UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Adaptación de la aplicación SVS PLC Controller para integrar el PLC del fabricante Mitsubishi Electric

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: de la Fe Robles, Raúl

Tutor/a: Sendra Compte, Sandra

Cotutor/a externo: García García, Marcial

CURSO ACADÉMICO: 2023/2024

Contenido

Índice de figuras

Índice de tablas

Resumen

El objetivo de este TFG es la adaptación de un proyecto de automoción de AUTIS para un nuevo cliente asiático. AUTIS dispone de un software desarrollado por ellos mismos llamado “SVS PLC CONTROLLER”, el cual tiene como objetivo, controlar el disparo de las cámaras del sistema SVS (Surface Verification System) y el sistema de iluminación dependiendo del modelo a inspeccionar. Este sistema de control está basado en un PLC. Normalmente, en las fábricas europeas, el fabricante más común de PLCs es Siemens y en las fábricas americanas es Rockwell. Ahora, con la implantación del sistema de inspección en Asia, surge la necesidad de integrar el PLC del fabricante Mitsubishi. Por lo tanto, se trata de desarrollar esta implementación.

A continuación, pondremos en el foco la estructura y el funcionamiento de la “Arquitectura Autis”, esencial en la gestión, sus procedimientos operativos y la forma en que esos se integran en cualquier proyecto de la empresa. Examinaremos detalladamente el software “SVS PLC CONTROLLER”, para comprender su lógica y los engranajes que lo componen, llamados módulos. Finalmente, expondremos la solución implementada al problema de la adaptación.

Abstract

The objective of this TFG is the adaptation of an AUTIS automotive project in Asia. AUTIS has software developed by itself called “SVS PLC CONTROLLER”, which aims to control the triggering of the SVS (Surface Verification System) system cameras and the lighting system depending on the model to be inspected. This control system is based on a PLC. Typically, in European factories, the most common manufacturer is Siemens and in American factories it is Rockwell. Now, with the implementation of the inspection system in Asia, the need arises to integrate the PLC of the manufacturer Mitsubishi. Therefore, it is about developing this implementation.

Next, we will focus on the structure and operation of the “Autis Architecture”, essential in management, its operating procedures and the way in which these are integrated into any company project. We will examine the “SVS PLC CONTROLLER” software in detail to understand its logic and the gears that make it up, called modules. Finally, we will present the solution implemented to the problem of adapting said project to an Asian client.

Resum

L'objectiu d'aquest TFG és l'adaptació d'un projecte d'automoció d'AUTIS a Àsia. AUTIS disposa d'un programari desenvolupat per ells mateixos anomenat "SVS PLC CONTROLLER", el qual té com a objectiu controlar el tret de les càmeres del sistema SVS (Surface Verification System) i el sistema d'il·luminació depenent del model a inspeccionar. Aquest sistema de control està basat en un PLC. Normalment, a les fàbriques europees, el fabricant més comú és Siemens i a les fàbriques americanes és Rockwell. Ara, amb la implantació del sistema d'inspecció a Àsia, sorgeix la necessitat d'integrar el PLC del fabricant Mitsubishi. Per tant, es tracta de desenvolupar aquesta implementació.

A continuació, posarem al focus l'estructura i el funcionament de la "Arquitectura Autis", essencial en la gestió, els seus procediments operatius i la manera com aquests s'integren de qualsevol projecte de l'empresa. Examinarem detalladament el programari "SVS PLC CONTROLLER", per comprendre'n la lògica i els engranatges que el componen, anomenats mòduls. Finalment exposarem la solució implementada al problema de l'adaptació del projecte a un client asiàtic.

1. Introducción

AUTIS Ingenieros es una empresa especializada en el diseño e implementación de sistemas de automatización industrial para diversos sectores, incluyendo electrodomésticos, automoción, servicios y procesamiento de datos. Con sede principal en Gandía, han desarrollado proyectos para varias empresas españolas y a nivel internacional, como BSH, Ford, Tesla, Toyota y Mitsubishi, entre otras.

1.1. SVS

Uno de sus proyectos y productos más demandados y exitosos es SVS (Surface Verification System). Este, es un proyecto de alta tecnología que funciona como solución industrial para la detección y clasificación de defectos en superficies con capa electrónica, de imprimación y capa transparente.

SVS procesa imágenes adquiridas de varias cámaras de alta resolución bajo una variedad de condiciones de iluminación para detectar y clasificar defectos mientras las unidades se están moviendo a través una estación de inspección. Aplicando los algoritmos adecuados, SVS es capaz de procesar las imágenes adquiridas y detectar una gran variedad de defectos incluyendo sellador, rayones, suciedad, gotas de agua, estallidos, forúnculos, gotas de pintura, cabello humano, etc. Por medio de técnicas avanzadas de inteligencia artificial, el sistema identifica y clasifica los defectos para facilitar su reparación eficiente.

SVS ayuda a las empresas del sector de la automoción que quieren mejorar sus procesos de calidad de pintura aumentando la precisión de la detección y clasificación de defectos mediante visión avanzada por ordenador y tecnología de inteligencia artificial.

Estas son las características más a destacar de SVS:

1. El uso de tecnología dinámica elimina la necesidad de que los automóviles se detengan en la estación de inspección, lo que permite una integración más sencilla en la línea de producción, ya que no hay necesidad de modificaciones en el transportador.
2. SVS puede inspeccionar superficies con revestimiento e-coat (electrónico), primer (imprimación) y clear coat (capa transparente). Por lo que el diseño de cada estación de inspección de SVS tiene en cuenta las características de las superficies sobre las que se va a realizar la inspección (fig.1).
3. SVS es capaz de inspeccionar hasta el 98% de la superficie de la carrocería y está equipado con cámaras de alta resolución capaces de detectar defectos tan pequeños como 0,2 mm (0,079 pulgadas).
4. La estación de inspección SVS es un sistema flexible, preparado para el futuro cambios y modernizaciones de nuevos modelos que permitan la instalación de equipamiento adicional.
5. SVS es una solución “ready to repair”, capaz de comunicarse con sistemas que reparan automáticamente los defectos previamente detectados por SVS durante la inspección.
6. Soporte y mantenimiento 24/7 por ingenieros especializados de AUTIS.

Figura 1: Tipos de superficies de chasis

Veamos una representación visual de cada uno de los elementos que componen el proyecto SVS:

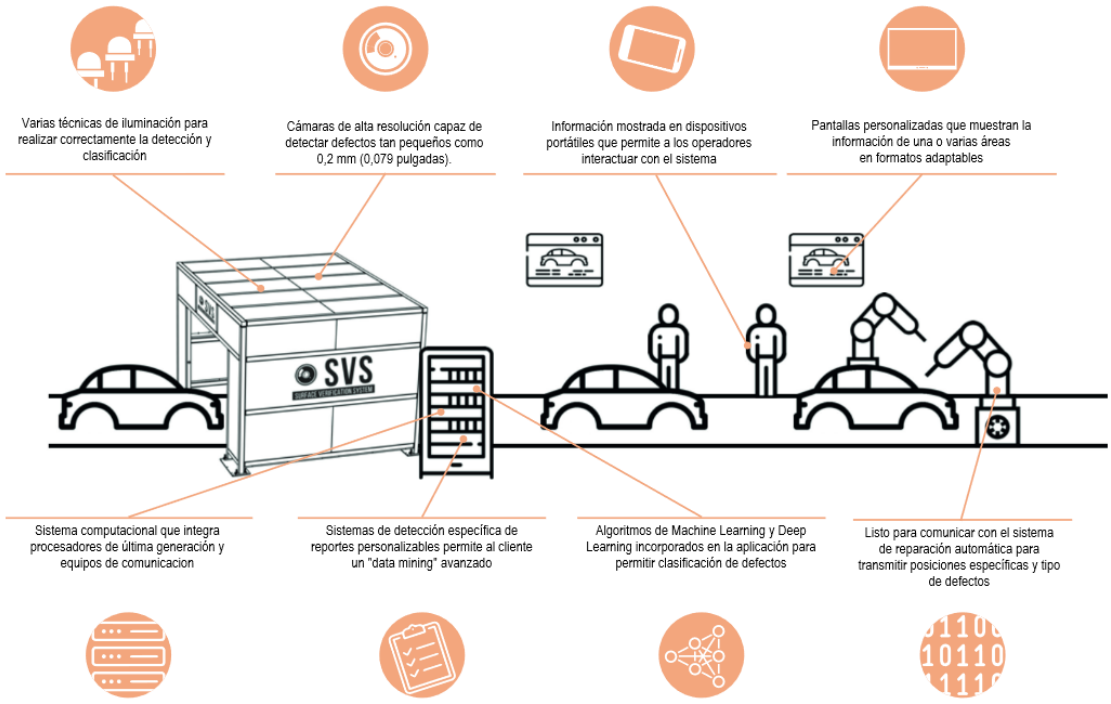


Figura 2: Representación de elementos clave de SVS

1.2. PLCs

Dentro de lo que es llamado “túnel” (fig.3), nos encontraremos con el arco de iluminación, compuesto por un número maleable de segmentos de leds regulables de alta potencia, y otro número maleable de cámaras de alto rendimiento. Esto permite que, al pasar el chasis por el túnel, se obtengan imágenes de alta calidad sobre la superficie de este. SVS hace uso de los PLC (Programmable Logic Controller) para controlar de forma automática y personalizada todo este proceso.

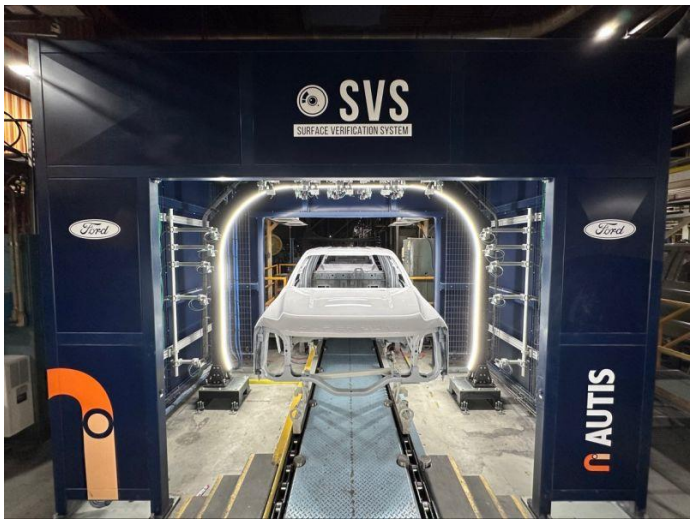


Figura 3: Túnel de detección SVS

Un PLC es el "cerebro" de distintos componentes, un salto significativo en la automatización de todos los procesos de un proyecto industrial. Por lo tanto, son muy utilizados en industrias y máquinas. A diferencia de las computadoras de propósito general, el PLC está diseñado de tal forma que permite múltiples señales tanto de entrada como de salida, inmunidad electromagnética, resistencia a señales externas, al ruido e impacto…

AUTIS adapta sus PLCs dependiendo de la procedencia de los clientes. Para clientes europeos, el fabricante de componentes más común es Siemens, y en las fábricas americanas es Rockwell. En el caso asiático, AUTIS ha tenido que encontrar otra alternativa, por lo que decidió optar por Mitsubishi (fig.4).

Figura 4: PLC de Mitsubishi

AUTIS hace uso de LabVIEW para la creación del hardware encargado del control de los PLCs utilizados.

1.3. LabVIEW

LabVIEW es una plataforma y entorno de desarrollo pensado para el diseño de sistemas hardware y software de pruebas (entornos de laboratorio), y control y diseño simulado o real. Una de sus principales peculiaridades es que es un lenguaje de programación visual gráfico.

Los programas desarrollados con LabVIEW son llamados Instrumentos Virtuales o VI ya que, en un primer momento, fue diseñado teniendo en cuenta el uso constante de instrumentación, aunque, hoy en día, sus usos son mucho más amplios. A continuación, se enumeran las principales características de LabVIEW:

1. LabVIEW utiliza un lenguaje de programación gráfico llamado G, que permite a los usuarios crear programas (llamados VI o Virtual Instruments**) mediante la conexión de nodos con cables en un diagrama de bloques, en lugar de escribir líneas de código.
2. Permite el diseño de interfaces de usuario (front panels) personalizables y fáciles de usar, donde se pueden colocar controles (botones, deslizadores, etc.) e indicadores (gráficos, displays, etc.) para interactuar con los VI.
3. Incluye una amplia gama de bibliotecas para análisis de señales, procesamiento de datos, generación de informes, control de instrumentos y comunicación con hardware externo.
4. Ofrece una excelente integración con hardware de National Instruments y otros fabricantes, permitiendo la adquisición de datos, generación de señales y control de dispositivos en tiempo real.
5. Facilita la programación de aplicaciones paralelas y multitarea mediante la utilización de múltiples lazos de ejecución en un mismo VI, aprovechando las capacidades de los procesadores multinúcleo.
6. Soporta múltiples protocolos de comunicación, como TCP/IP, UDP, GPIB, USB, Serial, CAN, y muchos otros, permitiendo la interconexión de diversos dispositivos y sistemas.
7. Ofrece herramientas avanzadas para el análisis y procesamiento de datos, incluyendo filtros, transformadas de Fourier, análisis de espectro, y más.
8. Permite el diseño de sistemas de control avanzados, como controladores PID, y la automatización de procesos industriales y de laboratorio.
9. LabVIEW se puede extender mediante la creación de sub-VI (subrutinas gráficas) y la integración de código externo en lenguajes como C, C++, Python y MATLAB.
10. Compatible con múltiples plataformas, incluyendo Windows, macOS, y Linux, y también ofrece soporte para sistemas embebidos y tiempo real a través de LabVIEW Real-Time y LabVIEW FPGA.
11. Incluye herramientas para la gestión de proyectos y la colaboración en equipo, facilitando el control de versiones y el desarrollo en conjunto.
12. Permite la simulación y modelado de sistemas antes de su implementación real, lo cual es útil para la prueba y validación de diseños.

LabVIEW es ampliamente utilizado en investigación, desarrollo, y aplicaciones industriales debido a su flexibilidad, facilidad de uso y capacidad para manejar tareas complejas de medición y control.

1.4. Arquitectura Autis

AUTIS ha desarrollado su propia estructura de proyectos y método de programación para poder simplificar y optimizar el código lo máximo posible. La Arquitectura Autis hace uso de módulos, máquinas de estados, eventos y colas para aumentar significativamente la efectividad del programa.

A continuación, voy a explicar brevemente cada uno de los elementos que he mencionado.

1.4.1. Módulos

Un módulo es, en esencia, un VI. Tiene su propio Block Diagram y Front Panel. Se le llama módulo ya que, este, engloba y cumple una serie de funciones particulares dentro de una aplicación. Un ejemplo de módulo sería el módulo "Adquisición de datos". Esto módulo sería un VI que se encargaría de la adquisición de los datos. Para ello, utilizaría todos los SubVIs desarrollados para esa funcionalidad.

Otro ejemplo, y el que mejor tiene que ver con el uso de los módulos, sería el llamado "Module Launcher" (fig.5). Este módulo es el primer módulo en ejecutarse, y se encarga de ejecutar los demás módulos en un orden en específico.

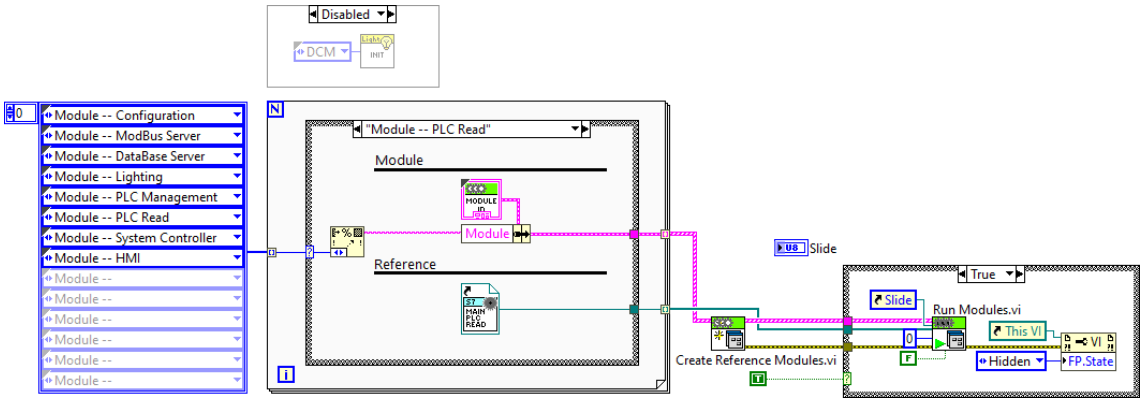


Figura 5: Módulo de lanzamientos de módulos

Como se puede ver, hay un array con referencias a los nombres de los otros módulos. Una vez procesados los nombres en el bucle foráneo, se hace uso del VI "Create Reference Modules" para crear, como su nombre indica, un array con las referencias a cada módulo. Luego, este array es introducido en el VI "Run Modules" para ejecutar cada módulo en su orden especificado.

Como podemos ver, los VIs son ejecutados en forma "Hidden", para que no aparezcan en pantalla, es decir, para que se ejecuten en segundo plano.

AUTIS suele ejecutar primero el "Móule Launcher", después el módulo encargado de la configuración inicial (carga la base de datos, el .ini, inicializa variables globales…) y luego, los demás. Dependiendo del proyecto, el orden varía.

1.4.2. Colas

Las colas (fig.6) es un concepto avanzado dentro del LabVIEW. Las colas deben de ser utilizadas en los casos en los que se quieran transmitir datos entre bucles paralelos. Es el caso de los módulos, los cuales son, en esencia, bucles (lo veremos en el siguiente apartado) que se están ejecutando al mismo tiempo.

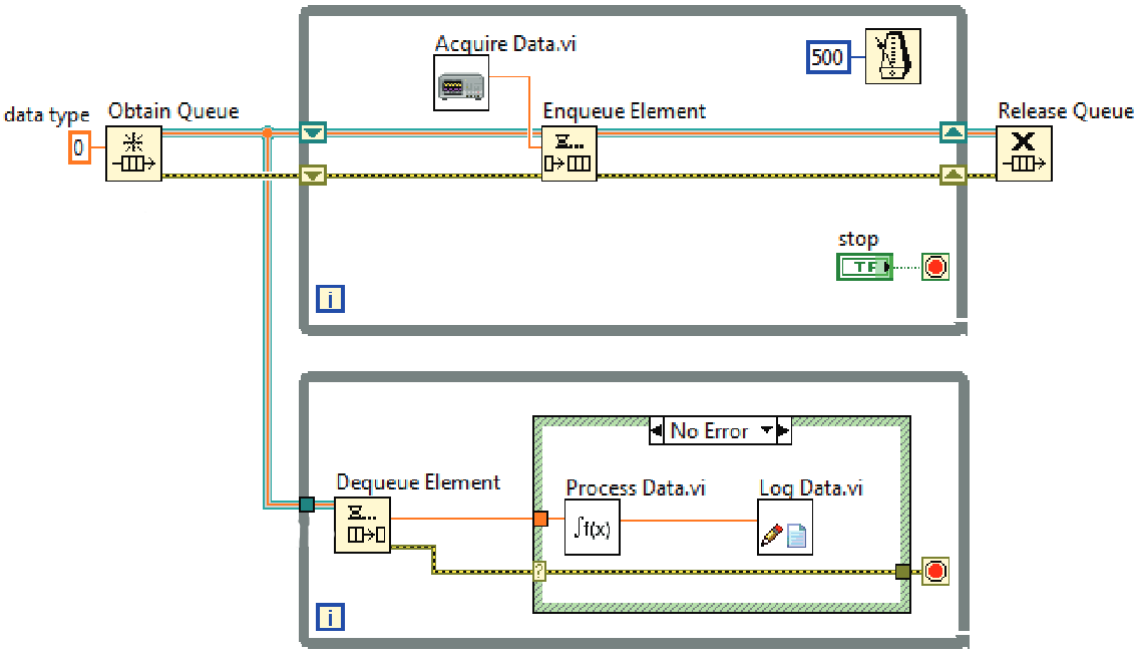


Figura 6: Ejemplo de uso típico de colas

El bucle superior suele llamarse bucle de adquisición, mientras que el inferior, suele llamarse bucle de consumición.

Al inicio del programa, se hace uso del VI "Obtain Queue", al que le introducimos por un terminal el tipo de dato que se va a pasar por la cola. En el caso del ejemplo, el tipo de dato es un double.

A continuación, se ejecutarán paralelamente los dos bucles, que no se detendrán hasta que se cumplan ciertas condiciones. En el bucle de arriba, cuando se pulse el botón "stop"; en el de abajo, cuando haya un error.

Esto haría que fuese imposible enviarse datos entre sí, y ahí es cuando la cola cumple su función. En el bucle de adquisición, cada vez que es ejecutado, se obtiene del VI "Acquire Data", un valor de tipo double que es introducido en el VI "Enqueue Element", el cual añade ese valor dentro de la propia cola. La próxima vez que los dos bucles se vuelvan a ejecutar, el bucle de consumición utilizará el VI "Dequeue Element" para obtener el dato obtenido anteriormente y poder procesarlo.

AUTIS ha desarrollado un tipo específico de colas, el cual se comunica no solo entre bucles, si no entre distintos VIs. Además, las colas de AUTIS están modificadas para que solo puedan enviar mensajes, esto es, strings.

La paleta de VIs desarrollada se llama Message Queue. Sus VIs son los siguientes:

Tabla 1: Comparación colas de LabVIEW con colas de AUTIS

Nombre	Objetivo	Diferencias	Icono
Create Message Queue	Creación de la cola	No necesitas especificar el tipo de cola, ya que siempre es la misma	
Enqueue Message	Obtiene la referencia de la cola y encola el mensaje al final de la cola	Solo admite tipo string	
Dequeue Message	Elimina el último elemento de la cola	Ninguna	

1.4.3. Máquinas de Estados

Una máquina de estados (fig.7) es un útil patrón de diseño de LabVIEW que suele tener un estado de arranque y cierre, aunque puede haber otros. Las máquinas de estados implementan cualquier algoritmo que pueda describir explícitamente un diagrama de estado o un diagrama de flujo. Las máquinas de estado se suelen utilizar en procesos secuenciales o procesos manejados por interfaces. Veamos un ejemplo de máquina de estados aplicada en la Estructura AUTIS.

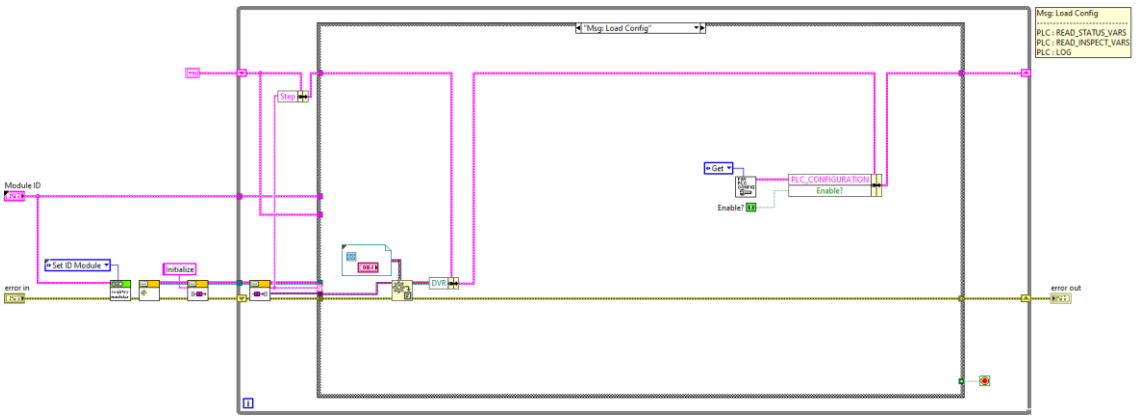


Figura 7: Ejemplo de máquina de estados

Como podemos observar, la máquina de estados está compuesta por un bucle while, que se ejecutará hasta que una condición sea cumplida (como pulsar un botón de “stop”), y un case, que se encarga del orden de ejecución de la secuencia. Cada módulo está compuesto por una máquina de estados.

Cada Módulo y máquina de estados de la Arquitectura Autis, tiene un caso llamado “Event Check” (fig.8). Este caso utiliza la estructura de eventos para encolar mensajes en la cola y así poder ejecutar casos en particular o casos de otros módulos.

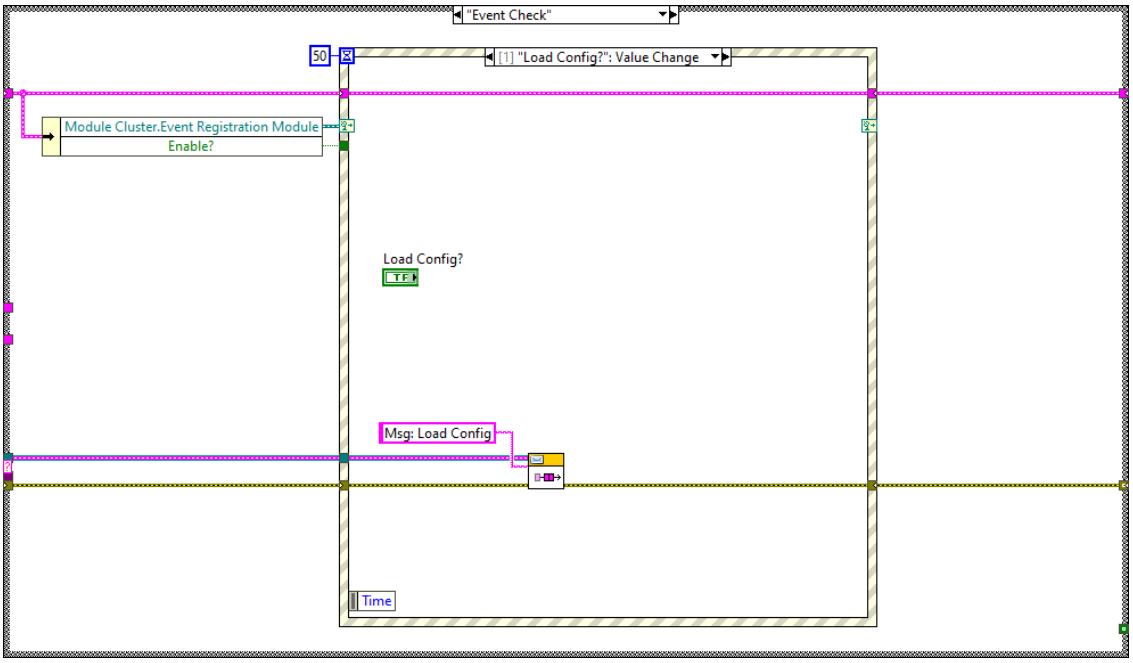


Figura 8: Ejemplo de uso de eventos en máquina de estados

Como ningún evento será llamado en un principio, se ejecutará el evento "Timeout" en el tiempo establecido. El "Timeout", en este caso, se encarga de encolar el siguiente mensaje "Msg: Load Config", el cual se encarga de ejecutar el caso que se ve en la imagen de arriba, es decir, el caso que carga la configuración procesada por el módulo de configuración.

1.4.4. FGVs

Los "FGVs" son SubVIs que se están ejecutando en segundo plano haciendo uso de un bucle while. Estos SubVI sirven como "getters" y "setters", es decir, son llamados para obtener un dato o cambiar el valor de un dato en específico que va a ser utilizado a lo largo de toda la aplicación. Este es un ejemplo de una llamada de inicialización de un FGV:

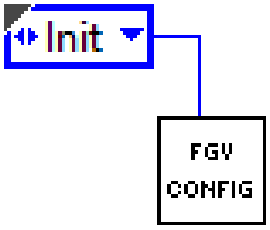


Figura 9: Inicialización de un FGV

Como podemos ver (fig.9), lo único que debemos hacer es especificar que queremos inicializarlo introduciéndole un "Init". Si introducimos un "Get", obtendríamos los valores que tuviera almacenados. Para que tengan algún valor almacenado, primero debemos introducírselo enviándole un "Set" y los valores. Veamos un ejemplo de "Set":

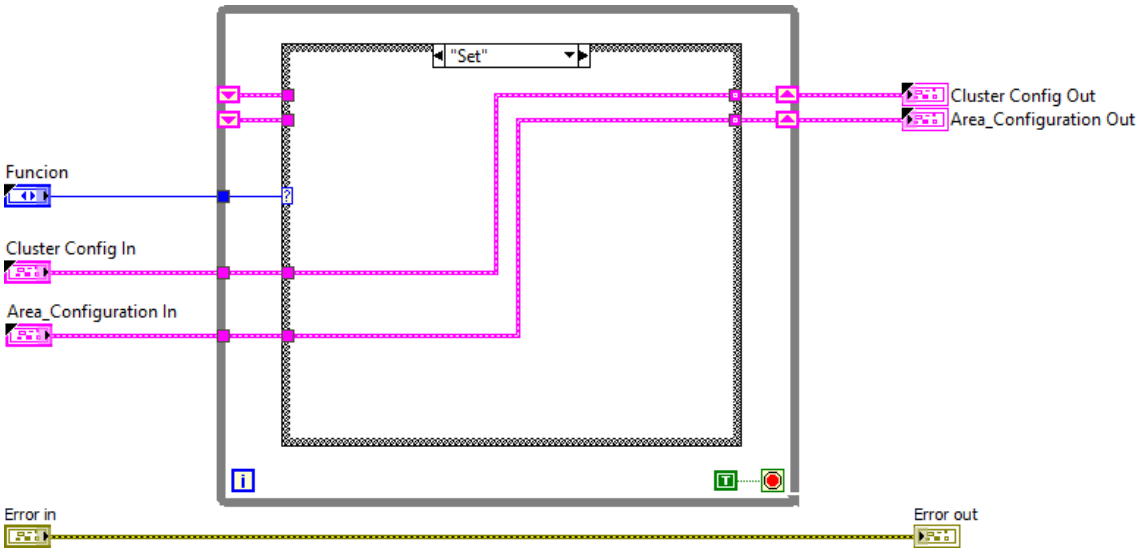


Figura 10: Ejemplo de uso de "Set" en un FGV

Como se puede ver (fig.10), en este FGV, la información que almacena son dos clústeres. Cuando introducimos nuevos clústeres y enviamos un "Set", estos reemplazan la antigua información que teían con la nueva. Por ello, el cluster con la información que queremos que sea la nueva, tiene que tener exactamente la misma estructura que el cluster que está declarado en el FGV, de no ser así, habrá un error en la ejecución del VI.

En caso de haber enviado un "Get", no se reemplazaría ninguna información, si no que, devolvería los clústeres con la información almacenada. Esto sirve para almacenar información de configuración general de la aplicación, como calibración, tiempos de establecimiento, información para conexiones con bases de datos de la compañía, información de dispositivos, listas de tareas, características de dispositivos…

En la siguiente figura (fig. 11), podemos apreciar un ejemplo de uso de "Get" en un FGV.

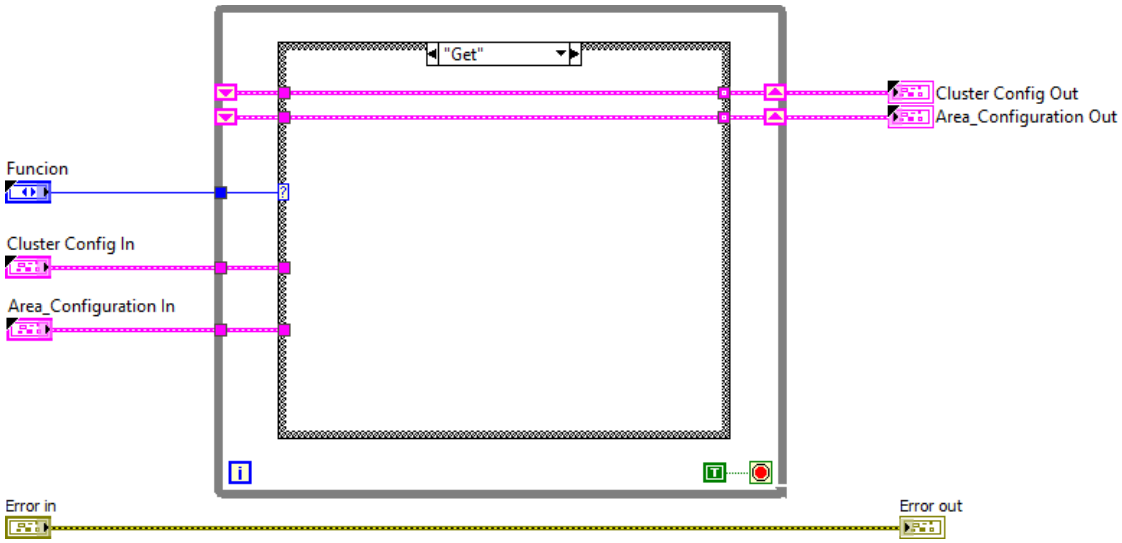


Figura 10: Ejemplo de uso de "Get" en un FGV

De esta forma (fig.11), podemos ser capaces de almacenar y utilizar información a lo largo de toda una aplicación mediante llamadas simples a SubVIs. Es un método muy recomendado por su facilidad de uso y su escalabilidad, permitiendo modificar el tipo de dato que se quiere utilizar cambiando únicamente el parámetro de entrada del FGV.

1.5. Motivación

Mi deseo es establecerme profesionalmente en el sector tecnológico. Desde pequeño me ha apasionado las nuevas tecnologías, así como la programación, y encontré un nicho en el desarrollo de automoción en LabVIEW, donde se puede trabajar con grandes profesionales día a día.

Con este TFG pretendo demostrar que soy capaz de trabajar con múltiples sistemas y desarrollar soluciones rápidas y dinámicas a problemas complejos, así como poder trabajar en marcas y clientes a nivel internacional.

He invertido mucho tiempo y motivación en el desarrollo de las habilidades que han hecho falta para poder llevar este proyecto, ya que llevo más de un año trabajando codo a codo con profesionales del sector en AUTIS Ingenieros. Este TFG es, por lo tanto, no solo un salto importante en mi expediente académico, si no también, en el laboral.

Mis objetivos van más allá de terminar la tesis. Me propuse demostrar que puedo seguir en esta empresa y, además, contribuir de forma constante en su desarrollo a la par que evoluciono como trabajador y como persona.

Además de demostrar mi valía, pretendo avanzar en la empresa, asumiendo más responsabilidades y gestionando proyectos a nivel internacional en distintos países del globo.

Este TFG es, por lo tanto, solo el inicio de este viaje en el que pienso construirme como un gran profesional. Con mi pasión, mi incansable sed de conocimiento y dedicación por el sector, sé que lo conseguiré.

1.6. Presupuesto

A continuación, se presenta el desglose del presupuesto para la realización del Trabajo Fin de Grado (TFG), considerando únicamente el tiempo invertido por el trabajador y los materiales específicos empleados:

- Se ha trabajado durante un periodo de 3 meses.
- El coste por hora para el cliente por la contratación de un empleado de la empresa es de 60 euros. Por lo tanto, con una media de 21 días laborables al mes, la remuneración mensual de una persona a cargo del cliente asciende a aproximadamente 10.080 euros.
- Con esta información, podemos calcular una estimación del coste total del sueldo por 2,5 meses para el cliente: 10.080€ x 2,5 = 25.200€.
- La licencia anual de LabVIEW tiene un costo de 3.000 euros.
- El coste de maquinaria y/o equipamiento asciende a 2.500 euros.

Considerando el tiempo invertido, el precio de la licencia de LabVIEW y la adquisición de equipos específicos, el presupuesto total del trabajo realizado en este TFG asciende a aproximadamente 30.700 euros.

Es importante señalar que, en caso de cualquier inconveniente atribuible a factores externos a AUTIS, se cobrará una tarifa superior al salario durante la creación de la aplicación (60€/h).

Es fundamental destacar que este presupuesto solo contempla los costes directamente relacionados con una sola persona encargada de completar el proyecto. No incluye ningún otro coste indirecto, gastos generales de la empresa, ni el trabajo realizado por otros departamentos o el equipo utilizado por estos.

1.2. Planificación temporal

El diagrama de Gantt es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. Antes de realizar cualquier tarea de este TFG, por lo tanto, me propuse a crear un diagrama Gantt para tener los objetivos bien claros, una especie de hoja de ruta, para saber en cada momento cuál era el siguiente objetivo que debería de abordar.

A continuación, voy a mostrar dicho diagrama de Gantt:

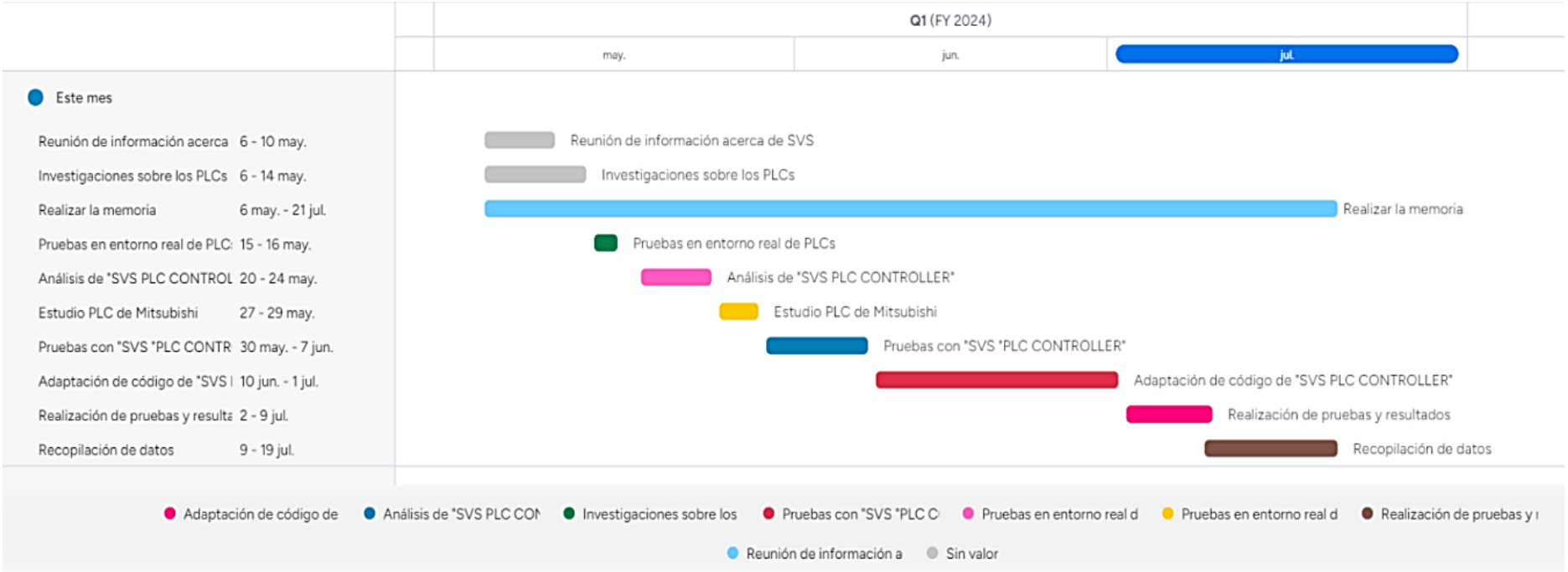


Figura 11: Diagrama de Gantt

2. Trabajos previos

En este apartado se pretenden abordar dos aspectos clave en el desarrollo de LabVIEW a partir de la comparación con varios códigos de diferentes TFG/TFM donde también se han utilizado LabVIEW. El primer aspecto para abordar será señalar los puntos en común y las diferencias, así como las técnicas utilizadas en cada proyecto. El segundo tendrá que ver con la utilización de máquinas de estado, donde se examinará la arquitectura, el diseño, los SubVIs y la eficiencia.

Gracias a este análisis, veremos cómo se utiliza LabVIEW en diferentes proyectos académicos, pudiendo detectar errores, patrones, y valorar el toque profesional que ofrece la empresa con la que estoy realizando este TFG.

2.1. Comparación del código de diferentes TFG/TFM

Comparando objetivos

La diferencia más clara en los proyectos la encontramos en los objetivos de este. El objetivo del TFG de Sergio Ferrer es “reproducir artificialmente la Recirculación de Gases (EGR) de un Motor Convencional para que el estudio de la combustión se asemeje a todas las condiciones que se pueden dar en un motor”. Para ello, ha hecho uso de LabVIEW para desarrollar el controlador de manera que se pueda interactuar con él.

El objetivo de mi TFG, en cambio, es la realización de una adaptación de un software que trabaja con un conjunto muy amplio y complejo de dispositivos. Con la ayuda de la Arquitectura Autis, paletas especializadas, control de procesos, representación en tiempo real de la información (módulo HMI), gestión de erroes…

Esto concluye con que el TFG de AUTIS aborda un marco mucho más complejo y amplio con gestión sofisticada y especializada que van más allá de un estudio de combustión. Mi TFG constituye una solución más compleja para un proyecto de mayor tamaño.

Comparación de códigos

En mi TFG seguí utilizando la arquitectura empleada en cada proyecto de AUTIS: la Arquitectura Autis (fig.13), explicada anteriormente. Lo cual es supone una estructura modular, bien estructurada, eficiente y escalable. La decisión de afrontar así el proyecto viene dada por la complejidad de mismo, haciendo falta el uso de código adaptable a cualquier cambio a futuro.

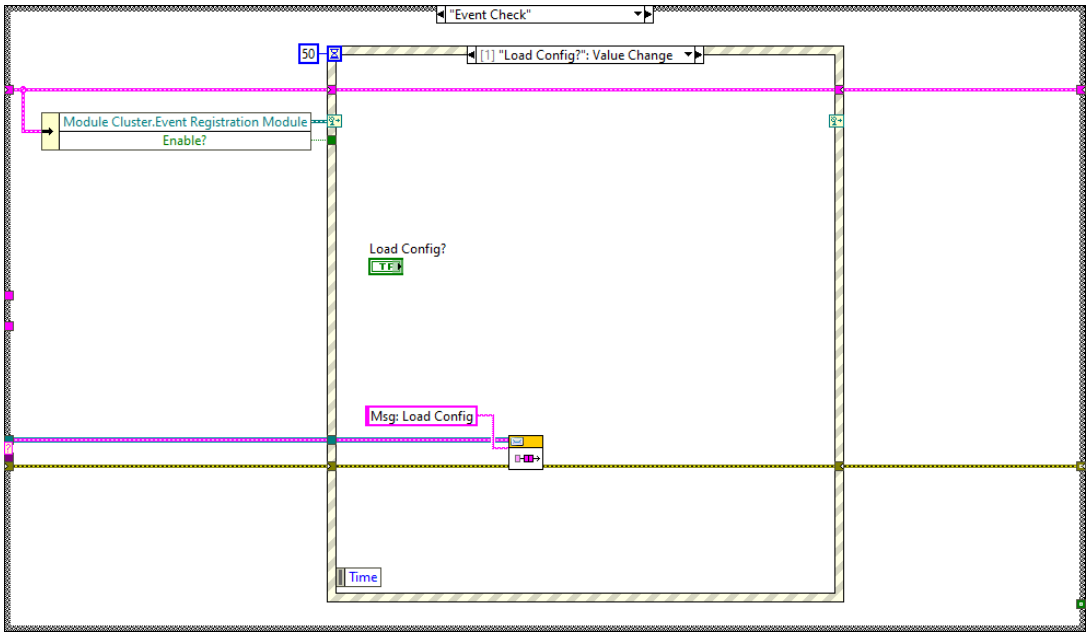


Figura 12: Ejemplo de comparación de mi código

Por otro lado (fig.14), el TFG de Sergio Ferrer es mucho más primitivo en el uso de herramientas, arquitectura, nodos, paletas y algoritmos complejos. Su código podría realizarse de tal forma que fuese más sencillo de leer y comprender, y peca de no cumplir con algunos requisitos básicos a la hora de programar con LabVIEW.

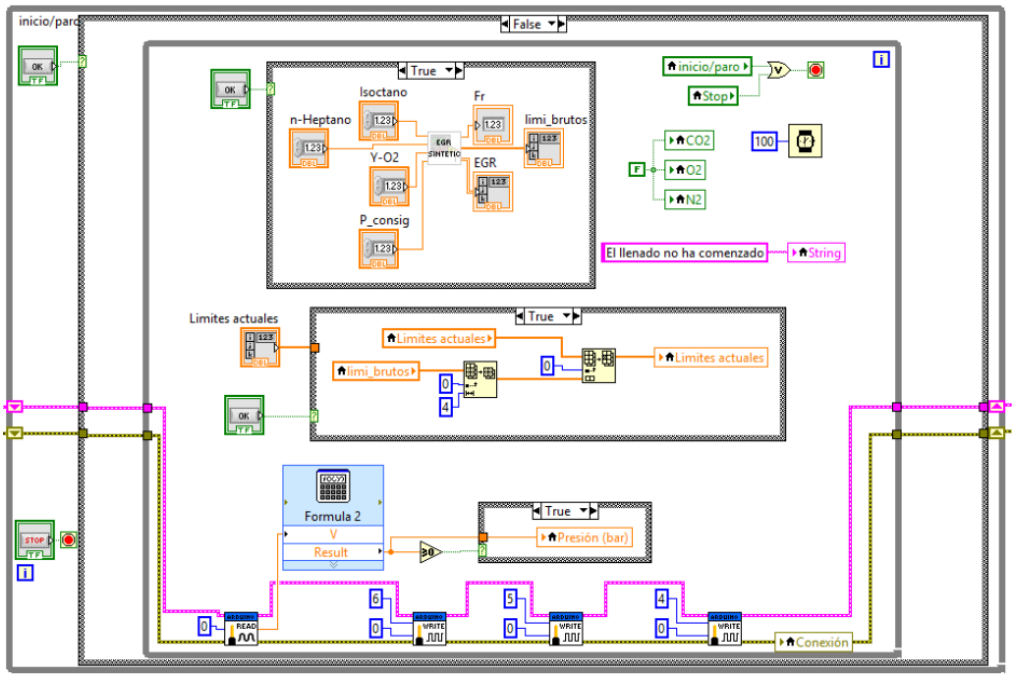


Figura 13: Ejemplo de comparación de otro código

Las decisiones arquitectónicas y de diseño a la hora de realizar cada código tienen un fuerte impacto a la hora de la comprensión de este. El proyecto de Sergio Ferrer emplea una estructura más simple y menos comprensiva.

2.2. Comparación de máquinas de estado

Gracias a apartados anteriores, se conoce cuál es la técnica que voy a utilizar a la hora de estructurar el proyecto. Dicho esto, se comparará esa estructura, la Estructura Autis, que es, en esencia, una máquina de estados más compleja, con la máquina de estados del TFG de Juantian Zhu Ji, “Implementación de un sistema de monitorización para un laboratorio de termotecnia” (fig.15).

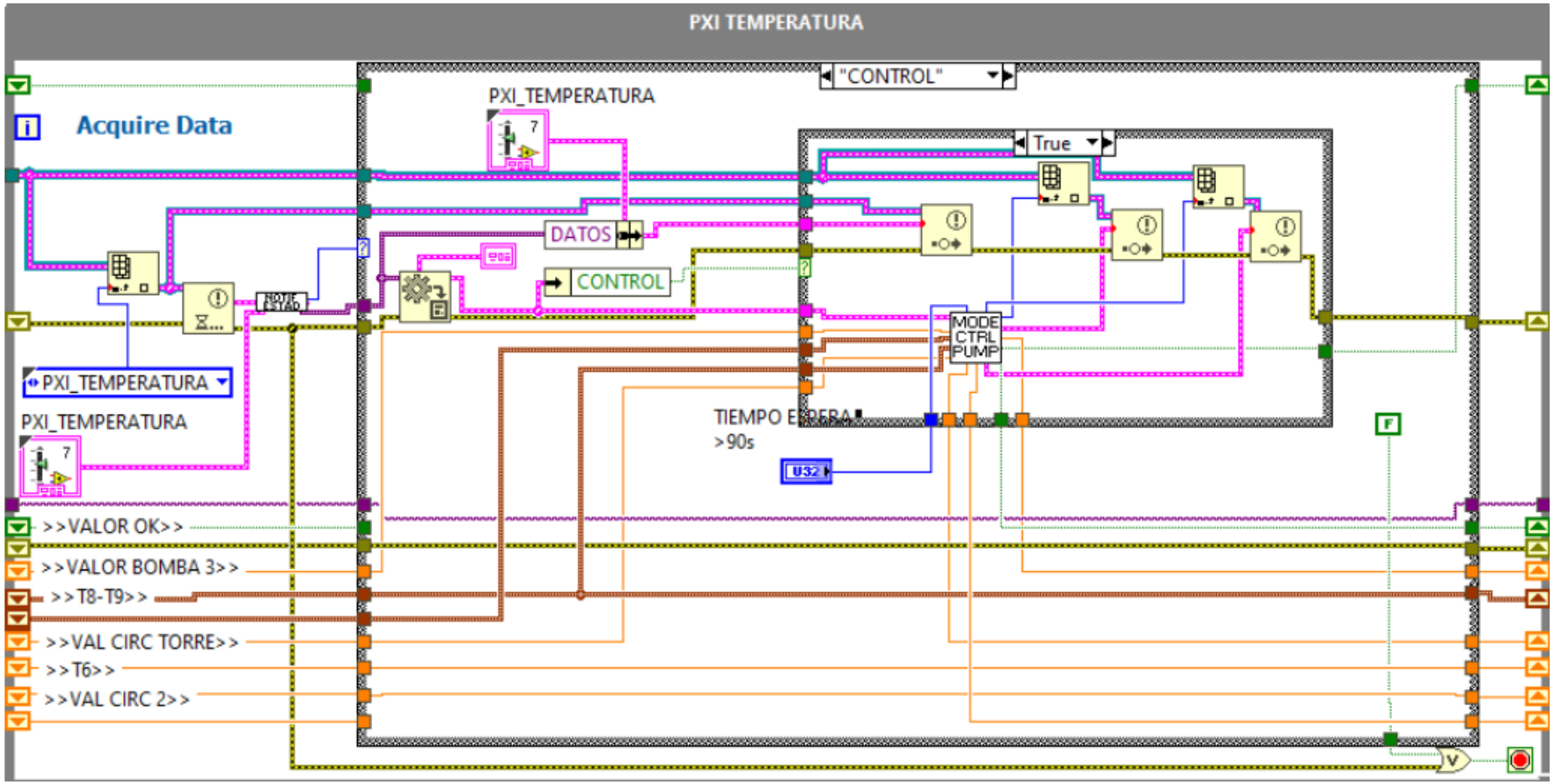


Figura 14: Ejemplo de comparación de otra máquina de estados

Como se puede apreciar, se echa de menos dos componentes clave: el uso de colas y un clúster principal. Las colas permiten cambiar entre casos sin tener que almacenar los casos anteriores en diferentes variables, lo que agiliza mucho más el desarrollo y facilita la lectura del flujo. Lo mismo ocurre con el uso de un clúster principal.

En la parte inferior del código, tiene muchas variables que se podrían almacenar en un solo clúster, haciendo el código mucho más pequeño. También se recomienda el uso del flujo estándar de LabVIEW, eso es, que los cables siempre deben de ir de izquierda a derecha, sin atravesar "suelos" o "techos" de estructuras. Para que esto no ocurra, el código debe de ser más simple, para que los cables puedan pasar en el orden acordado sin colisionar con otros.

Veamos un ejemplo de lo mencionado en mi código:

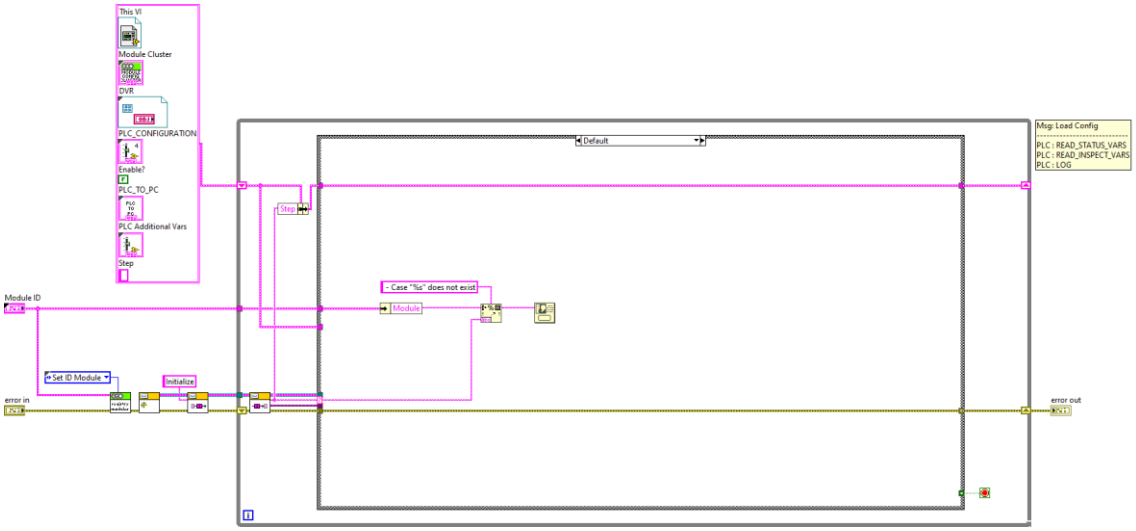


Figura 15: Ejemplo de comparación de mi maquina de estados

La estructura es mucho más simple, pese a que, en esencia, es el mismo concepto. Almaceno todas las variables en un solo clúster, y hago uso del "unbundle" cuando es necesario hacer uso de alguna o varias de ellas.

Otra diferencia reside en la localización del cable de error. Este, se sitúa en la parte inferior, ya que este no es un cable muy utilizado y suele ser usado más para mantener un orden de flujo.

3. Propuesta del sistema a desarrollar

En este capítulo se presentan los aspectos fundamentales a la hora de realizar la propuesta de este TFG. Esta propuesta consta de dos partes: un análisis exhaustivo de un desarrollo base ("SVS PLC CONTROLLER"), y la adaptación a dicho desarrollo de otra funcionalidad. Dicha funcionalidad es añadir la posibilidad de incorporar PLCs de la marcha Mitsubishi al proyecto mencionado de forma totalmente automática. Esto es, sin alterar el funcionamiento del proyecto, permitiendo la máxima comodidad a la hora de la personalización y modularidad en la implementación del producto al cliente.

Es decir, se debe de tener claro todos los aspectos del programa al que se le va a realizar la implementación antes de comenzar a realizarla. Por lo tanto, voy a exponer, punto a punto, y basándome en la explicación previa que he realizado del entorno de programación gráfico LabVIEW, el software "SVS PLC CONTROLLER" desarrollador por AUTIS Ingenieros.

3.1. Análisis del software "SVS PLC CONTROLLER"

Como comentamos en apartados anteriores, AUTIS hace uso de los PLCs para el control de la iluminación, calibración de dispositivos y obtención de datos. Para poder sacar el máximo partido a este elemento, se ha desarrollado un software de control del PLC llamado "SVS PLC CONTROLLER", desarrollado en LabVIEW 2018.

Este software está compuesto de 9 módulos, los cuales enumeraremos a continuación y en los que vamos a profundizar más adelante.

- Module Launcher: Se encarga de ejecutar todos los otros módulos en el orden especificado.
- Module Configuration: Carga la configuración inicial almacenada en un ".ini", y se encarga de enviarla a los demás módulos antes de que estos se ejecuten.
- Module ModBus Server: Se encarga de ejecutar el servidor ModBus para que otros módulos puedan utilizarlo. También se encarga de cerrarlo cuando sea necesario.
- Module DataBase Server: Se encarga de todo lo relacionado con la base de datos. Solamente inserta, de dos formas: "Multiquery" o "query" sencilla.
- Module Lighting Management: Se encarga de la iluminación. Inicializa la iluminación del arco, y revisa periódicamente su estado. Todo esto, ajustándose al tipo de iluminación que requiera el proveedor.
- Module PLC Management: Se encarga de cargar la configuración del PLC dependiendo del proveedor establecido. Como hemos especificado anteriormente, AUTIS trabaja con 2 proveedores: Siemens para proyectos europeos y Rockswell para americanos. Al final de este TFG habremos incorporado Mitsubishi para el asiático.
  También se encara de la escritura del PLC.
- Module PLC Read: Se encarga de la lectura de los datos para el PLC.
- Module System Controller: Este módulo es ampliamente utilizado por AUTIS en muchos de sus proyectos y no me es permitido compartir su contenido, así que no profundizaremos en él más adelante.
- Module HMI: Este módulo se encarga únicamente de la interfaz, esto es, la visualización de los datos obtenidos: parámetros, procesos, datos… Abordaré el funcionamiento de este módulo en el capítulo 5, "Resultados".

3.1.1. Module Configuration

Como ya hemos explicado, cada módulo es una máquina de estados, compuesta por un bucle "while", un "Case Structure" y un "Event Structure". Cada caso es un paso dentro de una secuencia establecida y controlada por el "Even Structure", que encola las "ordenes", estos son, los mensajes, en la cola inicializada antes del inicio del bucle.

Como en todos los módulos, la cola comienza con el mensaje "Initialize", que nos lleva al caso homónimo que se puede observar en la imagen adjunta (fig.17). En este caso especificamos que se ha inicializado correctamente el módulo haciendo uso del VI "Init Módule", y pones la variable "Loading Modules" del clúster de configuración a "True" para indicar que podemos continuar con la configuración.

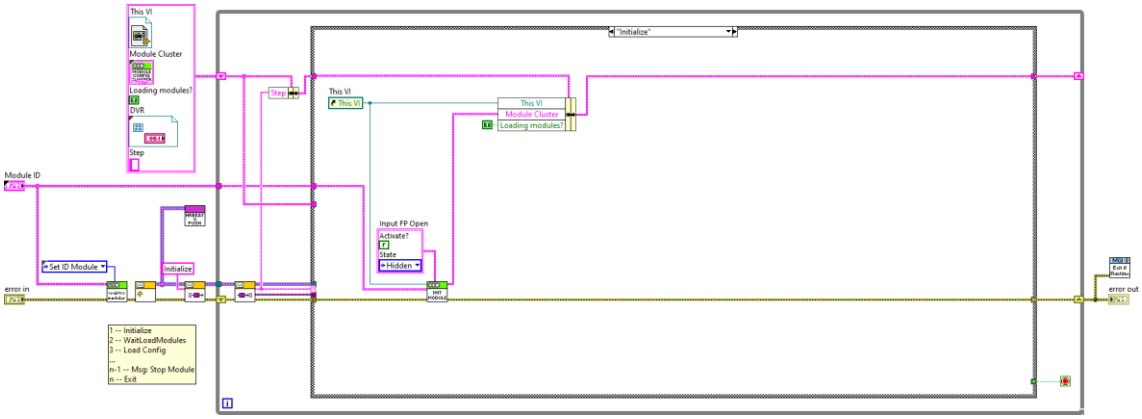


Figura 16: Module Configuration

Como no hemos especificado el siguiente caso, y como hemos puesto a "True" la variable mencionada, a los 250 ms, se ejecutará el "TimeOut" del "Event Structure" del caso "Event Check" (fig.18). Esos 250 ms son el tiempo estándar que asegura que los otros módulos se hayan inicializado con seguridad.

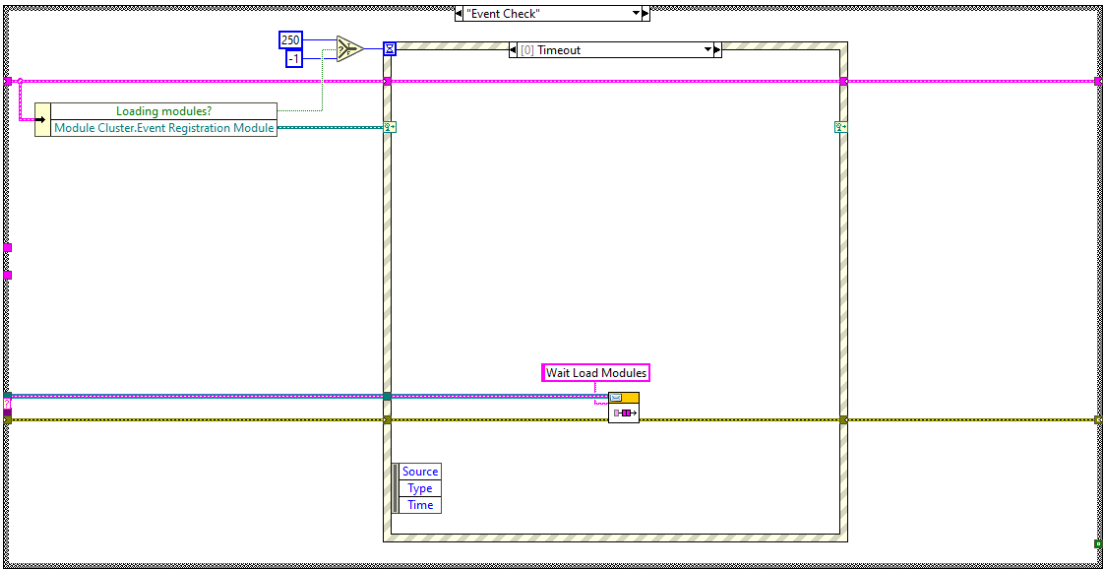


Figura 17: Explicación Event Check en Module configuration

Como se puede ver en la siguiente imagen (fig.19), se encola el mensaje "Wait Load Modules", que nos llevará al siguiente caso con el mismo nombre.

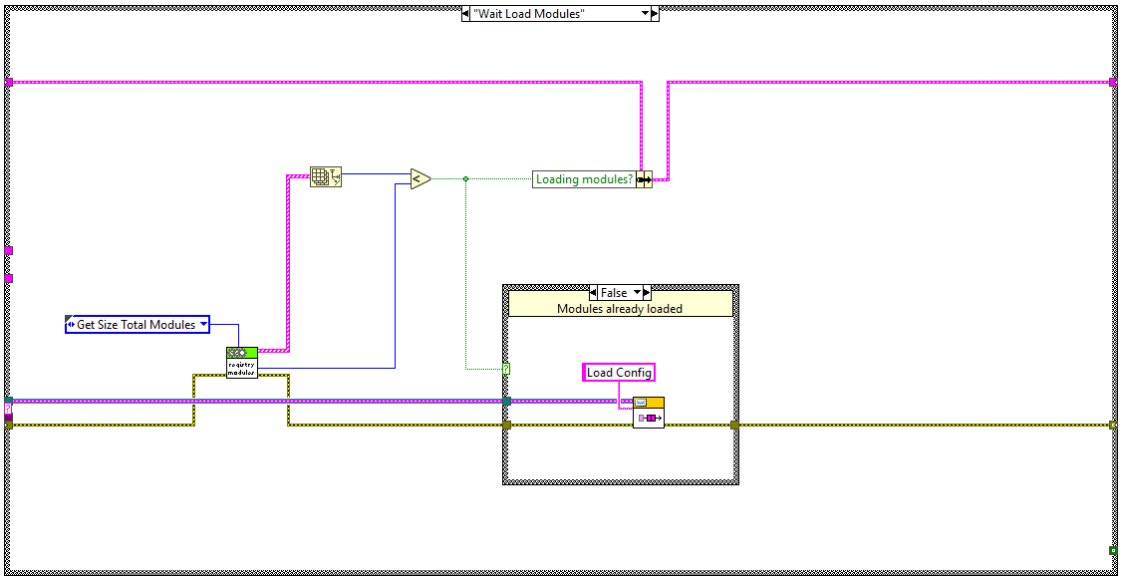


Figura 18: Caso "Wait Load Modules" en Module Configuration

En este caso, se hace uso del módulo "FGV Registry Modules" para obtener el número de módulos que han sido registrados (registrados como se ha registrado este módulo en el caso "Initialize"). Si el número de módulos registrados coincide con el que debería de ser, entonces se encola el mensaje "Load Config" y se ejecutaría su caso homónimo (fig.20), si no, no se hace nada, por lo que se volvería a esperar 250 ms hasta volver a comprobarse.

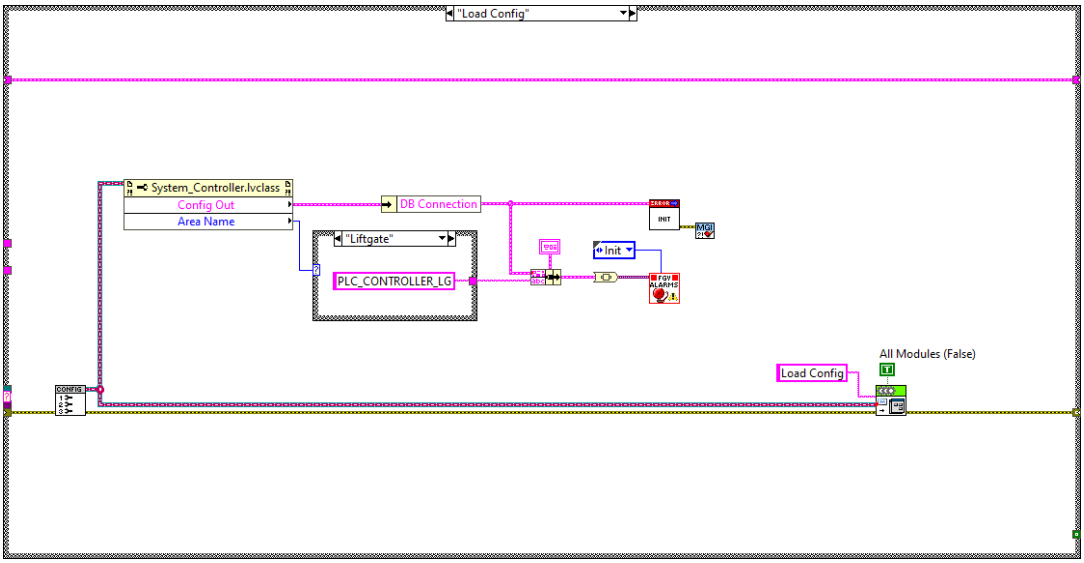


Figura 19: Caso "Load Config" en Module Configuration

En la imagen anterior (fig.20) podemos ver el caso "Load Config". Este es el objetivo principal del módulo. Dentro de este caso, debemos de poner el foco en el SubVI llamado "LoadConfig", que se encargará de cargar la configuración establecida para el tipo de proyecto especificado, tendiendo en cuenta la procedencia del cliente, el hardware utilizado, las dimensiones del producto…

No me está permitido mostrar capturas del código de dicho VI, ya que es creación del equipo de ingenieros de AUTIS, pero voy a resumir su funcionamiento.

El SubVI es un bucle foráneo que recorre un array con una estructura de casos. Dependiendo del elemento del array, se ejecutará un caso u otro. Esto es, en esencia, una máquina de estados.

Cada elemento del array es un dato o conjunto de datos que se quiere cargar: “Read Config”, “Get Area Setup”, “Get Modbus Server Port” … para algunos casos, lee de un .init; para otros, utiliza la paleta de LabVIEW que permite conectarse a un servidor de MariaDB, donde AUTIS tiene datos confidenciales sobre cada uno de los proyectos que ha desarrollado. De esta base de datos, se obtienen datos de calibración. Veamos un ejemplo de la utilización de esta base de datos en la siguiente imagen:

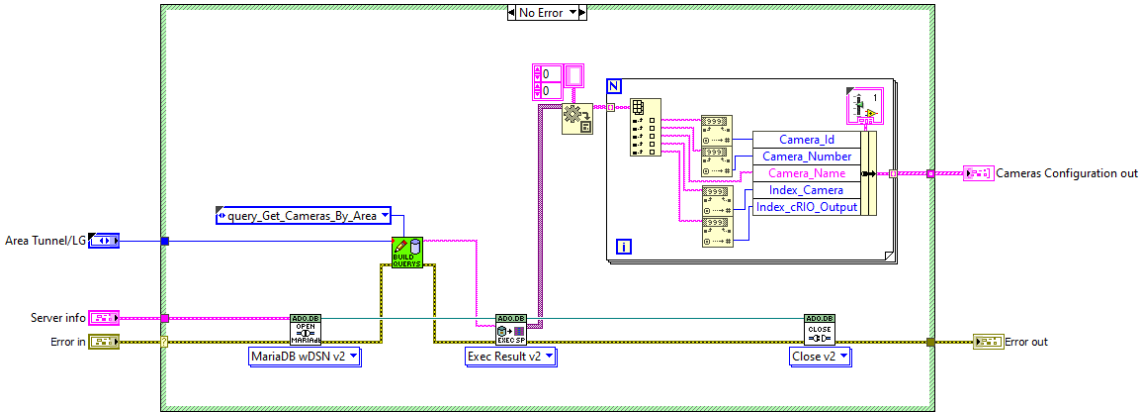


Figura 20: Petición de datos de configuración mediante MariaDB

En la figura anterior (fig.21), utilizamos una paleta de VIs proporcionada por el propio LabVIEW para crear la conexión. El VI que crea la conexión exige un clúster con datos del servidor, esto es, el clúster llamado “Server info” que se puede ver la imagen anterior. Una vez creada la conexión, hacemos una llamada a un SubVI que, dependiendo del tipo de túnel que sea (el área del túnel puede variar por muchas cosas tales como: tamaño de chasis, cantidad de túneles, cantidad de leds, cantidad de cámaras…), crea una query u otra. Esta query, junto a una referencia a la conexión establecida, son introducidos en el VI de ejecución de MariaDB, que obtiene los datos pedidos en la query. Finalmente, estos datos son procesados y situados en un array de clústeres (fig.22), en la que cada cámara viene representada por un clúster, y se cierra la conexión con el servidor mediante otro VI haciendo uso de la referencia de la conexión.

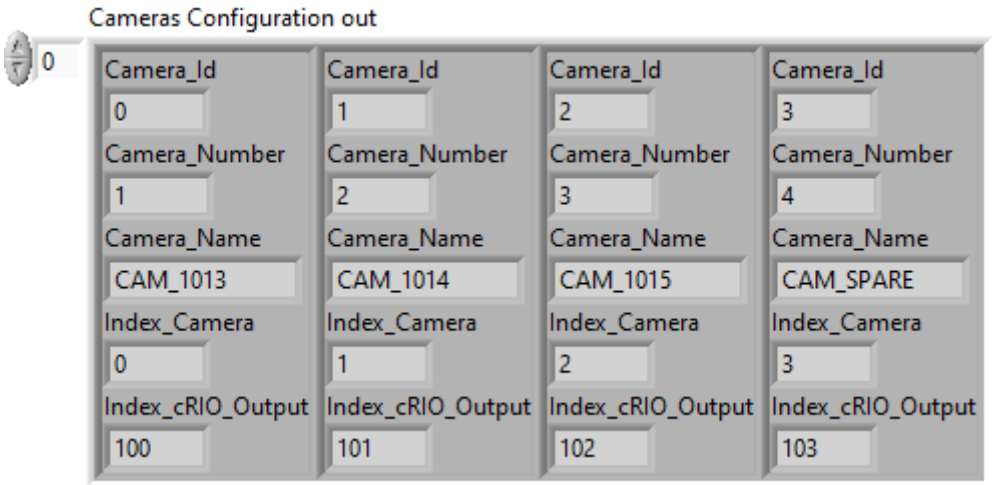


Figura 21: Array de clústers con información de las cámaras SVS

El SubVI “LoadConfig” también se encarga de inicializar los llamados “FGV”. En este proyecto se hace uso del SubVI “LoadConfig_InitFGV” para inicializar todos los FGVs que se van a utilizar. En la siguiente imagen (fig.23), podemos ver un extracto de dicho SubVI donde se aprecian múltiples inicializaciones de FGVs.

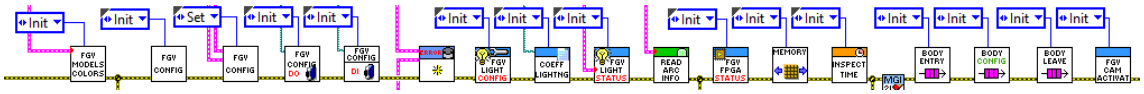


Figura 22: Extracto del SubVI de inicialización de FGVs

3.1.2. Module ModBus Server

Este módulo, igual que todos los que le siguen, tiene la misma estructura especificada en el apartado anterior, con la diferencia de que, en vez de esperar 250 ms para configurarse, esta espera a recibir del módulo de configuración el mensaje “Msg: Load Config”. Como ya he explicado este proceso, no lo repetiré en los siguientes módulos, y me centraré más en sus funciones específicas.

Una vez inicializado el módulo, recibido el mensaje del módulo de configuración y cargada la información, el siguiente caso que ejecutará será "Run ModBus Server" (fig.24).

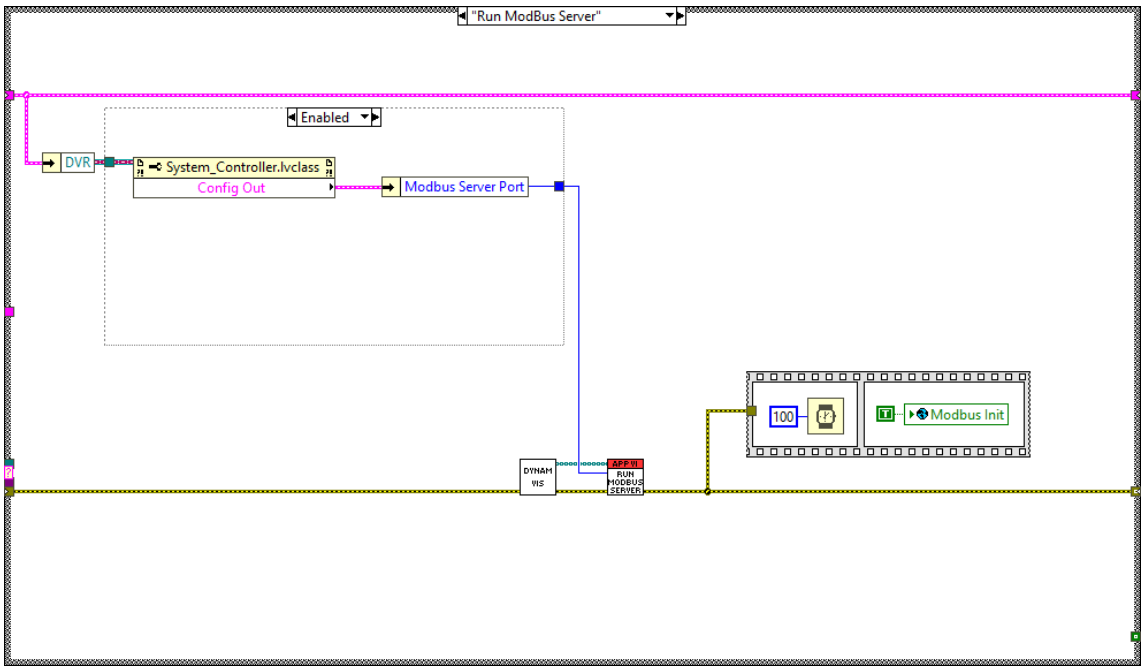


Figura 23: Caso “Run ModBus Server” de Module ModBus Server

DVR es una variable obtenida en el módulo de configuración. Esta variable sirve para especificar el puerto del servidor del Modbus. Haciendo uso de esta variable y el VI proporcionado por LabVIEW llamado “Run_Modbus_Server_Triggering”, inicializamos el servidor.

3.1.3. Module DataBase Server

En este módulo, nos vamos a centrar únicamente en el caso “Msg: DB_QUEUE” (fig.25), un caso que solo puede ser llamado desde otros módulos. Dicho caso sirve para hacer llamadas a la base de datos. Como podemos ver, de la cola obtenemos un clúster compuesto por un

"Message" de tipos string y otro clúster llamado "Data" donde vendrán los datos que se quieran introducir en la base de datos.

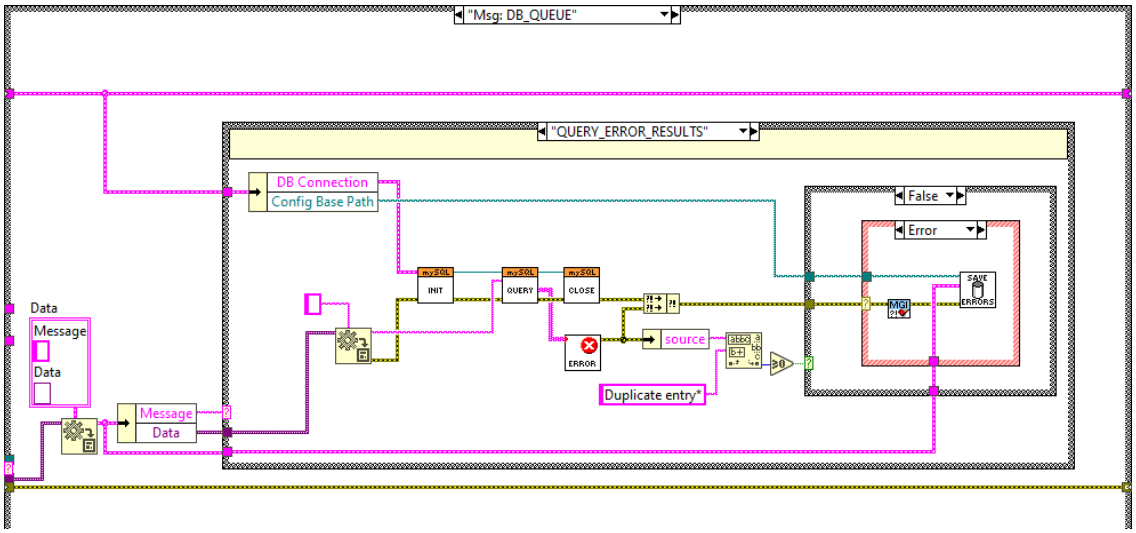


Figura 24: Caso "DB_QUEUE" de Module ModBus Server

El elemento "Message" del clúster, sirve para especificar el tipo de procesamiento de datos que se quiere realizar.

El caso utilizado para la inserción de datos es el llamado "QUERY_ERROR_RESULTS". Para ello, hace uso de una paleta de inserción a base de datos mediante SQL creada por AUTIS. No puedo mostrar el código, aun así, voy a explicar su funcionamiento:

Tabla 2: Paleta de AUTIS para mySQL en LabVIEW

Nombre	Función	Icono
lv_mysql_conn_init	Inicializa la conexión con la base de datos y obtiene su referencia	mySQL INIT
lv_mysql_conn_query	Obtiene una "query" y la ejecuta.	mySQL QUERY
lv_mysql_conn_close	Cierra la conexión	mySQL CLOSE

3.1.4. Module Lighting Management

Este módulo es el encargado de la configuración, inicialización y el registro de errores de la iluminación. AUTIS trabaja con unos dos proveedores de segmentos, que son leds de iluminación de alta potencia y calidad. Estas dos marcas son DCM y Contaval. Dependiendo de las especificaciones del proyecto, se selecciona un proveedor u otro.

AUTIS ha desarrollado dos paletas distintas para el control de la iluminación de los segmentos dependiendo del proveedor (fig.26).

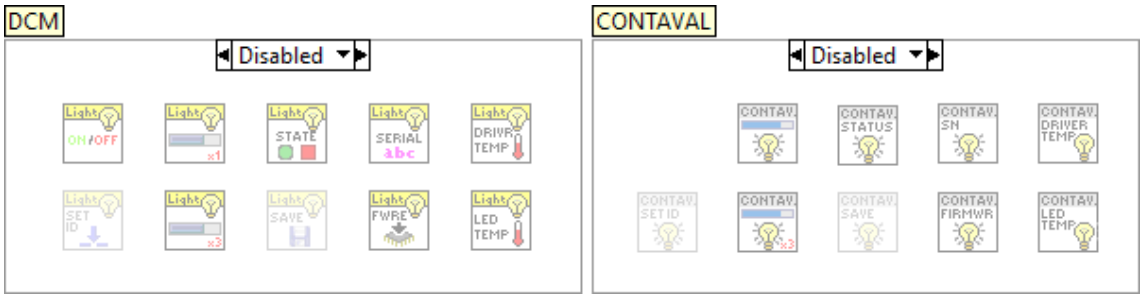


Figura 25: Paletas personalizadas para diferentes tipos de segmentos SVS

En el caso de inicialización de la iluminación, hacemos uso de este fragmento de código (fig.27):

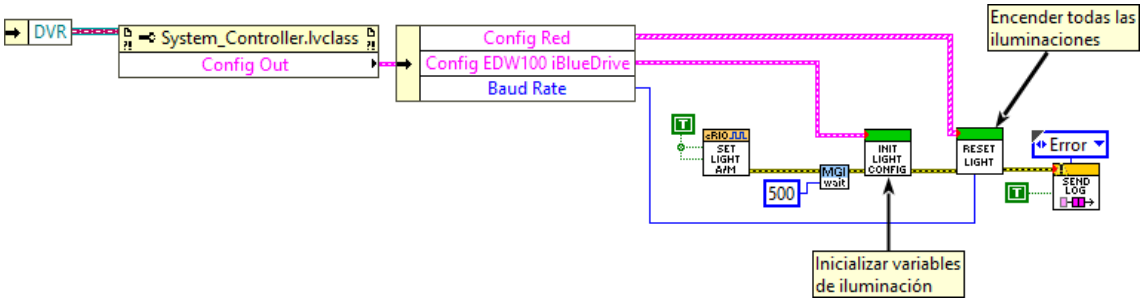


Figura 26: Inicialización de iluminación de segmentos SVS

A partir de la configuración obtenida por parte del módulo de configuración, podemos obtener la configuración del EDW100 (una pasarela Serial-Ethernet), con lo que podremos configurar la iluminación mediante el SubVI "initialization light config". Esta configuración deja las luces apagadas, por eso se reinician mediante el SubVI "Rest lights".

3.1.5. Module PLC Management

Tanto este módulo como el siguiente son los más importantes a la hora de crear la adaptación que queremos realizar, ya que tienen que ver con la configuración, inicialización, escritura, registro y lectura del PLC.

Este módulo en específico se encarga de abrir y cerrar sesión con el PLC, y la escritura de datos de este. Veamos los 3 casos que se encargan de ello:

"Open Session":

Para abrir sesión con el PLC, lo esencial es conocer qué tipo de PLC se va a utilizar, como ya hemos mencionado anteriormente, esto tiene que ver con el tipo de proveedor que vamos a usar. En este proyecto, hay incorporado dos: Rockwell y Siemens. La información sobre el tipo de PLC que se va a utilizar es adquirida por el módulo de configuración y almacenada en un "FGV". Este "FGV" es llamado en el caso "Load Config".

Como se puede apreciar en la siguiente ilustración (fig.28), utilizamos el clúster de configuración del PLC para obtener el dato del tipo (type) del PLC del que se va a hacer uso. En el caso de la imagen adjunta (fig.28), el tipo es "SIEMENS".

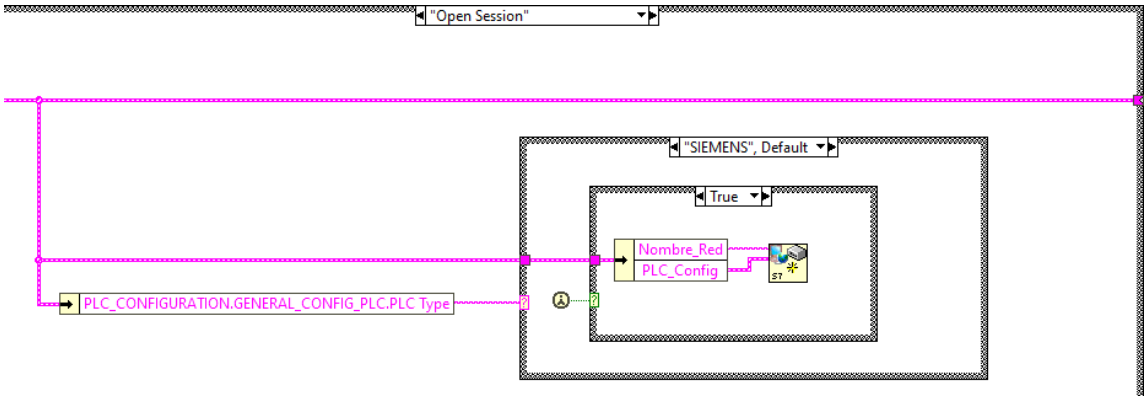


Figura 27: Ejemplo de inicio de conexión TCP con un PLC

LabVIEW nos permite establecer conexión mediante el nombre de red y un clúster con información detallada del PLC (fig.29). Se debe de hacer uso de un VI privado, es decir, que no podemos ver cómo es su código.

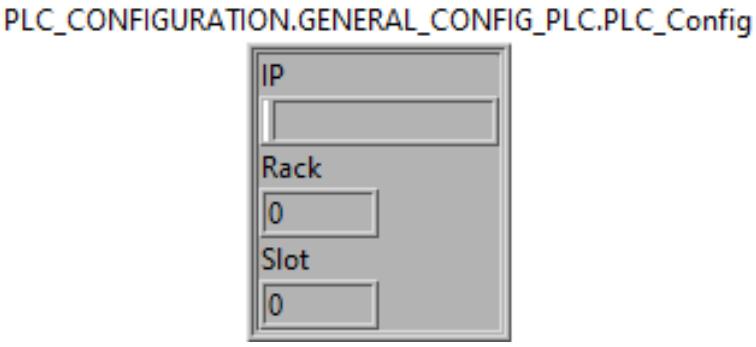


Figura 28: Clúster con datos de conexión a un PLC

"Close Session":

Utiliza los mismos elementos para obtener los datos del PLC y poder cerrar la conexión con este. En este caso, también utilizamos un VI privado (fig.30), pero solo es necesario introducir el nombre de red para cortar la conexión.

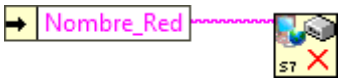


Figura 29: Cierre de conexión TCP con el PLC

"Write PLC":

Este caso sirve para mandar órdenes desde el PLC a las cámaras o segmentos (fig.31).

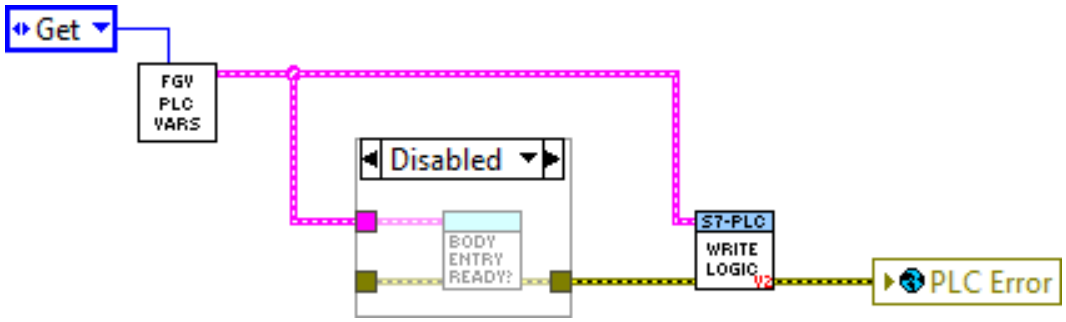


Figura 30: Case "Write PLC" del Module PLC Management

Obtiene un clúster con todas las variables establecidas del PLC según su tipo, y envía esa información al SubVI "PLC_Write_Logiv v2". El funcionamiento de este SubVI es muy importante, así que, a continuación, voy a explicarlo con profundidad.

PLC_Write_Logiv v2:

Este SubVI (fig.32) utiliza dos elementos claves del PLC: Los "BODY_ENTRY" y el "HEARTBEAT".

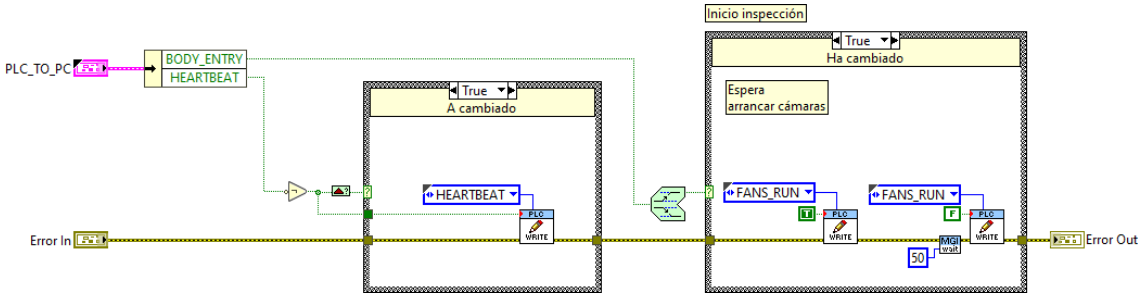


Figura 31: VI PLC_Write_Logiv v2

El "heartbeat" es una señal periódica generada mediante hardware o software que sirve para la sincronización con otros componentes del sistema. El "heartbeat" es representado de dos formas: Mediante una subida cuando el bit tiende a 1 y una bajada cuando a 0, o al revés. La más común es la primera interpretación.

En este SubVI, cuando se produce un cambio en el tipo de "heartbeat", se sabe que es porque se ha interrumpido la ejecución, por lo que se registra dicho suceso mediante el SubVI "PLC_Write Vars", que explicaremos más tarde.

El segundo elemento que utiliza, el "BODY_ENTRY", sirve para saber si se ha iniciado una inspección. La inspección es el proceso en el que el chasis del coche comienza a recorrer toda la pasarela y entra por el túnel para ser analizado. Cuando esto ocurre, el "BODY_ENTRY" pasa de ser un "False" a ser un "True". Para detectar este cambio, utilizamos el nodo "Boolean Trigger", que devuelve un "True" solo si el valor recibido ha sido anteriormente un "False". En ese caso, hacemos uso otra vez del SubVI "PLC_Write Vars" para mandar la orden de encender y, después de pasados 50 ms, el apagado de todos los ventiladores de los segmentos (los leds de alta potencia que componen el arco del túnel). Esto último permite que se enciendan automáticamente sin riesgo.

PLC_Write Vars:

Como este VI es muy grande, voy a explicarlo paso a paso, para que quede todo bien claro, ya que es de mucha importancia.

Este VI sirve como mediador. Cuando es llamado, revisa el tipo de PLC que tiene asociado el proyecto, el tipo de dato que se quiere mandar, para qué componente y si quiere enchufar o apagarlo, y llama a VIs privados para realizarlo.

La variable "VARIABLE" es el elemento en el que se envía sobre qué componente se quiere trabajar. En el caso anterior vimos los ventiladores de los segmentos, que se especificaban como "FANS_FUN". La variable "Body Info" es un clúster que guarda información sobre el PLC (fig.33).

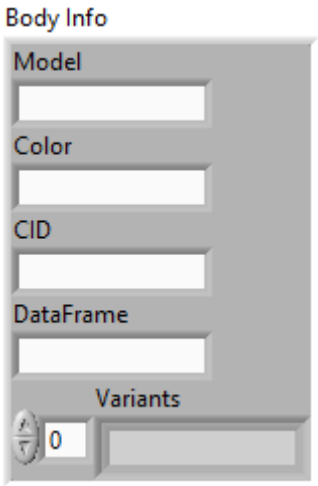


Figura 32: Clúster de información de un PLC

Dependiendo del tipo de PLC que sea, entra en un caso u en otro (fig.34). En dichos casos, obtendremos los datos de identificación y configuración necesarios para mandar los "funciones" que deben de realizar los PLCs y las configuraciones que deberán de realizar.

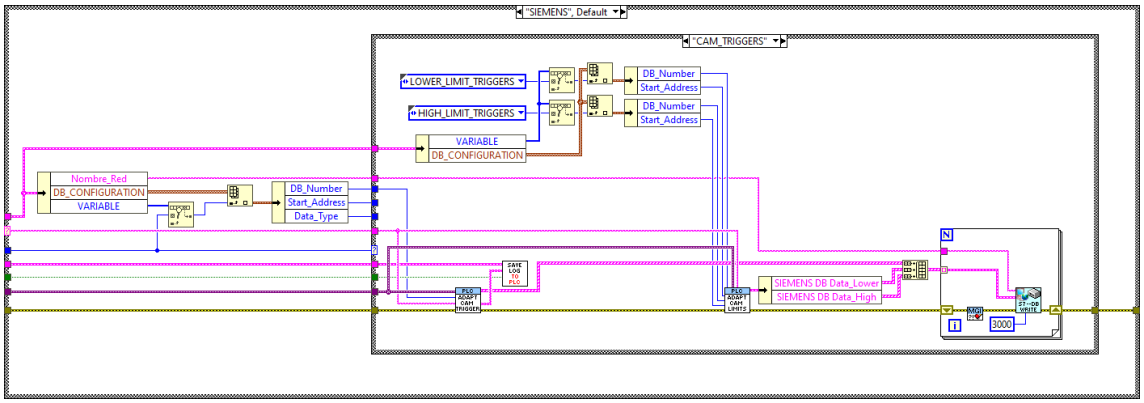


Figura 33: Caso de Seamens configuración triggers cámaras

Tanto en Siemens como en RocksWell, se utiliza "VARIABLE" para saber si estamos actuando sobre "CAM_TRIGGERS" o cualquier otro elemento.

La variable "Data" es utilizada solo en el caso "CAM_TRIGGERS". En este campo aparece la información relacionada con las cámaras y los triggers que han obtenido cada una. Más adelante, explicaré el significado de triggers de la cámara.

3.1.6. Module PLC Read

En este módulo obtendremos las variables del PLC, que son utilizadas en la aplicación. El caso que se centra en obtener esos datos es "PLC: READ_INSPECT_VARS" (fig.35), donde se sitúa este código.

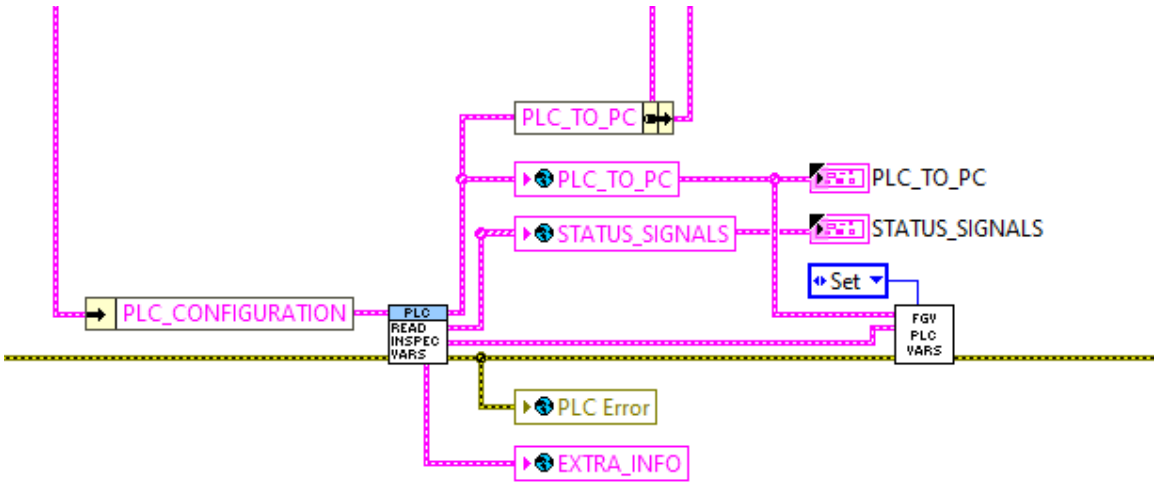


Figura 34: Caso READ_INSPECT_VARS del Module PLC Read

Como podemos ver, hacemos uso de un SubVI para obtener las variables que se almacenan en variables globales o, en el caso de “STATUS_SIGNALS”, en un FGV.

El SubVI mencionado se llama “PLC_Read_Insection_Vars” y, dependiendo de qué tipo de PLC sea, ejecuta un caso u otro para obtener esos datos. En el caso de Seamens, forzamos a recibir los datos del PLC en forma de un cluster con los elementos especificados (fig.36).

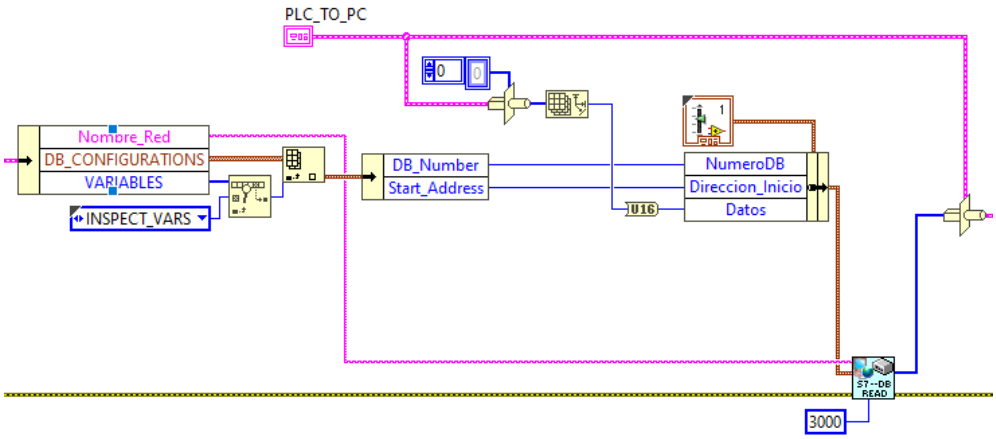


Figura 35: Ejemplo lectura de datos del PLC

El clúster “PLC_TO_PC” está compuesto por una serie de elentos que almacenan bit a bit la información obtenida del PLC en el formato que este utiliza.

Una vez obtenido el clúster con los bits almacenados, utilizamos un SubVI especial para cada tipo de PLC para convertirlo en el tipo de dato que necesitamos. No puedo mostrar estos SubVIs, ya que son confidenciales.

3.2. Análisis del PLC a implantar

Cuando AUTIS alcanzó el mercado asiático, se comprometió a realizar todas las modificaciones necesarias para satisfacer a sus nuevos clientes. A la hora de implantar su proyecto estrella, SVS, la planta de fabricación de automóviles a la que iba a vender el producto entregó una lista de modificaciones del mismo. Normalmente, estas modificaciones tienen que ver con el número de cámaras, segmentos de iluminación, o con la longitud y número de túneles que se desean aquirir. Pero, esta vez, el cliente, una planta de Mitshibishi, decidió cambiar el PLC que se había estado utilizando hasta ahora.

Mitshubishi fabrica sus propios PLCs de alta gamma, y pensaron que era oportuno hacer uso de sus propios productos, no solo para mejorar la eficiencia de SVS, si no, también, para abaratar costes. Por eso mismo, desde el equipo de AUTIS Ingenieros, nos propusimos a implementar dicho PLC en el sistema.

Una vez implementado a nivel de hardware, se necesitaba una adaptación a nivel de software, una forma de que, con los mínimos cambios posibles, se puediese icorporar al software desarrollado que controla el funcionamiento del PLC, el “SVS PLC CONTROLLER”.

Para poder adaptar dicho PLC al software ya desarrollado, se tuvo que hacer un estudio de las distintas caractrísticas que lo diferenciaba del resto de los PLCs utilizados hasta la fecha. A continuación, voy a listar dichas especificaciones, centrándome más en las que influyen considerablemente en el desarrollo de este TFG.

Aspectos básicos de Mitsubishi R63RP

Tabla 3: FA-MOD PLC: Power Supply Module

Serie	MELSEC IQ-SERIES
Tipo	REDUNDANT POWER SUPLY
Fuente de alimentación (V)	24
Tipo de corriente	DC
Potencia de entrada máxima (W)	50
Corriente de salida 5V (A)	6.5

Tabla 4: Dimensiones y peso del PLC

Ancho (mm)	54.6
Altura (mm)	98 (106)
Largo (mm)	110
Peso (kg)	0.41

Transmisión de la información

A la hora de recibir la información, el software "SVS PLC CONTROLLER" utiliza un "molde" idéntico para todos. A partir de este molde, un clúster con distintos elementos, procesamos los datos, los nombramos y los ordenamos en distintos clústeres.

Mitsubishi funciona con transmisión de datos de 16 bits, por lo que deberemos tener en cuenta la diferencia a la hora de aplicar este molde. Por ejemplo, Siemens trabaja con transmisión de datos de 8 bits.

En la figura que se va a mostrar a continuación (fig.37), voy a poner, uno al lado del otro, el array de numéricos de 16 bits de Mitsubishi y el de 8 de Siemens, para que se pueda apreciar cómo los representa LabVIEW.

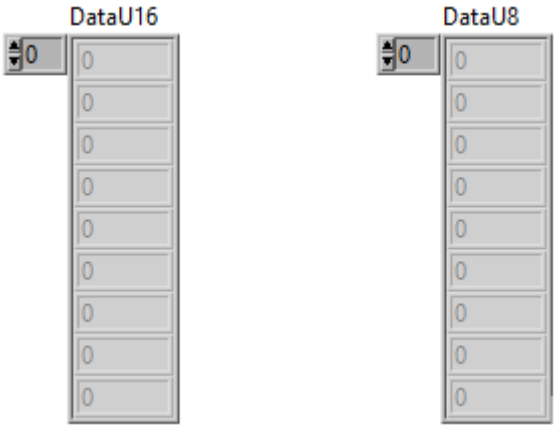


Figura 36: Diferencia entre arrays de datos de PLCs

Haciendo un análisis de conexión con el PLC de Mitsushima mediante el SubVI mencionado en este apartado, nos hemos fijado que esta es la estructura de datos que se recibe desde el PLC vía conexión TCP (fig.38).

PLC_TO_PC_SYSTEM_1	DATATYPE	
HEARTBEAT	Bool	D1500.0
LIFEBIT_OK	Bool	D1500.1
DATA_READY	Bool	D1500.2
START_ACQUISITION	Bool	D1500.3
BODY_ENTRY	Bool	D1500.4
CYCLE_ON_I1	Bool	D1500.5
CYCLE_ON_I2	Bool	D1500.6
BYPASS_MODE	Bool	D1500.7
PULSES_I1	Dint	D1501
PULSES_I2	Dint	D1503
CURRENT_INSP_1	Bool	D1505.0
CURRENT_INSP_2	Bool	D1505.1
TEST_INIT_CONT	LReal	D1506
TEST_END_CONT	Lreal	D1510
ENCODER_SWITCHING	Bool	D1514.0
DIRECTION_LH_TO_RH	Bool	D1514.1
DIRECTION_RH_TO_LH	Bool	D1514.2
FANS_LH	Bool	D1514.3
FANS_RH	Bool	D1514.4
CAMERA_RESET	Bool	D1514.5
BODY_1_N_STOPS	Int	D1515
BODY_1_N_SETBACKS	Int	D1516
BODY_1_MAX_SPEED	LReal	D1517
BODY_1_AVERAGE_SPEED	LReal	D1521
BODY_2_N_STOPS	Int	D1525
BODY_2_N_SETBACKS	Int	D1526
BODY_2_MAX_SPEED	LReal	D1527
BODY_2_AVERAGE_SPEED	LReal	D1531
ARRAY_SPARE_SVS	Array Int [0..19]	D1535
STATUS_SIGNALS	Array BOOL [0..63]	D1555.0
ENCODER_ACTUAL_POSITION_MM	LReal	D1559
ENCODER_ACTUAL_SPEED_MM_S	LReal	D1563
TRIGGER_TEST_COUNTER	Int	D1567
TRIGGERS_DONE_BODY_1	Array Int [0..63]	D1568
TRIGGERS_DONE_BODY_2	Array Int [0..63]	D1632
MM_ABS_ENCODER	Real	D1696

Figura 37: Encapsulación de la información del PLC de Mitsubishi

Obviamente, los datos recibidos no tienen nombre, ni tipo, si no que son, como hemos visto anteriormente, una serie de número de 16 bits que debemos convertir mediante un molde.

Nuestro objetivo, por lo tanto, es hacer posible que, con el array mencionado, consigamos varios clústeres con la información de la imagen, poniendo el tipo de dato que es cada uno, el nombre y el orden establecido.

3.3. Adaptación propuesta

Teniendo en cuenta las diferencias de funcionamiento y especificaciones con los otros PLCs implementados anteriormente, es fundamental cambiar aspectos importantes de los módulos "Module PLC Management" y "Module PLC Read". Los demás módulos, gracias a su modularidad, están exentos de recibir cambios.

A continuación, voy a enumerar los cambios que voy a realizar en cada uno de los módulos mencionados:

3.3.1. Cambios en "Module PLC Management"

Caso "Open Session"

En dicho caso, ya mencionado en apartados anteriores, se inicia sesión de dos formas distintas, una por cada tipo de PLC. Por lo tanto, deberemos crear otro caso por el PLC nuevo.

Caso "Close Session"

Lo mismo que en el apartado anterior. En este caso, hay una estructura de casos en el que cada PLC tiene su forma específica de cerrar la sesión o conexión. Deberemos hacer lo mismo con el nuevo.

VI "PLC_Write_Vars"

Debemos adaptar el VI para que permita el uso del PLC de Mitsubishi en los casos en los que se quiera encender o apagar algún elemento, o en el caso en el que se quieran calibrar los "CAM_TRIGGERS"

VI "Adapt_PLC_CameraTriggering"

Para el caso anterior del VI "PLC_Write_Vars", en el que se calibran los triggers de la cámara, se hace uso del VI "Adapt_PLC_CAM_TRIGGERS", que devuelve la información necesaria para la adaptación de los triggers de la cámara haciendo uso de la información aportada. Esta adaptación es diferente para cada PLC.

VI "Adapt_PLC_Camera_Limits"

En este VI se busca establecer los límites en los triggers de la cámara. Tanto el límite superior como el inferior. En el caso de Siemens, por ejemplo, estos límites componen un clúster como el que voy a mostrar en la siguiente imagen (fig.39).

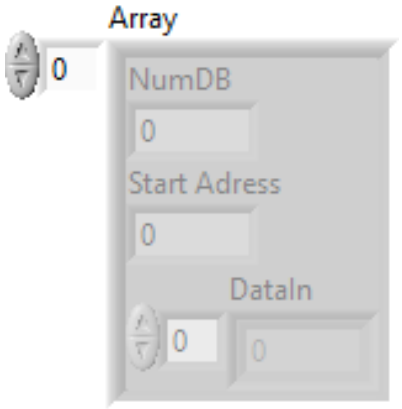


Figura 38: Array que representa los límites de triggers

Mientras que el de Rockswell requiere un simple array de números reales. Esto nos indica que, lo más probable, sea que se necesite una adaptación al nuevo PLC según sus especificaciones.

3.3.2. Cambios en “Module PLC Read”

VI “PLC_Read_Inspection_Vars”

Con este VI, podemos obtener la información que nos llega de la conexión con el PLC. Existen dos casos, uno por cada PLC, ya que estos trabajan con distintas señales y encapsulado de información. Es necesario, por lo tanto, crear un caso particular para el caso del nuevo PLC.

Creación de un VI para procesamiento de los datos

Veo necesaria la creación de un VI que convierta la información que llega de la conexión con el PLC en el caso que se establecerá (punto anterior) a datos procesables, basándome en VIs como “PLC_Transform_Inspection_Vars”, del PLC de Siemens.

4. Implementación

En este capítulo se mostrará los diferentes cambios realizados en desarrollo base “SVS PLC CONTROLLER” que sigue la estructura de cambios mencionada en el capítulo “Adaptación propuesta”. Mostraré el porqué de dichos cambios, el proceso de desarrollo de estos, y adjuntaré el código para su correcto entendimiento.

Para que se entienda todo mejor, estructuraré el capítulo en dos partes, una que explique la adaptación en el módulo de control de “órdenes”, establecimiento y cierre de conexión del PLC nuevo, y otro que explique cómo adaptamos el módulo de lectura para que pueda obtener los datos del PLC y procesarlos para ser óptimos para la lectura.

4.2. Adaptando “Module PLC Management”

Caso “Open Session”

La primera adaptación que debimos tener en cuanta, es la de conexión directa entre software y PLC. Para ello, y como se ha visto en el apartado 3.1.5 (“Module PLC Management”), haremos uso del VI “MODBUS_TCP_OPEN”, un VI desarrollado por AUTIS de código cerrado, que nos permitirá una conexión TCP directa desde el PC al PLC.

Este VI tiene dos terminales de entrada: Uno para el ID de la conexión y otro que conforma la configuración al SOCKET. Para ello, debemos de obtener estos datos de alguna forma. La solución que se encontró fue obtenerla directamente del “Module Configuration”.

Como ya hemos explicado, todos los datos necesarios para los módulos, se genera en el módulo de configuración. En el caso de la configuración de los PLCs, sus datos se almacenan en un FGV. Por lo tanto, modificando ese FGV, podemos hacer que también contenga la información necesaria para el PLC de Mitsubishi.

Pero, antes que nada, debemos de modificar los directorios en los que se almacena la información del PLC que es cargada por el módulo de configuración. Dicha carga se realiza tanto en directorios del propio proyecto como en bases de datos de la compañía. Por temas de confidencialidad, no puedo mostrar el archivo modificado.

La carga de dichos archivos se realiza en este fragmento del código del módulo de configuración (fig.40):

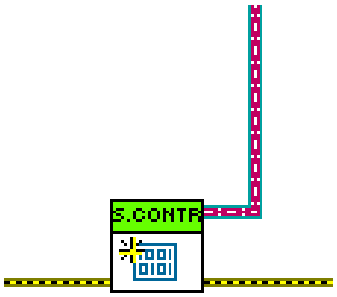


Figura 39: VI de carga de información general

Ese tipo de cable, de color marrón rojizo, es un “Data Value Reference”. Esto es un concepto muy complejo dentro del lenguaje de LabVIEW. A modo de resumen, ese cable representa una referencia a un objeto. Cuando un clúster se hace demasiado grande, es recomendable convertirlo en objeto.

Adaptando el objeto, podemos permitir que almacene las variables que queremos del PLC. El caso en el que se carga información del objeto al FGV que almacena datos de los PLCs es el siguiente (fig.41):

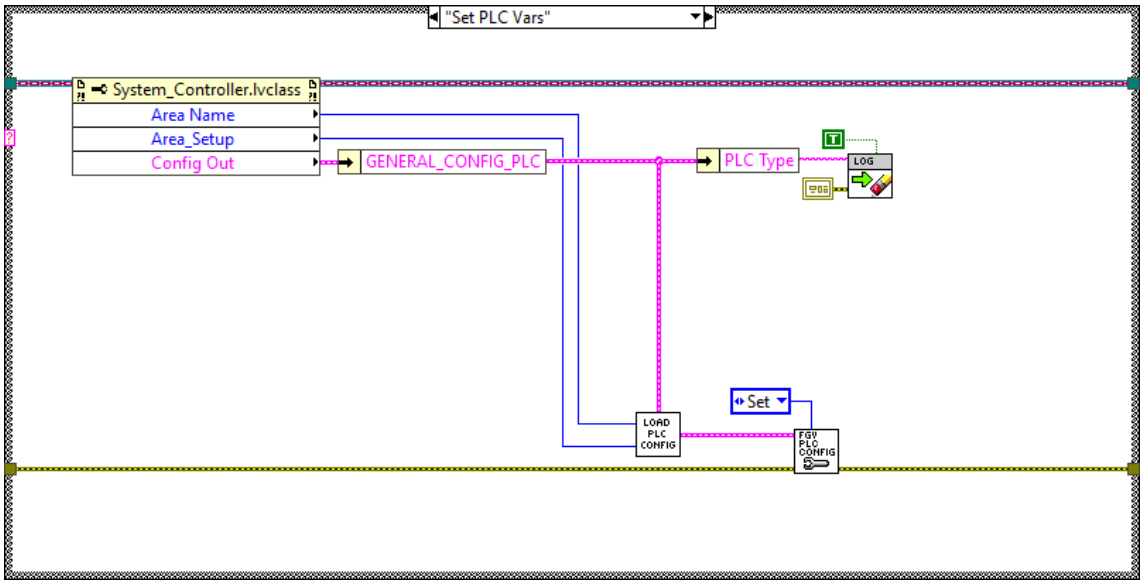


Figura 40: Caso de almacenamiento de información del PLC en FGVs

En este caso, nos encontramos con el SubVI llamado “LoadConfig_GetPLCconfig”, el cual adapta la información del objeto al clúster de información del PLC que nosotros buscamos. Este clúster se almacena en el FGV de variables de PLC, llamado “FGV_PLC_Configuration”.

El último paso para obtener los datos que queremos del PLC del Mitsubishi al FGV mencionado, sería adaptar” LoadConfig_GetPLCconfig” para que procese también los datos del nuevo PLC. Dicho SubVI “pregunta” el tipo de PLC del proyecto antes de procesar los datos del objeto, después, pone los datos en un clúster general, donde solo se rellenan los datos del PLC que se va a utilizar (los otros quedan vacíos, pero siguen formando parte del clúster).

A continuación, voy a mostrar la solución que he implementado (fig.42):

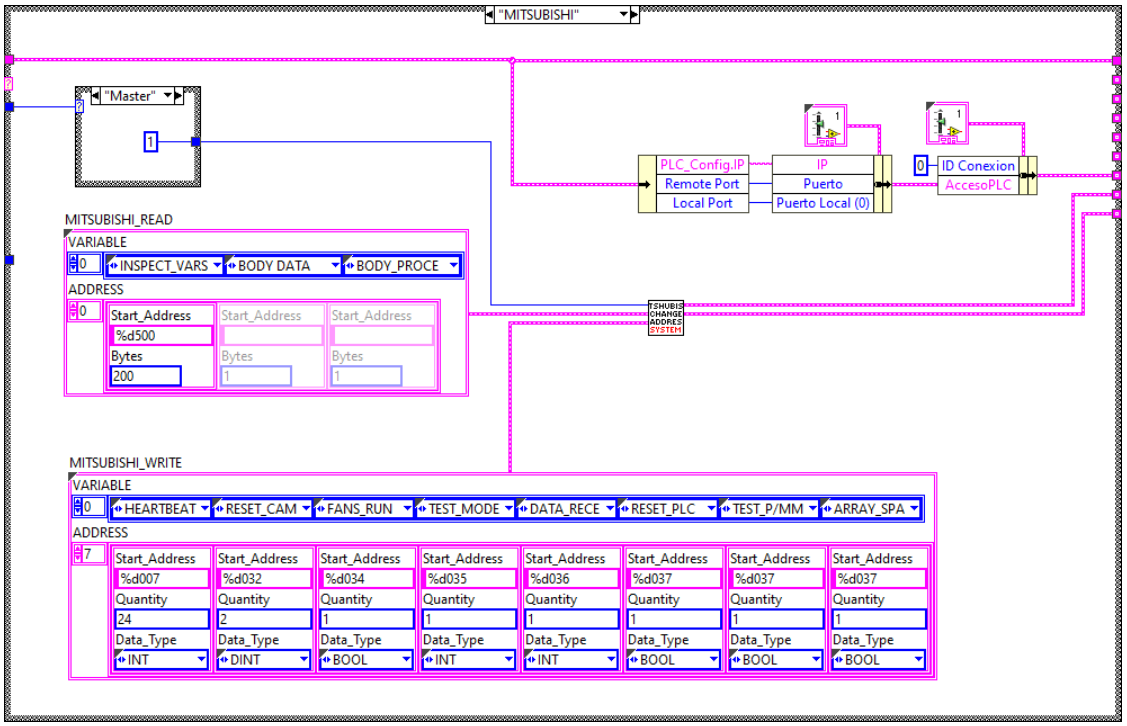
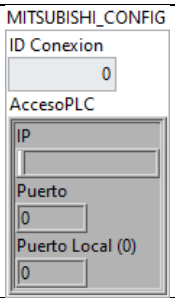
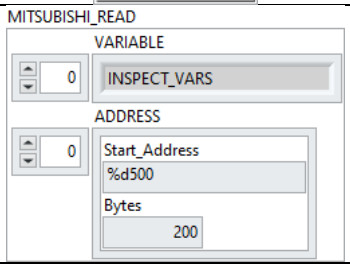
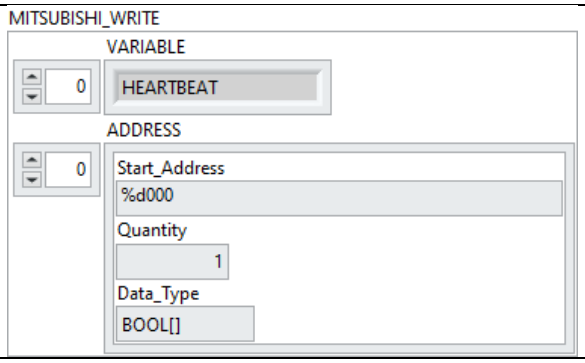


Figura 41: Solución creación de clúster de configuración para el PLC de Mitsubishi

De este código, obtenemos 3 clústeres con información del PLC que buscábamos:

Tabla 5: Clústeres con información de los PLCs

Visualización	Explicación
	Este cluster contiene los elementos necesarios para establecer una conexión directa de tipo TCP con él
	Contiene la dirección del PLC para su lectura, y los componentes que componen su mensaje de lectura en el array "VARIABLE".

	Contiene información acerca de los distintos nombres de los componentes que puede controlar el PLC (en el array llamado "VARIABLE") y un array con las direcciones de cada uno de estos componentes

Almacenados esos clústeres en el cluster principal, que a su vez es introducido en el FGV de PLCs, ya podemos obtener la información que necesitábamos para la conexión con el PLC en el "Module PLC Management". Por lo tanto, el resultado de dicha adaptación será la siguiente (fig.43):

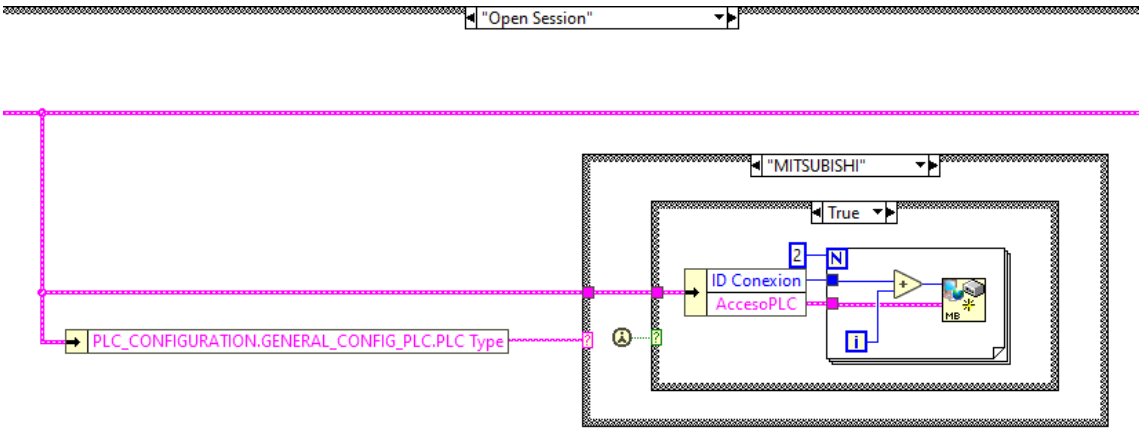


Figura 42: Solución a la conexión TCP con el PLC de Mitsubishi

Como se puede apreciar, he añadido a la estructura de casos el caso "MITSUBISHI". Dentro del caso, compruebo que sea la primera vez que lo ejecuto (no es necesario establecer conexión continuamente) mediante el nodo "First Call?", si es así, establezco conexión haciendo uso del cluster "MITSUBISHI_CONFIG", creado anteriormente.

Caso "Close Session"

Esta implementación ha sido sencilla de realizar gracias a los cambios hechos en el punto anterior. Dentro del caso "Close Session", he añadido otro caso para el PLC de Mitsubishi (fig.44).

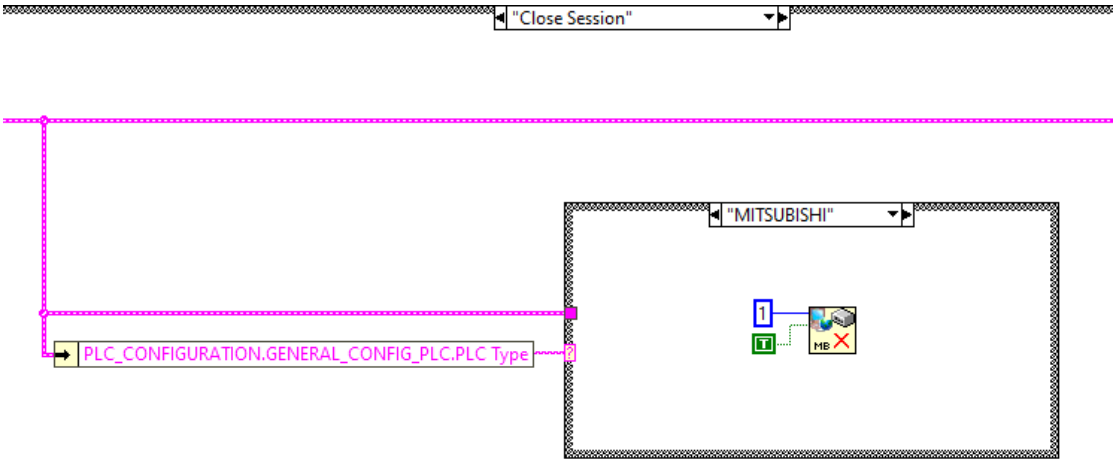


Figura 43: Solución al cierre de conexión TCP con el PLC de Mitsubishi

He utilizado el SubVI "MODBUS_TCP_Close", creado por AUTIS y de código cerrado, para cerrar la conexión TCP al PLC.

VI "Adapt_PLC_CameraTriggering"

El caso de Mitsubishi para este VI es mucho más sencillo que los otros, ya que solamente se necesita información del registro de los triggers. Optamos por almacenarlos en un array de strings.

El orden de ejecución de este VI es el siguiente: Forzamos la "Data" a convertirse en un clúster y procesamos ese clúster para obtener el array de registros. Echemos un vistazo al código (fig.45):

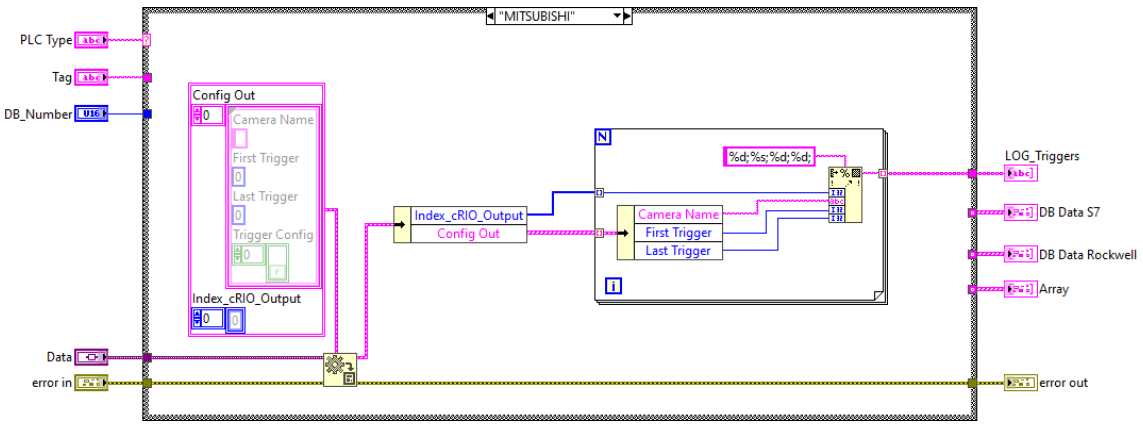


Figura 44: Solución a la creación del registro de triggers para el PLC de Mitsubishi

La data se obtiene como un tipo "variant", el cual es un tipo de variable utilizada en LabVIEW para elementos que pueden tener todo tipo de tamaño, procedencia y contenido, sin especificar nada previamente. De esta forma, podemos forzar su "forma", esto es, añadir los elementos uno en uno en un clúster con el orden, el tipo de elementos y las especificaciones que nosotros queramos. Para hacer eso, utilicé el nodo "Variant to Data".

Una vez convertido la “Data” en un clúster fácilmente manipulable, modificamos cada uno de los elementos para que se muestren como cadenas de texto, y los almacenamos en un array.

VI “Adapt_PLC_Camera_Limits”

De nuevo, en el caso del nuevo PLC, la adaptación resulta más fácil que el de otros proveedores. Siguiendo la misma estructura que en el punto anterior, conseguí obtener un clúster con la “forma” exacta en la que quería almacenados los elementos. Haciendo uso de un bucle foráneo, separé los “First Triggers” y “Last Triggers” en dos arrays numéricas diferentes, que se almacenaron dentro del clúster principal llamado “Data Limits”. Si nos fijamos, sigue el mismo funcionamiento que el clúster principal de PLC, el cual puede almacenar todos los datos de todos los distintos PLCs, pero solo obtiene la información del PLC que está siendo utilizado en ese proyecto.

Cabe destacar que, antes de almacenar los datos en arrays, los convertí, uno a uno, de variables reales de 32 a 16 bits, ya que es con este formato con el que trabaja el PLC de Mitsubishi. Para hacerlo, hice uso del nodo “To Unsigned Wold Integer”. A continuación (fig.46), voy a mostrar el resultado de la adaptación:

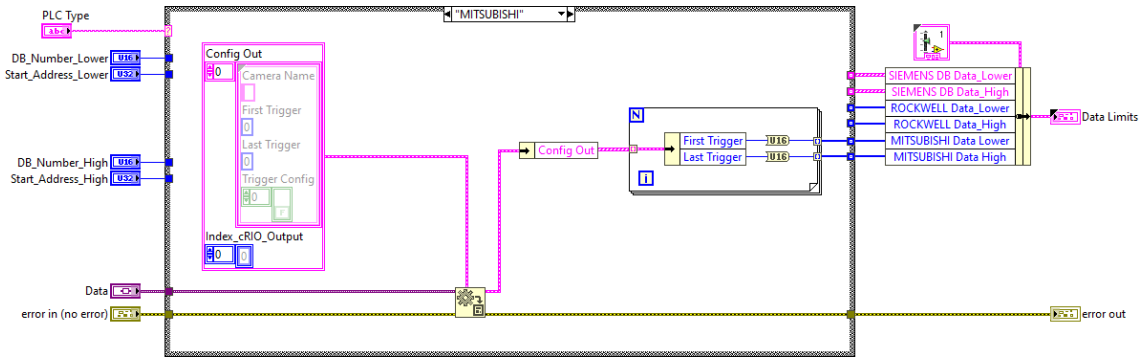


Figura 45: Solución almacenamiento de datos de límites de triggers para PLC de Mitsubishi

VI “PLC_Write_Vars”

Para implementar esta modificación, se debe conocer el funcionamiento del SubVI llamado “MODBUS TCP Read_Write”, otro desarrollado por AUTIS y de código cerrado. Como no puedo explicarlo su código por confidencialidad, explicaré su objetivo y sus requisitos.

Este SubVI es utilizado para escribir “ordenes” o “comandos” al PLC mediante conexión del tipo TCP. Tiene 4 terminales de entrada obligatorios para su funcionamiento: ID de la conexión con el PLC (número real), un clúster con la configuración al SOCKET, los comandos (clúster) y un “timeout” que pondré siempre en 3000 ms.

Los dos primeros terminales ya hemos visto como obtenerlos en el punto “Caso “Open Session””. El tercer terminal puede hacerse de muchas formas, lo veremos más a delante.

Una vez explicado esto, centrémonos en la adaptación del “PLC_Write_Vars”. Para ello, debemos saber que la adaptación se tiene que hacer para dos casos: “CAM_TRIGGERS” y “Default”. “Default” hace referencia a cualquier “orden” o “comando” que se quiera realizar sobre

un componente del proyecto que controle el PLC que no tenga que ver con los triggers de la cámara.

A continuación (fig.47), voy a mostrar la implementación que he hecho para el caso "Default", que servirá para comprender mejor el otro caso:

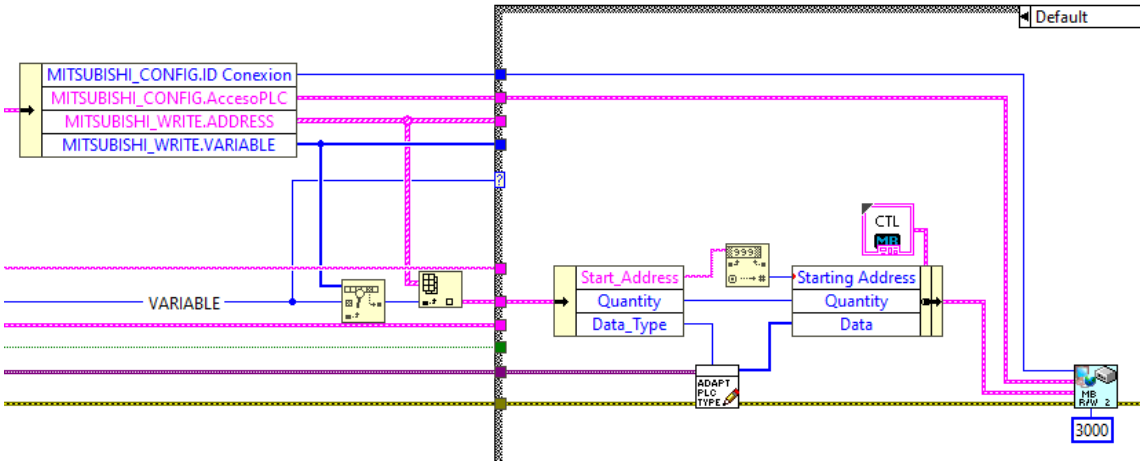


Figura 46: Solución escritura para PLC de Mitsubishi mediante conexión TCP

Hacemos uso del array de clústeres de "MITSHUBISHI_WRITE", explicado en el punto "Caso "Open Session"", para obtener las "ordenes" o "comando" que será enviado al SubVI "MODBUS TCP Read_Write".

Del array de clústeres de "MITSHUBISHI_WRITE", obtengo todos los clústeres con "VARIABLE" igual a la obtenida. Un ejemplo de variable en este caso sería "FANS_FUN", que valdría para encender o apagar los ventiladores de los segmentos de iluminación del arco de iluminación del túnel. Si la "VARIABLE" es igual a "CAM_TRIGGERS", se ejecutaría el otro caso.

Para continuar, es necesario entender qué es exactamente a lo que se refiere como "CAM_TRIGGERS". Un trigger, se entiende como una muestra de dato o, en el caso que nosotros le damos, un momento en el que una cámara o conjunto de cámaras van a tomar una fotografía.

En el caso de "CAM_TRIGGERS" se define el número de triggers que se van a utilizar en cada inspección del chasis que se va a realizar. Además, se especifica cuál es el número mínimo y máximo de triggers que se asignan a cada cámara. Cada inspección tiene una media de 900 triggers. De normal, cada cámara obtiene una media de 230 fotografías, es decir, participa en 230 triggers. Estos "triggers de participación" son asignados dependiendo de la posición de la cámara. Una cámara muy cerca del inicio del túnel, por ejemplo, participará en los primero 230 triggers, mientras que uno situado al final, puede participar en los últimos 230. Cada 58 mm se pasa de un trigger a otro. En la siguiente imagen (fig.47), habrá expuesta la solución que he realizado.

Como podemos ver en la imagen anterior (fig.48), hacemos uso de los SubVIs "Adapt_PLC_CamerTriggering" y "Adapt_PLC_CameraLimits" para obtener, por una parte, los registros de los triggers; por la otra, información acerca de los límites superiores e inferiores de los triggers. Haciendo uso del segundo SubVI mencionado, y del array de clústeres "MITSHUBISHI_WRITE", componemos dos clústeres, cada uno de ellos conteniendo órdenes de calibración de los límites de triggers de las cámaras del túnel.

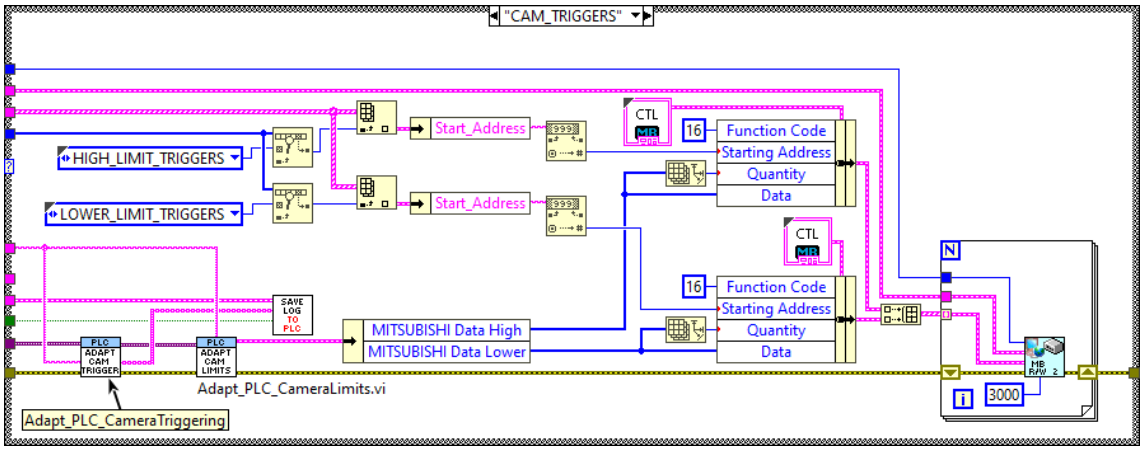


Figura 47: Solución escritura de triggers para cámaras mediante el PLC de Mitsubishi

4.3. Adaptando "Module PLC Read"

VI "PLC_Read_Inspection_Vars"

De nuevo, hacemos uso del SubVI "MODBUS TCP Read_Write", del que obtenemos un array de numéricos de 16 bits.

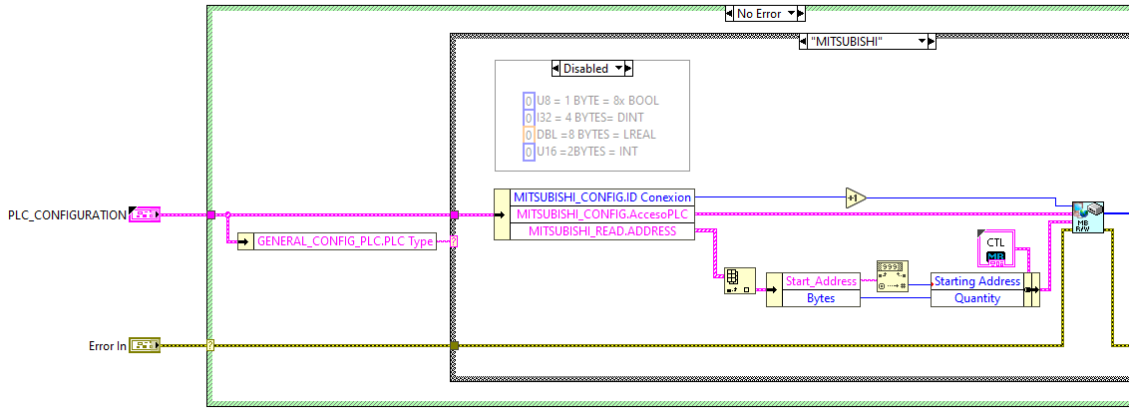


Figura 48: Solución lectura de datos para el PLC de Mitsushima

Pero, para poder leer correctamente los datos obtenidos, debemos transformarlos uno a uno a los tipos de variantes que nosotros buscamos. Para ello, debemos crear otro VI.

Creación de un VI para procesamiento de los datos

Para poder procesar correctamente los datos, debemos saber qué datos son los que nos llegan del PLC y cómo lo vamos a mostrar. Para ello, se recomienda hacer un repaso del capítulo 3.2, "Análisis del PLC a implantar".

Como hemos visto en el apartado anterior, ya tenemos el array con los valores de numéricos de 16 bits almacenados. El siguiente paso, será forzar a estos bits a colocarse en orden sobre un

molde establecido. Para ello, hacemos uso del nodo “Type cast”, al que le introduciremos el array junto a un “molde”:

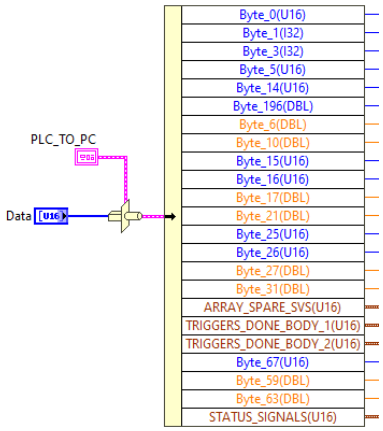


Figura 49: Desencapsulación de datos de un clúster

Como podemos ver (fig.50), el molde es un cluster, al que le hemos puesto a cada elemento, un molde, como se puede ver en el “Unbundle By Name”. Ahora, volveremos a utilizar el nodo “Type Cast” para asignar los nombres que se ven en la tabla del capítulo 3.2, “Análisis del PLC a implantar”. El resultado será el siguiente:

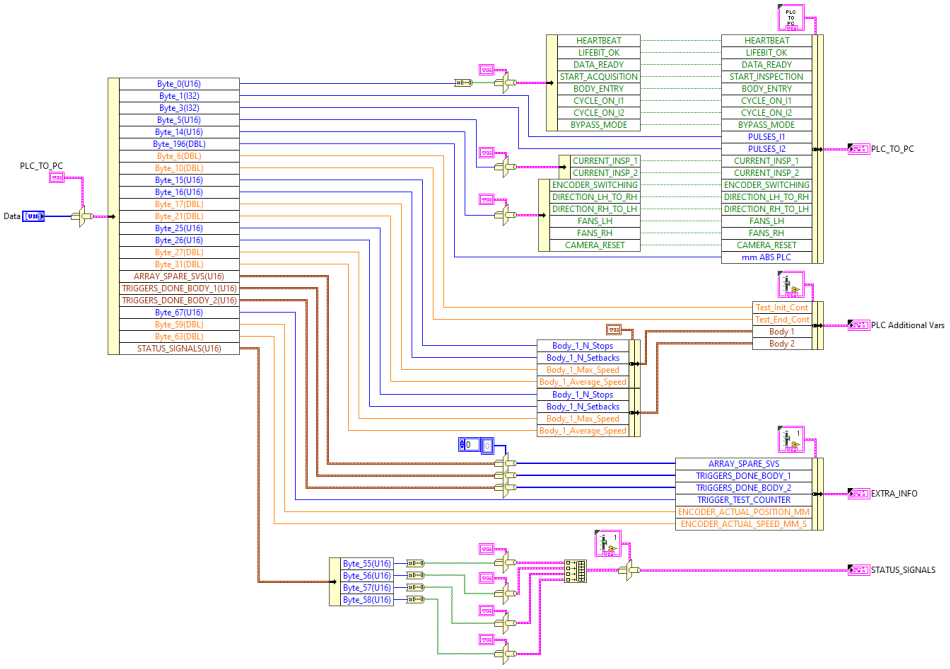


Figura 50: Solución para almacenamiento de información leída para el PLC de Mitsubishi

5. Resultados

En este capítulo, se pretenden mostrar los resultados obtenidos de la implementación del código al software “SVS PLC CONTROLLER”. Para ello, debemos conocer qué resultados son los que buscamos obtener. Por lo tanto, voy a dividir este capítulo en 3 partes: “Breve explicación del funcionamiento del Module HMI”, “Muestra de datos de otro PLC”, “Resultados de la implementación”

5.1. Breve explicación del funcionamiento del Module HMI

En este apartado, no pretendo explicar el funcionamiento y flujo del código del Module HMI, en parte, porque no se me está permitido mostrar dicho código, en parte, porque no es necesario para alcanzar el punto al que quiero llegar. Solo pretendo explicar la interfaz de este, para que, más adelante, comprendamos mejor los resultados mostrados por el PLC.

Normalmente, cada SVS está siendo controlado por mínimo tres PCs llamados WorkStation o ACQs, y un solo PC especializado en la parte de la producción, llamado PRO. Las ACQs se centran en el procesado de la información captada por las cámaras, mientras que las PRO ejecutan “SVS PLC CONTROLLER” (y otros muchos más programas), para controlar el proceso general de la captación de datos, calibración de dispositivos, control de estos… es decir, se encargan de gestionar el correcto funcionamiento del proyecto.

“SVS PLC CONTROLLER” se ejecuta automáticamente en la PRO1 (pueden haber más de una), cuando esta es encendida. Desde ese momento, el programa detecta en qué proyecto de SVS está siendo ejecutado, y dependiendo de esto, genera unas especificaciones u otras. En el caso de estar en el proyecto de Mitsushima, por ejemplo, generará datos para el PLC de Mitsubishi, en vez de el de Roswell o Siemens. Una vez obtenidos todos los parámetros, el programa espera hasta que comienza una inspección. La inspección es el proceso por el que el chasis del coche comienza a recorrer el interior del túnel para que este, mediante la iluminación y las cámaras, detecten sus posibles defectos.

En apartados anteriores, hemos explicado todos los engranajes que componen dicho programa para que realice lo mencionado. Ahora, explicaremos como muestra los datos.

El “Module HMI” tiene una interfaz bastante simple que divide los datos y procesos obtenidos de la inspección en submenús. Nosotros nos vamos a centrar en uno específico, el submenú de los PLCs. A continuación (fig.52), voy a mostrar una captura de la interfaz mencionada:

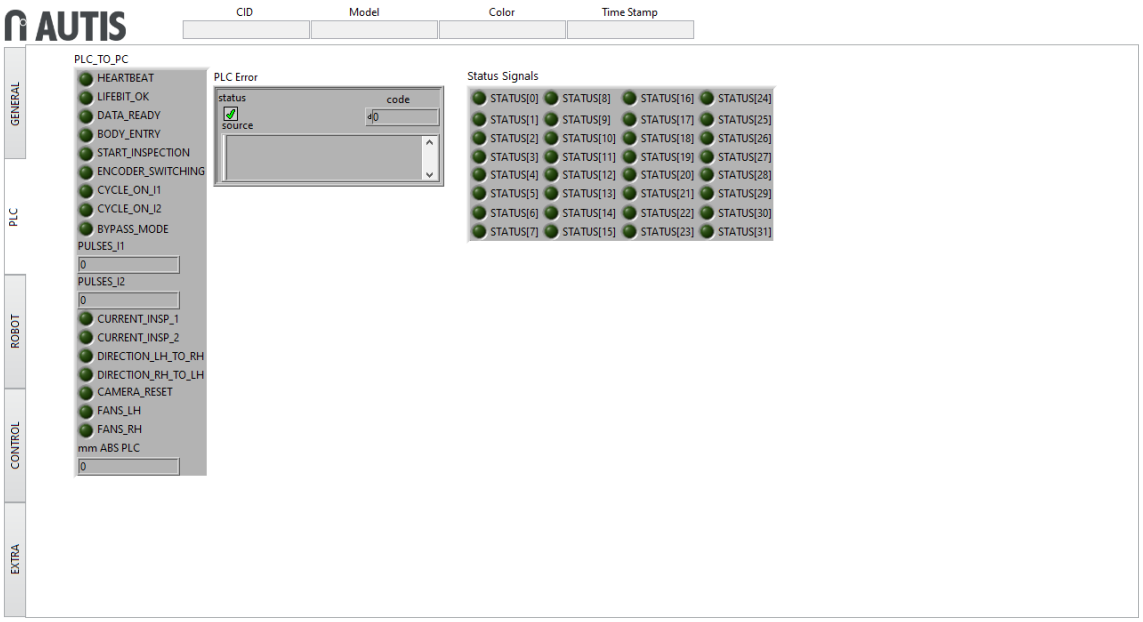


Figura 51: Interfaz de "SVS PLC CONTROLLER"

En este submenú, se muestran dos clústeres con información y un clúster de error. EL clúster de error, como es normal, nos mostrará si ha surgido algún error en el proceso de la inserción. Los otros dos clústeres los vamos a explicar a continuación.

Clúster PLC_TO_PC

Hablamos de este clúster en el capítulo 4.3, "Adaptando "Module PLC Read"". En dicho capítulo, creábamos un VI llamado "PLC_MITSHIBISHI_Read_Inspection", el cual, nos devolvía varios clústeres. Uno de ellos, era este. En este clúster, tenemos información recibida por parte del PLC en particular que haya asignado al proyecto. Veamos los componentes relevantes que lo forman:

- HEARTBEAT: Mientras esta señal siga enviando periódicamente un 0 y 1 (encender y apagar el led que representa en el clúster), significa que existe una conexión.
- DATA_ENTRY: Una planta puede estar procesando varios tipos de chasis al mismo tiempo. Como el PLC no tiene forma de saber si estamos con un chasis diferente o no, AUTIS ha desarrollado un programa llamado "GET BODY INFO", que detecta cuando la carrocería es distinta. Cuando esto ocurre, se envía una señal distinta al PLC. El PLC recibe esa señal, que encapsula de otra forma. Cuando esto llega al software "SVS PLC CONTROLLER", detecta la encapsulación distinta con un 1 en el valor del bit que nosotros llamamos "DATA_ENTRY". Esto permite al programa seleccionar la trama y obtener la información del nuevo chasis.
- BODY_ENTRY y START_INSPECTION: Hay dos sensores de posición en el túnel de inspección. Uno llamado BODY_ENTRY, que detecta cuando una carrocería nueva está entrando en el túnel haciendo que se enciendan los leds, y otro llamado START_INSPECTION, que indica cuando el chasis está en posición para iniciar la captación de datos.
- BYPASS_MODE: Este bit sirve para que la carrocería pase de un lado a otro del túnel sin que se realice ninguna inspección.
- CYCLE_ON_I1 y CYCLE_ON_I2: Es posible que, en una misma inspección, se estén analizando dos chasis a la vez. Para que esto ocurra, debe de haber una separación considerable entre una carrocería y otra.

Normalmente, el proceso es el siguiente: Se enciende el primero de los leds, se termina la inspección, se apaga el primer led y se enciende el segundo. Pero es posible que se encienda el segundo led sin haberse apagado el primero, por lo que, se estarían analizando dos al mismo tiempo.

- PULSES_I1 y PULSES_I2: Los pulsos están estrechamente relacionados con los triggers. Cada pulso, es una señal que comienza al mismo tiempo que se enciende el led CYCLE_ON_I1 o 2. Sirve para calcular la posición donde se encuentra el chasis respecto al inicio de la inspección. Cada pulso, que representa el valor de un 1 en un bit, se ejecuta de manera periódica. Como el chasis se mueve siempre a la misma velocidad, podemos obtener la posición de este. De esta forma, sabemos cuándo se estarán realizando los triggers (recordemos que cada trigger se ejecuta a los 58 mm).
  El PULSES_I1 representa los pulsos para la primera carrocería y el 2 para la segunda.
- CAMERA_RESET: Cuando este led se encienda, significa que las cámaras están siendo reseteadas.
- FANS_LH Y FANS_RH: Cuando esto leds sean encendidos, significa que los ventiladores de los segmentos han sido conectados. LH representa el lado izquierdo del arco de segmentos y RH el derecho.

Clúster Status Signals

Este segundo clúster es de información extra, y es exclusivo del tipo de PLC utilizado. Es información no contemplada para el proyecto, y puede representar y ser utilizada de muchas formas y fines dependiendo de la planta.

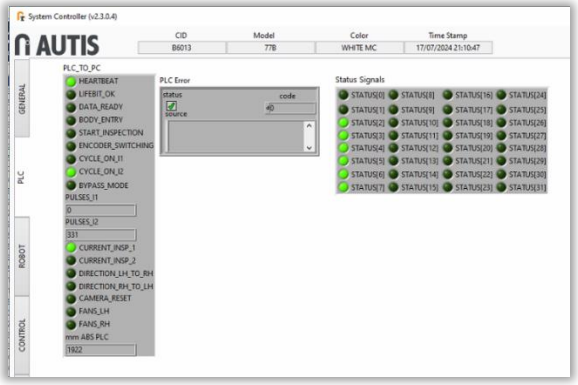
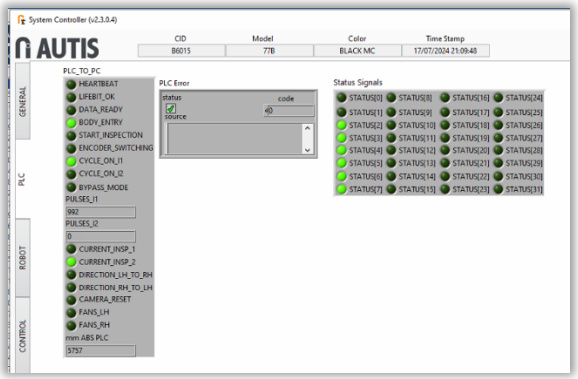
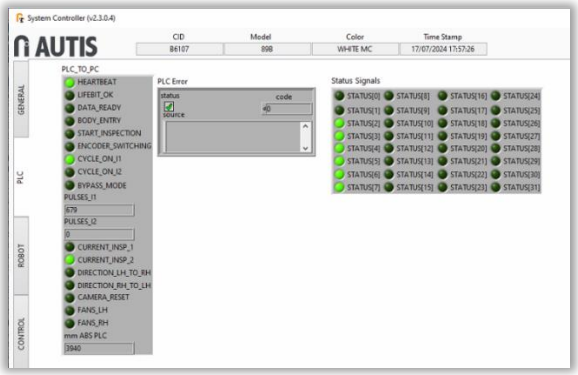
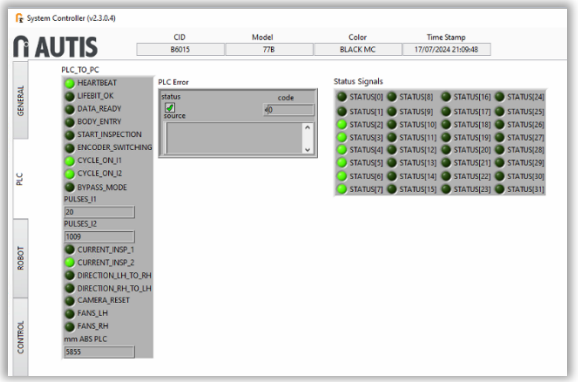
5.2. Muestra de datos de otro PLC

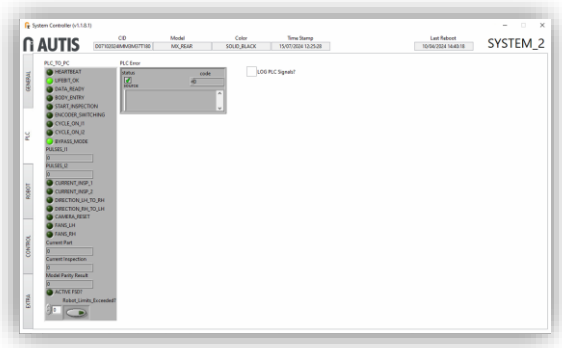
En el momento de la realización de este TFG, no hay ningún proyecto de SVS configurándose en el taller de AUTIS, por lo que, para poder obtener los datos de otros proyectos, haré uso de una herramienta ampliamente utilizada hoy en día en el sector y que, en AUTIS, se utiliza para monitorizar, dar soporte y actualizar parámetros de servicios o productos vendidos a nivel internacional: TeamViewer.

AUTIS tiene muchos clientes a lo largo del globo, muchos de los cuales, están disfrutando de los beneficios de SVS. Una de las marcas que más recientemente ha apostado por nosotros es FAW, una marcha de origen chino.

A continuación, voy a mostrar qué datos son los plasmados en la interfaz de “SVS PLC CONTROLLER”:

***Tabla 6**: Proceso de inspección desde PLC de Siemens*

Capturas de proceso	Explicación
	Existe una conexión con el PLC, ya que recibimos señal en el HEARTBEAT. El PLC espera recibir señal de un nuevo chasis.
	Al iluminarse el led de BODY_ENTRY, significa que se ha detectado un chasis acercándose al túnel, por lo que se habrán encendido las luces de los leds, comenzado con el proceso de calibrado de cámaras (triggering)
	Se ha iluminado el led de CYCLE_ON_I1, así que ha comenzado el primer ciclo (inspección de la primera carrocería) Vemos como el contador de pulsos ha aumentado (PULSES_I1). Con esto, podemos saber la posición del coche.
	Durante unos segundos, se ha detectado un segundo chasis entrando en el túnel, por lo que se ha encendido también el led CYCLE_ON_I2. Podemos apreciar cómo ha aumentado también el contador de pulsos (PULSES_I2).

	Aquí vemos un caso de BYPASS_MODE. Tesla, Fremont, estaba haciendo test en su planta, así que aproveché para obtener esta captura.

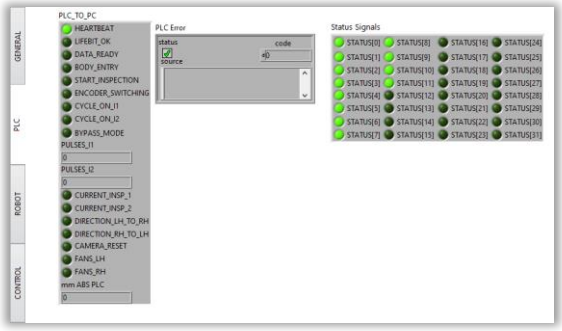
Hay algunos pasos que no me han sido posible capturarlos. Estaba haciendo capturas de un producto vendido a un cliente, por lo que no era posible alterar el proceso para obtener los datos de todos los casos posibles. Aun así, las capturas obtenidas, muestran bastante bien el proceso por el cual, el PLC, captura cada punto de la inspección.

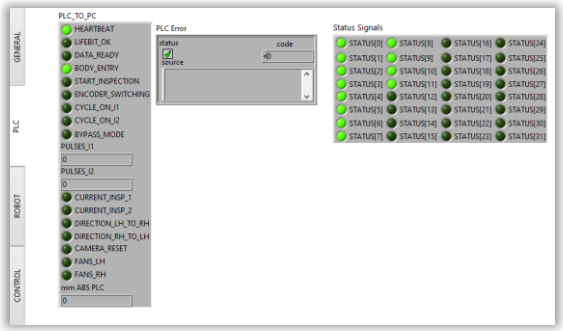
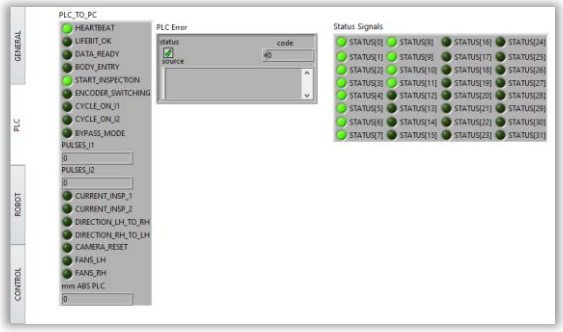
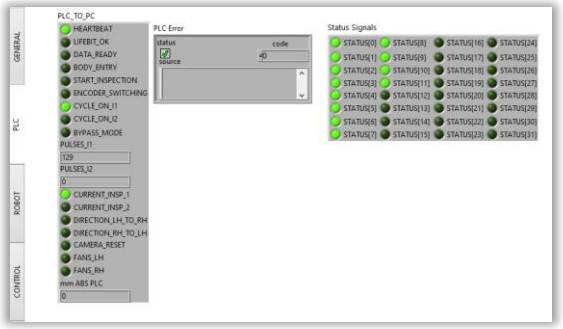
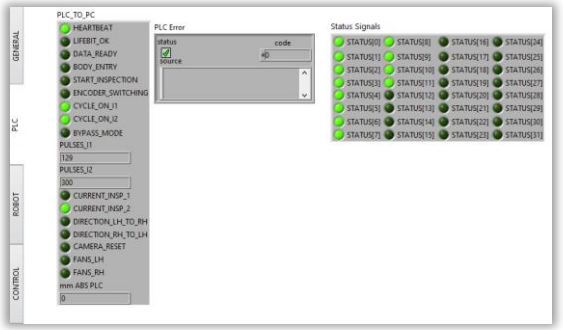
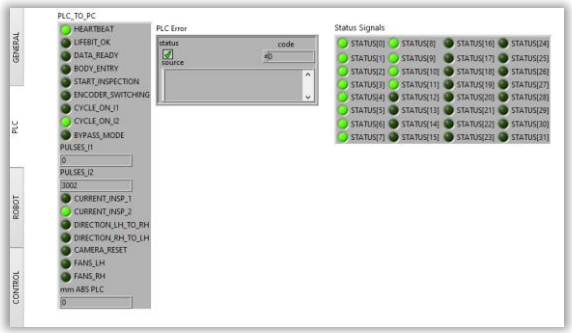
5.3. Resultados de la implementación

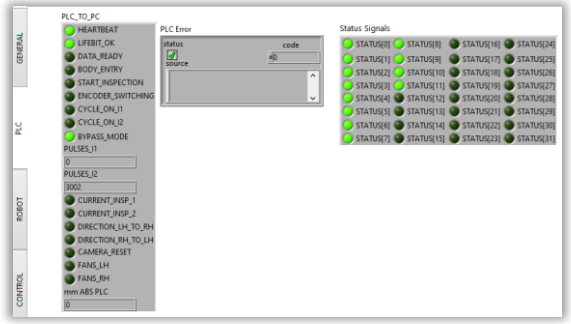
En este capítulo, mostraré los datos obtenidos de la implementación para el caso del PLC de Mitsubishi. Este PLC se estaba probando por primera vez en los talleres de AUTIS, para el proyecto de SVS de Mitsushima, por lo que es fácil obtener imágenes del proceso y testearlo manualmente. Aun así, para obtener capturas de los resultados obtenidos, voy a seguir haciendo uso de TeamViewer, ya que, aunque SVS tiene una pantalla que puede alternar entre cada PC de la estación, no podría pasármelas.

Como es un entorno controlado, puedo ser yo quien ejecute cada proceso de la inspección para ver, paso a paso, cómo reacciona el PLC. Por ello, voy a mostrar, a continuación, el proceso de inserción de dos chasis paso a paso:

Tabla 7: Proceso del funcionamiento del PLC de Mitsushima en una inspección de SVS

Capturas de proceso	Explicación
	Al ejecutarse el programa, a los pocos segundos, ya aparece la señal representada en el led HEARTBEAT. Por lo que la conexión PC a PLC ha sido establecida y está preparado para realizar la inspección.

	Al iniciar la inspección con otro software también desarrollador por AUTIS, el chasis comienza a moverse hacia la entrada del túnel, pasando por un sensor (BODY_ENTRY) que envía una señal al PLC para que encienda la iluminación y comience con la calibración de las cámaras (triggering).
	Al entrar al túnel, el sensor llamado START_INSPECTION envía una señal de un solo bit con valor 1 al PLC para indicar que el chasis está dentro del túnel y, por lo tanto, va a comenzar la captura de datos.
	Se enciende el led CYCLE_ON_I1 para indicar que ha comenzado el primer ciclo de inspección de chasis. Además, se aprecia como el contador de pulsos ha aumentado, indicando la distancia recorrida en pulsos controlados de una señal periódica.
	A los pocos segundos, se enciende CYCLE_ON_I2 (he obviado los pasos anteriores para este chasis), indicando que se están inspeccionando dos al mismo tiempo. Comienza el conteo de pulsos para este también.
	Al cabo de un tiempo, se apaga el primer led, indicando que ya se ha terminado la primera inspección. También, se reinicia el contador de pulsos del primer chasis.

	Finalmente, hice una prueba con el modo BYPASS, en el que el chasis pasa a través del túnel sin haber una inspección, es decir, sin recogida de datos.

Los resultados obtenidos han sido muy positivos, mucho más de lo que cabría pensar. Algunos PLCs de otras marcas pueden causar errores que deben de ser procesados y descartados luego de muchas inspecciones. En cambio, al realizar más de diez inspecciones distintas, con dos chasis en cada una, el nuevo PLC ha demostrado ser mucho más fiable.

Los resultados muestran claramente el correcto funcionamiento de la implementación del nuevo PLC, ya que este, se comporta igual que los demás. Algo destacable es la diferencia que hay entre el los tres PLCs respecto al uso del clúster “Status Signals”. El PLC de Roswell no hace uso de este clúster, ya que, las señales que envía conforman una trama más pequeña (no hay otros bits reservados). En el caso de Siemens y Mitsubishi, vemos como sí que aportan una serie de bits extra, los cuales, tienen una disposición diferente.

6. Conclusiones

El proyecto final de grado se centra en el análisis de un software complejo, comprensión del funcionamiento de un PLC y adaptación de código a dicho software. Con dicho fin, se hizo uso del sistema de programación gráfico llamado LabVIEW, así como distintas herramientas e ideas, algunas de ellas, la cuales estaban creadas previamente por la empresa.

Haciendo uso de la arquitectura Autis, basada en la modularidad y las máquinas de estados, me fue fácil la comprensión y desarrollo de la adaptación que buscaba, aprendiendo la importancia de realizar un código estructurado y ordenado.

Gracias al equipo del taller y el tutor asignado en la empresa, pude ver el proceso de ensamblado del túnel de SVS, con el que comprendí el funcionamiento, a varios niveles, de un proyecto de automatización de un nivel profesional muy alto y con demanda a nivel internacional.

Uno de los puntos claves del proyecto fue la comprensión del funcionamiento de la lectura y escritura del nuevo PLC, ya que este, pese a enviar señales parecidas a los otros que utiliza la compañía, tiene un encapsulado y funcionamiento distinto. Por lo tanto, fue crucial la optimización de la conversión de datos.

Finalmente, los datos tuve que compararlos con los de otros PLCs, los cuales, al no estar en las instalaciones de la compañía en el momento de la realización de este TFG, tuve que comunicarme con compañeros de la empresa y clientes de otros países para poder, mediante el uso de TeamViewer, obtener los resultados buscados.

Referencias

[1] Mitsubushi Electrics:

https://es.mitsubishielectric.com/fa/products/cnt/plc/plcr (Último acceso: 21 de julio de 2024)

[2] National Instruments. Consideraciones de diseño en LabVIEW [Artículo en línea]. Disponible en:

https://www.ni.com/es/support/documentation/supplemental/22/design-considerations-in-labview-.html (Último acceso: 21 de julio de 2024).

[3] National Instruments. Explicación del panel frontal de LabVIEW [Artículo en línea]. Disponible en:

https://www.ni.com/es/support/documentation/supplemental/08/labview-front-panel-explained.html (Último acceso: 21 de julio de 2024).

[4] National Instruments. Explicación de Arrays y Clusters de LabVIEW [Artículo en línea]. Disponible en:

https://www.ni.com/es/support/documentation/supplemental/08/labview-arrays-and-clusters-explained.html (Último acceso: 21 de julio de 2024).

[5] National Instruments. Event Structure [Artículo en línea]. Disponible en:

https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/structures/event-structure.html (Último acceso: 21 de julio de 2024).

[6] National Instruments. Máquinas de estado [Artículo en línea]. Recuperado de:

https://www.ni.com/es/support/documentation/supplemental/16/simple-state-machine-template-documentation.html (Último acceso: 21 de julio de 2024).

[7] National Instruments. ¿Qué es un Queue? [Artículo en línea]. Recuperado de:

https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P7OfSAK&l=es-ES (Último acceso: 21 de julio de 2024).

[8] National Instruments. Arquitectura de Producto/consumidor en LabView [Artículo en línea]. Recuperado de:

https://www.ni.com/es/support/documentation/supplemental/21/producer-consumer-architecture-in-labview0.html (Último acceso: 21 de julio de 2024).

[9] National Instruments. Comunicación TCP/IP [Artículo en línea]. Recuperado de:

https://www.ni.com/es/support/documentation/supplemental/06/basic-tcp-ip-communication-in-labview.html

[10] National Instruments. Foro y comunidad de LabVIEW:

https://forums.ni.com