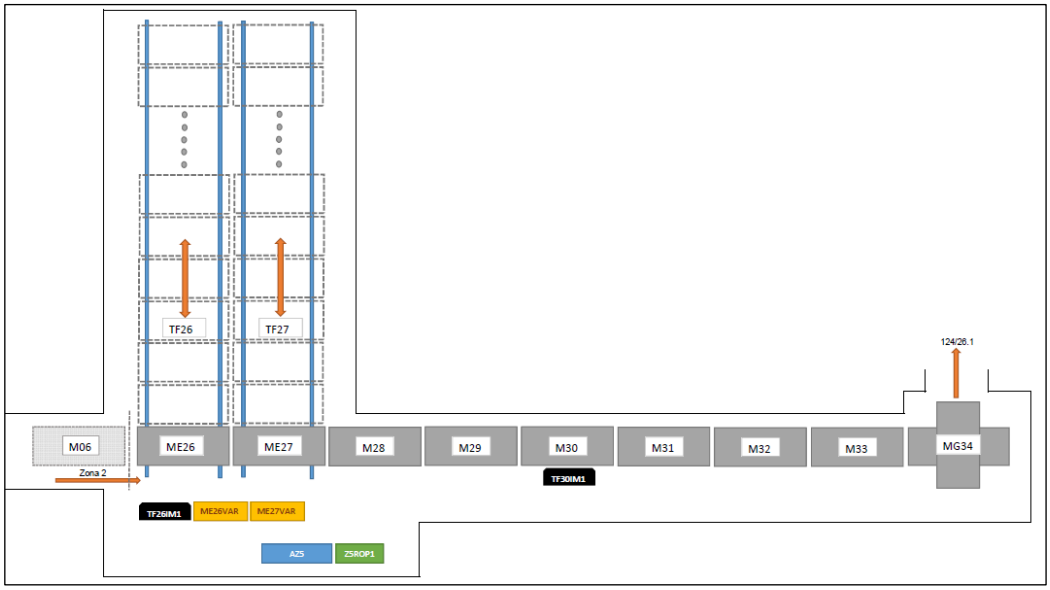


Figura 70 – Descripción de la Zona 5

Esta zona cuenta con los siguientes elementos:

- ME26: Mesa de Giro pivotante.
- TF26: Stock de Cadena para almacenamiento de patines.
- ME27: Mesa de Giro pivotante.
- TF27: Stock de Cadena para almacenamiento de patines.
- M28: Mesa de Paso.
- M29: Mesa de Paso.
- M30: Mesa de Paso.
- M31: Mesa de Paso.
- M32: Mesa de Paso.
- M33: Mesa de Paso.
- MG34: Mesa de Giro pivotante.

7.3. ANEXO 3: INSTALACIÓN Y CONFIGURACIÓN DEL SOFTWARE

Para la realización de este proyecto se han necesitado los siguientes programas, ordenados por el orden de instalación, tal y como se desarrollará a continuación (González 2015).

1. Siemens SIMATIC STEP 7 v5.5 SP1 + PLCSIM v5.4 SP5 64bits (Siemens 2017)
    1.1. Siemens SIMATIC STEP 7 v5.5 SP1
2. Factory I/O 2.1.3 (Factory I/O s.f.)
3. Nettoplcsim v1.2.1.0 (NetToPLCSim s.f.)
4. KEPServerEX 6.1.601.0 (KepServerEX s.f.)
5. Anaconda con Phyton 2.7 v4.3.1 32bits (Anaconda s.f.)
6. OpenOPC 1.3.1 32bits Phyton 2.7 (OpenOPC s.f.)
7. PyQt4 v4.11.4 para Phyton 2.7 Qt4.8.7 32bits (PyQt s.f.)
8. Openpyxl 2.4.1 para Phyton 2.7 (OpenPyXL s.f.)

He de comentar que para la instalación de este software se ha partido desde un ordenador, con Windows 7 Profesional de 64bits, recién formateado para que no queden restos de otras instalaciones previas.

7.3.1. INSTALACIÓN SIEMENS STEP 7 + PLCSIM

En primer lugar, instalaremos el STEP7 v.55 con SP1, el cual incluye la versión 5.4 con SP5 de PLCSIM.

Para ello seguiremos el asistente a la instalación y cuando nos pida elegir qué componentes queremos instalar debemos seleccionar todos los componentes como vemos en la siguiente figura.

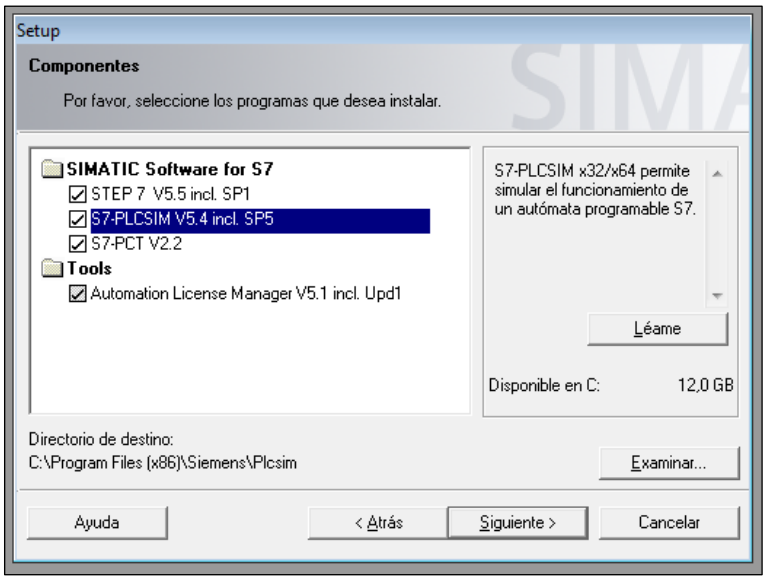


Figura 71 – Instalación Componentes STEP7

Una vez escogidos todos los componentes esperaremos a que la instalación finalicé. Cuando esta finalicé, será el momento de abrir el programa que gestiona las licencias e introducir nuestra licencia. Si no contamos con licencia la versión del software que adjuntamos en ese DVD cuenta con una versión de prueba de 14 días.

Una vez instalada esta versión procederemos a instalar la versión con SP4 de STEP7. Muy importante, al ejecutar el instalador, se descomprimirán los archivos de instalación en una ruta especificada y marcaremos la opción de no instalar después de extraer:

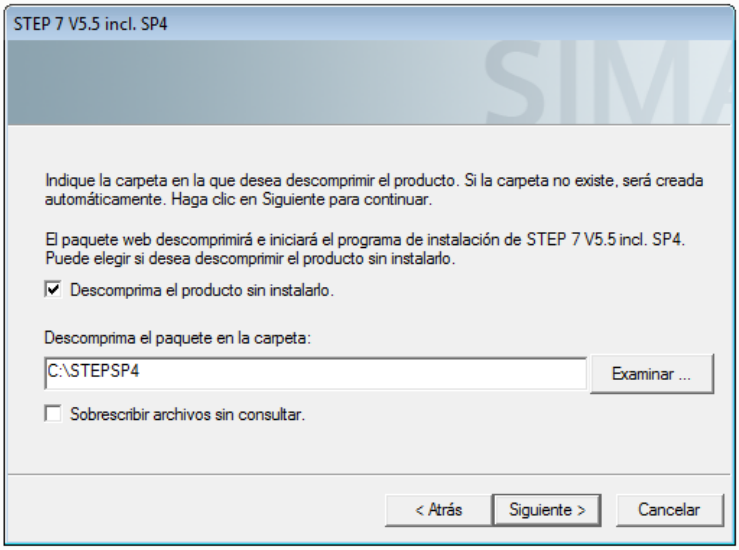


Figura 72 – Instalación STEP7 SP4

Una vez descomprimido debemos entrar en la ruta elegida, en la carpeta "InstData" y borrar la carpeta denominada "Automation License Manager" para evitar problemas con las licencias.

Una vez borrado este archivo procederemos con su instalación, y cuando ésta acabe habremos completado la instalación de este software.

7.3.2. CONFIGURACIÓN SIEMENS STEP 7 + PLCSIM

Una vez tenemos instalado el software y creado el proyecto debemos configurar el interfaz de conexión para conectar el programa de autómata con el de simulación. Para ello dentro de SIMATIC Manager, en la pestaña de herramientas, pulsamos sobre "Ajustar Interface PG/PC" y en la nueva ventana seleccionamos la opción PLCSIM.TCPIP.1

Dentro de PLCSIM debemos de cerciorarnos que la opción seleccionada es PLCSIM(TCP/IP).

7.3.3. INSTALACIÓN FACTORY I/O

Para instalar este software solo hay que ejecutar el archivo de instalación y seguir con el asistente de la instalación hasta que este finalice.

Una vez instalado contaremos con una licencia de evaluación de un mes, y posteriormente si queremos seguir usando el programa deberemos introducir una licencia valida.

7.3.4. CONFIGURACIÓN FACTORY I/O

En este programa se deben realizar diversas configuraciones. En primer lugar, se debe configurar para que establezca conexión con PLCSIM. Para ello necesitaremos tener el modelo de Factory I/O creado y tener configuradas las cartas de entradas salidas del autómata.

Para conjurarlo entraremos en File ► Drivers y pulsaremos en la opción de Configuración. Deberemos marcar "Siemens S7-PLCSIM" y configurarlo como se puede ver en la siguiente figura.

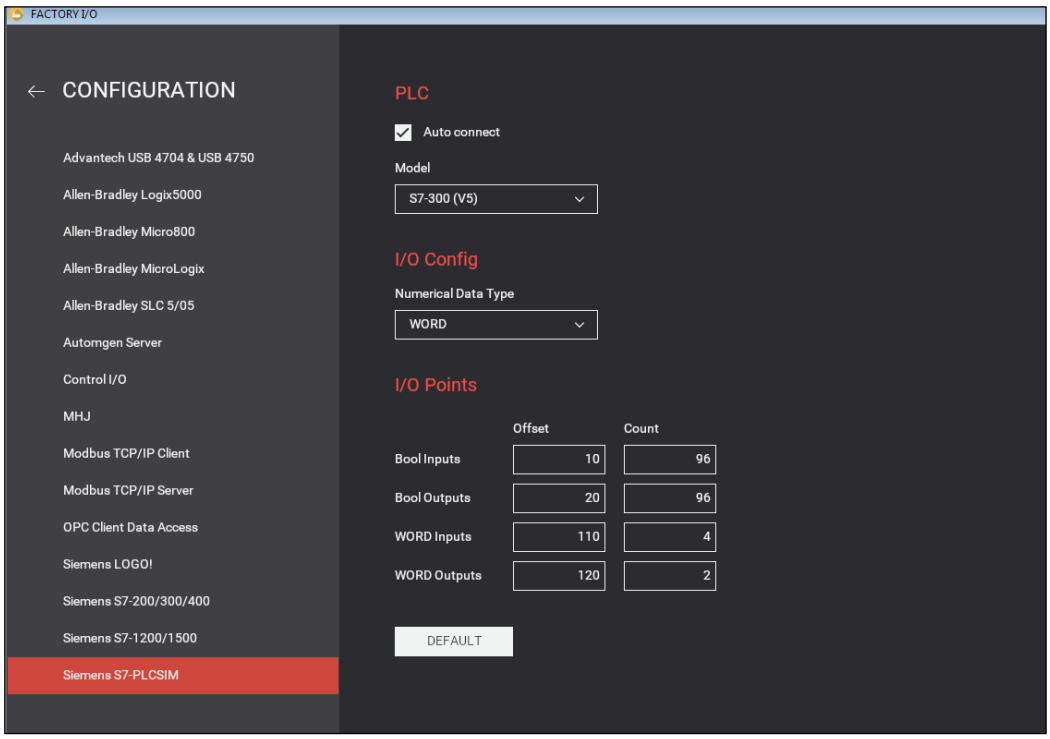


Figura 73 – Configuración Driver Factory I/O

Seleccionamos la opción de autoconectar, para que cada vez que abrimos el software de simulación se inicie la conexión, seccionamos el tiempo de CPU elegida, y configuramos las entradas y salidas en función de lo configurado en el Hardware de STEP7. En nuestro caso como las entradas comienzan en la I10.0 y tenemos 6 cartas de 16 bits, tendremos desde la I10.0 hasta la I21.7, es decir tendremos 96 entradas configurables. Para conectar las variables de Factory I/O con las entradas salidas únicamente deberemos de arrastrar la entrada o salida que queremos conectar con el puerto simulado de autómata, y nos debe quedar algo como vemos en la siguiente figura.

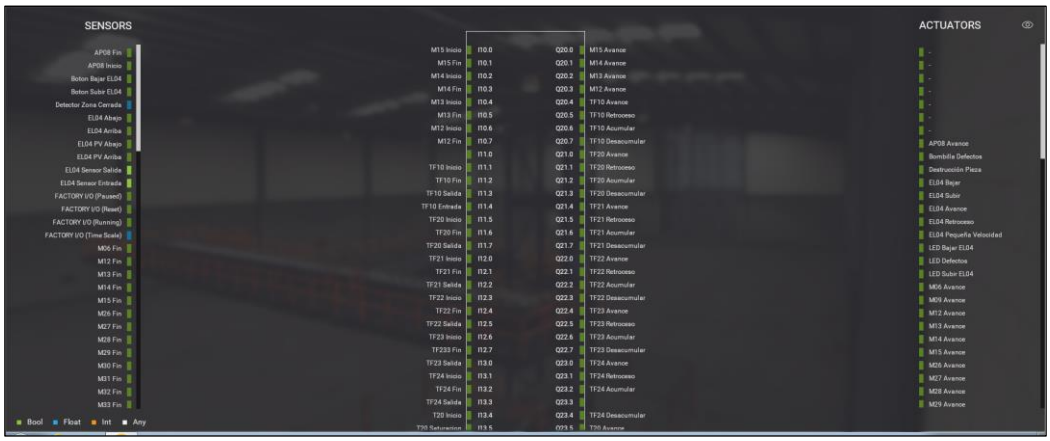


Figura 74 – Conexión Factory I/O con PLCSIM

7.3.5. CONFIGURACIÓN NETTOPLCSIM

Esto software no necesita ser instalado ya que es un software portable, es decir, una vez descomprimido ejecutamos el instalable y el software funcionará. Muy importante es ejecutar este programa todas las veces en modo administrador.

Cada vez que ejecutemos el programa nos pedirá usar el puerto 102, lo cual tendremos que aceptar ya que es el puerto que usará para conectarse con PLCSim.

Para efectuar la configuración debemos tener PLCSim en funcionamiento, dentro de NetToPLCSim pulsaremos en añadir. Se nos abrirá una ventana donde le daremos un nombre al autómata, en este caso elegiremos PLC, para las direcciones IP su pulsamos sobre el botón con tres puntos suspensivos y nos sale una ayuda de las posibles direcciones IP seleccionables siempre y cuando tengamos PLCSIM en funcionamiento.

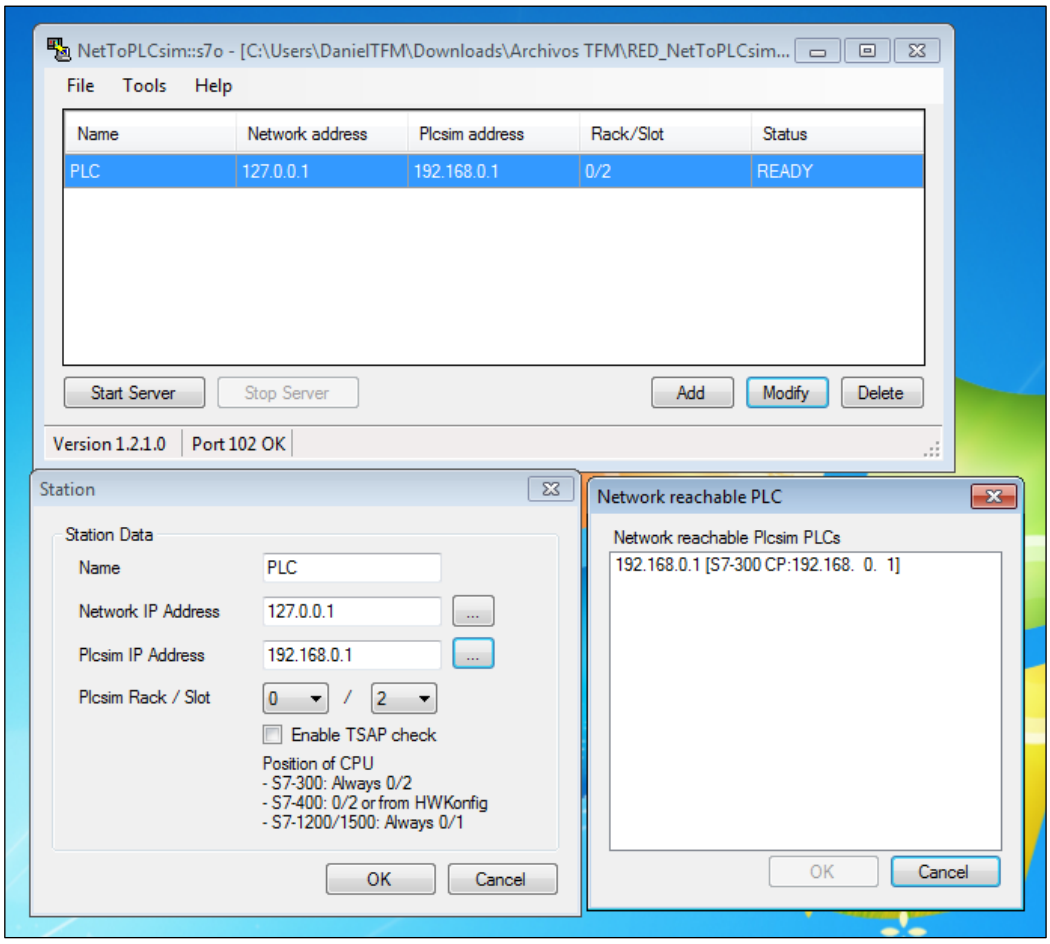


Figura 75 – Configuración NetToPLCSim

Una vez configurado deberemos de pulsar en "Start Server" para iniciar la conexión.

7.3.6. INSTALACIÓN KEPSERVEREX6

Para instalar este software solo hay que ejecutar el archivo de instalación y seguir con el asistente de la instalación hasta que éste finalice.

Una vez instalado contaremos con una licencia de evaluación para uso del programa limitada a 2 horas de uso por cada ejecución, necesitando reiniciar el programa si queremos seguir usándolo, pero suficiente para la realización de este trabajo.

7.3.7. CONFIGURACIÓN KEPSERVEREX6

En este caso deberemos configurar diferentes parámetros para poder crear el servidor OPC. En primer lugar, en Conectividad deberemos de crear un nuevo canal y seleccionaremos Siemens TCP/IP Ethernet y de nombre le daremos PLC.

Una vez tenemos el canal creado deberemos añadir un dispositivo que será nuestro dispositivo de PLCSim a través de NetToPLCSim. Para ello dando con el botón derecho en el canal seleccionaremos añadir nuevo dispositivo. En el asistente de creación le daremos un nombre, en nuestro caso también le llamaremos PLC, posteriormente debemos de seleccionar el modelo de CPU, y para este trabajo será S7-300, a continuación, nos pide una IP y debe ser la misma que la dirección de red hemos configurado en NetToPLCSim. Siguiendo nos pedirá unos parámetros característicos de una conexión OPC, los cuales los dejaremos por defecto.

Nos pedirá el puerto a usar que será el 102 tal y como nos pedía el programa NetToPLCSim y por último seleccionaremos el programa de autómata para poder importar todos los tags creados como vemos en la siguiente figura.

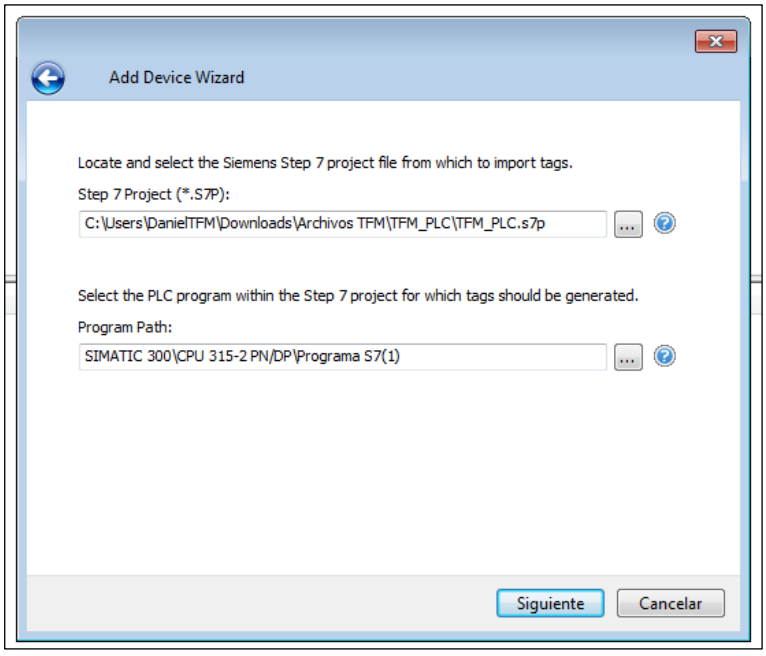


Figura 76 – Configuración TAGS en KEPSERVEREX6

Para acabar crearemos un grupo con todos estos tags para que un cliente pueda importarlos de una sola vez. Para ello en el dispositivo creado con el

botón derecho seleccionaremos añadir nuevo grupo de tags y le llamaremos TAGS, arrastraremos todos los tags que tenemos en el dispositivo PLC a este nuevo grupo y tenemos que ver algo como lo siguiente:

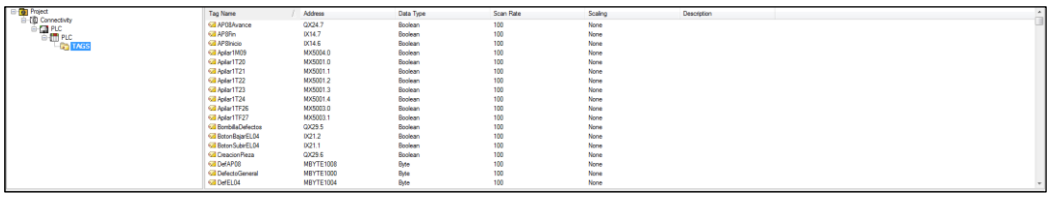


Figura 77 – Grupo TAGS en KEPSERVEREX6

Ahora tendremos el software funcionando y podremos probar que todo funciona correctamente con la herramienta QuickClient que trae este software. Para ello necesitaremos tener en funcionamiento PLCSIM, con NetToPLCSim y KEPSERVEREX6. Abriremos la herramienta y si vemos el valor de los TAGS la conexión habrá quedado establecida. Deberemos ver algo como en la siguiente imagen.

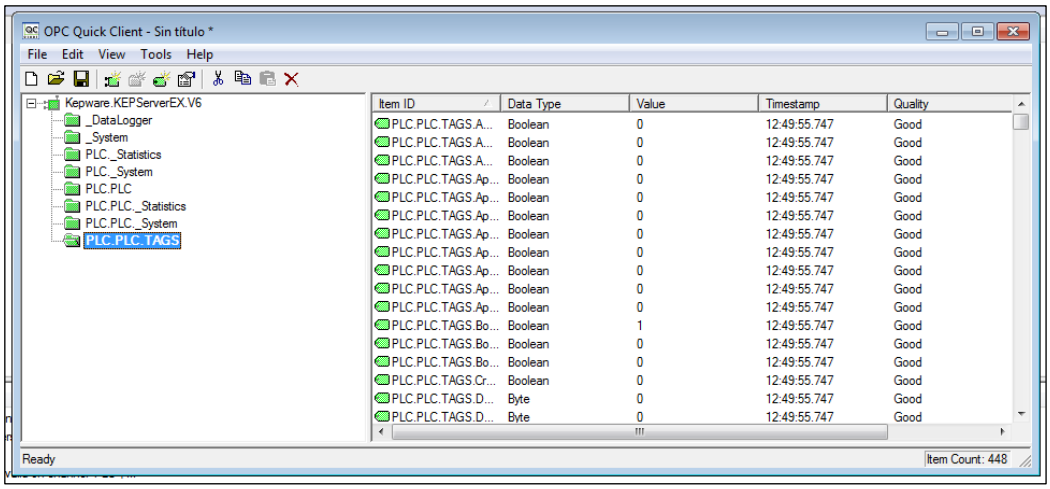


Figura 78 – QuickClient

7.3.8. INSTALACIÓN ANACONDA CON SPYDER

Para instalar este software solo hay que ejecutar el archivo de instalación y seguir con el asistente de la instalación hasta que éste finalice.

Si bien instalaremos el programa en C:\Anaconda como vemos en la siguiente figura.

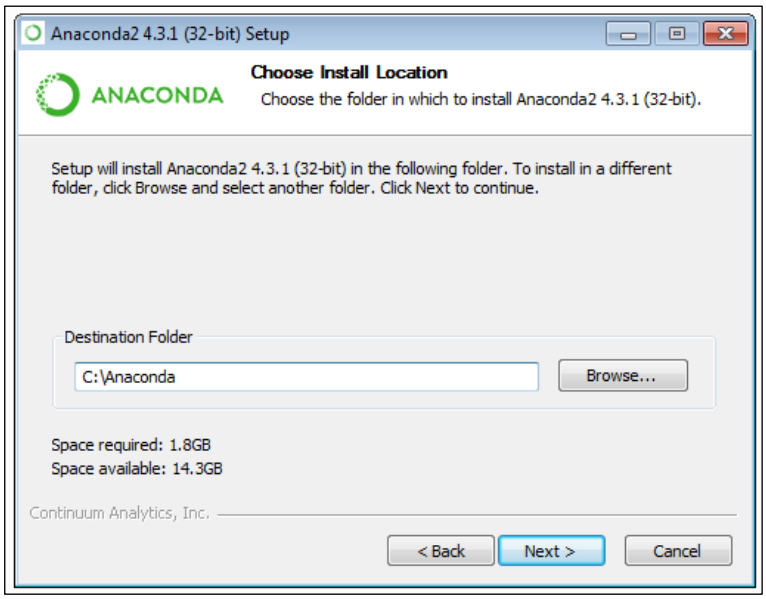


Figura 79 – Instalación Anaconda

Este programa no necesitará configuración adicional, si bien para la realización de este proyecto solo usaremos el entorno de programación Spyder con Phyton (Jones 2013) y no usaremos el resto de características que trae Anaconda.

7.3.9. INSTALACIÓN OPENOPC

Para instalar este software solo hay que ejecutar el archivo de instalación y seguir con el asistente de la instalación hasta que este finalice.

El software se instalará en la ruta por defecto que es en el directorio raíz de C, deberemos copiar la carpeta OpenOPC y pegarla en "C\Anaconda\Lib\site-packages" si es que hemos instalado Anaconda en la ruta por defecto.

7.3.10. INSTALACIÓN PYQT

Para instalar PYQT lo haremos directamente desde Anaconda (Physicist s.f.). Para ello abriremos Anaconda Prompt y escribiremos "conda install pyqt=4.11". Cuando el proceso finalice la librería habrá quedado instalada. Tendremos que ver algo como en la siguiente figura. Para ese paso nuestro equipo tiene que disponer de conexión a internet.

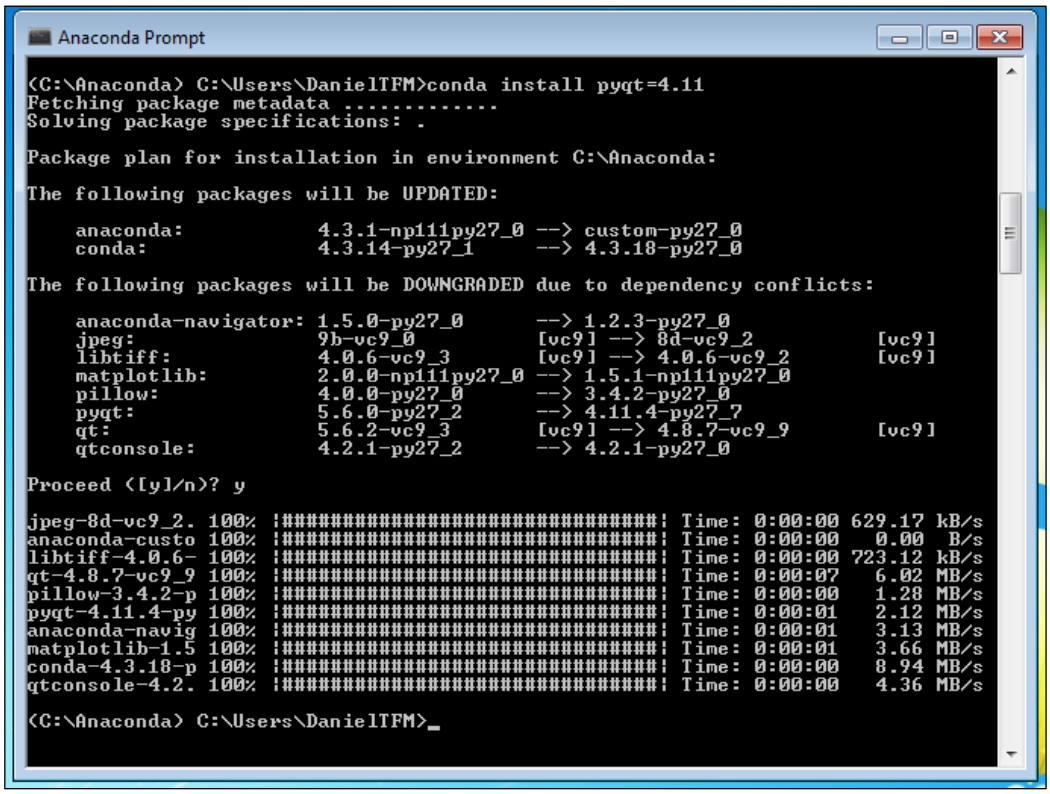


Figura 80 – Instalación PYQT

Una vez tengamos el software instalado, dentro de la ruta "C:\Anaconda\Library\bin" encontraremos la aplicación designer, que nos permitirá diseñar nuestras interfaces gráficas.

7.3.11. INSTALACIÓN OPENPYXL

Para instalar OpenPYXL lo haremos directamente desde Anaconda. Para ello abriremos Anaconda Prompt y escribiremos "conda install opepyxl". Cuando el proceso finalice la librería habrá quedado instalada. Tendremos que ver algo como en la siguiente figura. Para ese paso nuestro equipo tiene que disponer de conexión a internet.

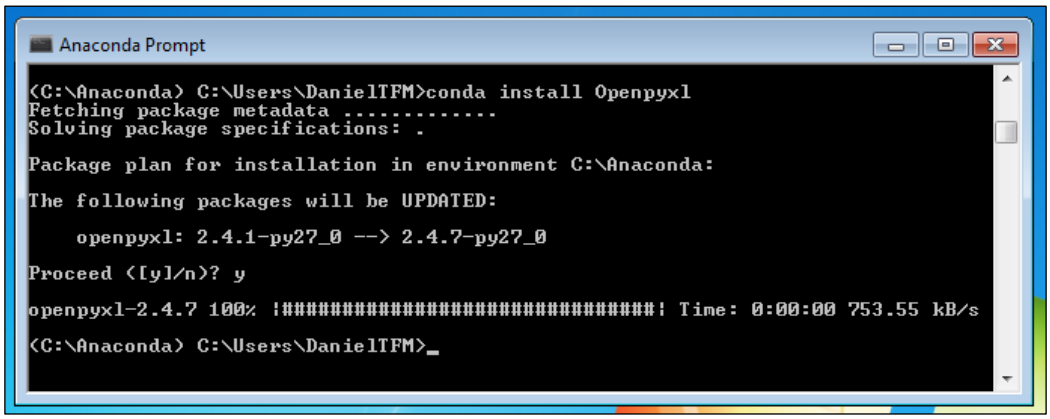


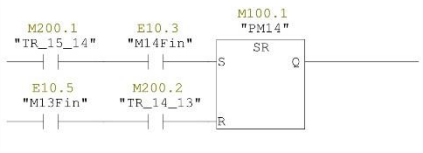
Figura 81 – Instalación OPENPYXL

7.4. ANEXO 4: EJEMPLOS CÓDIGOS EN IMPLEMENTACIÓN

7.4.1. CÓDIGO PROGRAMACIÓN LADDER MESA TRANSITO

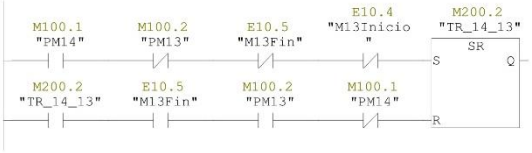
Segm.: 1

Memoria Pieza Pesente



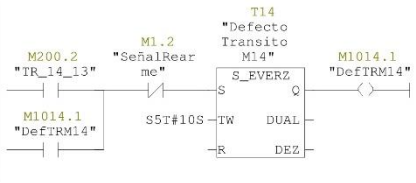
Segm.: 2

Gestión Transferencia A Mesa 13



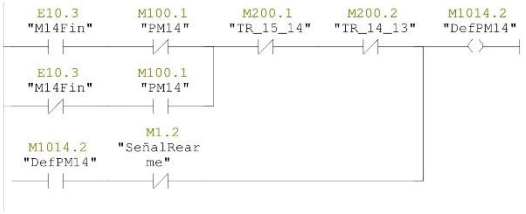
Segm.: 3

Defecto Tiempo de Transito Excesivo



Segm.: 4

Defecto Presencia



Segm.: 5

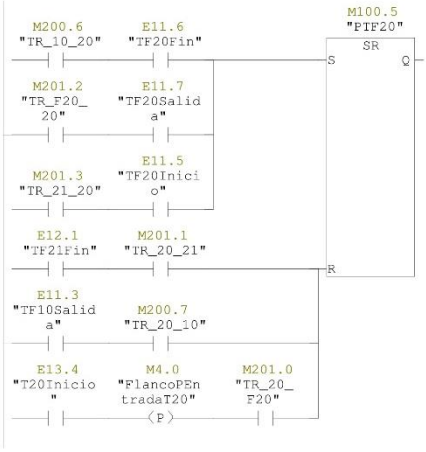
Orden de Avance

```
      U(
      U     "TR_14_13"          M200.2
      U(
      L     "DefM13"            MB1013
      L     0
      ==I
      )
      O
      U     "TR_15_14"          M200.1
      U(
      L     "DefM15"            MB1015
      L     0
      ==I
      )
      )
      U(
      L     "DefM14"            MB1014
      L     0
      ==I
      )
      U(
      L     "DefectoGeneral"    MB1000
      L     0
      ==I
      )
      =     "M14Avance"         A20.1
```

7.4.2. CÓDIGO PROGRAMACIÓN LADDER TRANSFER

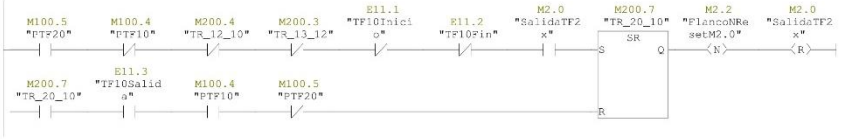
Segm.: 1

Memoria Pieza Pesente



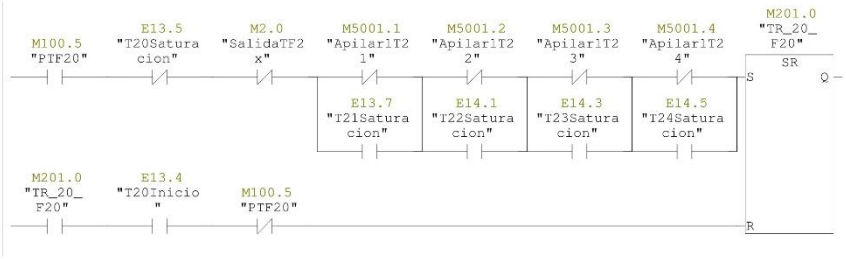
Segm.: 2

Gestión Transferencia A Trasfer 20



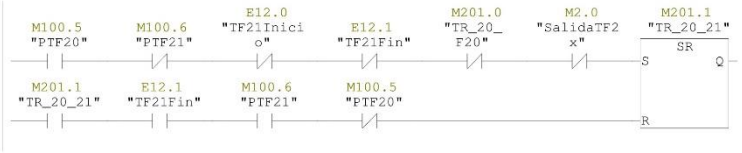
Segm.: 3

Gestión Transferencia A Trasfer 20



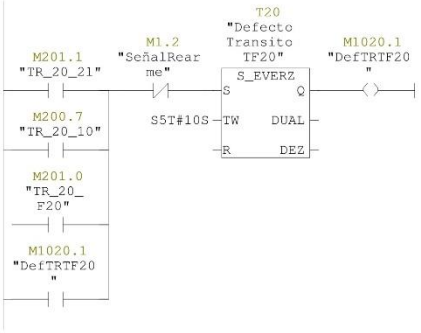
Segm.: 4

Gestión Transferencia A Trasfer 21



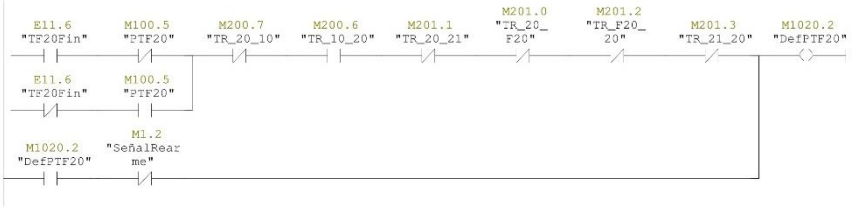
Segm.: 5

Defecto Tiempo de Transito Excesivo



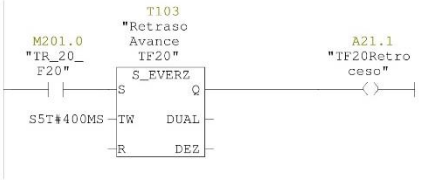
Segm.: 6

Defecto Presencia



Segm.: 7

Orden de Retroceso



Segm.: 8

Orden de Acumular

```
U(
U     "TR_10_20"          M200.6
U(
L     "DefTF10"           MB1010
L     0
==I
)
O
U     "TR_20_21"          M201.1
U(
L     "DefTF21"           MB1021
L     0
==I
)
)
U(
L     "DefTF20"           MB1020
L     0
==I
)
U(
L     "DefectoGeneral"    MB1000
L     0
==I
)
=     "TF20Acumular"      A21.2
```

Segm.: 9

Orden de Desacumular

```
U(
U     "TR_20_10"          M200.7
U(
L     "DefTF10"           MB1010
L     0
==I
)
O
U     "TR_21_20"          M201.3
U(
```

```
L     "DefTF21"          MB1021
L     0
==I
)
)
U(
L     "DefTF20"          MB1020
L     0
==I
)
U(
L     "DefectoGeneral"   MB1000
L     0
==I
)
=     "TF20Desacumular"  A21.3
```

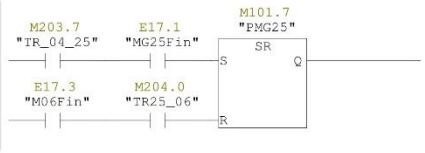
Segm.: 10
Orden de Avance

```
U     "TR_F20_20"        M201.2
U(
L     "DefT20"           MB10200
L     0
==I
)
U(
L     "DefTF20"          MB1020
L     0
==I
)
U(
L     "DefectoGeneral"   MB1000
L     0
==I
)
=     "TF20Avance"       A21.0
```

7.4.3. CÓDIGO PROGRAMACIÓN LADDER MESA GIRATORIA

Segm.: 1

Memoria Pieza Pesente



Segm.: 2

Ggestion Giro Para Entrada

```
UN    "PMG25"              M101.7
UN    "MG25GiradaEntrada"  E16.6
U(
L     "DefMG25"            MB1025
L     0
==I
)
U(
L     "DefectoGeneral"     MB1000
L     0
==I
)
=     "MG25GirarEntrada"   A26.5
```

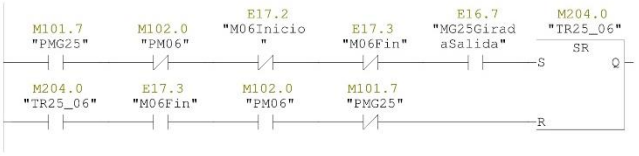
Segm.: 3

Gestion Giro Para Salida

```
U     "PMG25"              M101.7
UN    "MG25GiradaSalida"   E16.7
U(
L     "DefMG25"            MB1025
L     0
==I
)
U(
L     "DefectoGeneral"     MB1000
L     0
==I
)
=     "MG25GirarSalida"    A26.6
```

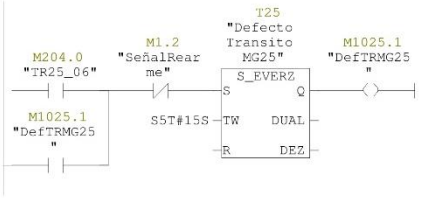
Segm.: 4

Gestión Transferencia A Mesa 06



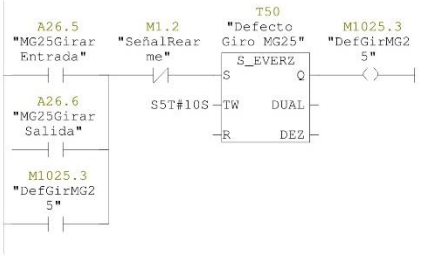
Segm.: 5

Defecto Tiempo de Transito Excesivo



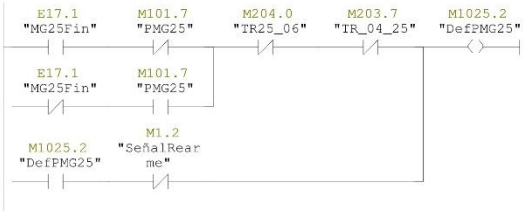
Segm.: 6

Defecto Tiempo de Gito Excesivo



Segm.: 7

Defecto Presencia



Segm.: 8

Orden de Avance

```
U(
U     "TR_04_25"          M203.7
U(
L     "DefEL04"           MB1004
L     0
==I
)
O
U     "TR25_06"           M204.0
U(
L     "DefM06"            MB1006
L     0
==I
)
)
U(
L     "DefMG25"           MB1025
L     0
==I
)
U(
L     "DefectoGeneral"    MB1000
L     0
==I
)
=     "MG25Avance"        A26.7
```

Página 3 de 3

7.4.4. CÓDIGO PROGRAMACIÓN LADDER ELEVADOR

Segm.: 1

Flanco Entrada Pieza

Segm.: 2

Memoria Pieza Pesente

Segm.: 3

Gestión Elevacion Nivel 1

```
U(
U     "PEL04"               M101.6
UN    "EL04Arrba"           E16.1
U     "SelectorAutomatico"  E20.7
O
U     "SelectorManual"      E21.0
U     "BotonSubirEL04"      E21.1
)
U(
L     "DefEL04"             MB1004
L     0
==I
)
U(
L     "DefectoGeneral"      MB1000
L     0
==I
```

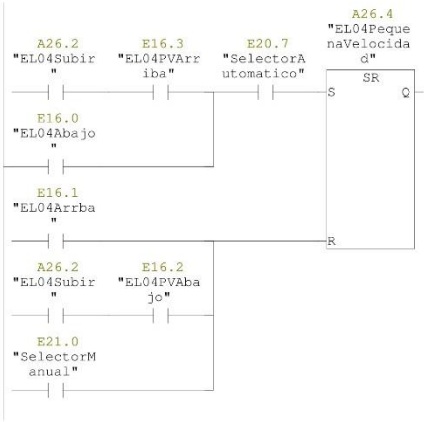
Página 1 de 4

```
      )
      =     "EL04Subir"           A26.2
```

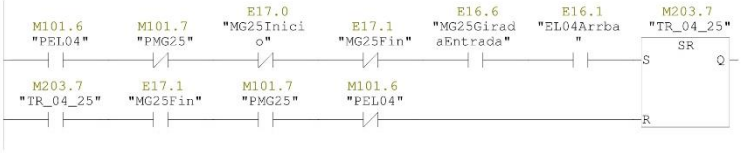
Segm.: 4
Gestión Elevacion Nivel 0

```
      U(
      UN    "PEL04"               M101.6
      UN    "EL04Abajo"           E16.0
      U     "SelectorAutomatico"  E20.7
      O
      U     "SelectorManual"      E21.0
      U     "BotonBajarEL04"      E21.2
      )
      U(
      L     "DefEL04"             MB1004
      L     0
      ==I
      )
      U(
      L     "DefectoGeneral"      MB1000
      L     0
      ==I
      )
      =     "EL04Bajar"           A26.3
```

Segm.: 5
Gestion Pequeña Velocidad

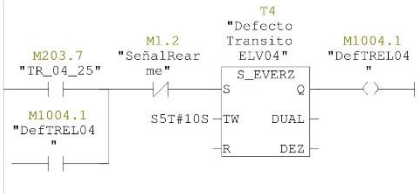


Segm.: 6
Gestión Transferencia a Mesa Giratoria 25



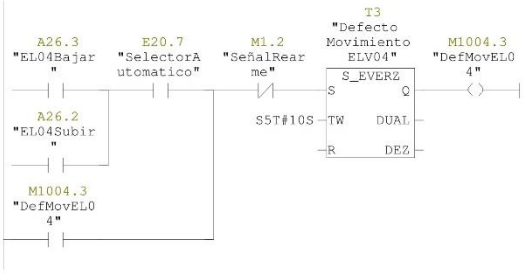
Segm.: 7

Defecto Tiempo de Transito Excesivo



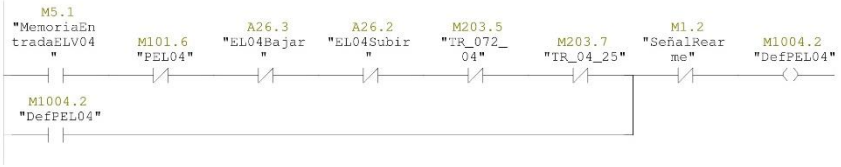
Segm.: 8

Defecto Tiempo de Movimiento en Subida o Bajada



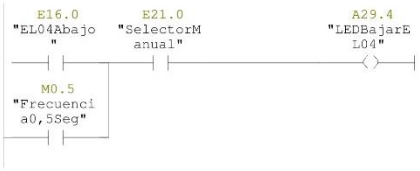
Segm.: 9

Defecto Presencia



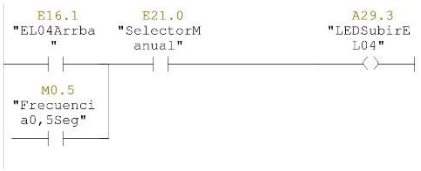
Segm.: 10

LED Bajar EL04



Segm.: 11

LED Subir EL04



Segm.: 12

Orden de Avance

```
U     "TR_072_04"        M203.5
U(
L     "DefEL04"          MB1004
L     0
==I
)
U(
L     "DefIF07_2"        MB10072
L     0
==I
)
U(
L     "DefectoGeneral"   MB1000
L     0
==I
)
=     "EL04Avance"       A26.0
```

Segm.: 13

Orden de Retroceso

```
U     "TR_04_25"         M203.7
U(
L     "DefEL04"          MB1004
L     0
==I
)
U(
L     "DefMG25"          MB1025
L     0
==I
)
U(
L     "DefectoGeneral"   MB1000
L     0
==I
)
=     "EL04Retroceso"    A26.1
```

7.4.5. CÓDIGO PROGRAMACIÓN PHYTON CLIENTE OPC

```
#Hilo en el que programamos el cliente OPC
def clienteOPC(obj):
    global itemsOPC

    opc=OpenOPC.client()
    opc.connect('Kepware.KEPServerEX.V6')

    while not obj.finalizarClienteOPC:
        itemsOPC = opc.read(opc.list('PLC.PLC.TAGS'))

        if((obj.ui.OPCRearme.isChecked()==True) and ((itemsOPC[238])[1]==False)):
            opc.write(('PLC.PLC.TAGS.RearmeOPC', True))
        if((obj.ui.OPCRearme.isChecked()==False) and ((itemsOPC[238])[1]==True)):
            opc.write(('PLC.PLC.TAGS.RearmeOPC', False))
        if((obj.ui.OPCDeshabilitarT2X.isChecked()==True) and ((itemsOPC[121])[1]==Fa
            opc.write(('PLC.PLC.TAGS.DesahabilitarT2x', True))
        if((obj.ui.OPCDeshabilitarT2X.isChecked()==False) and ((itemsOPC[121])[1]==T
            opc.write(('PLC.PLC.TAGS.DesahabilitarT2x', False))
        if((obj.ui.OPCDeshabilitarTF26TF27.isChecked()==True) and ((itemsOPC[122])[1
            opc.write(('PLC.PLC.TAGS.DesahabilitarTF26TF27', True))
        if((obj.ui.OPCDeshabilitarTF26TF27.isChecked()==False) and ((itemsOPC[122])[
            opc.write(('PLC.PLC.TAGS.DesahabilitarTF26TF27', False))
        if((obj.ui.OPCApilarTF26.isChecked()==True) and ((itemsOPC[9])[1]==False)):
            opc.write(('PLC.PLC.TAGS.Apilar1TF26', True))
        if((obj.ui.OPCApilarTF26.isChecked()==False) and ((itemsOPC[9])[1]==True)):
            opc.write(('PLC.PLC.TAGS.Apilar1TF26', False))
        if((obj.ui.OPCDesapilarTF26.isChecked()==True) and ((itemsOPC[129])[1]==Fals
            opc.write(('PLC.PLC.TAGS.Desapilar1TF26', True))
        if((obj.ui.OPCDesapilarTF26.isChecked()==False) and ((itemsOPC[129])[1]==Tru
            opc.write(('PLC.PLC.TAGS.Desapilar1TF26', False))
        if((obj.ui.OPCApilarTF27.isChecked()==True) and ((itemsOPC[10])[1]==False)):
            opc.write(('PLC.PLC.TAGS.Apilar1TF27', True))
        if((obj.ui.OPCApilarTF27.isChecked()==False) and ((itemsOPC[10])[1]==True)):
            opc.write(('PLC.PLC.TAGS.Apilar1TF27', False))
        if((obj.ui.OPCDesapilarTF27.isChecked()==True) and ((itemsOPC[130])[1]==Fals
            opc.write(('PLC.PLC.TAGS.Desapilar1TF27', True))
        if((obj.ui.OPCDesapilarTF27.isChecked()==False) and ((itemsOPC[130])[1]==Tru
            opc.write(('PLC.PLC.TAGS.Desapilar1TF27', False))
        if((obj.ui.OPCApilarM09.isChecked()==True) and ((itemsOPC[3])[1]==False)):
            opc.write(('PLC.PLC.TAGS.Apilar1M09', True))
        if((obj.ui.OPCApilarM09.isChecked()==False) and ((itemsOPC[3])[1]==True)):
            opc.write(('PLC.PLC.TAGS.Apilar1M09', False))
        if((obj.ui.OPCDesapilarM09.isChecked()==True) and ((itemsOPC[123])[1]==False
            opc.write(('PLC.PLC.TAGS.Desapilar1M09', True))
        if((obj.ui.OPCDesapilarM09.isChecked()==False) and ((itemsOPC[123])[1]==True
            opc.write(('PLC.PLC.TAGS.Desapilar1M09', False))
        if((obj.ui.OPCApilarT20.isChecked()==True) and ((itemsOPC[4])[1]==False)):
            opc.write(('PLC.PLC.TAGS.Apilar1T20', True))
        if((obj.ui.OPCApilarT20.isChecked()==False) and ((itemsOPC[4])[1]==True)):
            opc.write(('PLC.PLC.TAGS.Apilar1T20', False))
        if((obj.ui.OPCDesapilarT20.isChecked()==True) and ((itemsOPC[124])[1]==False
            opc.write(('PLC.PLC.TAGS.Desapilar1T20', True))
        if((obj.ui.OPCDesapilarT20.isChecked()==False) and ((itemsOPC[124])[1]==True
            opc.write(('PLC.PLC.TAGS.Desapilar1T20', False))
        if((obj.ui.OPCApilarT21.isChecked()==True) and ((itemsOPC[5])[1]==False)):
            opc.write(('PLC.PLC.TAGS.Apilar1T21', True))
        if((obj.ui.OPCApilarT21.isChecked()==False) and ((itemsOPC[5])[1]==True)):
            opc.write(('PLC.PLC.TAGS.Apilar1T21', False))
        if((obj.ui.OPCDesapilarT21.isChecked()==True) and (itemsOPC[125][1]==False)):
            opc.write(('PLC.PLC.TAGS.Desapilar1T21', True))
```

```
    if((obj.ui.OPCDesapilarT21.isChecked()==False) and ((itemsOPC[125])[1]==True
        opc.write(('PLC.PLC.TAGS.Desapilar1T21', False))
    if((obj.ui.OPCApilarT22.isChecked()==True) and ((itemsOPC[6])[1]==False)):
        opc.write(('PLC.PLC.TAGS.Apilar1T22', True))
    if((obj.ui.OPCApilarT22.isChecked()==False) and ((itemsOPC[6])[1]==True)):
        opc.write(('PLC.PLC.TAGS.Apilar1T22', False))
    if((obj.ui.OPCDesapilarT22.isChecked()==True) and ((itemsOPC[126])[1]==False
        opc.write(('PLC.PLC.TAGS.Desapilar1T22', True))
    if((obj.ui.OPCDesapilarT22.isChecked()==False) and ((itemsOPC[126])[1]==True
        opc.write(('PLC.PLC.TAGS.Desapilar1T22', False))
    if((obj.ui.OPCApilarT23.isChecked()==True) and ((itemsOPC[7])[1]==False)):
        opc.write(('PLC.PLC.TAGS.Apilar1T23', True))
    if((obj.ui.OPCApilarT23.isChecked()==False) and ((itemsOPC[7])[1]==True)):
        opc.write(('PLC.PLC.TAGS.Apilar1T23', False))
    if((obj.ui.OPCDesapilarT23.isChecked()==True) and ((itemsOPC[127])[1]==False
        opc.write(('PLC.PLC.TAGS.Desapilar1T23', True))
    if((obj.ui.OPCDesapilarT23.isChecked()==False) and ((itemsOPC[127])[1]==True
        opc.write(('PLC.PLC.TAGS.Desapilar1T23', False))
    if((obj.ui.OPCApilarT24.isChecked()==True) and ((itemsOPC[8])[1]==False)):
        opc.write(('PLC.PLC.TAGS.Apilar1T24', True))
    if((obj.ui.OPCApilarT24.isChecked()==False) and ((itemsOPC[8])[1]==True)):
        opc.write(('PLC.PLC.TAGS.Apilar1T24', False))
    if((obj.ui.OPCDesapilarT24.isChecked()==True) and ((itemsOPC[128])[1]==False
        opc.write(('PLC.PLC.TAGS.Desapilar1T24', True))
    if((obj.ui.OPCDesapilarT24.isChecked()==False) and ((itemsOPC[128])[1]==True
        opc.write(('PLC.PLC.TAGS.Desapilar1T24', False))
    time.sleep(periodo)

obj.clienteOPCAcabado=True
return
```

7.4.6. CÓDIGO PROGRAMACIÓN PHYTON INTERFAZ GRÁFICA GENERAL CON QT

```
#Clase Qt para ejecución grafica de la ventana general
class ProyectoTFM(QtGui.QMainWindow):
    M15Red = QtCore.pyqtSignal(); M14Red = QtCore.pyqtSignal(); M13Red = QtCore.pyqt
    TF22Red = QtCore.pyqtSignal(); TF23Red = QtCore.pyqtSignal(); TF24Red = QtCore.p
    T24Red = QtCore.pyqtSignal(); AP08Red = QtCore.pyqtSignal(); M09Red = QtCore.pyc
    M06Red = QtCore.pyqtSignal(); M26Red = QtCore.pyqtSignal(); M27Red = QtCore.pyqt
    M30Red = QtCore.pyqtSignal(); M31Red = QtCore.pyqtSignal(); M32Red = QtCore.pyqt
    EmergenciaEn = QtCore.pyqtSignal();   IntrusionEn = QtCore.pyqtSignal(); DefRece
    M15Grey = QtCore.pyqtSignal(); M14Grey = QtCore.pyqtSignal(); M13Grey = QtCore.p
    TF22Grey = QtCore.pyqtSignal(); TF23Grey = QtCore.pyqtSignal(); TF24Grey = QtCor
    T24Grey = QtCore.pyqtSignal(); AP08Grey = QtCore.pyqtSignal(); M09Grey = QtCore.
    M06Grey = QtCore.pyqtSignal(); M26Grey = QtCore.pyqtSignal(); M27Grey = QtCore.p
    M30Grey = QtCore.pyqtSignal(); M31Grey = QtCore.pyqtSignal(); M32Grey = QtCore.p
    EmergenciaDis = QtCore.pyqtSignal();   IntrusionDis = QtCore.pyqtSignal(); DefRe
    OPCNOOK = QtCore.pyqtSignal(); OPCOK = QtCore.pyqtSignal();
    M15=True; M14=True; M13=True; M12=True; M11=True; TF10=True; TF20=True; TF21=Tru
    TF071=True; TF072=True; M09=True; EL04=True; MG25=True; M06=True; M26=True; M27=

    def __init__(self,parent=None):
        self.finalizarClienteOPC = False
        self.finalizarActualizarDatosOPC = False
        self.clienteOPCAcabado = False
        self.actualizarDatosOPCAcabado = False

        QtGui.QMainWindow.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.labelOPCOK.hide();
        self.startTheThread()

        self.M15Red.connect(lambda: self.ui.M15Button.setStyleSheet("background-colo
        self.TF22Red.connect(lambda: self.ui.TF22Button.setStyleSheet("background-co
        self.T24Red.connect(lambda: self.ui.T24Button.setStyleSheet("background-colo
        self.M06Red.connect(lambda: self.ui.M06Button.setStyleSheet("background-colo
        self.M30Red.connect(lambda: self.ui.M30Button.setStyleSheet("background-colo
        self.EmergenciaEn.connect(lambda: self.ui.labelDefectoEmergencia.show()); se
        self.M15Grey.connect(lambda: self.ui.M15Button.setStyleSheet("background-col
        self.TF22Grey.connect(lambda: self.ui.TF22Button.setStyleSheet("background-c
        self.T24Grey.connect(lambda: self.ui.T24Button.setStyleSheet("background-col
        self.M06Grey.connect(lambda: self.ui.M06Button.setStyleSheet("background-col
        self.M30Grey.connect(lambda: self.ui.M30Button.setStyleSheet("background-col
        self.EmergenciaDis.connect(lambda: self.ui.labelDefectoEmergencia.hide()); s
        self.OPCNOOK.connect(lambda: self.ui.labelOPCOK.hide()); self.OPCNOOK.connec
        self.M15Red.connect(lambda: self.escribeDefecto('M15', 'Aparece', (itemsOPC[
        self.TF22Red.connect(lambda: self.escribeDefecto('TF22', 'Aparece', (itemsOP
        self.T24Red.connect(lambda: self.escribeDefecto('T24', 'Aparece', (itemsOPC[
        self.M06Red.connect(lambda: self.escribeDefecto('M06', 'Aparece', (itemsOPC[
        self.M30Red.connect(lambda: self.escribeDefecto('M30', 'Aparece', (itemsOPC[
        self.EmergenciaEn.connect(lambda: self.escribeDefecto('Emergencia', 'Aparece
        self.M15Grey.connect(lambda: self.escribeDefecto('M15', 'Desaparece', (items
        self.TF22Grey.connect(lambda: self.escribeDefecto('TF22', 'Desaparece', (ite
        self.T24Grey.connect(lambda: self.escribeDefecto('T24', 'Desaparece', (items
        self.M06Grey.connect(lambda: self.escribeDefecto('M06', 'Desaparece', (items
        self.M30Grey.connect(lambda: self.escribeDefecto('M30', 'Desaparece', (items
        self.EmergenciaDis.connect(lambda: self.escribeDefecto('Emergencia', 'Desapa
        self.ui.M15Button.clicked.connect(self.ventanaM15)
        self.ui.M14Button.clicked.connect(self.ventanaM14)
        self.ui.M13Button.clicked.connect(self.ventanaM13)
        self.ui.M12Button.clicked.connect(self.ventanaM12)
```

```
        self.ui.TF10Button.clicked.connect(self.ventanaTF10)
        self.ui.AP08Button.clicked.connect(self.ventanaAP08)
        self.ui.TF071Button.clicked.connect(self.ventanaTF071)
        self.ui.TF072Button.clicked.connect(self.ventanaTF072)
        self.ui.M09Button.clicked.connect(self.ventanaM09)
        self.ui.TF20Button.clicked.connect(self.ventanaTF20)
        self.ui.TF21Button.clicked.connect(self.ventanaTF21)
        self.ui.TF22Button.clicked.connect(self.ventanaTF22)
        self.ui.TF23Button.clicked.connect(self.ventanaTF23)
        self.ui.TF24Button.clicked.connect(self.ventanaTF24)
        self.ui.T20Button.clicked.connect(self.ventanaT20)
        self.ui.T21Button.clicked.connect(self.ventanaT21)
        self.ui.T22Button.clicked.connect(self.ventanaT22)
        self.ui.T23Button.clicked.connect(self.ventanaT23)
        self.ui.T24Button.clicked.connect(self.ventanaT24)
        self.ui.EL04Button.clicked.connect(self.ventanaEL04)
        self.ui.MG25Button.clicked.connect(self.ventanaMG25)
        self.ui.MG34Button.clicked.connect(self.ventanaMG34)
        self.ui.M06Button.clicked.connect(self.ventanaM06)
        self.ui.M28Button.clicked.connect(self.ventanaM28)
        self.ui.M29Button.clicked.connect(self.ventanaM29)
        self.ui.M30Button.clicked.connect(self.ventanaM30)
        self.ui.M31Button.clicked.connect(self.ventanaM31)
        self.ui.M32Button.clicked.connect(self.ventanaM32)
        self.ui.M33Button.clicked.connect(self.ventanaM33)
        self.ui.M26Button.clicked.connect(self.ventanaM26)
        self.ui.M27Button.clicked.connect(self.ventanaM27)
        self.ui.TF26Button.clicked.connect(self.ventanaTF26)
        self.ui.TF27Button.clicked.connect(self.ventanaTF27)
        self.ui.diganosticoButton.clicked.connect(self.ventanaDiagnostico)
        self.ui.cerrarButton.clicked.connect(self.cerrarApp)

    def startTheThread(self):
        self.threadOPC = threading.Thread(target=clienteOPC, args=(self,))
        self.threadOPC.start()
        threadQt = threading.Thread(target = actualizarDatosOPC, args=(self,))
        threadQt.start()

    def ventanaM15(self):
        cicladoM15(self).show()
    def ventanaM14(self):
        cicladoM14(self).show()
    def ventanaM13(self):
        cicladoM13(self).show()
    def ventanaM12(self):
        cicladoM12(self).show()
    def ventanaTF10(self):
        cicladoTF10(self).show()
    def ventanaAP08(self):
        cicladoAP08(self).show()
    def ventanaTF071(self):
        cicladoTF071(self).show()
    def ventanaTF072(self):
        cicladoTF072(self).show()
    def ventanaM09(self):
        cicladoM09(self).show()
    def ventanaTF20(self):
        cicladoTF20(self).show()
    def ventanaTF21(self):
        cicladoTF21(self).show()
```

```
def ventanaTF22(self):
    cicladoTF22(self).show()
def ventanaTF23(self):
    cicladoTF23(self).show()
def ventanaTF24(self):
    cicladoTF24(self).show()
def ventanaT20(self):
    cicladoT20(self).show()
def ventanaT21(self):
    cicladoT21(self).show()
def ventanaT22(self):
    cicladoT22(self).show()
def ventanaT23(self):
    cicladoT23(self).show()
def ventanaT24(self):
    cicladoT24(self).show()
def ventanaEL04(self):
    cicladoEL04(self).show()
def ventanaMG25(self):
    cicladoMG25(self).show()
def ventanaMG34(self):
    cicladoMG34(self).show()
def ventanaM06(self):
    cicladoM06(self).show()
def ventanaM28(self):
    cicladoM28(self).show()
def ventanaM29(self):
    cicladoM29(self).show()
def ventanaM30(self):
    cicladoM30(self).show()
def ventanaM31(self):
    cicladoM31(self).show()
def ventanaM32(self):
    cicladoM32(self).show()
def ventanaM33(self):
    cicladoM33(self).show()
def ventanaM26(self):
    cicladoM26(self).show()
def ventanaM27(self):
    cicladoM27(self).show()
def ventanaTF26(self):
    cicladoTF26(self).show()
def ventanaTF27(self):
    cicladoTF27(self).show()
def ventanaDiagnostico(self):
    muestraDiagnostico(self).show()

def escribeDefecto(self, elemento, estado, defecto):
    if self.ConOPC==True:
        semaforo.acquire();
        doc = openpyxl.load_workbook('../REGISTRO.xlsx')
        hoja = doc.get_sheet_by_name('Hoja1')
        fila=hoja.max_row+1
        day = time.strftime("%x")
        hour = time.strftime("%X")
        hoja.cell(row=fila,column=1).value=day
        hoja.cell(row=fila,column=2).value=hour
        hoja.cell(row=fila,column=3).value=estado
        hoja.cell(row=fila,column=4).value=elemento
        if(defecto==2 and estado=='Aparece'):


            hoja.cell(row=fila,column=5).value='Defecto Transito'
        elif(defecto==4 and estado=='Aparece'):
            hoja.cell(row=fila,column=5).value='Defecto Presencia'
        elif(defecto==8 and estado=='Aparece'):
            hoja.cell(row=fila,column=5).value='Defecto Movimiento'
        elif(defecto!=0 and estado=='Aparece'):
            hoja.cell(row=fila,column=5).value='Desconocido'
        doc.save("../REGISTRO.xlsx")
        doc.close
        semaforo.release();

def cerrarApp(self):
    while not self.clienteOPCAcabado:
        self.finalizarClienteOPC = True
    while not self.actualizarDatosOPCAcabado:
        self.finalizarActualizarDatosOPC = True
    opc.close()
```

7.4.7. CÓDIGO PROGRAMACIÓN PHYTON INTERFAZ GRÁFICA ELEMENTO FUNCIONAL CON QT

```
#Hilo para actualizar graficamente la ventana M14 en funcion de los datos OPC leidos
def threadM14(obj):
    while not obj.finalizarThread:
        if(itemsOPC[163])[1]==True:
            obj.sensorInicio1.emit()
        else:
            obj.sensorInicio0.emit()
        if(itemsOPC[162])[1]==True:
            obj.sensorFin1.emit()
        else:
            obj.sensorFin0.emit()
        if(itemsOPC[217])[1]==True:
            obj.presencia1.emit()
        else:
            obj.presencia0.emit()
        if(itemsOPC[335])[1]==True:
            obj.transitoM131.emit()
        else:
            obj.transitoM130.emit()
        if(itemsOPC[45])[1]==True:
            obj.defectoPresencia1.emit()
        else:
            obj.defectoPresencia0.emit()
        if(itemsOPC[94])[1]==True:
            obj.defectoTransito1.emit()
        else:
            obj.defectoTransito0.emit()
        time.sleep(periodo)

    obj.threadAcabado=True
    return


#Clase Qt para ejecución grafica de la ventana de M14
class cicladoM14(QtGui.QMainWindow):
    sensorInicio0 = QtCore.pyqtSignal(); sensorFin0 = QtCore.pyqtSignal(); presencia
    sensorInicio1 = QtCore.pyqtSignal(); sensorFin1 = QtCore.pyqtSignal(); presencia

    def __init__(self,parent=None):
        self.finalizarThread = False
        self.threadAcabado = False

        QtGui.QMainWindow.__init__(self,parent)
        self.ui = Ui_widgetM14()
        self.ui.setupUi(self)

        self.sensorInicio0.connect(lambda: self.ui.labelSensorInicio.setStyleSheet('
        self.sensorFin0.connect(lambda: self.ui.labelSensorFin.setStyleSheet("backgr
        self.presencia0.connect(lambda: self.ui.labelPresencia.setStyleSheet("backgr
        self.transitoM130.connect(lambda: self.ui.labelTransitoM13.setStyleSheet("ba
        self.defectoPresencia0.connect(lambda: self.ui.labelDefectoPresencia.hide())
        self.defectoTransito0.connect(lambda: self.ui.labelDefectoTransito.hide());
        self.ui.buttonPresencia0.clicked.connect(self.forzarPresencia0)
        self.ui.buttonPresencia1.clicked.connect(self.forzarPresencia1)
        self.ui.buttonTransitoM130.clicked.connect(self.forzarTransitoM130)
        self.ui.buttonTransitoM131.clicked.connect(self.forzarTransitoM131)
        self.ui.buttonCerrar.clicked.connect(self.cerrarWidget)

        self.opctemp=OpenOPC.client()
        self.opctemp.connect('Kepware.KEPServerEX.V6')
```

```
        threadQtM14 = threading.Thread(target = threadM14, args=(self,))
        threadQtM14.start()

    def forzarPresencia0(self):
        self.opctemp.write(('PLC.PLC.TAGS.PM14', False))
    def forzarPresencia1(self):
        self.opctemp.write(('PLC.PLC.TAGS.PM14', True))
    def forzarTransitoM130(self):
        self.opctemp.write(('PLC.PLC.TAGS.TR_14_13', False))
    def forzarTransitoM131(self):
        self.opctemp.write(('PLC.PLC.TAGS.TR_14_13', True))
    def cerrarWidget(self):
        while not self.threadAcabado:
            self.finalizarThread = True
        self.opctemp.close()
```

7.4.8. CÓDIGO PROGRAMACIÓN PHYTON INTERFAZ GRÁFICA VENTANA DIAGNÓSTICO CON QT

```
#Clase Qt para ejecución grafica de la ventana de diganostico
class muestraDiagnostico(QtGui.QMainWindow):

    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.ui = Ui_diagnostico()
        self.ui.setupUi(self)

        self.filasTablaExcel=0
        self.times=np.zeros((10, 33))
        self.ui.dateFin.setDate(QtCore.QDate.currentDate())
        self.actualizaTablaExcel()

        self.ui.tablaExcel.itemChanged.connect(self.guardarComentario)
        self.ui.elementos.currentIndexChanged.connect(self.actualizaTablaExcel)
        self.ui.dateFin.dateChanged.connect(self.actualizaTablaExcel)
        self.ui.dateInicio.dateChanged.connect(self.actualizaTablaExcel)
        self.ui.analisisButton.clicked.connect(self.ventanaAnalisis)

    def guardarComentario(self, item):
        fila=item.row()
        if(item.column()==5):
            semaforo.acquire();
            doc = openpyxl.load_workbook('../REGISTRO.xlsx')
            hoja = doc.get_sheet_by_name('Hoja1')
            filas=hoja.max_row-1
            elemento=self.ui.tablaExcel.item(fila,3)
            estado=self.ui.tablaExcel.item(fila,2)
            hora=self.ui.tablaExcel.item(fila,1)
            dia=self.ui.tablaExcel.item(fila,0)
            comentario=item.text()
            for f in range(filas):
                if ((elemento.text()==str(hoja.cell(row=f+2,column=4).value))and(est
                        hoja.cell(row=f+2,column=6).value=comentario
            doc.save("../REGISTRO.xlsx")
            doc.close
            semaforo.release();

    def actualizaTablaExcel(self):
        self.ui.tablaExcel.clearContents()
        self.ui.tablaExcel.setRowCount(0)

        fechaTemp=self.ui.dateInicio.date()
        fechaInicial=fechaTemp.toString('MM/dd/yy')
        fechaTemp=self.ui.dateFin.date()
        fechaFinal=fechaTemp.toString('MM/dd/yy')
        elementoFiltro=self.ui.elementos.currentText()

        semaforo.acquire();
        doc = openpyxl.load_workbook('../REGISTRO.xlsx')
        hoja = doc.get_sheet_by_name('Hoja1')
        doc.close
        semaforo.release();

        filas=hoja.max_row-1
        fila_act=0;
        for f in range(filas):
            dia = QtGui.QTableWidgetItem(str(hoja.cell(row=f+2,column=1).value))
            dia.setFlags(QtCore.Qt.ItemIsSelectable |  QtCore.Qt.ItemIsEnabled)
            hora = QtGui.QTableWidgetItem(str(hoja.cell(row=f+2,column=2).value))
```

```
            hora.setFlags(QtCore.Qt.ItemIsSelectable |  QtCore.Qt.ItemIsEnabled)
            estado = QtGui.QTableWidgetItem(hoja.cell(row=f+2,column=3).value)
            estado.setFlags(QtCore.Qt.ItemIsSelectable |  QtCore.Qt.ItemIsEnabled)
            elemento = QtGui.QTableWidgetItem(hoja.cell(row=f+2,column=4).value)
            elemento.setFlags(QtCore.Qt.ItemIsSelectable |  QtCore.Qt.ItemIsEnabled)
            tipo = QtGui.QTableWidgetItem(hoja.cell(row=f+2,column=5).value)
            tipo.setFlags(QtCore.Qt.ItemIsSelectable |  QtCore.Qt.ItemIsEnabled)
            comentario = QtGui.QTableWidgetItem(hoja.cell(row=f+2,column=6).value)

            if ((elemento.text()==elementoFiltro or elementoFiltro=='TODOS') and (fe
                self.ui.tablaExcel.insertRow(fila_act)
                self.ui.tablaExcel.setItem(fila_act, 0, dia)
                self.ui.tablaExcel.setItem(fila_act, 1, hora)
                self.ui.tablaExcel.setItem(fila_act, 2, estado)
                self.ui.tablaExcel.setItem(fila_act, 3, elemento)
                self.ui.tablaExcel.setItem(fila_act, 4, tipo)
                self.ui.tablaExcel.setItem(fila_act, 5, comentario)
                fila_act=fila_act+1
        self.filasTablaExcel=fila_act
        self.actualizaTablaDiagnostico()

    def actualizaTablaDiagnostico(self):
        self.ui.tablaDiagnostico.clearContents()
        elementos = ['M15','M14','M13','M12','TF10','TF20','TF21','TF22','TF23','TF2
        numDef=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
        timeDefIni=['00:00:00','00:00:00','00:00:00','00:00:00','00:00:00','00:00:0
        timeDefTotal=['00:00:00','00:00:00','00:00:00','00:00:00','00:00:00','00:00:
        timeMax=['00:00:00','00:00:00','00:00:00','00:00:00','00:00:00','00:00:00','
        timeMin=['00:00:00','00:00:00','00:00:00','00:00:00','00:00:00','00:00:00','

        for j in range(len(elementos)):
            contDef=0
            for i in range(self.filasTablaExcel):
                item=self.ui.tablaExcel.item(i,3)
                estado=self.ui.tablaExcel.item(i,2)
                hora=self.ui.tablaExcel.item(i,1)
                dia=self.ui.tablaExcel.item(i,0)

                if (item.text() == elementos[j] and estado.text() == 'Aparece'):
                    numDef[j]=numDef[j]+1
                    timeDefIni[j]=hora.text()

                if (item.text() == elementos[j] and estado.text() == 'Desaparece'):
                    resta=self.restar_hora(timeDefIni[j], hora.text())
                    timeDefTotal[j]=self.sumar_hora(timeDefTotal[j], self.restar_hor
                    if(self.comparar_hora(timeMax[j], resta)==True):
                        timeMax[j]=resta
                    if((timeMin[j]=='00:00:00') or self.comparar_hora(resta, timeMir
                        timeMin[j]=resta
                    if(contDef<10):
                        self.times[contDef,j] = sum(int(x) * 60 ** i for i,x in enum
                        contDef=contDef+1

            self.ui.tablaDiagnostico.setItem(j, 0, QtGui.QTableWidgetItem(str(numDef
            self.ui.tablaDiagnostico.setItem(j, 1, QtGui.QTableWidgetItem(str(timeDe
            self.ui.tablaDiagnostico.setItem(j, 2, QtGui.QTableWidgetItem(str(timeMa
            self.ui.tablaDiagnostico.setItem(j, 3, QtGui.QTableWidgetItem(str(timeMi
            self.ui.tablaDiagnostico.setEditTriggers(QtGui.QAbstractItemView.NoEditT
        self.analisis_boxplot(elementos)
        self.analisis_clustering()
```

```
def ventanaAnalisis(self):
    widgetAnalisis(self).show()

def restar_hora(self,hora1,hora2):
    formato = '%H:%M:%S'
    h1 = datetime.strptime(hora1, formato)
    h2 = datetime.strptime(hora2, formato)
    resultado = h2 - h1
    return str(resultado)

def comparar_hora(self,hora1,hora2):
    formato = '%H:%M:%S'
    h1 = datetime.strptime(hora1, formato)
    h2 = datetime.strptime(hora2, formato)
    return h1 < h2

def sumar_hora(self, hora1,hora2):
    formato = "%H:%M:%S"
    lista = hora2.split(":")
    hora=int(lista[0])
    minuto=int(lista[1])
    segundo=int(lista[2])
    h1 = datetime.strptime(hora1, formato)
    dh = timedelta(hours=hora)
    dm = timedelta(minutes=minuto)
    ds = timedelta(seconds=segundo)
    resultado1 =h1 + ds
    resultado2 = resultado1 + dm
    resultado = resultado2 + dh
    resultado=resultado.strftime(formato)
    return str(resultado)

def analisis_boxplot(self, elementos):
    df = DataFrame(self.times, columns=[elementos])
    im = df.plot(kind='box', figsize=(18,6))
    im.set(xlabel="Elemento", ylabel="Duracion (segundos)")
    #Éplt.savefig("../boxplot.png")

def analisis_clustering(self):
    df_fin = DataFrame(self.times)
    Z = hac.linkage(df_fin, 'single', 'correlation')
    plt.figure(figsize=(9, 3))
    plt.xlabel('Numero de Defecto')
    plt.ylabel('Distancia CLustering Jerarquico')
    hac.dendrogram(Z, leaf_rotation=90.,leaf_font_size=8.,)
    #plt.savefig("../clustering.png")
    #plt.show()
```