

**GIPUZKOAKO INGENIARITZA ESKOLA**

ESCUELA DE INGENIERÍA DE GIPUZKOA

**GRADU AMAIERAKO LANA / TRABAJO FIN DE GRADO**

---

Grado en	Ingeniería Electrónica Industrial y Automatismos	Gradua
Fecha	XXXX XX XX	Data

---

**Desarrollo de una interfaz de usuario  
estándar para entornos Industria 4.0**

---

**Nombre y apellidos del alumno/a:**

Aner Villanueva Garcia

---

**Nombre y apellidos del director/a:**

Jose Luis Jodra

**Nombre y apellidos del codirector/a:**

Ander Ansuategi

## **RESUMEN**

La evolución de la comunicación entre Persona y Máquina deriva, principalmente, de la constante aplicación de las tecnologías modernas en el día a día de las personas. Se busca una fácil y rápida comprensión de las nuevas tecnologías, permitiendo así, un uso óptimo del producto.

Este proyecto tiene como objetivo desarrollar una comunicación entre Personas y Máquinas intuitiva, mediante el uso de interfaces gráficas de usuario. La GUI (interfaz gráfica de usuario) estará dirigida principalmente a la visualización de información de distintos dispositivos o softwares. Esto se consigue con un sistema genérico, permitiendo la compatibilidad con varios conectores que permiten la recepción y emisión de información, así como la flexibilidad para configurar, en manos del usuario, diversos aspectos y componentes relativos a la GUI y los conectores.

## ÍNDICE

Resumen	2
Índice	3
Índice de tablas	6
Índice de figuras	7
<b>1. Introducción</b>	<b>9</b>
1.1. Contexto y motivación del proyecto	10
1.1.1. Tekniker	10
1.1.2. Planteamiento del problema y motivación	11
1.2. Objetivos del proyecto	11
1.2.1. Creación de Interfaz	11
1.2.2. Creación de conectores	11
1.2.3. Creación de IWidgets	12
1.2.4. Documento de Configuración	12
1.2.5. Implementación	12
1.3. Planificación	13
1.3.1. Planificación del proyecto	13
1.3.2. Presupuesto	14
1.4. Análisis de riesgos y viabilidad	16
1.4.1. Análisis de riesgos	16
1.4.2. Análisis de viabilidad	16
<b>2. Estado del arte</b>	<b>17</b>
2.1. ROS	17
2.2. OPC UA	18
2.3. OpenCV	18
2.4. GigE Vision, Genicam	19
2.5. Qt	19
2.6. Bibliotecas de código para crear interfaces gráficas	20
<b>3. Recursos necesarios</b>	<b>21</b>
3.1. Hardware	21
3.1.1. Cámaras	21
3.1.1.1. Camara Logitech TESSAR 2.0/3.7	21

3.1.1.2. IDS imaging u3-3890CP	21
3.1.1.3. IDS imaging GV-5890CP	21
3.1.1.4. OPTOCam COE-032-M-POE-040-IR-C	22
3.1.1.5. Gobi 640	23
3.1.2. SIEMENS SITOP smart GEPI333-2BA01	23
3.1.3. WEIDMULLER UC20-WL2000-AC	24
3.1.4. Switch D-Link	25
3.1.5. Globalsat BU-353-54 GPS	25
3.2. Software	26
3.2.1. Python 3.8.10	26
3.2.1.1. Pipenv (entornos virtuales)	26
3.2.2. ROS	27
3.2.3. OPC UA	28
3.2.3.1. UA Expert	28
3.2.4. Open CV	28
3.2.5. Aravis	29
3.2.6. Qt	29
3.2.6.1. PySide2	29
3.2.6.2. Qt Designer	29
3.2.7. PyCharm	30
3.2.8. GitLab	30
3.2.9. Google Docs	30
3.2.10. Text Editor	30
<b>4. Requerimientos del sistema</b>	<b>31</b>
4.1. Funcionalidades	31
4.1.1. Requisitos funcionales	32
4.1.2. Requisitos no funcionales	34
<b>5. Diseño y arquitectura de la solución</b>	<b>35</b>
5.1. Diseño de alto nivel	35
5.2. Diseño de bajo nivel	41
5.2.1. Primera versión del sistema	41
5.2.1.1 BackEnd	41
5.2.1.2 FrontEnd	58
5.2.1.3 Conclusiones de la primera version	64
5.2.2. Segunda versión del sistema	66

5.2.2.1 Planteamiento del Problema	66
5.2.2.2 Hipótesis: Plugins	67
5.2.2.3 Conclusiones de la segunda versión	68
<b>6. Resultados obtenidos</b>	<b>69</b>
<b>7. Conclusiones y trabajo futuro</b>	<b>77</b>
<b>8. Acrónimos y glosario</b>	<b>78</b>
8.1. Glosario	78
8.2. Acrónimos	80
<b>9. Bibliografía</b>	<b>82</b>
<b>Anexos</b>	<b>86</b>
Anexos I	88
Anexos II	98
Anexos III	121

## ÍNDICE DE TABLAS

<i>Tabla 1.1 Planificación del proyecto</i>	<b>13</b>
<i>Tabla 1.2 Presupuesto del proyecto</i>	<b>14</b>
<i>Tabla 1.3 Tabla de análisis de riesgos</i>	<b>16</b>

## ÍNDICE DE FIGURAS

Figura 3.1 Logitech QuickCam pro 9000	22
Figura 3.2 IDS Imaging U3-3890CP	22
Figura 3.3 IDS Imaging GV-5890CP	22
Figura 3.4 OPTOcam COE-032-M-POE-040-IR-C	23
Figura 3.5 Gobi+ 640	24
Figura 3.6 SIEMENS SITOP smart GEPI333-2BA01	
	24
Figura 3.7 WEIDMULLER UC20-WL2000-AC	25
Figura 3.8 SWITCH D-Link DGS-105	25
Figura 3.9 GLOBALSAT BU-353-S4 GPS	26
Figura 5.1 Diagrama de clases de la arquitectura del entorno del programa	35
Figura 5.2 Prototipo inicial de ventana principal	37
Figura 5.3 Representación de un conector sobre la GUI	37
Figura 5.4 Representación de conexión del conector para recibir datos de un componente y mostrarlas en la GUI	38
Figura 5.5 Representación de conexión del conector para enviar datos introducidos en la GUI a un componente	38
Figura 5.6 Representación estructural del módulo <i>interface IConnector</i>	39
Figura 5.7 Diagrama de flujo del programa principal	41
Figura 5.8 Ejemplos de conectores en el documento de configuración	43
Figura 5.9 Asignación de <i>parser</i> y <i>logger</i> en el documento de configuración	43
Figura 5.10 Ejemplo de conector con <i>parser</i> y formato en el documento de configuración	44
Figura 5.11 Asignacion de parametros en conector OPC UA	44
Figura 5.12 Asignación de parámetros en conector ROS	44
Figura 5.13 Gestión de Pestañas en documento de configuración	45
Figura 5.14 Diagrama de flujo de la Ventana principal	46

Figura 5.15 Diagrama de flujo de IConnector	49
Figura 5.16 Diagrama de flujo de conector estándar	51
Figura 5.17 Diagrama de flujo de conector OPC UA	52
Figura 5.18 Diagrama de flujo de conector ROS	54
Figura 5.19 Diagrama de flujo de un IWidget estándar	55
Figura 5.20 Diagrama de flujo de IWidget específico PLCWidget	57
Figura 5.21 GUI de la Ventana principal	59
Figura 5.22 GUI de la ventana multiconexiones Layout	60
Figura 5.23 GUI de la Ventana de comunicación PLC	61
Figura 5.24 GUI de la Ventana Secundaria de Visualización	62
Figura 5.25 GUI de la Ventana de comunicación ROS	63
Figura 5.26 resultado de la implementación de imagen en el Layout principal	64
Figura 5.27 resultado de la implementación de PLC en la GUI	64
Figura 5.28 resultado de implementación de imagen en la ventana secundaria de visualización	65
Figura 5.29 resultado de implementación de ROS en la GUI	65
Figura 6.1 Configuración de inicial con Webcam	69
Figura 6.2 Interfaz inicial con Webcam	70
Figura 6.3 Configuración final con GenICam	70
Figura 6.4 Interfaz final con GenICam	71
Figura 6.5 Configuración inicial de widgets del conector RS232	72
Figura 6.6 Interfaz inicial de los widgets del Layout principal	73
Figura 6.7 Configuración final de widgets del conector RS232	73
Figura 6.8 Interfaz final de los widgets del Layout principal	74
Figura 6.9 Configuración inicial de formato de textos	74
Figura 6.10 Interfaz inicial de formato de textos	75
Figura 6.11 Configuración final de formato de texto	75
Figura 6.12 Interfaz final de formato de textos	75

## 1. INTRODUCCIÓN

En este capítulo se resumen el contenido del trabajo, las definiciones necesarias para ser capaz de comprenderlo, y la motivación del proyecto.

Desde las máquinas informáticas más primitivas, hasta el superordenador más avanzado, el ser humano ha intentado aproximar el lenguaje que utilizamos con los ordenadores al suyo propio.

La evolución informática nos ha permitido, mediante la interacción Persona y Ordenador, entender y manipular cada vez mejor los secretos que guarda el universo informático. Este avance en las interfaces es el que permite, sin darnos cuenta, que tanto niños como personas de la tercera edad hayan dado un salto al uso de dispositivos informáticos. Esto lo confirma el incremento progresivo del uso de un dispositivo complejo que todos utilizamos en el día a día, el smartphone.

Las denominadas GUI (*Graphical User Interface*) son programas informáticos que actúan de interfaz para un operador o usuario, estas interfaces se componen de un conjunto de objetos denominados “widget” y en ocasiones imágenes para representar una información. La interfaz permite al usuario interactuar con la misma información que nos aporta, y brinda una serie de acciones disponibles para la manipulación de esta.

El objetivo de este proyecto es desarrollar una interfaz de usuario intuitiva para facilitar la interacción entre persona y el dispositivo o dispositivos a manejar, que sea polivalente en cuanto a varias funcionalidades, como por ejemplo, la visualización de datos, la gestión de información entrante y el seguimiento de la misma, así como la edición y actualización manual de dichos datos.

La comunicación es la acción consciente de intercambiar información entre dos o más participantes con el fin de transmitir o recibir información o mensajes. Para ello, los humanos utilizamos el lenguaje. Los estudios indican que el 55% de la comunicación se da por medio del lenguaje corporal, un 38% por el lenguaje y el 7% restante por el lenguaje oral, visual y escrito.

[1]

Uno de los objetivos de este TFG se basa en reforzar el 7% de este lenguaje facilitando un mensaje o información simple e intuitivo mediante el uso de una interfaz gráfica de usuario.

## **1.1. Contexto y motivación del proyecto**

Este proyecto está siendo apoyado por el centro de investigación Tekniker porque es de interés para el centro disponer de una interfaz aplicable a ciertas secciones y departamentos de la empresa.

Este sistema se implementará para que pueda interactuar con otras tecnologías, dispositivos, y protocolos. Con esa intención, se busca una interfaz genérica que acepte varios métodos de comunicación que sea tanto fácil de configurar como intuitivo de manipular.

### **1.1.1. Tekniker**

Tekniker es un centro de investigación tecnológico especializado en fabricación avanzada, ingeniería de superficies, ingeniería de producto y TICs para fabricación. Su misión es aportar crecimiento y bienestar a través de la I+D+i al conjunto de la sociedad, contribuyendo de manera sostenible a la competitividad del tejido empresarial.

Tekniker es una Fundación sin ánimo de lucro inscrita en el Registro de Fundaciones del país vasco con fecha 3 de Junio de 1996, bajo el número F 13 y clasificada en atención a sus fines en la sección 1º de dicho Registro, relativa a Fundaciones docentes y dedicadas a la investigación, a los efectos de lo dispuesto en el artículo 2 del Decreto 101/2007, de 19 de junio, que aprueba el reglamento del Registro de Fundaciones del País Vasco.

En Tekniker las disciplinas que se practican están agrupadas por unidades. La unidad para la que se hará el TFG es SAI (Sistemas Autónomos e Inteligentes). La unidad de SAI, a su vez, está compuesta por tres sub-disciplinas tecnológicas: Robótica, interacción y visión. A pesar de que todas estas sub-disciplinas sean distintas unas de otras, están totalmente ligadas, por lo que es común que durante los proyectos dentro de la unidad, incluido este, se trabaje en las diferentes sub-disciplinas.

### **1.1.2 Planteamiento del Problema y Motivación**

Tekniker y más específicamente, la unidad de SAI busca disponer de una interfaz de usuario que sea intuitiva, capaz de utilizar diferentes tipos de conexiones genéricas. Tekniker al tratar muchos campos tecnológicos, necesita disponer de soluciones de visualización de datos acorde con la cantidad de dispositivos utilizados en dichos campos tecnológicos, por ejemplo, cámaras industriales, térmicas, PLCs, sensores, etc ...

Tekniker propone que el TFG sea el contexto que permita el inicio del desarrollo de una solución a esa necesidad, consiguiendo así una base sólida, funcional y práctica que posterior a la finalización del TFG pueda seguir evolucionando.

## **1.2 Objetivos del proyecto**

Este proyecto se compone de los siguientes cinco objetivos principales: Creación de la interfaz, Creación de conectores, Creación de IWidgets, Documento de configuración e Implementación.

### **1.2.1 Creación de interfaz**

En este apartado el objetivo consiste en preparar la interfaz con la que el usuario tendrá que interactuar. Formar una estructura con un conjunto de “widgets” que facilitan la visualización de datos al usuario.

Las funcionalidades y requerimientos de la interfaz se describen más adelante.

### **1.2.2 Creación de conectores**

En este apartado el objetivo consiste en crear diferentes métodos de conexión entre los dispositivos y el programa, facilitando la comunicación entre el dispositivo y el desarrollador, que posteriormente agrega la información de dicha comunicación a la interfaz.

Las funcionalidades y requerimientos de los conectores se describen más adelante.

### **1.2.3 Creación de IWidgets**

IWidget hace referencia durante todo el proyecto al método de conexión entre la interfaz y el conector. El propósito de un IWidget se basa en recibir un mensaje tratado por los conectores, y aplicarlo a los diferentes “widgets” de la interfaz, de ahí su nombre.

En este apartado el objetivo consiste en crear un conjunto de IWidgets básicos, capaces de interactuar con los conectores principales y que sirvan de guía para la posterior creación y edición de nuevos IWidgets.

Las funcionalidades y requerimientos de los IWidgets se describen más adelante.

### **1.2.4 Documento de configuración**

En este apartado el objetivo consiste en crear un documento de configuración en base xml. Este documento facilitará al usuario la edición de varios apartados de la interfaz, así como de los conectores e IWidgets. El lenguaje xml permite una estructura de árbol similar al HTML, generando un formato intuitivo para la comprensión del documento.

Las funcionalidades y requerimientos del documento de configuración se describen más adelante.

### **1.2.5 Implementación**

En este último apartado el objetivo consiste en implementar un conjunto de dispositivos al proyecto. Dispositivos como: cámaras industriales, PLCs, etc... ayudarán a la verificación de las funcionalidades de los anteriores apartados, comprobando así, que el proyecto se desarrolla cumpliendo las planificaciones.

## **1.3 Planificación**

Para este apartado tendremos en cuenta la duración de la estancia en la empresa Tekniker en divisiones semanales (aproximadamente 1000 horas totales dedicadas al desarrollo del TFG).

A continuación se presenta un diagrama de gantt exponiendo las tareas necesarias para el desarrollo del proyecto.

### **1.3.1 Planificación del proyecto**

TAREAS	Feb				Mar				Abril				May				Jun				Jul			
	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	
Puesta a punto de Entorno	■																							
Prototipo config.xml		■																						
Crear Ventana principal			■																					
Diseño de Layout principal					■																			
Conektor Camara						■																		
Conektor ROS							■																	
Conektor RS232								■																
Desarrollo MVC									■															
Conectores IDS										■														
Conectores GenICam											■													
Conektor PLC											■													
Creacion IWidgets												■												
Implementar GUIs													■											
Implementacion Yapsy														■										
Actualizacion Config.xml															■									
Documentar dependencias																■								
Memoria TFG																	■							

Tabla 1.1 Planificación del proyecto

El proyecto ha sido desarrollado siguiendo la planificación de la Tabla 1.1. Al dividir el tiempo por semanas, se puede observar que la mayoría de las tareas requieren más de una semana para su realización, siendo común que varias tareas se realicen en paralelo. Esto se debe al continuo desarrollo de una tarea, que requiere de más tiempo para ajustarse a las necesidades de la siguiente.

### 1.3.2 Presupuesto

PROYECTO	PRECIO POR UNIDAD	UNIDAD	TOTAL
<b>Gestion de recursos</b>			
Camara Logitech Tессar 2.0/3.7	87,28 €	1	87,28 €
IDS Imaging U3-3890CP	612,00 €	1	612,00 €
IDS Imaging GV-5890CP	612,00 €	1	612,00 €
OPTOcam COE-032-M-POE-040-IR-C	599,00 €	1	599,00 €
Gobi+ 640	7.000,00 €	1	7.000,00 €
SIEMENS SITOP smart GEPI333-2BA01	200,00 €	1	200,00 €
WEIDMULLER UC20-WL2000-AC	396,45 €	1	396,45 €
SWITCH D-Link DGS-105	9,00 €	1	9,00 €
GLOBALSAT BU-353-S4 GPS	47,49 €	1	47,49 €
<b>Gestion de personal</b>			
Desarrollador principal	600 €/Mes	6 Meses	3.600,00 €
Sub-Total			10878,69
I.V.A. ( 21% )			2284,52
<b>TOTAL PRESUPUESTADO</b>			<b>13.163,22 €</b>

Tabla 1.2 Presupuesto del proyecto

En la tabla 1.2 figuran los recursos de valor utilizados en el proyecto. Dividiendo los apartados por la gestión de los recursos y la gestión del personal, podemos conseguir un valor aproximado del costo total utilizado en este proyecto.

El coste atribuido a la gestión de personal, es decir, al desarrollador principal, es considerado un pago mensual para un ingeniero en prácticas. Este factor se debería tener en cuenta, debido a que el salario medio de un ingeniero junior podría alcanzar los 2000 €/Mes [2], aumentando así el valor del total presupuestado.

La suma total del coste entre la gestión de recursos y la gestión del personal, asciende a la cifra de 13.163,22 €, y un sub-total bruto de 10878,69 €.

Varios de los productos utilizados en el proyecto han sido de uso meramente experimental, es decir, el proyecto trata de integrar indefinidas fuentes de información a una interfaz, fuentes como cámaras, GPS, PLCs, estaciones meteorológicas, conexiones USB, conexiones Ethernet, etc. Por esa razón, algunos productos utilizados en el proyecto son para verificar el desarrollo adecuado de ciertas aplicaciones para fuentes de información específicas, lo que significa que a medida que se añadan aplicaciones al proyecto, se deberán utilizar nuevos productos y dispositivos, haciendo así, un incremento en el total presupuestado del proyecto.

Se concluye que 13.163,22 € es un total mínimo para el desarrollo del proyecto, que a su vez, incrementa el coste a medida que se añaden nuevos dispositivos a la aplicación.

## **1.4 Análisis de riesgos y viabilidad**

### **1.4.1 Análisis de riesgos**

Los mayores riesgos que se tienen en cuenta son los recogidos en la siguiente tabla.

PROBLEMA	RIESGO	IMPACTO
Falta de tiempo	Medio	Alto
Enfermedad o Baja	Bajo	Alto
Pérdida de datos	Bajo	Medio
Hardware dañado	Bajo	Alto
Saturación de la transferencia de datos	Alto	Alto

Tabla 1.3 Tabla de análisis de riesgos

Con el propósito de reducir los riesgos presentados en la Tabla 1.3, aplicaremos las siguientes medidas:

1. Planificación y realización de reuniones periódicas para medir correctamente el tiempo de cada tarea.
2. Cumplimiento de las medidas y protocolos de seguridad y sanidad para prevenir el COVID-19.
3. Copias de seguridad periódicas mediante el software de control de versiones GitLab.
4. Debido uso del material, evitar sobrecalentamientos y su uso continuo prolongado.
5. Planificación de la estructura de gestión de datos dentro del programa, así como evitar el uso indebido de multithreading y el uso de frecuencias de actualización similares (obstrucción de flujo).

### **1.4.2 Análisis de viabilidad**

Teniendo en cuenta el análisis de riesgo, la planificación temporal y que el TFG se realizará en Tekniker, un centro de investigación y tecnología puntera, con profesionales en varias

disciplinas tecnológicas dispuestos a ayudar, podemos concluir que el proyecto es total y absolutamente viable dentro del tiempo previsto y cumpliendo los principales objetivos definidos.

## **2. ESTADO DEL ARTE**

En este apartado se recogen las investigaciones documentales necesarias para llevar a cabo el proyecto. Se realiza un análisis de las tecnologías, servicios, softwares y módulos que serán útiles para el desarrollo del proyecto, con la finalidad de entender lo que en el pasado se consiguió hacer, y lo que en el futuro se podrá lograr.

### **2.1 ROS**

Sistema Operativo Robótico (en inglés Robot Operating System, ROS) es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo.[\[3\]](#)

Gran parte de los desarrollos robóticos que se realizan en Tekniker utilizan ROS como sistema base. Es necesario, por tanto, que el interfaz desarrollado en el TFG sea capaz de comunicarse con ROS.

Para entender el uso de ROS durante el desarrollo del proyecto y el TFG, es importante definir los siguientes conceptos:

- *Nodo*: un nodo es un proceso que realiza cómputo, los nodos pueden comunicarse entre ellos a través de topics.
- *Roscore*: es una colección de nodos y el proceso encargado de que se comuniquen entre ellos correctamente.
- *Topic*: los topic son buses con nombre encargados de que se transmitan los mensajes del emisor al receptor.
- *Messages*: un *message* es una estructura de datos simple, es el contenido de lo que se envían los nodos entre ellos a través de los topics. Estos mensajes pueden ser de cualquier tipo, “int”, “strings”, imágenes, bytes, etc...
- *Publisher*: es la parte del nodo encargada de publicar (enviar) mensajes en un topic concreto.

- *Subscriber*: es la parte del nodo encargada de suscribirse a un topic concreto y recibir los mensajes.

## **2.2 OPC UA**

Open Platform Communications Unified Architecture se trata de un estándar extensible e independiente de la plataforma que permite el intercambio seguro de información en sistemas industriales. El objetivo de OPC UA es mejorar la responsabilidad entre hardware y software para la automatización.[\[4\]](#)

OPC UA se basa en un framework de comunicación cliente-servidor.

Un servidor proporciona a un cliente aplicaciones y sistemas de control, y acceso seguro a datos de automatización industrial utilizando modelos de información que especifican la forma en que se organizan, almacenan y recopilan dichos datos. El servidor puede ser un servidor virtual, como el que usaremos.

Un cliente puede soportar un modelo de información OPC UA. Los clientes solicitan y escriben datos en los componentes de un sistema a través de los servidores OPC UA.

## **2.3 OpenCV**

OPEN Computer Vision es una biblioteca de código abierto que contiene implementaciones que abarcan más de 2500 algoritmos. Además, está especializada en el sistema de visión artificial y machine learning.[\[5\]](#)[\[6\]](#)

Su utilización principal abarca la detección de objetos y rostros, especialmente en aspectos como la seguridad, el marketing o incluso la fotografía.

Algunos ejemplos de aplicaciones de OpenCV son: Detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes, etc...

## **2.4 GigE Vision, GenICam**

GigE Vision es un interfaz estándar para cámaras industriales de alto rendimiento. Proporciona un framework para la transmisión de video de alta velocidad y control de datos sobre conexiones Ethernet. [\[7\]](#)[\[8\]](#)

GenICam es una interfaz de programación genérica para cámaras de visión artificial. El objetivo del estándar es desacoplar la tecnología de las interfaces de cámaras industriales de la interfaz de programación de aplicaciones del usuario.

Ambas interfaces proporcionan un conjunto de funcionalidades básicas: Configuración de cámara, captura de imágenes, GUI de interacción con la cámara, transmisión de datos extra y diferentes interacciones con eventos.

## **2.5 Qt**

Qt es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.[\[9\]](#)

Con Qt, las GUI se pueden escribir directamente en C++ usando su módulo Widgets. Qt también viene con una herramienta gráfica interactiva llamada Qt Designer que funciona como un

generador de código para las GUI basadas en Widgets. Qt Designer se puede usar independiente pero también está integrado en Qt Creator.

## **2.6 Bibliotecas de código para interfaces gráficas**

**PyQt:** PyQt es uno de los frameworks de Qt para Python más populares. Contiene más de 620 clases que cubren tanto interfaces gráficas de usuario, manipulación de XML, comunicación con la red y muchas tecnologías referentes a Qt. [\[10\]](#)

**PySimpleUI:** PySimpleGUI es un contenedor de la API de Python del módulo Tkinter que permite al programador utilizar los mismos elementos de la interfaz de usuario que con Tkinter pero con una interfaz más intuitiva. PySimpleGUI se estableció con el propósito de crear un proceso de desarrollo de la GUI de Python más fácil de usar, y agrega la capacidad de definir diseños personalizados. [\[11\]](#)

**PySide:** PySide es una biblioteca para Python que contiene todas las herramientas de interfaz gráfica de usuario de Qt. Es una de las alternativas para programar GUI en Python, en vez de usar Tkinter el cual es usado por Python. [\[12\]](#)

PySide comparte funcionalidades con PyQt, pero aporta unas utilidades más intuitivas para la manipulación de interfaces mediante el Signal y Slot. Además, es la biblioteca oficialmente soportada por la compañía Qt. Lo que lo hace la mejor opción para el desarrollo del proyecto.

### **3. RECURSOS DEL PROYECTO**

En este capítulo se describe la colección de Hardware y Software que se va a utilizar en el proyecto.

#### **3.1 Hardware**

Para la realización de este TFG, Tekniker ha proporcionado el siguiente conjunto de hardware, que facilitarán la prueba y verificación de la funcionalidad del proyecto.

##### **3.1.1 Cámaras**

###### **3.1.1.1 Camara Logitech QuickCam Pro 9000**

La cámara Logitech QuickCam es una cámara RGB, capaz de percibir imágenes en color. Conexión mediante USB 3.0 y USB 2.0. Este tipo de cámaras es normalmente usado para aplicaciones de webcam. [\[13\]](#)



Figura 3.1 Logitech QuickCam pro 9000

###### **3.1.1.2 IDS Imaging U3-3890CP**

Esta es una cámara desarrollada por la empresa IDS, entre sus características se encuentran una calidad de imagen de 12 MegaPíxeles, un rango de colores monocromático y un bajo consumo de 1,8 Watts. Conexión mediante USB 3.0 Tipo B. Este tipo de cámaras están especialmente diseñadas para su uso industrial. [\[14\]](#)



Figura 3.2 IDS Imaging U3-3890CP

### **3.1.1.3 IDS Imaging GV-5890CP**

Se podría decir que este es el hermano gemelo de la anterior mencionada IDS Imaging U3-3890CP. [\[15\]](#) Comparten todas las características, a excepción de una. Esta cámara usa una conexión vía Ethernet, que le permite usar el protocolo GigE Vision, para una transferencia de video de alta velocidad.

El uso de esta variación de modelo nos permite verificar la compatibilidad de diferentes métodos de conexión para un mismo dispositivo de modelo similar.



Figura 3.3 IDS Imaging GV-5890CP

### **3.1.1.4 OPTOcam COE-032-M-POE-040-IR-C**

Si hablamos de cámaras industriales, las que proporciona la empresa OPTO Engineering [\[16\]](#) destacan ante las demás. Esta en particular es un miembro estándar de la serie COE-G, con una calidad de imagen de 3.1 MegaPixel y un rango monocromático.

La especialidad de la serie COE-G se basa en la conectividad GigE de alto frame-rate, haciendo posible la captura de la máxima cantidad de fotogramas posibles.



Figura 3.4 OPTOcam COE-032-M-POE-040-IR-C

### **3.1.1.5 Gobi+ 640**

Esta es la cámara térmica que se utilizara durante el proyecto, la cual viene de la mano de Xenics y sus serie de cámaras infrarrojas de onda larga.

Con un framerate de 60Hz es capaz de dar una imagen con valores cromáticos equivalentes a la medición de temperatura del rango de visión de la cámara.

Con una sensibilidad de error de 50 mK es capaz de medir amplios rangos de temperatura ( $\Delta 200$  K).

Gracias a estas características, la Gobi+ 640, en todos sus modelos, es usada en calibración térmica para aplicaciones termográficas.

Conección compatible con GigE Vision.



Figura 3.5 Gobi+ 640

### **3.1.2 SIEMENS SITOP smart GEPI333-2BA01**

SITOP Smart es una potente fuente de alimentación estándar para redes monofásicas y trifásicas. Esta en específico admite de entrada: AC 120/230V y salida: DC 24V, pudiendo soportar el 120% de la potencia nominal de forma permanente. [\[17\]](#)

Usaremos el SITOP junto al siguiente PLC.



Figura 3.6 SIEMENS SITOP smart GEPI333-2BA01

### **3.1.3 WEIDMULLER UC20-WL2000-AC**

Este es un controlador compacto de la mano de u-mation. Basado en web, el PLC es una herramienta de ingeniería integrada en web mediante u-create web para sistemas en tiempo

real y aplicaciones IoT. Su diseño modular permite la ampliación de la instalación para incluir otros componentes de la misma casa de productos. [\[18\]](#)

En el proyecto usaremos su servidor web para la previa configuración de variables y programación.



Figura 3.7 WEIDMULLER UC20-WL2000-AC

### **3.1.4 SWITCH D-Link DGS-105**

Este dispositivo es un switch de 5 puertos de 10/100/1000 Mbps. Tiene la funcionalidad de poder conectar hasta 5 puertos ethernet a la vez, lo que nos es útil ya que trabajamos con varias cámaras que transfieren datos mediante el protocolo GigE, que tiene una conexión directa con ethernet, al igual que el PLC.



Figura 3.8 SWITCH D-Link DGS-105

### **3.1.5 GLOBALSAT BU-353-S4 GPS**

El BU-353-S4 es un receptor GPS vía USB. Es altamente sensible y de bajo consumo, proporciona mayor rendimiento en zonas urbanas y en follaje denso. [\[19\]](#)

Para la transferencia de data utiliza el protocolo NMEA [\[20\]](#)[\[21\]](#), lo que le permite codificar la información en cuatro listas de datos: GGA (Datos fijos del sistema de posicionamiento global), GSA (Satélites activos), GSV (Satélites a la vista) y RMC (Datos de GPS específicos mínimos recomendados). Para el proyecto utilizaremos este último (RMC), que nos proporciona los datos de geoposicionamiento básicos, así como la latitud, longitud y velocidad de desplazamiento.



Figura 3.9 GLOBALSAT BU-353-S4 GPS

### **3.2 SOFTWARE**

Para la realización del proyecto, y durante el desarrollo del mismo, se ha determinado el uso del siguiente conjunto de software, que será utilizado en sincronía con el sistema operativo linux, Ubuntu 20.04, aunque será compatible con otros como Windows y macOS.

#### **3.2.1 Python 3.8.10**

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, ha sido seleccionado como lenguaje principal del código por su compatibilidad con distintos servicios, sistemas y numerosas librerías open source de las que disponen los usuarios.

La versión, aunque no es la más reciente, es la más óptima para sacarle el máximo provecho a varias librerías que serán utilizadas durante el desarrollo del proyecto, las cuales presentan algún error o fallos al implementarse en versiones más novedosas.

### **3.2.1.1 PIPENV (Entornos virtuales)**

Pipenv es una herramienta asociada a python que automáticamente crea y maneja un entorno virtual para tus proyectos, así como instalar paquetes o librerías y desinstalarlas desde un archivo “*pipfile*”, el cual almacena información sobre los paquetes y librerías agregadas al entorno virtual. [\[22\]](#)

También genera lo más importante, “*pipfile.lock*”, que se encarga de usar la información del archivo “*pipfile*” para descargar e instalar paquetes para el entorno virtual, haciendo así posible la principal causa por la que usamos entornos virtuales, permitir el uso de aplicaciones clonadas de nuestro proyecto original en otros dispositivos, sin la necesidad de preinstalar manualmente ningún requisito.

Estos son los paquetes y librerías más importante que se usarán en el proyecto:

- *PySide2*: Permite el uso de funciones asociadas a Qt en Python.
- *Rospy2*: permite el uso de funciones asociadas a ROS.
- *Pynmea2*: permite el uso de funciones asociadas al protocolo NMEA.
- *opcua*: permite el uso de funciones asociadas a OPC UA.
- *pillow*: permite el uso de funciones asociadas a la trata de imágenes y video, útil para editar los parámetros de las imágenes a nuestra necesidad.
- *opencv*: permite el uso de funciones asociadas a OpenCV, para el tratado de imágenes y videos en vivo.

- *pyueye*: permite el uso de funciones asociadas a uEye, para el tratado de imágenes de las cámaras IDS.
- *yapsy*: permite la implementación de plugins a nuestro proyecto.
- *numpy*: permite la creación y edición de vectores y matrices multidimensionales, junto con una colección de funciones matemáticas para operar con ellas. Útil para la edición de matrices creadas para codificar la información de las imágenes.
- *xmltodict*: permite la transcripción de los archivos estructurados en XML, convirtiendo el texto XML en un formato diccionario propio de python, lo que facilita la lectura y la búsqueda de datos del archivo XML.

### **3.2.2 ROS**

Para una completa información sobre ROS, véase el apartado “*2.1 ROS*”.

En el proyecto se utilizará la terminal de Ubuntu para publicar o recibir mensajes (De acuerdo con las necesidades) para así poder verificar la compatibilidad de la interfaz y ROS en una transferencia de datos de alta frecuencia, validando así, la futura conexión a un sistema robótico.

### **3.2.3 OPC UA**

Para una completa información sobre OPC UA, véase el apartado “*2.2 OPC UA*”.

En este proyecto se utilizará OPC UA para una conexión directa entre cliente-servidor, en el que el servidor, será virtual, para la edición y visualización de datos que nos proporciona el anterior hardware mencionado de la empresa Weidmüller(PLC), así como la suscripción de nodos para ser capaces de recibir esos datos, y actualizarlos con cierta frecuencia.

#### **3.2.3.1 UA Expert**

UA Expert funciona como un cliente experimental de OPC UA, un framework diseñado generalmente para la prueba de clientes OPC UA con todas las características que la SDK de OPC UA ofrece. [\[23\]](#)

Las funcionalidades más importantes son: Data Access View (Visualiza una tabla con nodos, exponiendo ciertos datos para cada uno de ellos), Attribute/Value (Visualiza los atributos y el valor de un nodo elegido, este puede ser editado), Address Space (Visualiza la estructuración de datos dentro del cliente).

Estas funcionalidades nos serán de utilidad para verificar el funcionamiento de nuestro propio cliente de OPC UA, según lo previsto, el estado y valor de los nodos modificados en nuestro cliente, debería de ser duplicado automáticamente en el Data Access View de UA Expert.

### **3.2.4 OpenCV**

Gracias a su extenso abanico de funcionalidades dirigidas a la manipulación de imágenes y videos en tiempo real, OpenCV será útil para gestionar la información de imágenes que recibimos de diferentes conectores, tanto para la propia recepción, como para la aplicación de diversas operaciones como pueden ser la redimensionación o la conversión de color. La mayoría de cámaras necesitarán ser tratadas por OpenCV.

### **3.2.5 Aravis**

Aravis es un software gratuito disponible en un repositorio de gitlab [\[24\]](#), desarrollado por usuarios de esta. Contiene unos archivos de código que permiten al usuario la manipulación de datos recibidos de las cámaras, así como lo hace OpenCV.

Algunos protocolos como GenICam rehusan al método de OpenCV para la captura de video, es por eso que aravis nos resultará útil. Aravis y OpenCV no interfieren, por lo que, es posible el

uso de Aravis para gestionar los datos de fotogramas y unificar el formato de imagen final al que propone OpenCV.

### **3.2.6 Qt**

#### **3.2.6.1 PySide2**

PySide2 es el módulo oficial de Python sobre Qt, sirve de enlace para las herramientas de interfaz gráfica de usuario de Qt. Es la mejor alternativa para programar GUI en Python gracias a toda la documentación publicada, la facilidad de crear y añadir atributos a widgets/páginas, y la intuitiva forma de implementarlo y manipularlo en código

#### **3.2.6.2 Qt Designer**

Qt Designer [\[25\]](#) es una herramienta de Qt para diseñar y crear interfaces gráficas de usuario. Permite diseñar y customizar ventanas usando QtWidgets, de la manera (como ellos lo llaman) “*lo que ves es lo que hay*”, la cual es una forma intuitiva de entender el propósito de cada widget sin tener que verificar continuamente la documentación.

Todos los widgets se integran perfectamente con el código de programación. Utilizando el mecanismo de “Signal y Slot”, podemos asignar diferentes comportamientos a los elementos gráficos, así como a otras partes independientes de la interfaz en el código.

Este mecanismo se basa en enviar una “Signal” en el momento en el que una causa específica ocurra, inmediatamente y en cualquier parte del programa el “Slot” asociado a la “Signal” se ejecuta, siendo los Slots funciones específicas creadas por el desarrollador, o las que predefinidamente ofrece Qt. Este mecanismo nos va a ser muy útil en el desarrollo de muchas funciones.

Todas las propiedades y atributos asignados en Qt Designer pueden cambiarse dinámicamente en código, así como cargar el archivo “.ui” creado por Qt Designer (Este archivo contiene la GUI) directamente desde el código y manipularlo usando una instancia del archivo cargado, no teniendo que implementar manualmente toda la interfaz en el código fuente.

### **3.2.7 PyCharm**

PyCharm es un IDE (entorno de desarrollo integrado) utilizado en programación especialmente para Python. Proporciona análisis de código, integración con sistemas de control de versiones, por ejemplo Gitlab, y soporta el desarrollo web con Django, así como la ciencia de datos con Anaconda. [\[26\]](#)

### **3.2.8 Gitlab**

Gitlab es un editor de control de versiones y desarrollo de software colaborativo basado en Git. Se usará para hacer control de versiones y gestionar las copias de seguridad que se van a realizar a lo largo del proyecto.

### **3.2.9 Google Docs**

Google Docs es un procesador de texto en línea, el TFG y otros documentos auxiliares se redactarán usando esta herramienta

### **3.2.10 Text Editor**

Text editor actúa como un block de notas en Ubuntu, y permite la escritura en varios lenguajes de programación. Se usará para la creación y edición del archivo de configuración en formato XML.

## **4. REQUERIMIENTOS DEL SISTEMA**

### **4.1 Funcionalidades**

El objetivo fundamental del proyecto es ofrecer al usuario una interfaz gráfica que facilite la comunicación persona-máquina de forma intuitiva, así como brindar al usuario la capacidad de añadir nuevas funcionalidades, conectores y GUI.

Dicho objetivo resume, a alto nivel, las particularidades de la aplicación. Sin embargo, para lograr este objetivo, se habrán de centrar las funcionalidades y requisitos en satisfacer una serie de objetivos más básicos que nos sirvan para alcanzar el objetivo final.

- En primer lugar, facilitar una interfaz básica donde el usuario podrá ver el contenido que él/ella quiera.
- En segundo lugar, permitir la edición de dicha interfaz libremente a decisión del usuario.
- En tercer lugar, brindar al usuario con unas muestras y/o ejemplos básicos para facilitar el uso y la edición de la interfaz.
- Por último, se requerirá que el programa sea lo más genérico posible, intentando que el usuario no tenga dificultades para su comprensión y posterior edición o implementación del mismo.

Para lograr estos objetivos de la manera más adecuada, se han definido distintos subsistemas en la aplicación que detallaran las funcionalidades necesarias.

### **Gestión de interfaz**

En este subsistema se agrupan las funcionalidades relacionadas con la interfaz y lo que el usuario va a ver en la pantalla, así como su estructura.

### **Gestión de IWidgets**

Este subsistema se encargará de los denominados “IWidget”, y de su funcionalidad de puente entre los conectores y la interfaz/GUI.

### **Gestión de conectores**

Este subsistema se ocupa de la gestión de los conectores, y las funcionalidades básicas para que se efectúe el envío y la recepción de la información.

### **Gestión de configuración**

Este apartado se ocupa de la estructura del archivo de configuración, las funciones y necesidades básicas que debe cumplir.

### **Gestion de documentacion y datos**

En este subsistema se agrupan tanto la documentación del programa como los datos archivados referentes a la información recibida por los conectores.

#### **4.1.1 Requisitos funcionales**

##### ***Gestión de interfaz***

- **RF-01** La interfaz será intuitiva y tendrá elementos de poca complejidad.
- **RF-02** La interfaz dispondrá de una gestión de pestañas que permiten el uso de varias páginas.
- **RF-03** Ciertas pestañas tendrán que disponer de interactividad con el usuario.
- **RF-04** El diseño se basará en una estructuración minimalista, con una gama de colores oscuros con intención de no saturar la recepción lumínica del usuario.
- **RF-05** La ventana principal deberá ser libremente redimensionable.

##### ***Gestión de IWidgets***

- **RF-06** Todos los IWidgets tienen que servir de puente entre los conectores y la interfaz.
- **RF-07** Todos los IWidgets tienen que implementar la información en el widget destinado por el usuario.
- **RF-08** Los IWidgets deben de permitir la implementación de varios tipos de información.

- **RF-09** Habrá un IWidget base diseñado como modelo para permitir la creación de una réplica a nivel estructural.
- **RF-10** Todos los IWidgets tienen que estar optimizados para ser lo más usuario-independiente posible.

### **Gestión de conectores**

- **RF-11** Los conectores tienen que gestionar la entrada de datos.
- **RF-12** Los conectores serán optimizados para ser capaces de gestionar diferentes dispositivos (siempre que sea posible).
- **RF-13** Habrá un conector base diseñado con las funciones que deben compartir todos los conectores.
- **RF-14** En caso de ser necesario, un conector tiene que permitir el “Parseo” del “mensaje” recibido.
- **RF-15** En caso de ser necesario, un conector tiene que permitir el “Loggeo” del “mensaje” recibido.
- **RF-16** En caso de ser necesario, un conector tiene que permitir modificar el formato del mensaje.
- **RF-17** Todos los conectores tienen que tener un protocolo de emisión del “mensaje”.

### **Gestión de configuración**

- **RF-18** El archivo de configuración tiene que ser intuitivo, con una estructura sencilla.
- **RF-19** La estructura debe dividirse en diferentes apartados de subsistemas.
- **RF-20** Los diferentes subsistemas tendrán una estructura ramificada para mejorar la comprensión y la navegación dentro del archivo.
- **RF-21** El archivo debe contener cada aspecto editable del programa, quitando algunas excepciones.

### **Gestion de documentacion y datos**

- **RF-22** Los archivos de documentación guardan datos e información útil.
- **RF-23** Unos archivos serán dedicados al almacenamiento de datos de cada conector (mediante “loggers”), para su postprocesamiento.
- **RF-24** Un archivo recogerá información dedicada a la explicación del programa e instrucciones para la edición del mismo.

#### **4.1.2 Requisitos no funcionales**

##### **Rendimiento**

- **RNF-01** El sistema tendrá la capacidad de soportar múltiples conexiones simultáneas.
- **RNF-02** El sistema deberá poder gestionar el acceso a los recursos en distintos dispositivos.
- **RNF-03** El sistema deberá tener una respuesta rápida para mostrar la información al usuario en el menor tiempo posible.

##### **Fiabilidad**

- **RNF-04** La aplicación deberá contar con un sistema de notificación para informar al usuario de errores y advertencias que se produzcan en el programa.
- **RNF-05** Para no interrumpir la operatividad del sistema, se debe plantear un flujo de trabajo para desarrollar mejoras y reparar errores mientras el programa está en uso.

## 5. DISEÑO Y ARQUITECTURA DE LA SOLUCIÓN

En este apartado se explica el diseño y la arquitectura del proyecto, la cual dividiremos en dos subapartados. En el primer subapartado se definirán el entorno y la estructura superficial del resultado que queremos conseguir, un diseño de alto nivel que explicará la correlación de los subsistemas a un nivel básico. A continuación, el segundo subapartado definirá el funcionamiento particular de los propios subsistemas, haciendo hincapié en los métodos y procesos utilizados a nivel de código para llegar al resultado deseado.

### 5.1 Diseño de alto nivel

A continuación se muestra un diagrama de clases con la estructura básica necesaria para definir el entorno del software:

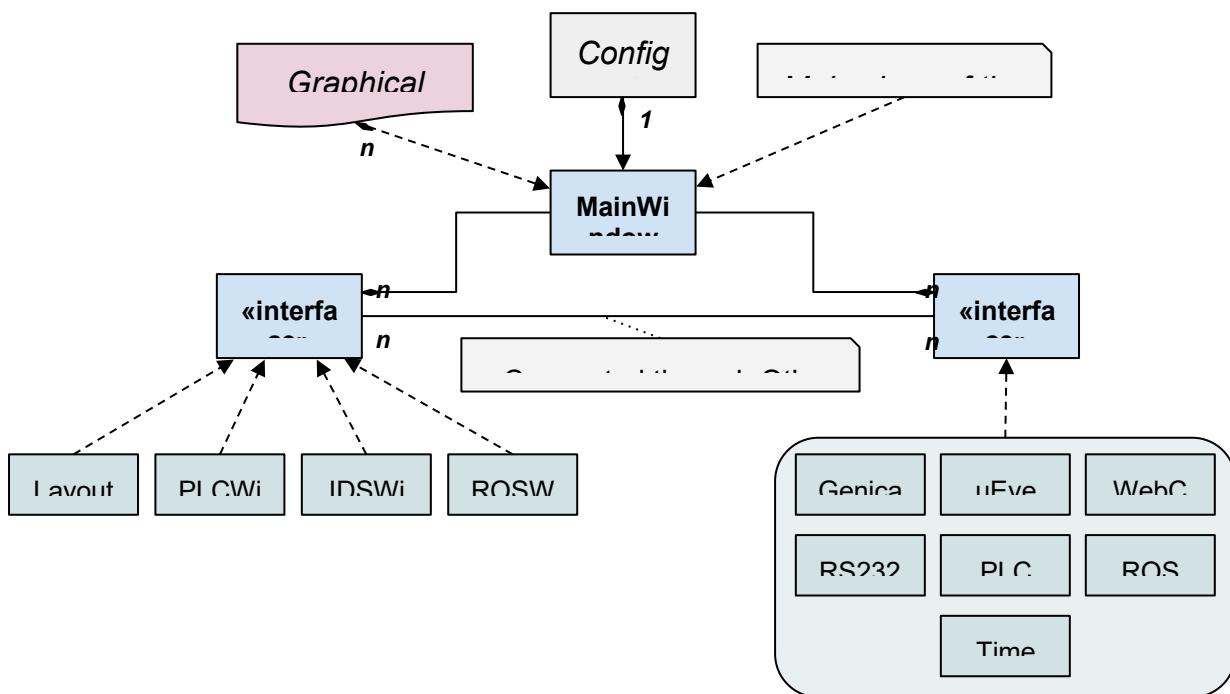


Figura 5.1 Diagrama de clases de la arquitectura del entorno del programa

Como se puede ver en la figura 5.1, el sistema está formado por:

- Una clase **MainWindow**, encargada de recoger y procesar información de otras clases, es la clase central del sistema.
- Un solo **archivo de Configuración**, implementado en *MainWindow* para gestionar la configuración de algunos aspectos relativos al sistema.
- Un conjunto de “n” cantidad de **GUI**, archivos “.ui” con una interfaz en bruto, implementados en *MainWindow* para la gestión de las interfaces.
- Un módulo “**interface IWidget**” utilizado como clase base para la implementación de “n” cantidad de **IWidgets** específicos, es un modelo a seguir para la creación de los mismos.
- Un conjunto de “n” cantidad de **IWidgets** derivados de la interfaz “**IWidget**”, conectados con el conjunto de **conectores** y *MainWindow*.
- Un módulo “**interface IConnector**” que se encarga del procesamiento básico de la información recogida de los **conectores**, aplicando “**parser**”, “**logger**” y un formato a los mensajes de texto. También utilizado como un nodo de conexión entre *MainWindow*, un **IWidget** y los **conectores**.
- Un conjunto de “n” cantidad de **Conectores** capaces de recibir datos e información de dispositivos o servidores ajenos al sistema.
- Por último, un módulo **Main** que contiene el programa de ejecución principal (“main”).

Una vez definidos los aspectos que conforman el entorno, se explicará el funcionamiento de las partes principales del sistema a una escala superficial.

Comenzando por el *MainWindow*, el cual cargará inicialmente una ventana principal, la cual seguirá la idea inicial del desarrollo. Esta ventana se constituirá principalmente por pestañas, a las cuales se añadirán sus respectivas GUIs, en función de la configuración por parte del usuario.

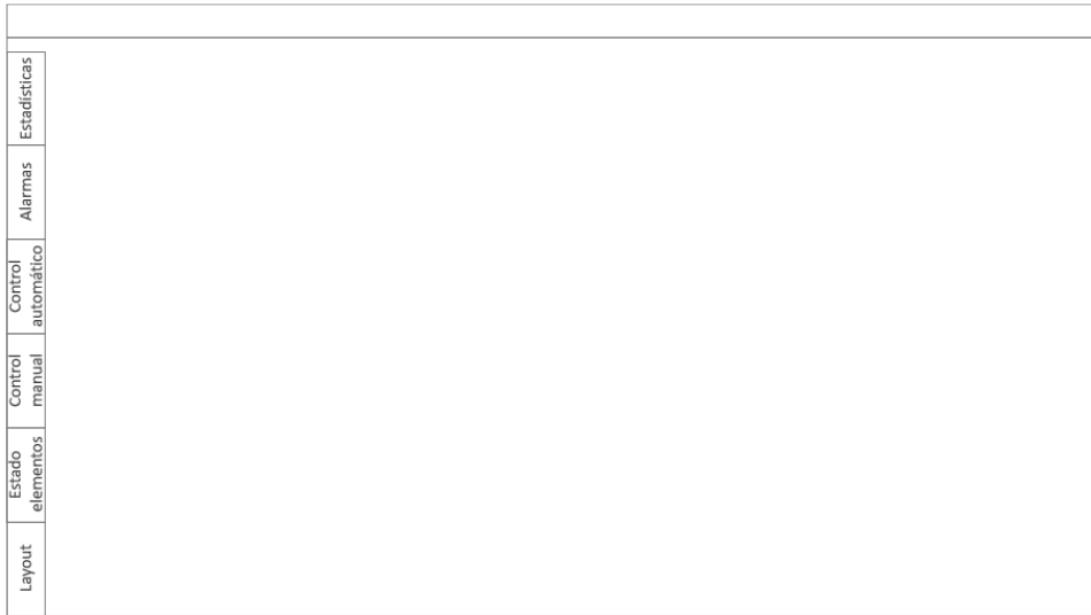


Figura 5.2 Prototipo inicial de ventana principal

A su vez, MainWindow tendrá constancia de los diferentes conjuntos de IWidgets y Conectores que el usuario decida utilizar. Mientras MainWindow se encarga de gestionar el archivo de configuración y diseñar la estructura de la ventana, los conectores empiezan a recibir mensajes de sus respectivos emisores.

Para entender el diseño de los conectores, se explicará su funcionalidad respecto al entorno del sistema:

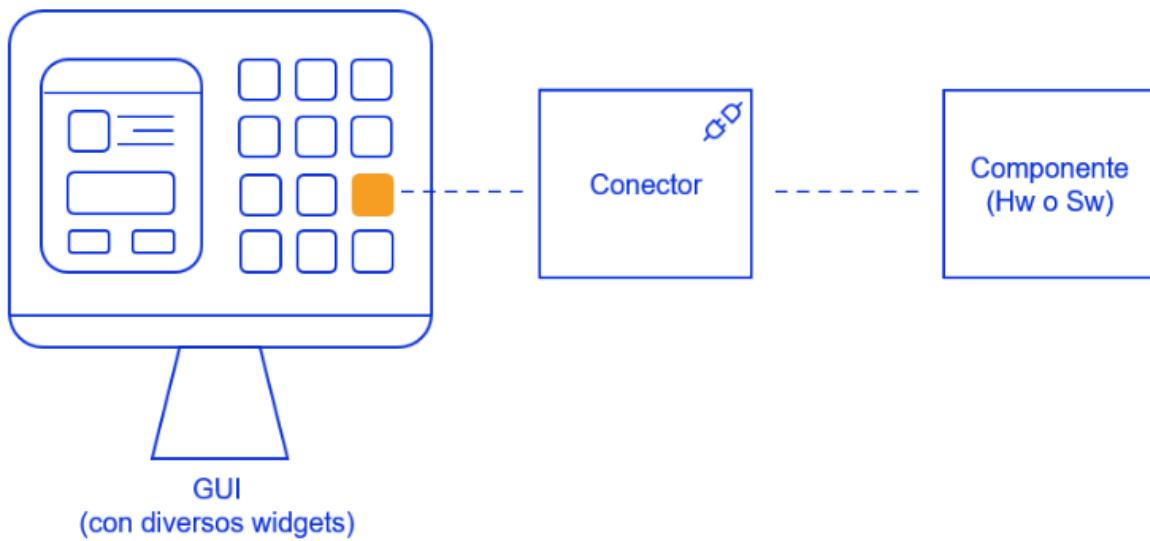


Figura 5.3 Representación de un conector sobre la GUI

Un conector permite crear un flujo de datos entre un componente y un widget dentro de la interfaz gráfica.

El componente a conectar puede ser tanto un elemento hardware (ej. un sensor) como un elemento software (ej. un topic de ROS).

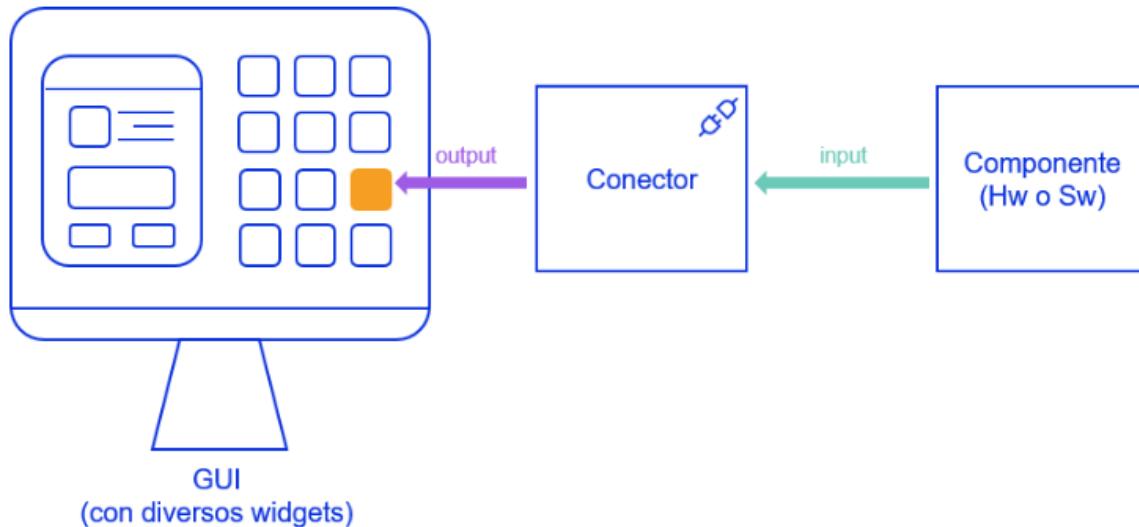


Figura 5.4 Representación de conexión del conector para recibir datos de un componente y mostrarlas en la GUI

Un conector puede utilizarse para mostrar datos provenientes de un componente en un widget de la interfaz gráfica (ej. Mostrar la imagen de una cámara).

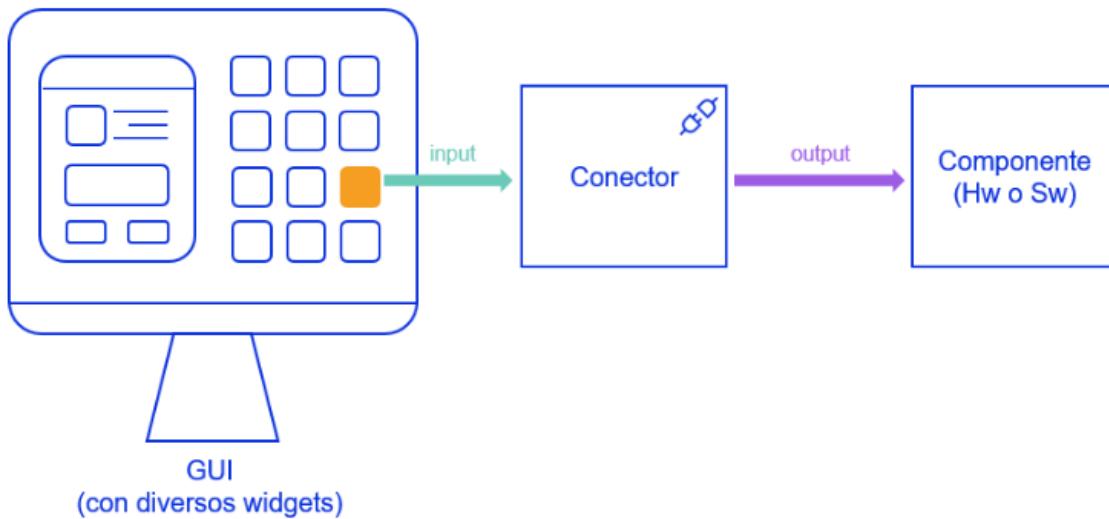


Figura 5.5 Representación de conexión del conector para enviar datos incluidos en la GUI a un componente  
De igual manera, un conector también puede utilizarse para enviar datos a un componente cuando ocurre un evento en un widget de la interfaz gráfica (ej. Cambiar el estado de una variable de un PLC al pulsar un botón).

Por la naturaleza de los conectores, que reciben datos de distintos componentes con distintos protocolos de envío de mensajes, el conector es capaz de gestionar el tipo de datos que recibe de ese componente. Para ello, este hace uso del módulo “*interface IConnector*”, el cual aplica un proceso a los datos recibidos para facilitar la emisión del mensaje, y su próxima implementación.

La siguiente figura representa estructuralmente la funcionalidad de este módulo:

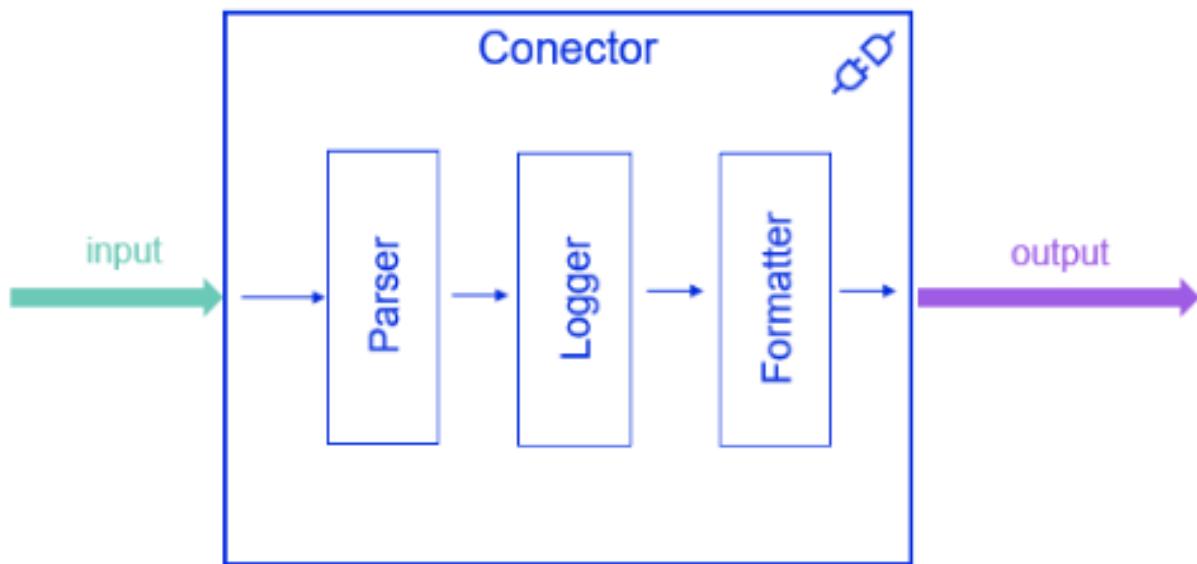


Figura 5.6 Representación estructural del módulo *IConnector*

Como se puede ver en la figura 5.6, “*interface IConnector*” está compuesto por un componente “*parser*”, un componente “*logger*”, y un componente “*formatter*”.

- **Parser:** Este componente analiza los datos de entrada para determinar su estructura lógica.  
Ejemplo: podemos utilizar el conector rs232 para obtener los datos transmitidos por un sensor GPS. Esos datos, sin embargo, vienen en bruto siguiendo el protocolo de comunicación NMEA 0183. Implementando un parser a este protocolo, y combinándolo con el conector, es posible dar como salida la información de interés del sensor GPS (ej. longitud, latitud, velocidad).
- **Logger:** Este componente permite generar registros (logs), lo cual puede resultar fundamental para el mantenimiento de un equipo.

Ejemplo: podemos crear un logger que tome como entrada una imagen y lo guarde en un directorio de disco. En combinación con un conector de webcam. Esto nos puede permitir capturar en disco imágenes. Otro ejemplo: podemos crear un logger que guarde en un fichero CSV el mensaje de entrada. Combinando este logger con el conector rs232 y el parser de NMEA 0183, podemos almacenar en un fichero CSV los datos de un sensor GPS para su posterior análisis.

- **Formatter:** Este componente permite dar formato a un mensaje de entrada. Está pensado solo para mensajes de tipo texto.

Ejemplo: El parser de NMEA 0183 devuelve un mensaje de tipo diccionario con el siguiente formato: {long: 23.3423, lat: 1.5234, speed: 4.12}. Nosotros queremos mostrarlo así:

GPS data: Latitud: 1.5234, Longitud: 23.3423

Velocidad: 4.12

Esto es posible gracias a la configuración previa del usuario, que define qué valores tomar, y en qué formato de texto aplicarlos.

## 5.2 Diseño de bajo nivel

### 5.2.1 Primera versión del sistema

En este apartado se describe el desarrollo del sistema principal, un sistema enfocado puramente en el planteamiento inicial de la solución, optimizado para que sea el resultado que aporte la solución correcta al entorno diseñado.

Para describir completamente el funcionamiento y el desarrollo del sistema, la solución de este se dividirá en distintos subsistemas, con el fin de profundizar en el desarrollo del código.

### 5.2.1.1 Backend

Backend es un término usado en informática y en diseño de software, más conocido en el mundo del desarrollo de aplicaciones web. Básicamente el Backend es la parte de un software que procesa la entrada desde el interfaz de usuario (frontend), así como la arquitectura interna del sistema.

En este apartado se describirán a nivel de código el desarrollo del Backend, dividido en conjuntos de “scripts” que tienen funcionalidades en común.

#### Programa principal

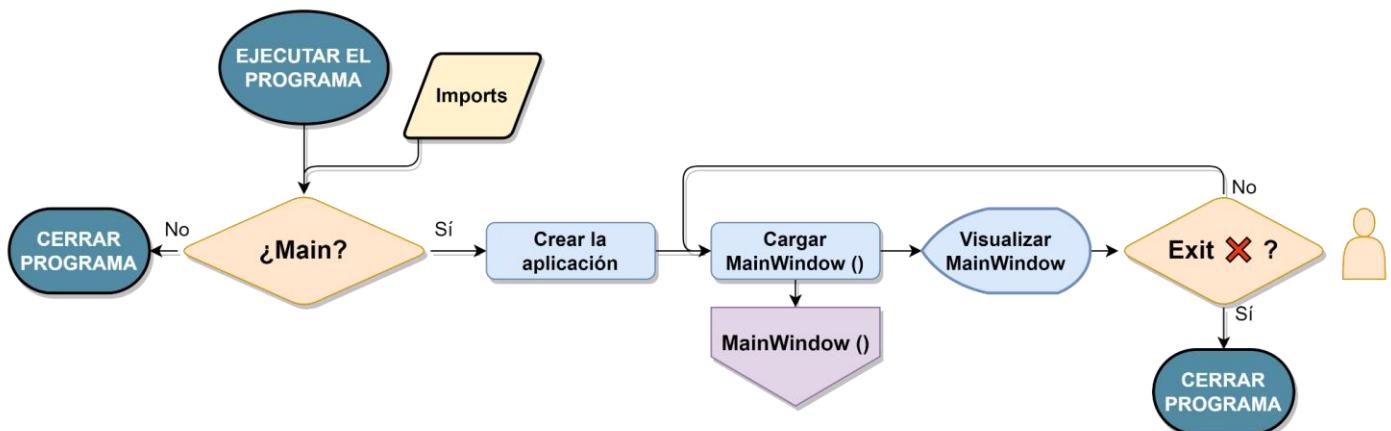


Figura 5.7 Diagrama de flujo del programa principal

La figura 5.7 representa el diagrama de flujo del programa principal, siendo así el “script” más importante para inicializar el programa.

La aplicación puede ejecutarse desde la propia IDE de PyCharm, ejecutando el propio archivo “main.py”, o como es más común, ejecutándose directamente desde la terminal..

Al iniciarse, el programa “importa” los módulos o librerías necesarias. A continuación se asegurará de que el programa que se ha ejecutado es el principal, si se intenta ejecutar

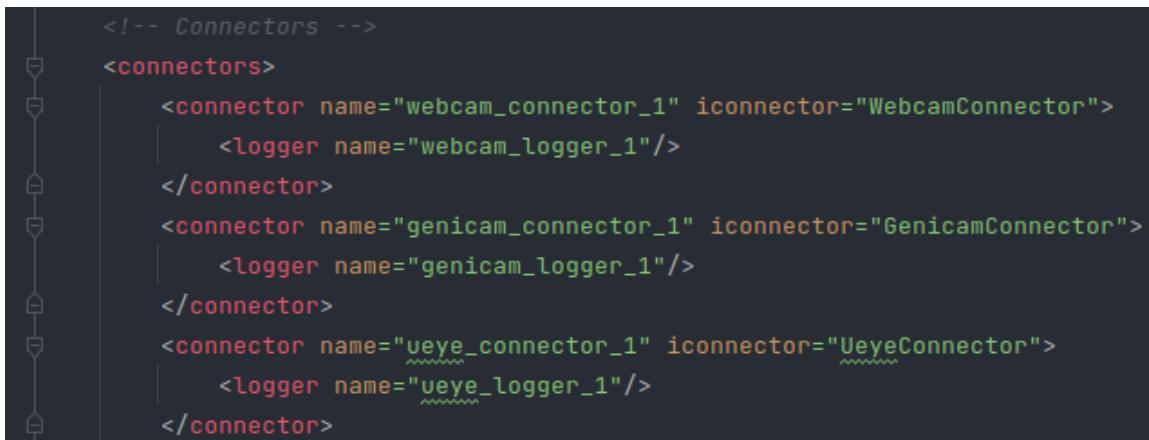
cualquier otro script, este no se ejecutará. Crea una instancia de la aplicación base sobre la cual aplicar todo el sistema y el sistema de cierre.

A partir de este punto comienza el bucle que iniciará y mantendrá abierta la ventana principal siempre que no se cierre con el botón de “Exit”. A su vez, se inicializará MainWindow ( Ventana principal ).

## **XML de configuración**

El documento de configuración es un documento de estructura sencilla, basado en el lenguaje XML, el cual proporciona una estructura de árbol al documento, añadiendo atributos y características únicas a cada rama. Gracias a esta estructura, la organización del documento es mucho más intuitiva para el usuario.

A continuación se mostraran los apartados que componen el documento:



```
<!-- Connectors -->
<connectors>
    <connector name="webcam_connector_1"  iconnector="WebcamConnector">
        <logger name="webcam_logger_1"/>
    </connector>
    <connector name="genicam_connector_1"  iconnector="GenicamConnector">
        <logger name="genicam_logger_1"/>
    </connector>
    <connector name="ueye_connector_1"  iconnector="UeyeConnector">
        <logger name="ueye_logger_1"/>
    </connector>
</connectors>
```

Figura 5.8 Ejemplos de conectores en el documento de configuración

Aunque no se explicará el funcionamiento del lenguaje XML, se puede apreciar que los colores representan diferentes atributos en el lenguaje, siendo: Rojo para los nodos, naranja para los atributos, y verde/blanco para los valores que cogen tanto atributos como nodos.

En la figura 5.8 se aprecia un conjunto de tres conectores, con los nombres: "webcam\_connector\_1", "genicam\_connector\_1" y "ueye\_connector\_1". A estos nombres se le atribuye el conector seleccionado en el atributo "iconnector" correspondiente. Bajo el conector, se aplica mediante la indentación, los "loggers" y "parser". Los "loggers" y "parsers" son instanciados mediante la asignación a un nombre, tal y como se ve en la figura 5.9.

```

51      <!-- Parsers -->
52      <parsers>
53          <parser name="rs232_parser_1" iparser="RS232parser"/>
54          <parser name="rs232_parser_2" iparser="RS232parser"/>
55          <parser name="rs232_parser_3" iparser="RS232parser"/>
56      </parsers>
57
58      <!-- Loggers -->
59      <loggers>
60          <logger name="webcam_logger_1" ilogger="WebcamLogger"/>
61          <logger name="genicam_logger_1" ilogger="GenicamLogger"/>
62          <logger name="ueye_logger_1" ilogger="UeyeLogger"/>
63          <logger name="rs232_logger_1" ilogger="RS232Logger"/>
64          <logger name="ros_logger_1" ilogger="MyCsvLogger"/>
65          <logger name="opcua_logger_1" ilogger="OPCUALogger"/>
66      </loggers>

```

Figura 5.9 Asignación de *parser* y *logger* en el documento de configuración

A diferencia de los anteriores conectores, hay un conector que requiere de "parser" y un formato propio, configurado por el usuario. En ese caso solo habría que añadir los sub-nodos correspondientes de la manera adecuada, como muestra la figura 5.10.

```

15      <connector name="rs232_connector_1" iconnector="RS232Connector">
16          <parser name="rs232_parser_1"/>
17          <logger name="rs232_logger_1"/>
18          <format string="COORDINATES :      {} , {}" values="DS,TS"/>
19      </connector>

```

Figura 5.10 Ejemplo de conector con *parser* y formato en el documento de configuración

En los casos más especiales, algunos conectores requerirán de inputs del usuario o argumentos. Estos argumentos son necesarios para el uso apropiado del conector al que se le aplican, por

ejemplo: asignación de parámetros, variables, nombres de topic de ROS, tipos de mensajes, etc...

```

38      <connector name="opcua_connector_1"  iconnector="OPCUAConnector">
39          <logger name="opcua_logger_1"/>
40          <parameters>
41              <parameter name="url" value="opc.tcp://10.1.24.225:4840"/>
42              <parameter name="read" value="True"/>
43              <parameter name="variable_node" value="ns=3;i=54053"/>
44              <parameter name="variables" value=""/>
45          </parameters>
46      </connector>

```

Figura 5.11 Asignacion de parametros en conector OPC UA

```

30      <connector name="ros_connector_1"  iconnector="ROSTopicConnector">
31          <logger name="ros_logger_1"/>
32          <parameters>
33              <parameter name="topic_name" value="My_topic"/>
34              <parameter name="comm_mode" value="subscriber"/>
35              <parameter name="msg_type" value="JointState"/>
36          </parameters>
37      </connector>

```

Figura 5.12 Asignación de parámetros en conector ROS

Tras el apartado de configuración y asignación de conectores, seguirá el apartado de gestión de pestañas.

```

68      <!-- Tabs -->
69      <tabs>
70          <tab name="Layout" page="0">
71              <widget iwidget="LayoutWidget">
72                  <gui>LayoutUI</gui>
73                  <connection connector="ueye_connector_1" widget="label"/>
74                  <connection connector="rs232_connector_1" widget="Coordinates"/>
75                  <connection connector="rs232_connector_2" widget="Velocity"/>
76                  <connection connector="rs232_connector_3" widget="Distance"/>
77                  <connection connector="time_connector_1" widget="Hora"/>
78          </widget>
79      </tab>
80      <tab name="PLC" page="1">
81          <widget iwidget = "PLCWidget">
82              <gui>PLC_TableView</gui>
83              <connection connector="opcua_connector_1" widget="" />
84          </widget>
85      </tab>
86      <tab name="IDS" page="2">
87          <widget iwidget = "IDSWidget">
88              <gui>ids</gui>
89              <connection connector="webcam_connector_1" widget="label"/>
90          </widget>
91      </tab>
92      <tab name="ROS" page="3">
93          <widget iwidget="ROSWidget">
94              <gui>ROS</gui>
95              <connection connector="ros_connector_1" widget="label"/>
96          </widget>
97      </tab>
98  </tabs>

```

Figura 5.13 Gestión de Pestañas en documento de configuración

En este apartado se definirá la estructura de la ventana principal, editando el nombre de la pestaña, y eligiendo en qué página se colocará esa pestaña en particular.

A cada pestaña se le asigna un IWidget óptimo para la gestión de los conectores que se van a aplicar a esa página. A ese mismo IWidget se le implementará la GUI deseada, así como un

conjunto de conectores (referenciados por nombre) junto con el nombre del widget al que se le quiere aplicar el mensaje del conector.

Ejemplo: A la tercera página con nombre “IDS” se le implementa el IWidget “IDSWidget”, junto con el gui (“ids.ui”) apropiado para el conector que se le quiere asignar, el **conector webcam**. El mensaje recibido del conector (un video en tiempo real) se mostrará en el widget con nombre “label”

## Ventana principal

Una vez descrito el documento de configuración, podemos retomar el punto de llamada a la ventana principal desde el programa main.

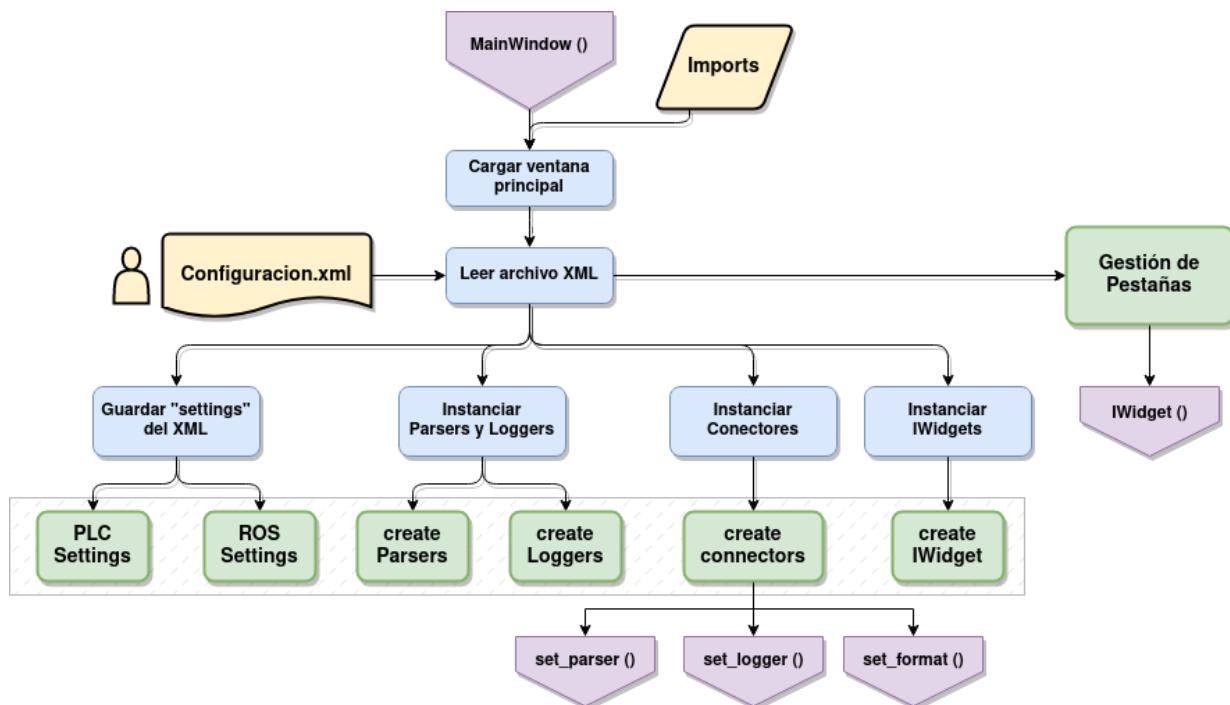


Figura 5.14 Diagrama de flujo de la Ventana principal

Al iniciar el *script* de la ventana principal, se carga la GUI referente a la ventana donde próximamente se irán añadiendo pestañas. Una vez creada la instancia a la GUI, se lee el documento de configuración en XML, cuya información se guarda en un diccionario que facilita el acceso a distintos valores que hacen referencia dentro del documento.

A continuación, con todas las referencias disponibles, se dispone a realizar distintos procesos.

- PLC Setting y ROS Setting: este proceso se encarga de aislar los valores del XML que hacen referencia a los parámetros asignados a los conectores de PLC y ROS.
- Create Parser y Create Logger: este proceso se encarga de aislar los valores del XML que hacen referencia a los *Parsers* y *Loggers*, creando una instancia de estos para su próximo uso.
- Create connectors: este proceso se encarga de aislar los valores del XML que hacen referencia a los *Conectores*, creando una instancia de estos, para luego asignar el *parser*, *logger* y/o *formato* necesarios a esos Conectores, usando las llamadas a *set\_parser()*, *set\_logger()* y *set\_format()*.
- Create IWidgets: este proceso se encarga de aislar los valores del XML que hacen referencia a los IWidgets, para su próxima asignación en las pestañas.

Todos los procesos devuelven un diccionario con estructura “clave: valor”, para facilitar la gestión de las referencias, y pudiendo así, emparejar conectores con sus respectivos mensajes a lo largo de todo el sistema.

Durante la ejecución de estos procesos, se inicia un proceso secundario, que se dedica a la gestión de pestañas usando las referencias del documento XML, así como las referencias recién creadas sobre los IWidgets.

Usando el método de los diccionarios, se puede enlazar, por el nombre asignado a los IWidgets, con el nombre asignado en las pestañas. Al igual que asigna los IWidgets a las pestañas, este proceso es capaz de asignar a esos IWidgets los conectores y GUI que corresponden con el

documento, así como enlazar la disposición de los conectores con los widgets elegidos de la misma GUI.

Todo el proceso de asignación se hace mediante los métodos genéricos que nos proporcionan y comparten las diferentes clases de IWidget.

## Conectores

Antes de describir el funcionamiento de un IWidget a nivel de código, se verán los métodos utilizados para la recepción de mensajes por parte de los conectores, comenzando por la clase base de los conectores: el *IConnector*. Las funciones descritas dentro de *IConnector* no se ejecutan junto con el *script*, estas son funciones o métodos que se llaman desde otro *script*, en este caso el *MainWindow*, para facilitar el uso de ciertas funcionalidades.

### ***IConnector***

En el constructor de la clase *IConnector* (en la inicialización), se definen los parámetros *has\_parser*, *has\_logger* y *has\_formatter*. Estos parámetros servirán para distinguir aquellos conectores que poseen esas cualidades, para más adelante manipular el mensaje.

Mirando en la figura 5.14 se aprecian tres llamadas de función, las cuales son: *set\_parser*, *set\_logger* y *set\_format*. Estas llamadas se ejecutan aquí, creando una instancia de los propios *parser* y *logger*, usando el argumento transferido como se ve en la figura 5.14. A su vez cambian el valor de *has\_parser* y *has\_logger* para posibilitar el acceso a estas funciones.

A diferencia del parser y logger, el formater se establece de forma diferente, requiriendo para ello de tres argumentos: el conector al que se le aplica el formato, el formato de texto a aplicar, y los valores que irán en las casillas destinadas. Estos argumentos se guardan para su posterior procesamiento, y al igual que los anteriores, cambiando el valor de *has\_format* a “True” para posibilitar el proceso de formateo.

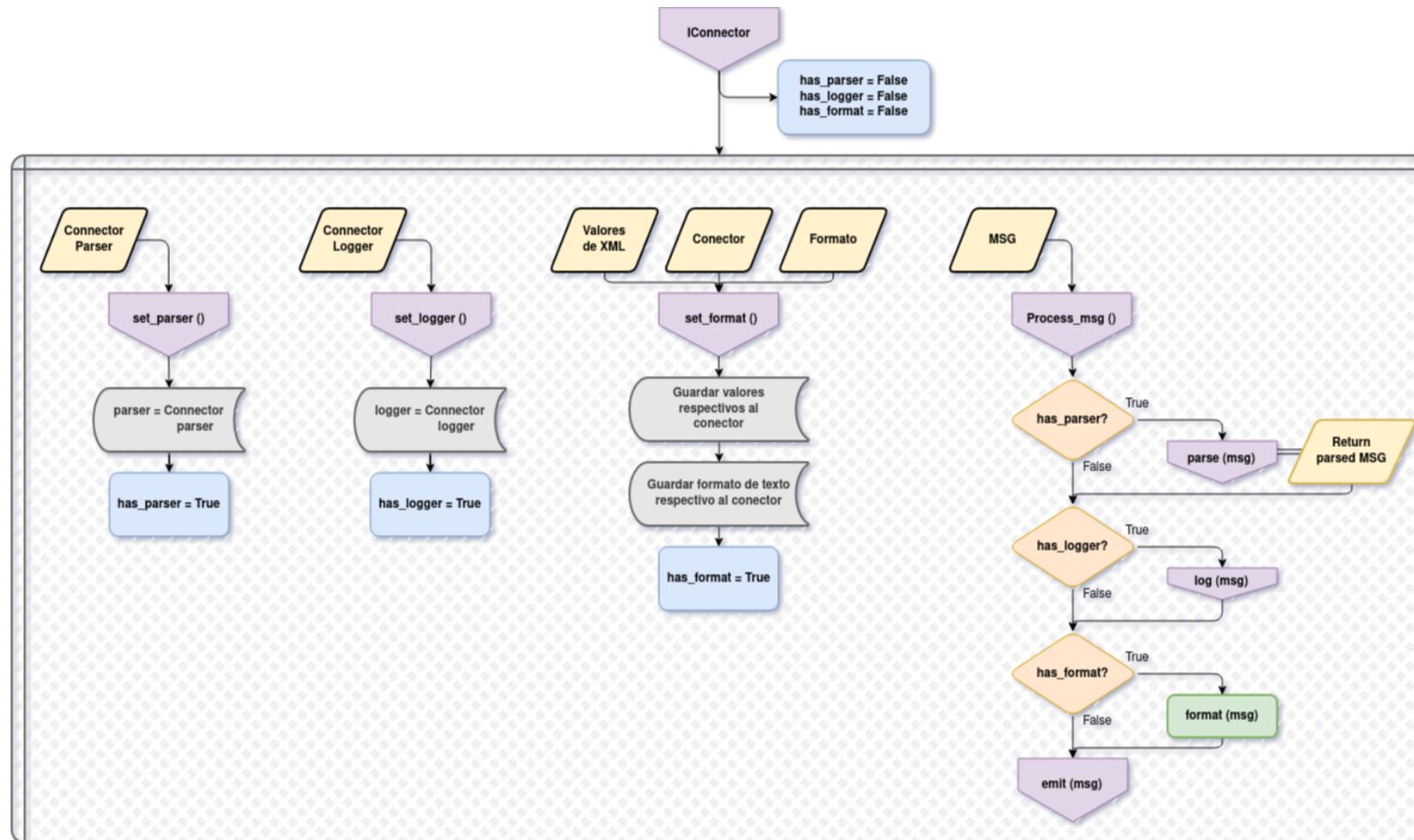


Figura 5.15 Diagrama de flujo de IConnector

Conjuntamente con los demás métodos, se encuentra la función *process\_msg*, la cual se encarga de recibir todos los mensajes de todos los conectores y procesar este mensaje en base a sus necesidades.

El *parser* actúa como “traductor”, se encarga de manipular el mensaje de entrada, el cual puede estar en un lenguaje ilegible, ya sea por métodos de protocolos de comunicación o por codificación en bytes, y transcribir ese mensaje a un formato que sea más legible para el usuario. En el caso del conector RS232, el mensaje recibido se compone por una lista de valores numéricos sin aparente orden ni descripción. El *parser* utiliza el posicionamiento de cada valor dentro de la lista y crea un diccionario añadiendo claves a esos valores, haciendo que el mensaje sea fácilmente manipulable y legible. Todos los *parsers* están pensados para devolver el mensaje en formato diccionario. Esto aporta facilidad al procesamiento de los datos del mensaje.

El *logger* simplemente actúa como un registro de datos. La primera vez que se ejecuta este proceso crea un archivo CSV (para cada uno de los conectores que lo requieran). A partir de la creación del archivo, el *logger* registrará todos los mensajes de todos los conectores. Esto es útil para el postprocesamiento de datos y/o análisis de estos.

Por último, se ejecuta el proceso *formatter*, a diferencia de los demás, este proceso es genérico, es decir, no depende del tipo de mensaje que recibe, ya que está pensado solo para los mensajes de tipo texto. Esto implica que no hay que crear un *formatter* nuevo por cada conector diferente que implementemos, a diferencia de otros conectores que quizás sí lo necesiten.

El *formatter* recibe los datos almacenados, tales como el *conector* al que se aplica, el formato de texto que debe de adquirir el mensaje, y los valores que se implementarán en el formato de texto elegido. Esto se consigue aislando los huecos del formato de texto donde los valores serán reemplazados (Ejemplo: mi nombre es {} y tengo {} años), y mediante el mecanismo de diccionario, emparejar el valor del mensaje deseado (nombre, edad), y aplicarlo al formato de

texto {nombre: “Aner”, edad: “23”}, obteniendo un resultado acorde con lo que pide el usuario ( mi nombre es Aner y tengo 23 años) .

Tras el proceso de manipulación del mensaje, este es emitido mediante el mecanismo de Signal/Slot, y esperará ser recibido en un apartado de otro *script*, permitiendo el envío de mensajes entre distintos puntos del programa.

### Conejor estándar

A continuación se mostrará el diagrama de flujo de un conector estándar:

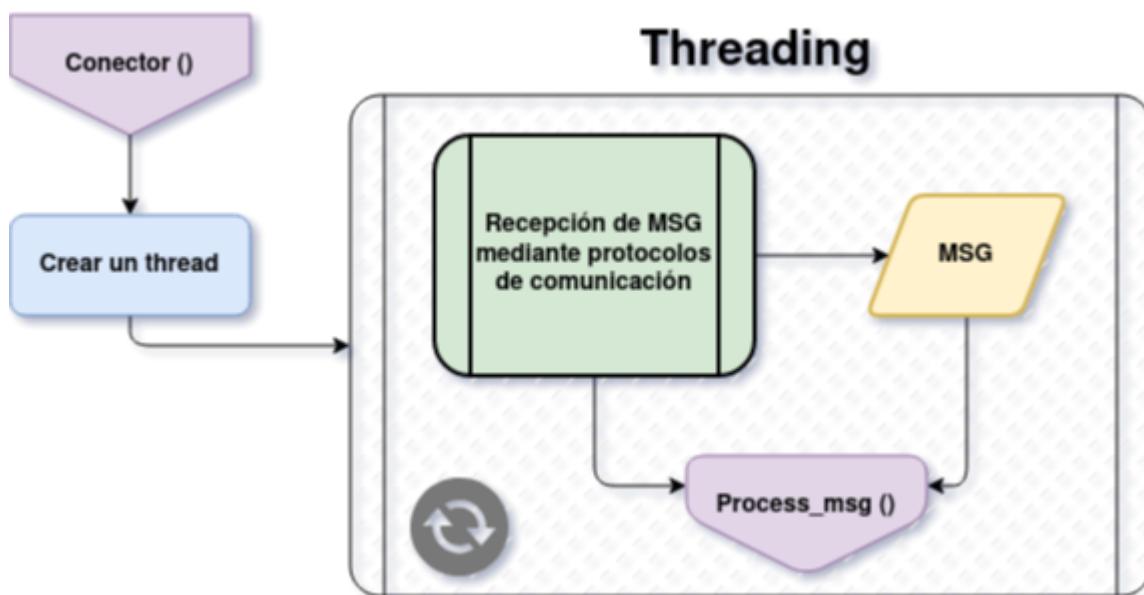


Figura 5.16 Diagrama de flujo de conector estándar

Este diagrama de flujo corresponde a la mayor parte de conectores que se utilizarán en el sistema, por ejemplo: conectores de cámaras, conectores de RS232 y conectores de tiempo.

La simplicidad del conector se debe al mecanismo de “*Threading*”. Este mecanismo se basa en crear un hilo paralelo al hilo de ejecución original, permitiendo ejecutar diversos procesos sin interrumpir el curso del programa principal. Esto resulta en una optimización de la fluidez del

código, varios conectores permanecerán continuamente recibiendo datos sin interrumpirse el uno al otro.

El proceso estándar es básico, cada dispositivo, componente o software, tiene un protocolo específico para enviar y recibir datos (Ejemplo: Protocolo NMEA 0183). Cada conector debe ser capaz de recibir mensajes usando un mismo protocolo. Esto permite que diferentes dispositivos puedan usar el mismo conector, utilizando el mismo protocolo.

El protocolo se encargará de recibir datos y guardarlos en forma de “*mensaje*”. En cuanto un conector reciba un “*mensaje*”, este será procesado por la función antes descrita: *process\_msg*.

#### **Conecotor específico: Conecotor OPC UA**

En algunos casos, el protocolo de comunicación utilizado es un poco más complejo y necesita de varios procesos antes de recibir y/o enviar datos.

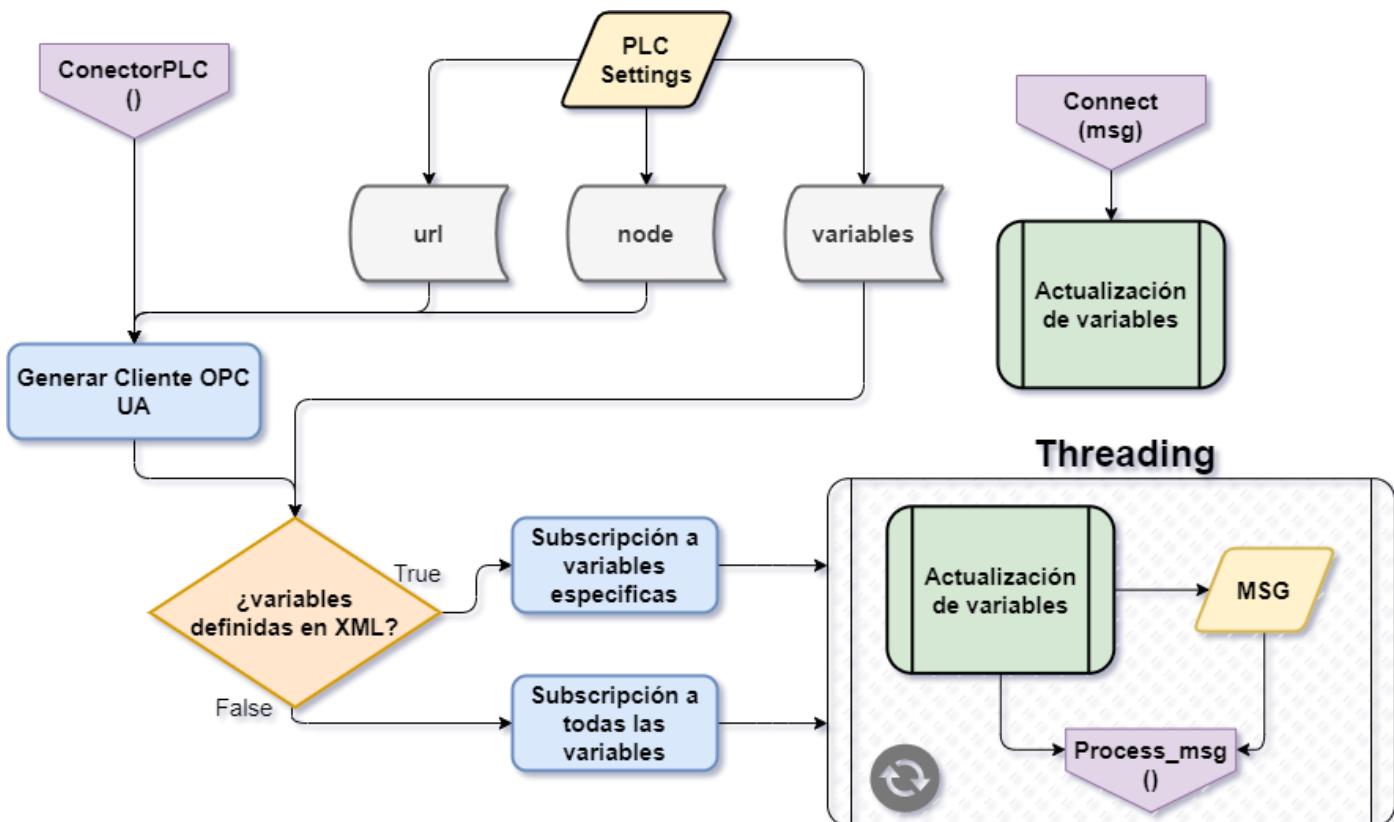


Figura 5.17 Diagrama de flujo de conector OPC UA

Para recibir y enviar mensajes o datos mediante OPC UA, antes se requiere la creación de un cliente virtual de OPC UA. Esto se consigue utilizando los parámetros “*url*” y “*node*”, extraídos del documento de configuración XML. Una vez creado el cliente virtual, es necesario “suscribirse” a las variables que contiene el PLC, para poder visualizar los datos de estas variables. En este caso, se da la opción al usuario de elegir mediante el documento de configuración las variables que quiere visualizar, en su defecto, se “subscribe” a todas las variables publicadas dentro del PLC.

La acción de “suscripción” permite al cliente recibir actualizaciones sobre la información de esas variables. Esta información es procesada por *process\_msg* y directamente enviada al IWidget correspondiente.

Este caso es un caso de comunicación bidireccional, lo que implica que el usuario tiene permitido interactuar con algunas variables a las que está suscrito (Ejemplo de variables: Arranque, Paro, Tiempo de Espera). Al interactuar con estas variables, el conector recibe la emisión del mensaje (proveniente del IWidget) usando el mecanismo Signal/Slot de Qt. Una vez los datos de la variable son cambiados por el usuario, el conector se encarga de notificar los cambios al cliente, para que este proceda con el funcionamiento definido del PLC.

#### **Conejor específico: Conejor ROS**

Al igual que el conector de OPC UA, el cliente de ROS necesita crear un nodo, el cual a elección de la configuración del usuario, servirá como *publisher* o *subscriber*, enviando o recibiendo datos respectivamente.

Para identificar y enlazar tanto el emisor del mensaje, como el receptor, se detallan dos parámetros configurados por el usuario: *topic\_name* (actúa como identificador de las vías de transmisión) y *msg\_type* (especifica el tipo de mensaje que va a recibir).

Ejemplo: el emisor (publisher) siendo un brazo robótico, está enviando constantemente datos sobre su posición. El cliente crea un nodo que servirá como receptor (subscriber). El emisor está enviando los datos por un canal (topic name) que llama “position”, y el mensaje que envía es de tipo “String”. El usuario deberá concretar qué quiere suscribirse al *topic\_name* = “position” y que *msg\_type* = “String”. Concordando así, entre publisher y subscriber, una vía de transmisión efectiva.

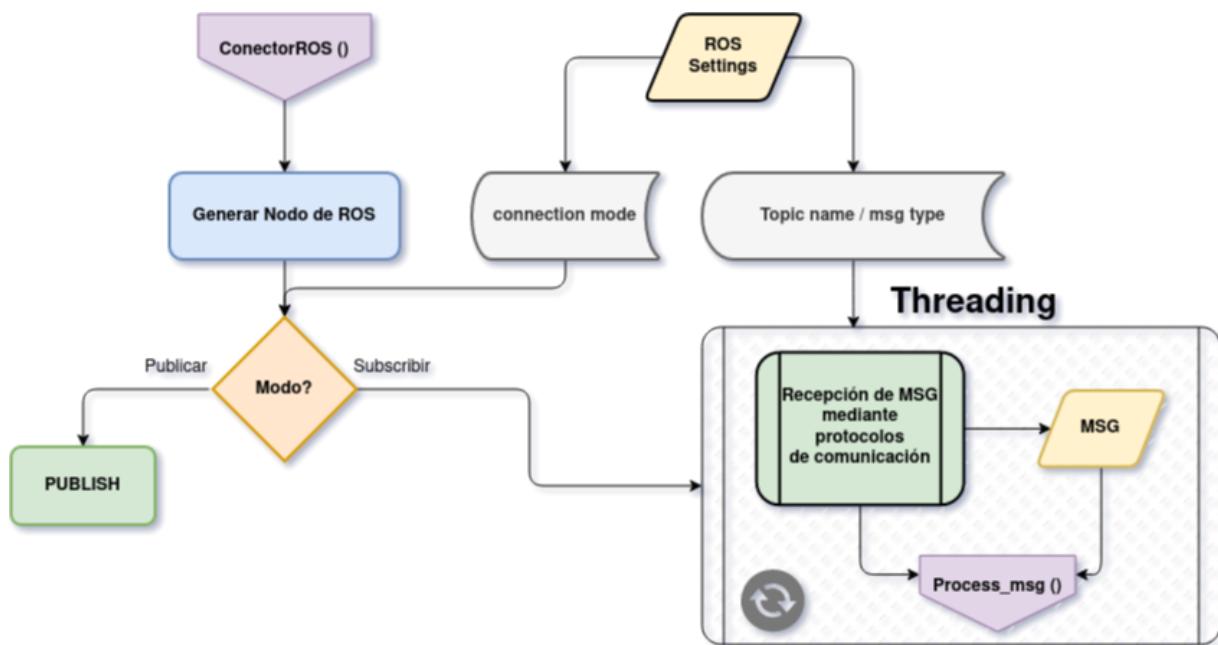


Figura 5.18 Diagrama de flujo de conector ROS

## IWidgets

Para describir el funcionamiento de los IWidgets conviene volver al proceso en el que se hace uso de estos por primera vez, véase la figura 5.14.

Un IWidget toma como parámetros los conectores que se asignan al IWidget, la GUI que implementa el IWidget dentro de la pestaña de la ventana principal, y el widget donde el mensaje recibido de los conectores serán aplicados.

### ***IWidget estándar***

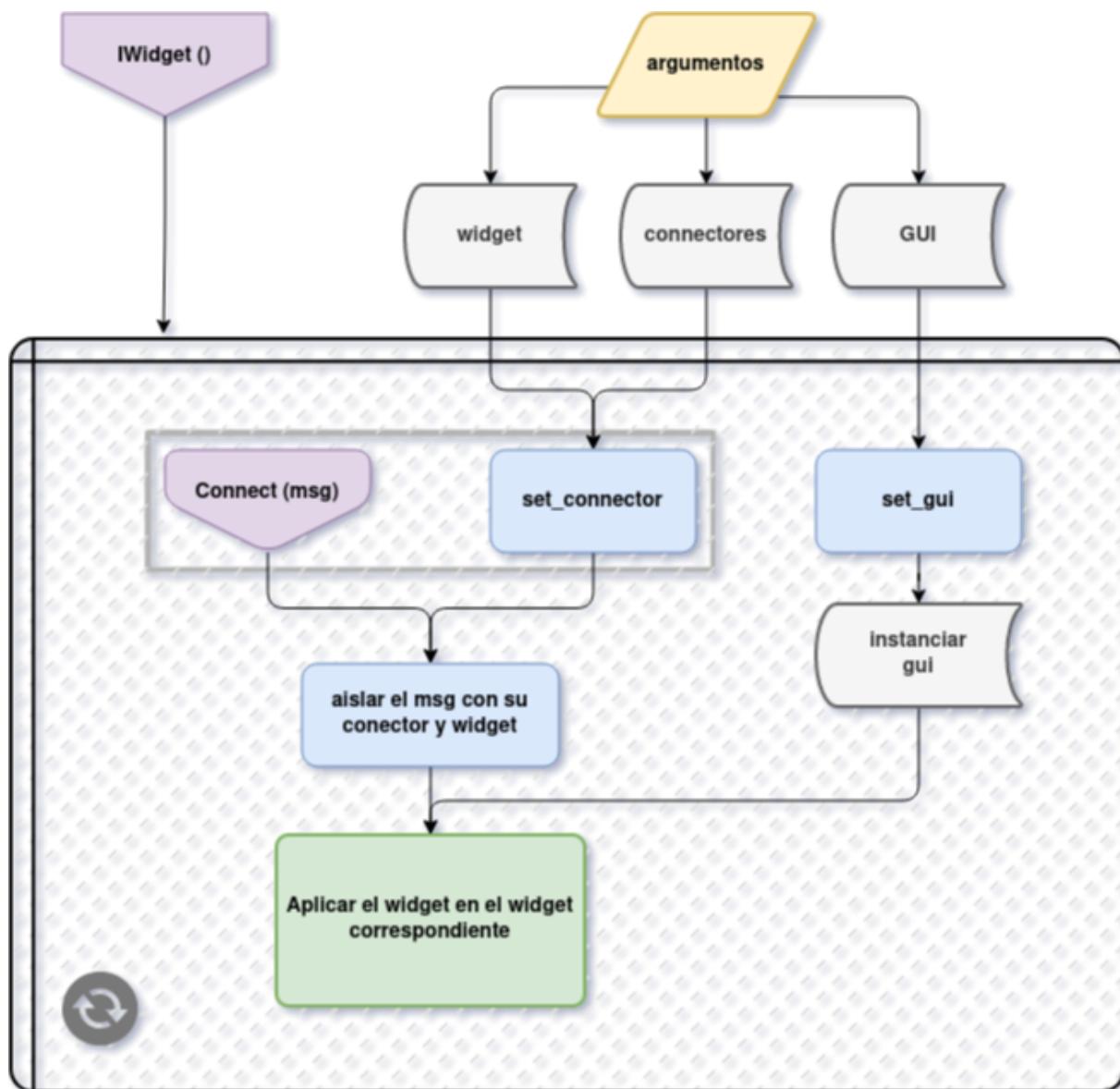


Figura 5.19 Diagrama de flujo de un IWidget estándar

Tomando el parámetro de la GUI, se crea una instancia de la interfaz, esta instancia permite referenciar widgets y aplicarles ciertos atributos, así como editar la interfaz usando solamente código.

A su vez `set_connector` guarda un diccionario con los conectores asignados a la `IWidget`, relacionando dichos conectores con los widgets a los que se aplicará cada mensaje(Ejemplo: `conector1: "label1", conector2: "label2"`). Mediante este proceso se crean instancias de los

conectores, permitiendo recibir mensajes emitidos por un conector específico utilizando el mecanismo Signal/Slot de Qt (Ejemplo: tras *process\_msg* del mensaje del conector1, emite el mensaje procesado, recibiendo instantáneamente en este apartado del código. El mensaje es atribuido al conector1). Con el mensaje recibido, el IWidget enlaza el mensaje y el widget correspondientes al mismo conector, implementándolo en el widget de la GUI (Ejemplo: conector1 a elegido colocar el mensaje en el widget “label1”, el mensaje que se reciba desde conector1 será aplicado directamente en el widget “label1”). Los mensajes implementados en la GUI son actualizados constantemente y mostrados cada vez que se recibe un mensaje nuevo.

#### **IWidget específicos: PLCWidget**

A diferencia de los IWidget comunes, el IWidget está basado en el patrón de arquitectura de software “Modelo-Vista-Controlador”[\[27\]](#), comúnmente utilizado en aplicaciones de interfaz gráfica de usuario.

Este método nos permite separar los componentes de nuestra aplicación dependiendo de la responsabilidad que tienen, es decir, una parte de tu código no debe de saber qué es lo que hace toda la aplicación, sólo debe de tener una responsabilidad.

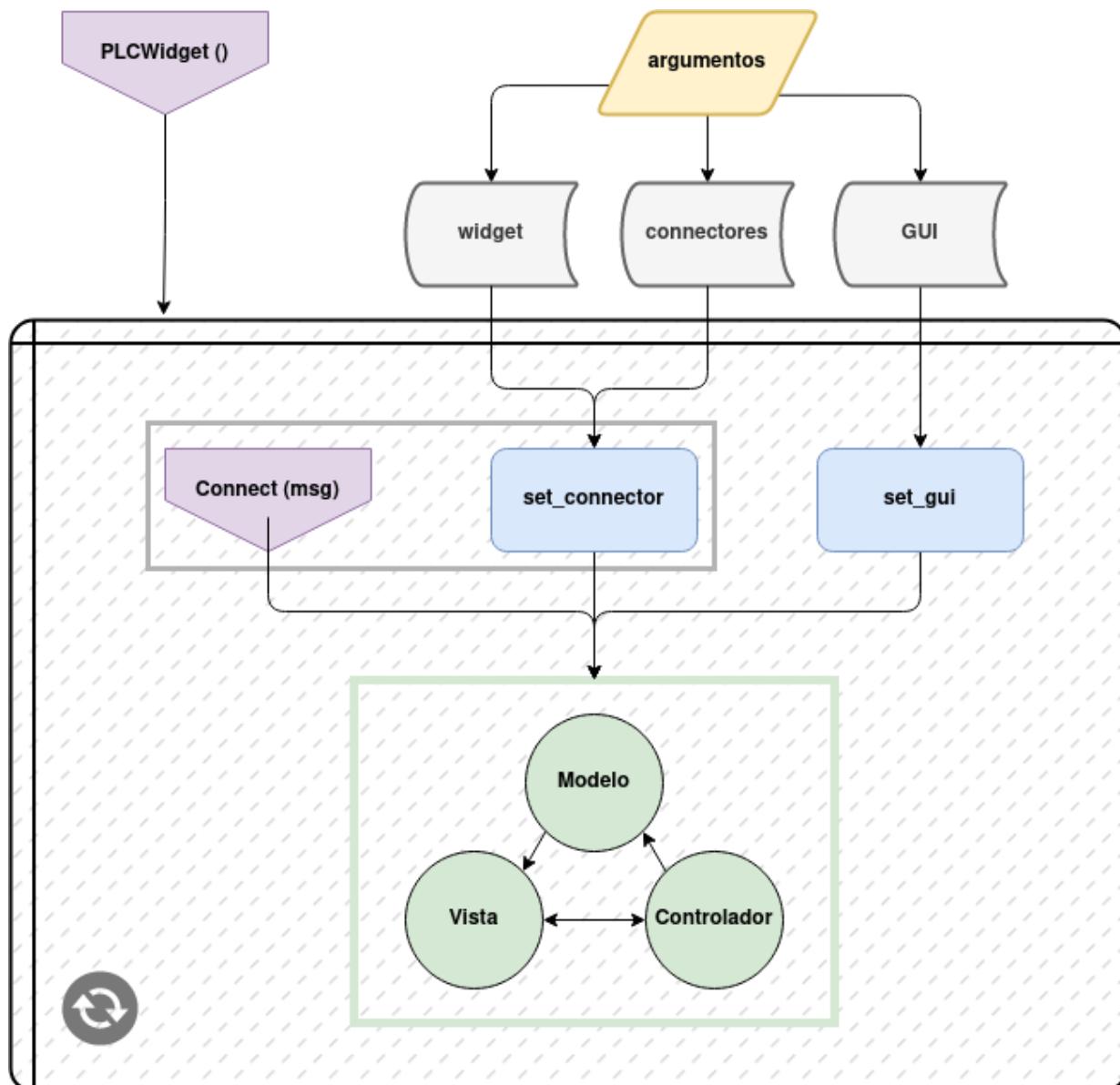


Figura 5.20 Diagrama de flujo de IWidget específico PLCWidget

- **Modelo:** Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc.
- **Controlador:** Se encarga de controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.
- **Vista:** Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

Para aplicar este método a nuestra IWidget, Qt tiene funcionalidades que facilitan la implementación de un patrón Modelo-Vista-Controlador.

En este sistema, la vista corresponderá a una tabla. Esta tabla visualizará la información necesaria recibida del conector del PLC.

El modelo y el controlador se encargan de gestionar la actualización de variables y la información atribuida a las variables, así como permitir y gestionar la interacción del usuario con las ciertas variables de la tabla. Y no menos importante, permite asignarle ciertos atributos a las celdas de la tabla, por ejemplo: editabilidad (permitir al usuario interactuar con esa celda), concretar el tipo de valor de la celda (si es solo tipo numérico o de texto), alineaciones (permitir ordenar las columnas o las filas), etc...

Este método proporciona una solución segura a la comunicación bidireccional con el servidor OPC UA, optimizando la actualización de variables, con la finalidad de no saturar la frecuencia de suscripción al servidor, creando una interacción más fiable con el usuario .

### **5.2.1.2 Frontend**

El frontend es la parte del desarrollo de aplicaciones que se dedica a la parte frontal o visual de la aplicación, como por ejemplo, diseño de la interfaz, estructura de la interfaz, diseño de widgets, colores, fondos, tamaños, etc...

En este apartado se incluyen todas las interfaces gráficas diseñadas y utilizadas en la aplicación, junto con una breve descripción de la estructura y las razones de diseño de las interfaces.

## MainWindow GUI



Figura 5.21 GUI de la Ventana principal

Esta GUI está diseñada específicamente para ser la ventana principal. Diseñada a partir del concepto inicial.

Esta GUI posee un mecanismo de pestañas, el cual permite el cambio a diferentes páginas. Esta estructura permite visualizar y gestionar diferentes GUIs en las pestañas seleccionadas.

## Layout GUI

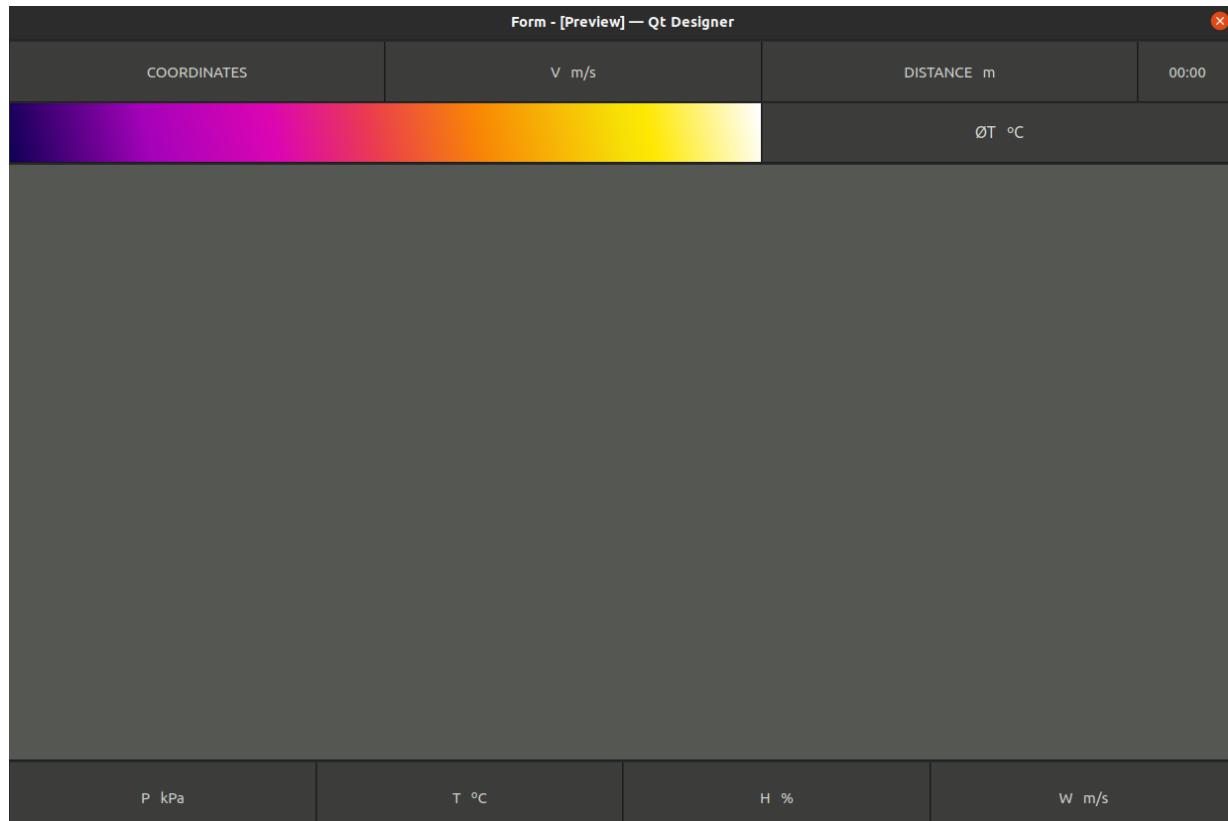


Figura 5.22 GUI de la ventana multiconexiones Layout

El diseño de esta interfaz se ha basado en el concepto de multicompatibilidad, permitiendo visualizar los diferentes mensajes recibidos por medio de un conjunto de conectores.

La disposición de widgets se ha basado en un ejercicio para probar la funcionalidad de la GUI, basado en un conjunto de conectores de ámbito industrial. Esto incluye:

Información de geolocalización:

- “COORDINATES”: Coordenadas del sensor. Latitud y Longitud
- “V m/s”: Velocidad de movimiento del sensor.
- “Distance”: Distancia recorrida.

Información Horaria:

- “Time”: Hora y minutos actuales.

Información de video:

- Widget Central: Implementación de imagen.
- Gradiante de color (Camara Termica): Específica para la camara termica, sirve para el analisis termico de la imagen.
- “ $\bar{\theta}T ^\circ C$ ” (Cámara Térmica): Específica para la cámara térmica, muestra la temperatura media de la medición.

Información meteorológica:

- “P kPa”: Presión atmosférica, unidad de kiloPascales
- “T  $^\circ C$ ”: Temperatura ambiente
- “H %”: Humedad ambiente
- “W m/s”: Velocidad de viento

**PLC GUI**

VARIABLE NAME	VALUE	DATA TYPE	SOURCE TIMESTAMP	SERVER TIMESTAMP	STATUSCODE

Figura 5.23 GUI de la Ventana de comunicación PLC

La GUI de la ventana de comunicación PLC está diseñada para la implementación del patrón Modelo-Vista-Controlador. La vista es la encargada de crear la GUI apropiada para la gestión y visualización de datos referentes al conector OPC UA.

A continuación se describen las diferentes columnas:

- **Variable name:** Nombre de la variable suscrita, permite ordenar la tabla alfabéticamente según el nombre.
- **Value:** Valor que poseen las variables.
- **Data type:** Tipo de dato de las variables (int, string, bool...).
- **SourceTimeStamp:** Hora exacta de actualización en el cliente
- **ServerTimeStamp:** Hora exacta de actualización en el servidor
- **StatusCode:** Estado de la variable, representa la correcta suscripción a la variable

## **IDS GUI**



Figura 5.24 GUI de la Ventana Secundaria de Visualización

Para la verificación de la conexión en paralelo de varias cámaras, se ha diseñado una interfaz simple que contiene un solo widget, encargado de implementar la imagen, originalmente para comprobar la conectividad con las cámaras IDS.

## **ROS GUI**

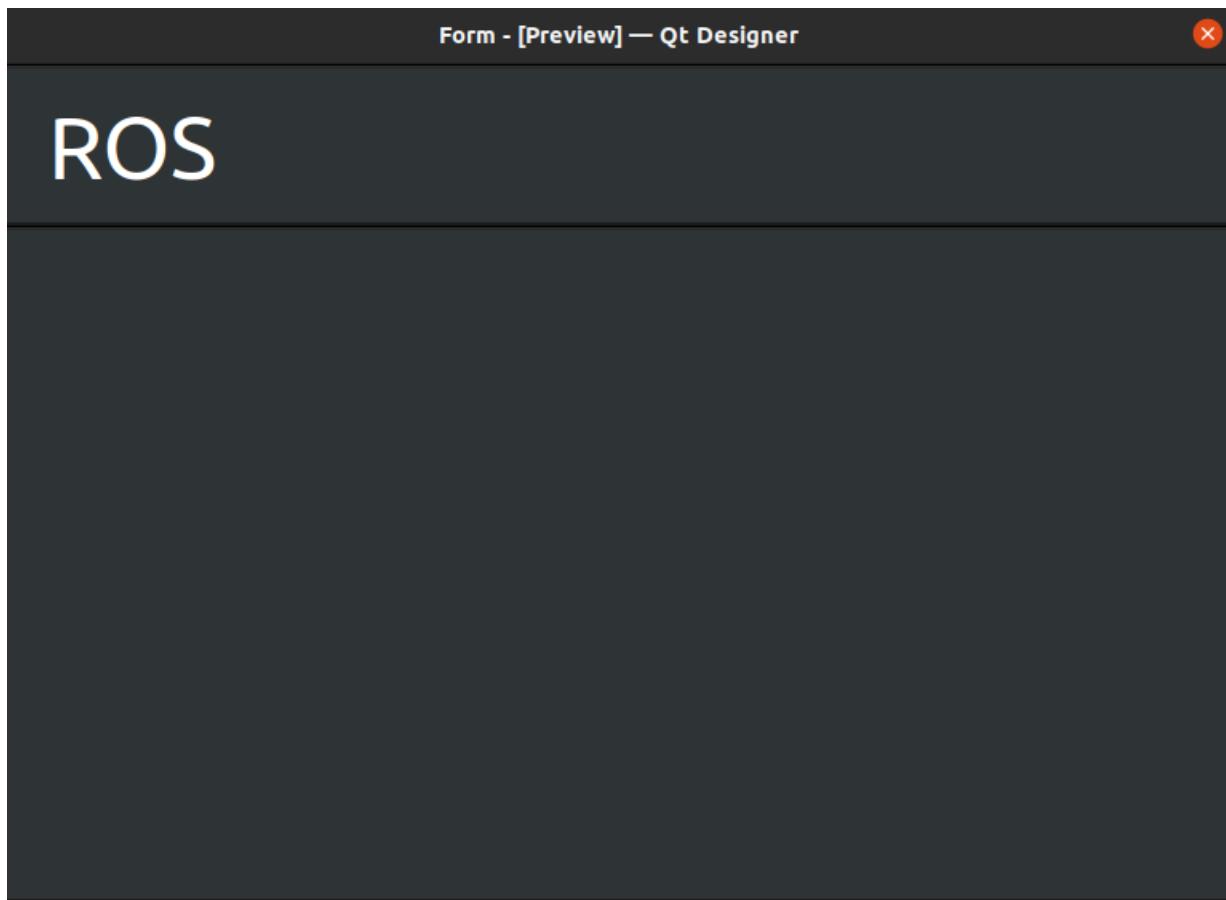


Figura 5.25 GUI de la Ventana de comunicación ROS

Esta GUI es un prototipo de diseño para un futuro desarrollo con el conector ROS. La GUI presenta un widget centrado, el cual mostrará únicamente cualquier mensaje recibido de ROS, permitiendo así la visualización de la comunicación entre máquina-usuario, pero no la comunicación bidireccional. Útil para la visualización rápida del estado de la máquina.

#### **5.2.1.3 Conclusiones de la primera versión**

La certificación del correcto funcionamiento de la primera versión de la solución, implica la conexión de dispositivos capaces de utilizar los conectores implementados en el sistema.

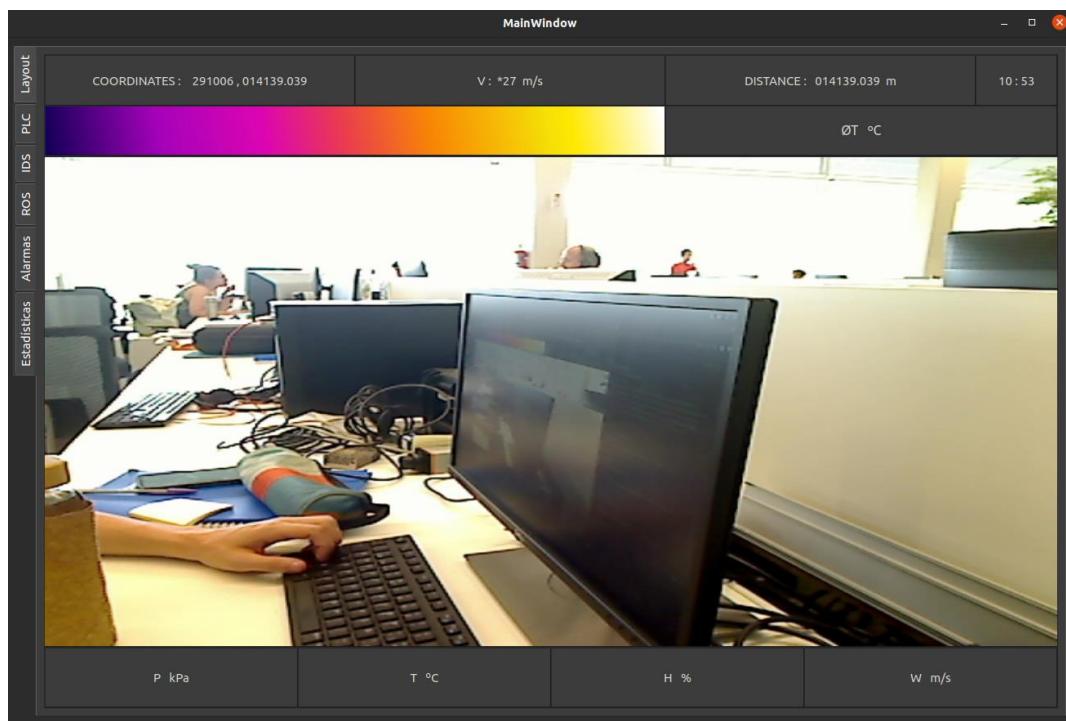


Figura 5.26 Resultado de la implementación de imagen en el Layout principal

MainWindow						
	VARIABLE NAME	VALUE	DATA TYPE	SOURCE TIMESTAMP	SERVER TIMESTAMP	STATUSCODE
Layout	ARRANQUE	True	Bool	08:41:08.667344	08:41:08.667343	Good
PLC	PARO	False	Bool	08:40:54.663361	08:40:54.663360	Good
IDS	SALIDA1	True	Bool	08:41:08.667396	08:41:08.667394	Good
ROS	SALIDA2	True	Bool	08:41:10.767325	08:41:10.767324	Good
Alarms	TIEMPOET	2000	Int	08:41:10.767273	08:41:10.767271	Good
Estadísticas	TIEMPOON	2000	Int	08:40:54.663172	08:40:54.663171	Good
Alarms	VALOR1	2143	Int	08:40:54.664357	08:40:54.664355	Good
Estadísticas	VALOR2	436	Int	08:40:54.664193	08:40:54.664191	Good

Figura 5.27 Resultado de la implementación de PLC en la GUI

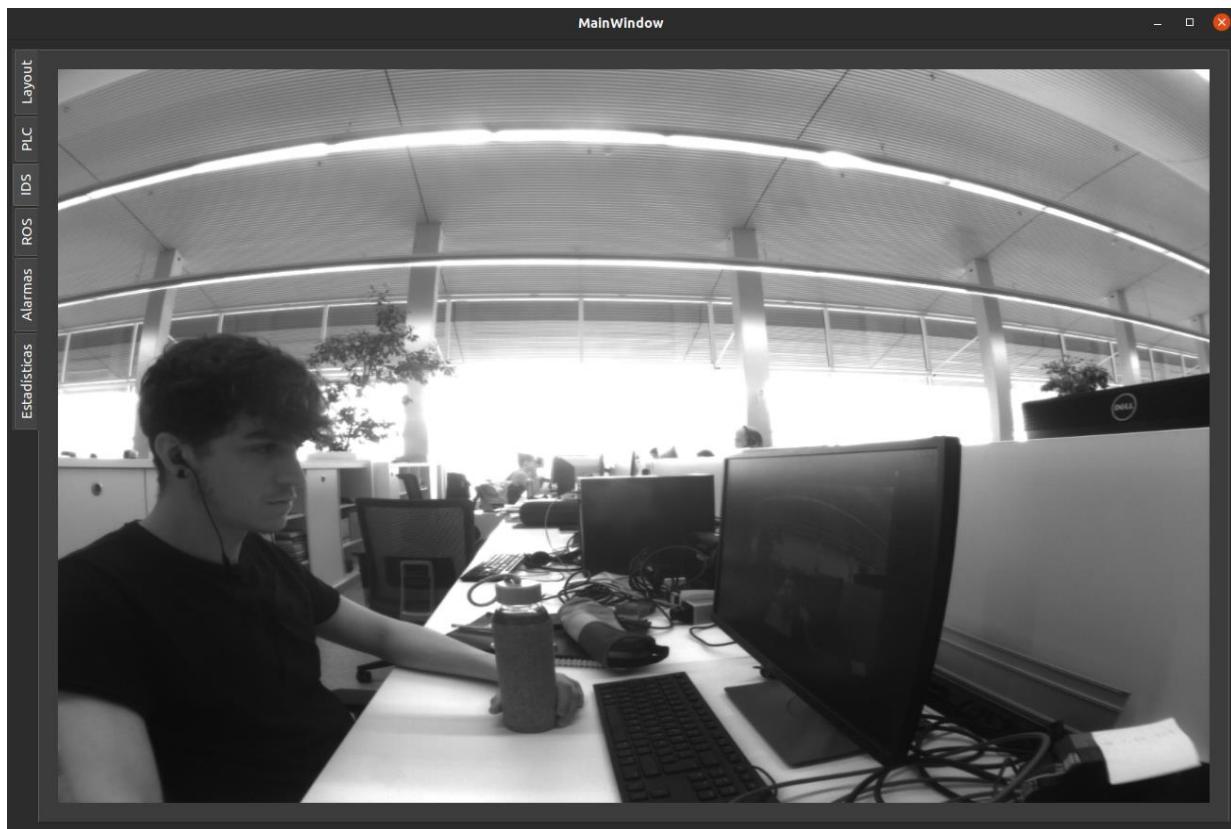


Figura 5.28 resultado de implementación de imagen en la ventana secundaria de visualización

The screenshot shows a window titled "MainWindow" with a dark theme. The title bar also says "MainWindow". On the left, there is a vertical toolbar with buttons for "Estatísticas", "Alarms", "ROS", "PLC", "IDS", and "Layout". The main content area has the word "ROS" displayed prominently. Below it, there is a JSON-like message structure:

```
seq: 16
stamp:
  secs: 0
  nsecs: 0
frame_id: ""
  name:
    - Robot
position: [5.0, 1.0]
velocity: [12.0]
effort: [34.0]
```

Figura 5.29 resultado de implementación de ROS en la GUI

Los resultados demuestran que la solución propuesta por la primera versión es perfectamente funcional.

Dadas las limitaciones de código, y por la propia estructura de la solución, el usuario está obligado a tomar parte y editar pequeñas partes del código, siempre que quiera implementar un nuevo conector o IWidget. Para facilitar esta tarea, se ha optimizado el código para que sea lo más fácil de entender posible. La estructura de código de IWidgets y Conectores es genérica, lo que permite clonar un ejemplo y editar algunos apartados para moldearlos a las nuevas implementaciones.

El problema requería una interfaz de usuario intuitiva, capaz de utilizar diferentes tipos de conexiones genéricas, y capaz de ser configurada por el usuario mediante un documento de configuración, habilitando también la implementación libre de nuevos conectores. La aplicación desarrollada ha demostrado ser una solución competente y funcional.

## **5.2.2 Segunda versión del sistema: Plugins**

### **5.2.2.1 Planteamiento del problema**

Tal y como se describe en el apartado anterior, aunque la solución si aporta una buena funcionalidad y es útil para el problema presentado, tiene aspectos a mejorar.

Algunos aspectos del código requieren de edición por parte de un desarrollador a la hora de implementar nuevos conectores e IWidgets. Esto no crea ningún problema importante, pero obliga al usuario a editar parte del código. En caso de no tener mucho conocimiento sobre el lenguaje de programación utilizado, este puede generar errores e inutilizar la operatividad de la aplicación.

### **5.2.2.2 Hipótesis: Plugins**

Una solución a este problema puede ser la implementación de los mecanismos de plugins.

En la primera versión de la solución, en algunas ocasiones es necesario acceder manualmente a los conectores y a los IWidgets utilizando su instancia por orden escrito específicamente del desarrollador, y no por el flujo de la aplicación, como se espera de una solución independiente al usuario.

El funcionamiento de los plugins aporta flexibilidad al manejo de diferentes *scripts* o funciones independientes al *script* principal, como pueden ser los Conectores o los IWidgets.

El mecanismo de plugins almacena tanto los conectores como los IWidgets dentro de un conjunto de “plugins” independientes a la estructura principal del sistema.

El método con el que se usan los plugins permite la interacción con el conjunto de plugins directamente desde el código, accediendo a la ubicación de estos. Permite gestionar el uso de los conectores e IWidgets de forma dinámica sin el llamamiento manual a estos. Esto significa que para los nuevos IWidgets que se puedan implementar en un futuro no es necesario modificar ni siquiera una línea de código en el proyecto principal.

La intención es usar los métodos que aporta el mecanismo para instanciar los conectores e IWidgets, facilitando la gestión de los mismos.

La librería de código que permite el uso de plugins está en fase de desarrollo, por lo que no consta de suficientes funcionalidades como para realizar el uso correcto del mecanismo. Una de las limitaciones que más impacto producen es la falta de gestión individual de los plugins. Estos son cargados en conjunto y de forma simultánea.

Esta limitación produce varios errores en el diseño del sistema. La carga conjunta de conectores impide la implementación individual de parámetros, así como los del PLC o ROS. Además, provoca la saturación de la pila de memoria del procesador, haciendo que varios componentes del sistema tomen la misma referencia en la memoria. Este tipo de errores genera un mal funcionamiento dentro del mecanismo de conexión, conectores como PLC y ROS no son compatibles al uso simultáneo, por el simple hecho de que ambas frecuencias de actualización se colapsan.

### **5.2.2.3 Conclusiones de la segunda versión del sistema**

Durante el desarrollo de esta segunda versión del sistema, se ha demostrado que varios conectores no son compatibles al uso simultáneo, esto provoca un error fatal en el correcto funcionamiento de la solución.

Aunque si es una técnica prometedora, el mecanismo de plugins aún no está en una fase de desarrollo óptima para el uso en esta versión del sistema. Por lo que no es viable implementarlo al estado de la aplicación.

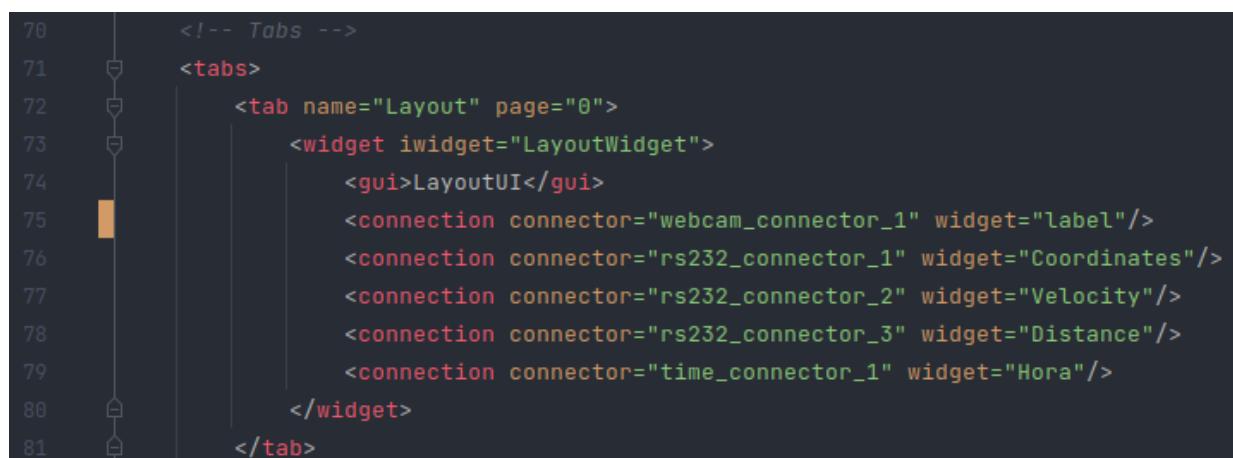
## 6. RESULTADOS OBTENIDOS

Tras realizar el proyecto, se ha logrado una interfaz gráfica de usuario perfectamente funcional. Es una solución al problema inicial notoriamente buena, a pesar de las mejoras a considerar anteriormente mencionadas, respectivas a los resultados de la primera versión de la solución.

Inicialmente se determinó que era un proyecto a largo plazo, y este TFG cubre la solución hasta establecer una base sólida y funcional con la que poder trabajar, aplicando funcionalidades propuestas para el cumplimiento de ciertos ejercicios.

El enfoque principal de la solución se basa en la flexibilidad de la interfaz para ser editada, hasta cierto punto, por el usuario. A continuación, se muestran mediante imágenes las consecuencias aplicadas en la interfaz frente a las posibles modificaciones en la configuración, hechas por un usuario ficticio.

### Selección de cámaras



```
70      <!-- Tabs -->
71      <tabs>
72          <tab name="Layout" page="0">
73              <widget iwidget="LayoutWidget">
74                  <gui>LayoutUI</gui>
75                  <connection connector="webcam_connector_1" widget="label"/>
76                  <connection connector="rs232_connector_1" widget="Coordinates"/>
77                  <connection connector="rs232_connector_2" widget="Velocity"/>
78                  <connection connector="rs232_connector_3" widget="Distance"/>
79                  <connection connector="time_connector_1" widget="Hora"/>
80          </widget>
81      </tab>
```

Figura 6.1 Configuración de inicial con Webcam

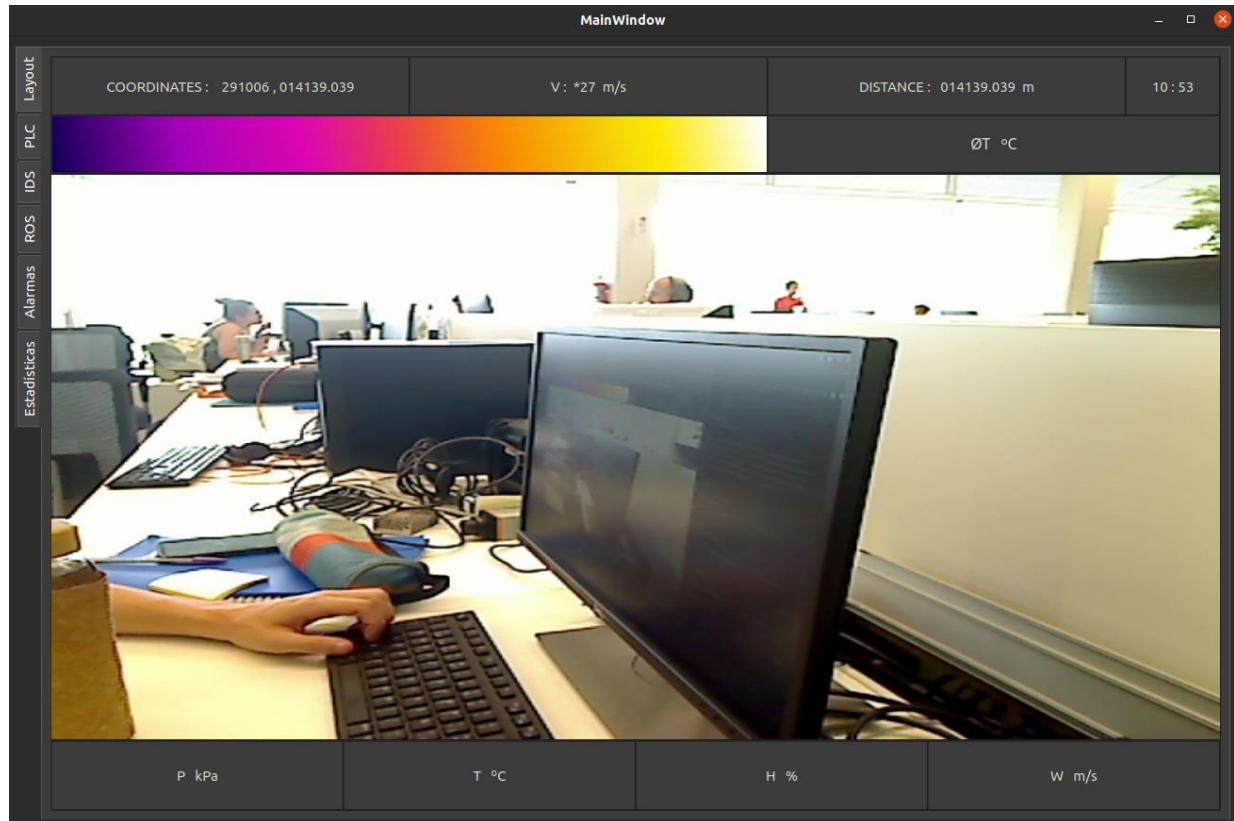


Figura 6.2 Interfaz inicial con Webcam

En estas imágenes se muestra la configuración inicial del Layout principal, donde la cámara que visualiza la imagen es una Webcam.

```
69      <!-- Tabs -->
70      <tabs>
71          <tab name="Layout" page="0">
72              <widget iwidget="LayoutWidget">
73                  <gui>LayoutUI</gui>
74                  <connection connector="genicam_connector_1" widget="label"/>
75                  <connection connector="rs232_connector_1" widget="Coordinates"/>
76                  <connection connector="rs232_connector_2" widget="Velocity"/>
77                  <connection connector="rs232_connector_3" widget="Distance"/>
78                  <connection connector="time_connector_1" widget="Hora"/>
79          </widget>
80      </tab>
81  </tabs>
```

Figura 6.3 Configuración final con GenICam

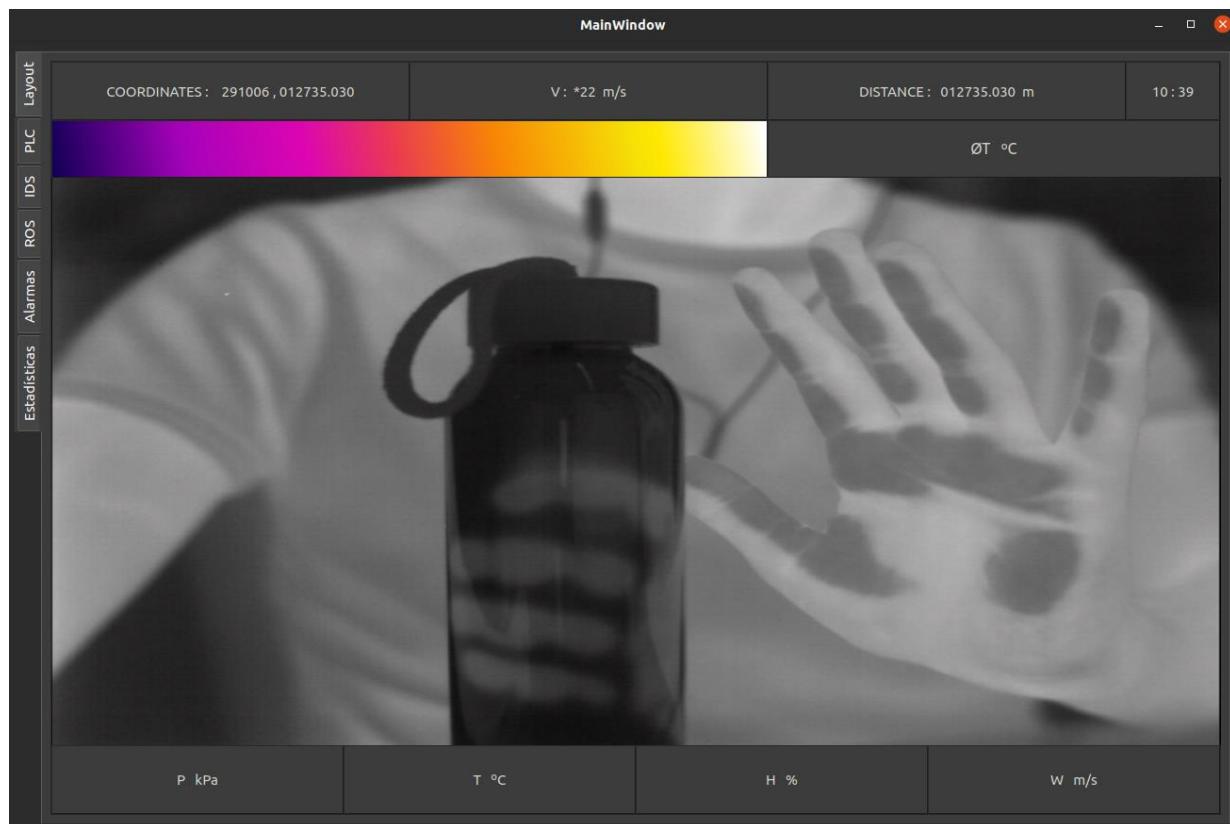


Figura 6.4 Interfaz final con GenICam

En la figura 6.4 se muestra la imagen recibida por una cámara térmica. La gama de colores de la imagen no concuerda con el gradiente de colores atribuido a esta misma cámara, esto se

debe al desconocimiento de la calibración de la cámara. La cámara asigna un color (en la escala de grises) a cada píxel correspondiente al valor de temperatura que reciba esa zona (los colores oscuros se asignan a temperaturas bajas, y los colores claros en cambio, a las altas). Por esa razón, sin una calibración adecuada y conocida, no es posible aplicarle un filtro de color RGB adecuado que se corresponda con la realidad. Pensando en el desarrollo que tendrá la aplicación en el futuro, el código está actualmente preparado para añadirle el código necesario para aplicar el mapeo de colores a la imagen recibida.

Aun así, la imagen claramente representa la propiedad térmica de los objetos frente a la cámara. Por ejemplo, en la figura 6.4 se muestra una botella de agua fría con un rastro de huellas marcadas por una mano a temperatura corporal, por otro lado, la mano refleja marcas de contacto con la botella, en las zonas en donde ha disminuido la temperatura de la piel.

Se demuestra mediante estas imágenes la correcta funcionalidad de selección de cámaras haciendo uso del documento de configuración, tal y como queda explicado en el apartado 5.2.1.1 Backend: XML de configuración.

### **Cambio de disposición de Widget**

```
68      <!-- Tabs -->
69      <tabs>
70          <tab name="Layout" page="0">
71              <widget iwidget="LayoutWidget">
72                  <gui>LayoutUI</gui>
73                  <connection connector="ueye_connector_1" widget="label"/>
74                  <connection connector="rs232_connector_1" widget="Coordinates"/>
75                  <connection connector="rs232_connector_2" widget="Velocity"/>
76                  <connection connector="rs232_connector_3" widget="Distance"/>
77                  <connection connector="time_connector_1" widget="Hora"/>
78          </widget>
79      </tab>
80      <tab name="PLC" page="1">
81          <widget iwidget = "PLCWidget">
82              <gui>PLC_TableView</gui>
83              <connection connector="opcua_connector_1" widget="" />
84          </widget>
85      </tab>
86      <tab name="IDS" page="2">
87          <widget iwidget = "IDSWidget">
88              <gui>ids</gui>
89              <connection connector="webcam_connector_1" widget="label"/>
90          </widget>
91      </tab>
92      <tab name="ROS" page="3">
93          <widget iwidget="ROSWidget">
94              <gui>ROS</gui>
95              <connection connector="ros_connector_1" widget="label"/>
96          </widget>
97      </tab>
98  </tabs>
```

Figura 6.5 Configuración inicial de widgets del conector RS232

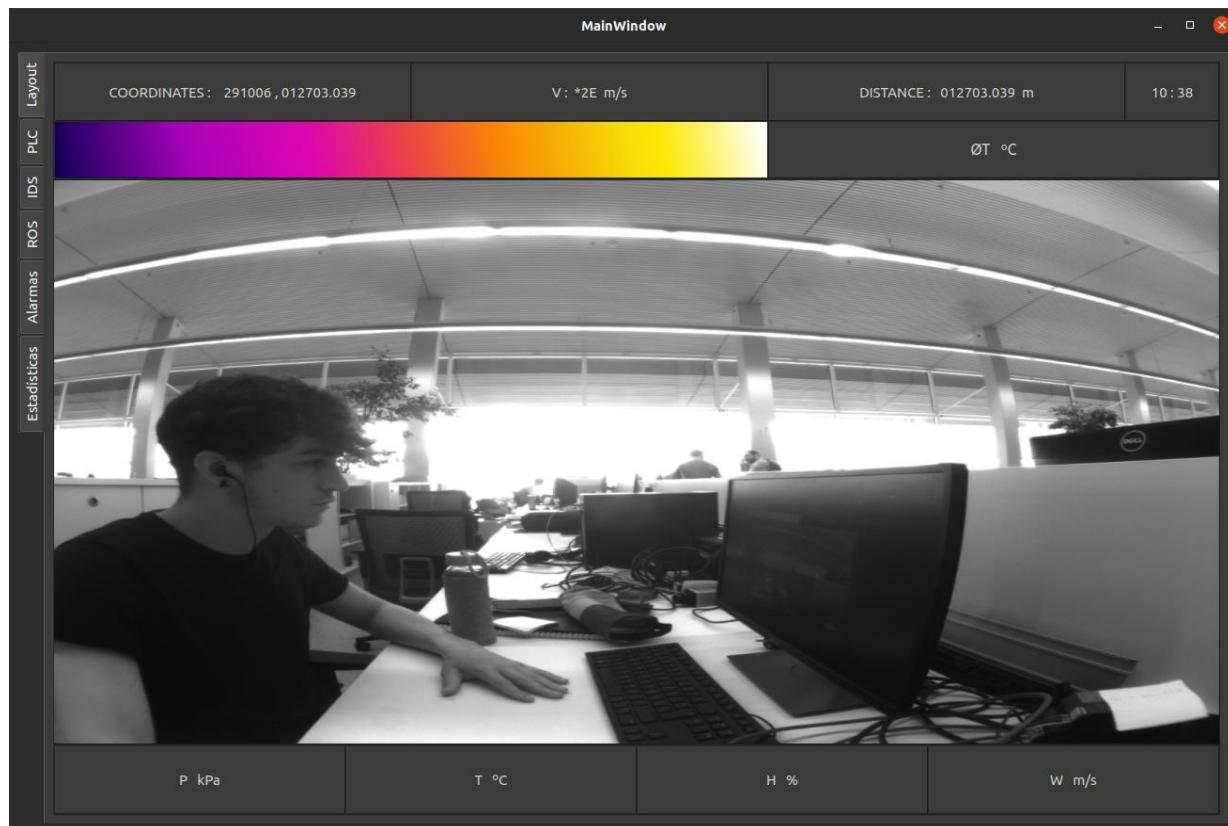


Figura 6.6 Interfaz inicial de los widgets del Layout principal

Estas imágenes muestran la configuración inicial correspondiente a la disposición de widgets para el conector RS232. Estos se visualizan en los widgets de “COORDINATES”, “VELOCITY” y “DISTANCE”.

```
67
68      <!-- Tabs -->
69      <tabs>
70          <tab name="Layout" page="0">
71              <widget iwidget="LayoutWidget">
72                  <gui>LayoutUI</gui>
73                  <connection connector="webcam_connector_1" widget="label"/>
74                  <connection connector="rs232_connector_1" widget="Pressure"/>
75                  <connection connector="rs232_connector_2" widget="Humidity"/>
76                  <connection connector="rs232_connector_3" widget="Temp"/>
77                  <connection connector="time_connector_1" widget="Hora"/>
78              </widget>
79          </tab>
```

Figura 6.7 Configuración final de widgets del conector RS232

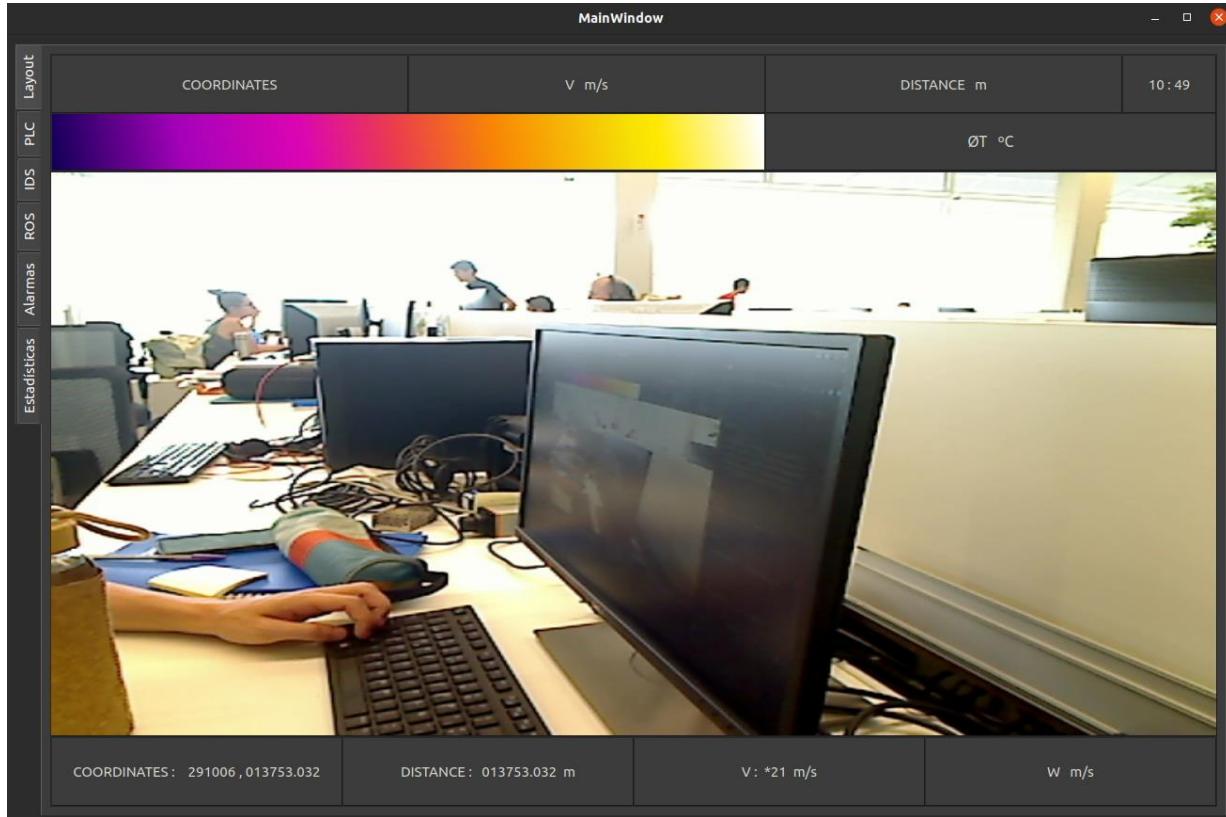


Figura 6.8 Interfaz final de los widgets del Layout principal

Tras el cambio de configuración en la asignación de widgets para el conector RS232 en la figura 6.7, en la siguiente figura 6.8 se muestra el cambio de widgets donde se visualizan los datos del conector. Ahora colocados en los widgets “PRESSURE”, “HUMIDITY” y “TEMP”.

### Configuración de formatos para textos

```
15      <connector name="rs232_connector_1"  iconnector="RS232Connector">
16          <parser name="rs232_parser_1"/>
17          <logger name="rs232_logger_1"/>
18          <format string="COORDINATES :  {} , {}" values="DS,TS"/>
19      </connector>
20      <connector name="rs232_connector_2"  iconnector="RS232Connector">
21          <parser name="rs232_parser_2"/>
22          <logger name="rs232_logger_1"/>
23          <format string="V :  {} m/s" values="Checksum"/>
24      </connector>
25      <connector name="rs232_connector_3"  iconnector="RS232Connector">
26          <parser name="rs232_parser_3"/>
27          <logger name="rs232_logger_1"/>
28          <format string="DISTANCE :  {} m" values="TS"/>
29      </connector>
```

Figura 6.9 Configuración inicial de formato de textos

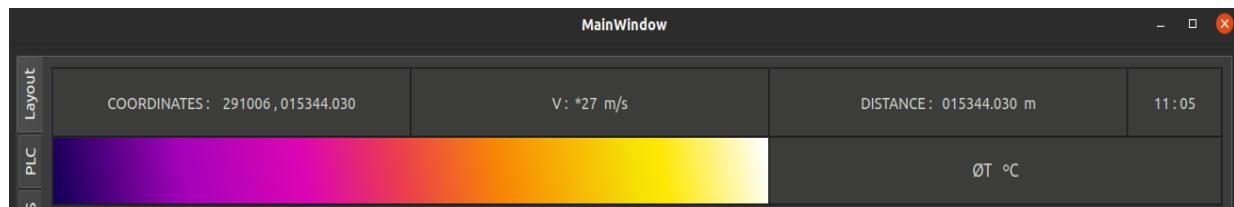


Figura 6.10 Interfaz inicial de formato de textos

En las anteriores imágenes se muestra la correspondencia de los formatos de texto configurados inicialmente, donde “string” representa el texto que acompañará a los valores aportados por los conectores, y “values” representa estos valores.

```

15 <connector name="rs232_connector_1" icontroller="RS232Connector">
16   <parser name="rs232_parser_1"/>
17   <logger name="rs232_logger_1"/>
18   <format string="Latitud: {} , Longitud: {}" values="DS,TS"/>
19 </connector>
20 <connector name="rs232_connector_2" icontroller="RS232Connector">
21   <parser name="rs232_parser_2"/>
22   <logger name="rs232_logger_1"/>
23   <format string="Speed: {} knots/s" values="Checksum"/>
24 </connector>
25 <connector name="rs232_connector_3" icontroller="RS232Connector">
26   <parser name="rs232_parser_3"/>
27   <logger name="rs232_logger_1"/>
28   <format string="TimeStamp: {} m" values="TS"/>
29 </connector>
  
```

Figura 6.11 Configuración final de formato de textos

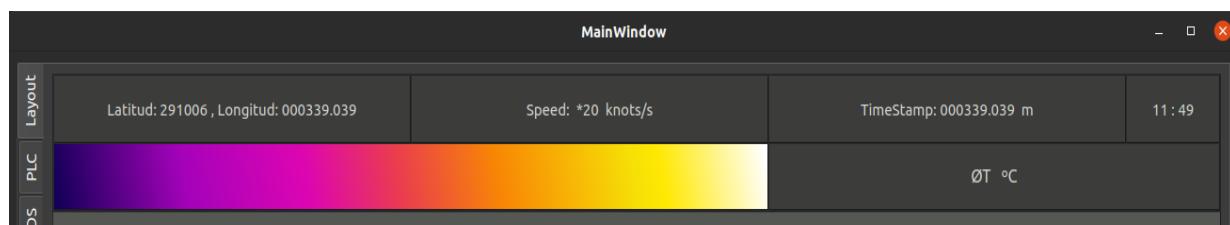


Figura 6.12 Interfaz final de formato de textos

Tras configurar los formatos en los que se aplicará el texto, en la figura 6.12 se muestran los cambios que corresponden a los nuevos formatos de texto, junto con sus nuevos valores.

Cabe la aclaración de los supuestos valores atribuidos a los componentes “longitud”, “latitud”, “velocidad” y “distancia”. Estos valores corresponden a la información referente a otros componentes como “TimeStamp” y “CheckSum”. Se ha elegido mostrar estos valores con la finalidad de verificar la funcionalidad del sistema, ya que desde un recinto cerrado (donde se sitúa el entorno de trabajo), el dispositivo GPS no es capaz de mostrar información válida sobre la posición y velocidad.

## **7. CONCLUSIONES Y TRABAJO A FUTURO**

Tras analizar los objetivos propuestos y los resultados, se puede concluir con total seguridad que el proyecto se ha desarrollado con éxito.

La solución final es capaz de visualizar información de diversos dispositivos y softwares mediante conectores, IWidgets y GUI. Con la flexibilidad de configurar varios componentes y factores que componen la interfaz gráfica usando únicamente un documento de configuración a disposición del usuario.

Este desarrollo servirá a Tekniker como una plataforma base para construir aplicaciones futuras en entornos tecnológicos. Aun así, el producto actual es completamente funcional y podría implementarse en cualquier momento aplicando nuevos conectores al sistema.

Tras la finalización de esta etapa de la solución, para el futuro quedarían diversas tareas, entre ellas:

1. Implementar nuevos conectores, ampliando el rango de dispositivos capaces de conectarse al sistema.
2. Implementar nuevas interfaces, logrando una visión diferente y adaptada al gusto del usuario en particular.
3. Crear nuevos IWidgets que contengan una disposición de información de utilidad en función de las necesidades de Tekniker.
4. Sustituir el código que depende del desarrollador, manteniendo su funcionalidad, optimizando el código para que sea aún más genérico.
5. Completar la documentación del sistema según se actualiza el estado de la aplicación.

Estas mejoras conseguirán un producto en continuo desarrollo, pero siempre funcional, que permite la adaptabilidad de la interfaz al usuario, tal y como se planeó inicialmente para el proyecto.

## **8. ACRÓNIMOS Y GLOSARIO**

Este apartado ofrece las definiciones referentes a conceptos técnicos, anglicismos y otros conceptos útiles para la comprensión del proyecto, así como los acrónimos utilizados en el desarrollo de la memoria del TFG.

### **8.1 Glosario**

- **Anaconda:** Anaconda es una distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos, y aprendizaje automático. Esto incluye procesamiento de grandes volúmenes de información, análisis predictivo y cómputos científicos
- **C++:** C++ es un lenguaje de programación. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos.
- **Clase:** Las clases proveen una forma de empaquetar datos y funcionalidad juntos. Al crear una nueva clase, se crea un nuevo tipo de objeto, permitiendo crear nuevas instancias de ese tipo. Cada instancia de clase puede tener atributos adjuntos para mantener su estado.
- **Conector rs232:** rs232 es una interfaz que designa una norma para el intercambio de datos binarios serie entre un DTE (Data Terminal Equipment), como por ejemplo una computadora, y un DCE (Data Communication Equipment), por ejemplo un módem
- **Django:** Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo–vista–controlador.
- **Estructura clave - valor:** La estructura clave - valor es un formato genérico en los diccionarios en Python, esto permite asociar una palabra “valor” a una palabra “clave” (Ejemplo: {Edad: 22}, el valor “22” referencia a la palabra clave “Edad”).
- **Frame-rate:** La tasa de fotogramas, expresada como fotogramas por segundo, es la frecuencia a la cual un dispositivo muestra imágenes llamadas fotogramas o cuadros.
- **Framework:** Un Framework o “entorno de trabajo” es una estructura previa que se puede aprovechar para desarrollar un proyecto. El Framework es una especie de

plantilla, un esquema conceptual, que simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo a lo que se quiere realizar.

- **Git:** Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.
- **Hilo de ejecución:** Un hilo de ejecución es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo.
- **Indentación:** Indentar significa hacer espacios hacia la derecha para mover una línea de código, esto facilita la estructuración de código y genera una jerarquía en algunos procesos.
- **Instancia:** Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro. En los lenguajes de programación orientada a objetos, como python, un objeto es una instancia de una clase.
- **Machine learning:** El Machine Learning es una disciplina del campo de la Inteligencia Artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones.
- **Plugin:** Los plugins son complementos que añaden funcionalidades extra o mejoras a los programas.
- **Script:** Un script en programación es un documento de texto donde se colocan instrucciones u órdenes que luego serán ejecutadas por un dispositivo inteligente.

- **Visión artificial:** La visión artificial es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador.
- **Widget:** Los widget o artilugios, tienen como objetivo dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

## 8.2 Acrónimos

- **API:** Interfaz de programación de aplicaciones, es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.
- **CSV:** Un archivo CSV (valores separados por comas) es un archivo de texto que tiene un formato específico que permite guardar los datos en un formato de tabla estructurada.
- **GUI:** Interfaz gráfica de usuario, es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.
- **HTML:** Siglas en inglés de HyperText Markup Language, hace referencia al lenguaje de marcado para la elaboración de páginas web.
- **I+D+i:** Investigación, desarrollo e innovación.
- **IDE:** Un entorno de desarrollo integrado es una aplicación de software que ayuda a los programadores a desarrollar código de software de manera eficiente.

- **IoT:** Internet of Things describe objetos físicos con sensores, capacidad de procesamiento, software y otras tecnologías que se conectan e intercambian datos con otros dispositivos y sistemas a través de internet u otras redes de comunicación.
- **NMEA 0183:** La norma NMEA 0183, utiliza un protocolo de comunicación simple en código ASCII. NMEA 0183 es una combinación de especificación eléctrica y de datos para comunicación entre equipos electrónicos marítimos, tales como GPS.
- **SDK:** SDK es el acrónimo de “Software Development Kit” (Kit de desarrollo de software). El SDK reúne un grupo de herramientas que permiten al desarrollador de software la programación de aplicaciones.
- **TICs:** Las TICs son todas las herramientas y soluciones tecnológicas que permiten eficientar, ordenar y procesar la información y las comunicaciones de cualquier tipo de persona, empresa u organización en pro de la eficiencia y la agilidad.
- **XML:** xml (eXtensible Markup Language) consiste en un archivo de texto sin formato que utiliza una serie de etiquetas personalizadas con la finalidad de describir tanto la estructura como otras características del documento.

## **9. BIBLIOGRAFÍA**

¿Qué es la comunicación?:

- [1] <https://es.wikipedia.org/wiki/Comunicaci%C3%B3n>

Sueldo base promedio de un ingeniero junior:

- [2] [https://www.glassdoor.es/Sueldos/ingeniero-junior-sueldo-SRCH\\_K00,16.htm](https://www.glassdoor.es/Sueldos/ingeniero-junior-sueldo-SRCH_K00,16.htm)

ROS:

- [3] [https://es.wikipedia.org/wiki/Robot\\_Operating\\_System](https://es.wikipedia.org/wiki/Robot_Operating_System)

OPC UA:

- [4] <https://www.paessler.com/es/it-explained/opc-ua>

¿Qué es Open CV?:

- [5] <https://www.crehana.com/blog/desarrollo-web/que-es-opencv/>

Aplicaciones de Open CV:

- [6] <https://es.wikipedia.org/wiki/OpenCV#Aplicaciones>

Protocolo GigE Vision:

- [7] [https://en.wikipedia.org/wiki/GigE\\_Vision](https://en.wikipedia.org/wiki/GigE_Vision)

Protocolo GenICam:

- [8] <https://en.wikipedia.org/wiki/GenICam>

¿Qué es Qt?:

- [9] [https://wiki.qt.io/About\\_Qt/es](https://wiki.qt.io/About_Qt/es)

Librería PyQt:

- [10] <https://wiki.python.org/moin/PyQt>

Libreria PySimpleGUI:

- [11] <https://es.wikipedia.org/wiki/PySimpleGUI>

Librería PySide:

[12] <https://es.wikipedia.org/wiki/PySide>

Cámara Logitech:

[13] <https://clipsset.com/logitech-quickcam-pro-9000-lo-ve-todo/>

Cámara industrial ids u3-3890:

[14] <https://es.ids-imaging.com/store/u3-3890cp-rev-2-2.html>

Cámara industrial ids gv-5890:

[15] <https://es.ids-imaging.com/store/gv-5890cp.html>

Cámara industrial Opto Engineering:

[16] <https://www.opto-e.com/products/COE-032-M-POE-040-IR-C>

Fuente de alimentación de SITOP:

[17] <https://es.wiautomation.com/siemens/fuente-de-alimentacion/sitop>

PLC Weidmuller

[18] <https://programacionsiemens.com/la-pasarela-iot-uc20-wl2000-iot-de-weidmuller/>

Manual de uso de GPS globalsat:

[19] <https://www.manualpdf.es/globalsat/bu-353s4/manual?p=5>

Protocolo NMEA:

[20] <https://www.informaticaabordo.com/2011/10/%C2%BFque-es-el-standar-nmea/>

Mensajes clave del protocolo NMEA:

[21] <http://aprs.gids.nl/nmea/#gga>

¿Qué es Pipenv?:

[22] <https://pipenv-es.readthedocs.io/es/latest/>

¿Qué es UA Expert?

[23] <https://www.unified-automation.com/products/development-tools/uaexpert.html>

Librería Aravis:

[24] <https://github.com/AravisProject/aravis>

¿Qué es Qt Designer?:

[25] <https://doc.qt.io/qt-6/qtdesigner-manual.html>

¿Qué es PyCharm?

[26] <https://es.wikipedia.org/wiki/PyCharm>

¿Qué es la arquitectura Modelo-Vista-Controlador?:

[27] <https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/>



# ANEXOS



## ANEXO I

### Manual de Instalación

#### ÍNDICE

Requisitos previos	89
Descarga de repositorio	92
Descarga de librerías	
94	

El siguiente manual describe los procedimientos necesarios para la descarga e instalación del sistema. Este manual es una guía práctica para la descarga del repositorio que almacena el proyecto, la instalación de librerías y la estructuración del repositorio.

En caso de consulta rápida, se dispone de una copia del Manual de Instalación dentro del repositorio.

## **Requisitos previos**

El sistema ha sido desarrollado en el sistema operativo Ubuntu 20.04, es recomendable utilizar el mismo sistema operativo para su instalación y uso. La aplicación es compatible con Ubuntu 20.04 y Windows 10, por lo tanto, el método de instalación puede variar dependiendo del sistema operativo que se quiera utilizar.

Para la instalación y mínimo uso del sistema, es necesario utilizar el lenguaje de programación Python, en concreto la versión de Python 3.8.10.

**Windows:** Para la instalación de Python en Windows hay que recurrir a la descarga del archivo desde el enlace que proporciona la pagina oficial de Python: «  
<https://www.python.org/downloads/release/python-3810/> ». En este mismo enlace, nos dirigiremos al final de la página, donde aparecerá una tabla con diferentes archivos enlazados

a un sistema operativo, en caso de Windows, lo recomendable sería descargar el archivo de ejecución: “Windows installer (64 bits)”. En la propia ejecución del archivo se mostrarán las instrucciones para proseguir con la instalación.

**Ubuntu:** Para la instalación de Python 3.8 en Ubuntu tendremos que seguir unos simples pasos:

1. Acceder a la terminal de Ubuntu, Ctrl + Alt + T.
2. Utilizar el siguiente comando para descargar Python 3.8:

```
$ sudo apt install python3.8
```

3. Verificar la correcta instalación con el comando:

```
$ python3.8 --version
```

Output

Python 3.8.0

Python finalmente está instalado y listo para utilizar.

4. En caso de querer actualizar la versión a una más reciente, se recomienda visitar la página oficial de python, donde hay instrucciones y archivos específicos para cada versión.

Con Python instalado, tendremos que utilizar “pipenv” para instalar las librerías necesarias para el sistema. Al instalar python, se ha instalado la herramienta de python llamada “pip”, la cual es un administrador de paquetes de python que usaremos para descargar “pipenv”. Primero nos aseguraremos de tener “pip” correctamente instalado:

1. Abrir la terminal correspondiente al sistema operativo.

**Windows:** Windows Powershell

**Ubuntu:** Ctrl + Alt + T

2. Usaremos este comando para asegurar que pip está instalado:

```
pip -V
```

Nos aparecerá la localización de “pip” dentro de la carpeta original de python, junto con la versión actual del administrador de paquetes.

A continuación usaremos una nueva línea en la terminal para instalar el paquete “pipenv”.

**Windows:**

```
pip install pipenv
```

**Ubuntu:**

```
sudo apt install pipenv
```

En el proceso de instalación de “pipenv”, la terminal nos guiará con el siguiente paso a tomar. Al terminar el proceso de instalación usaremos el siguiente comando para verificar la instalación del paquete:

```
pipenv -h
```

En caso de error en el proceso de instalación, se recomienda visitar los siguientes enlaces correspondientes a la instalación de “pipenv” en diferentes sistemas operativos:

« <https://www.pythontutorial.net/python-basics/install-pipenv-windows/> »

« <https://installati.one/ubuntu/20.04/pipenv/> »

Ante una duda específica sobre el proceso de descarga o instalación, así como la consulta de errores recurrentes, se recomienda acudir al enlace: « <https://es.stackoverflow.com/> » , y escribir el problema en el buscador de la página. Esta página se especializa en resolver dudas en el ámbito de la programación.

## Descarga de repositorio

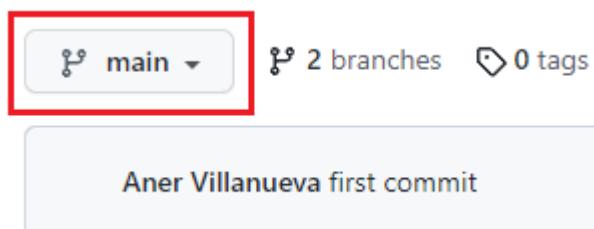
Para instalar el sistema se facilitará un enlace al repositorio en GitHub, utilizado para almacenar el proyecto, controlar sus actualizaciones, y facilitar la posterior descarga para el usuario.

A continuación se describirán una lista de instrucciones a seguir que ayudan al usuario en la descarga del repositorio.

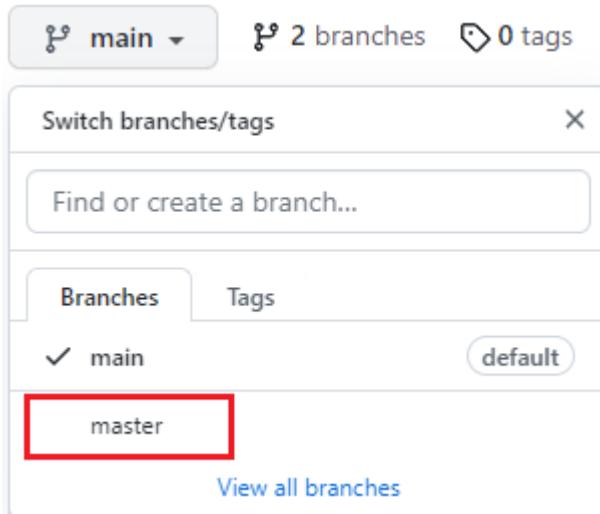
1. Acceder al enlace:

« <https://github.com/AnerVillanueva/GUI-Industry-4.0.git> »

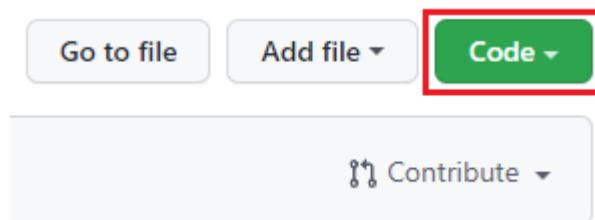
2. En la página actual, en la parte izquierda no encontraremos con este recuadro:



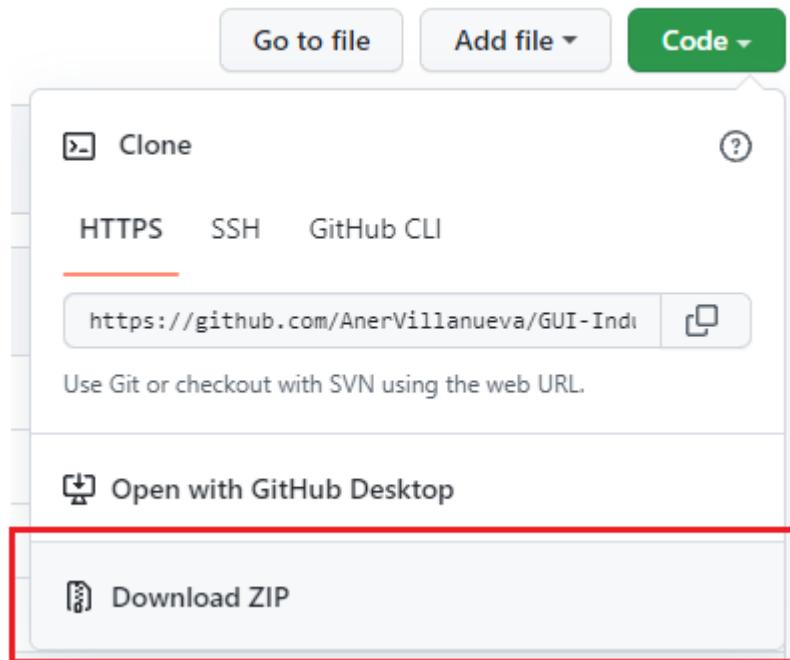
pulsaremos en “main” y aparecerán dos ramas del repositorio a elegir, y elegimos la rama “master”.



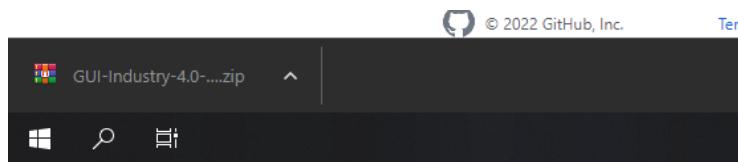
3. Dentro de esta rama, aparecerán nuevas carpetas y archivos referentes al proyecto. Estos archivos son los que se van a descargar. Para ello nos dirigimos al botón de color verde “Code”.



4. Este botón abrirá una lista de acciones. Hay varios métodos para descargar o usar un repositorio de github, pero la más sencilla es mediante la descarga de la carpeta comprimida en ZIP.



Esto descargara directamente el repositorio completo comprimido en una carpeta, que luego se podrá descomprimir usando métodos básicos como WinRAR o Zip.



## Descarga de librerías

Una vez terminada la descarga del repositorio, solo queda un requisito para poder comenzar a utilizar la aplicación, estas son las librerías. Las librerías son esencialmente herramientas que permiten al desarrollador implementar nuevas funcionalidades al código. Si un usuario o desarrollador nuevo quiere utilizar el código programado, tendrá que contar con las mismas librerías que el desarrollador anterior, ya que estas forman, en la mayoría de los casos, las funcionalidades generales del código.

Normalmente las librerías se descargan manualmente una a una, y eso sería un trabajo tedioso, ya que, este proyecto cuenta con más de 25 librerías diferentes.

Para agilizar este proceso, nos encontramos con los archivos “Pipfile” y “Pipfile.lock”, debemos asegurarnos de tenerlos en el repositorio descargado.

📁 .idea	GUI industry 4.0	18 hours ago
📁 gui_industry40	GUI industry 4.0	18 hours ago
📁 tests	GUI industry 4.0	18 hours ago
📄 .gitignore	GUI industry 4.0	18 hours ago
📄 Pipfile	GUI industry 4.0	18 hours ago
📄 Pipfile.lock	GUI industry 4.0	18 hours ago
📄 config.xml	GUI industry 4.0	18 hours ago
📄 env.sh	GUI industry 4.0	18 hours ago
📄 main.py	GUI industry 4.0	18 hours ago

Estos dos archivos se encargan de almacenar datos de las librerías instaladas en el proyecto. Antes de incluso abrir cualquier archivo de código, se recomienda instalar las librerías necesarias para el correcto funcionamiento de la aplicación.

En este momento entra en juego la herramienta previamente instalada, pipenv. Este gestor de paquetes de entornos (La palabra “entornos” hace referencia a proyectos localizados, si algo se aplica en un “entorno”, se entiende que es solamente a un proyecto focalizado, y a ningún otro) decodifica el archivo “pipfile.lock” e instala automáticamente las librerías almacenadas en el archivo “pipfile”.

Una vez entendido el funcionamiento de “pipenv” y los archivos complementarios, este es el simple proceso a seguir para la correcta instalación de librerías:

1. Abrir la terminal (funciona igual tanto para Windows como para Linux) y vamos al directorio donde se encuentra el repositorio:

```
cd myproject
```

Sustituimos “myproject” con la ubicación de la carpeta, ejemplo:

```
"cd C:\Users\Aner\Desktop\gui-industry-4.0 "
```

2. una vez nos situamos dentro del repositorio, continuamos usando el comando “pipenv shell”:

```
pipenv shell
```

Este comando creará un entorno virtual único para el proyecto, en el que se podrán instalar y desinstalar nuevas librerías.

3. Tras crear el entorno virtual, el siguiente y último paso consiste en instalar las librerías dentro del entorno, para ello usamos el siguiente comando:

```
pipenv install
```

Este comando se encarga de leer los archivos “pipfile” y “pipfile.lock” e instala automáticamente todo el contenido.

En caso de error o duda, se recomienda acudir a los siguientes enlaces:

« <https://pipenv-es.readthedocs.io/es/latest/> »

« <https://pipenv-fork.readthedocs.io/en/latest/basics.html> »

Esta es la página oficial de pipenv, ofrece explicaciones detalladas de las funcionalidades del paquete, así como el uso de los varios comandos de la terminal.



## ANEXO II

# Manual de usuario

## ÍNDICE

Requisitos previos	99
Inicio de la aplicación	100
Layout	103
Configuración	108
Excepciones, Errores y Soluciones	116

El siguiente manual contiene la información necesaria para el uso de la interfaz. Este manual es una guía práctica que aporta información sobre el inicio de la aplicación, estructuración de la interfaz y la configuración y manipulación del código. El manual cuenta con apartados de

excepciones y errores, contienen información sobre configuraciones que el usuario debería evitar para un buen funcionamiento de la aplicación.

En caso de consulta rápida, se dispone de una copia del Manual de Usuario dentro del repositorio.

## **Requisitos previos**

El sistema ha sido desarrollado en el lenguaje de programación python, en caso de no tener python instalado, se recomienda consultar el Manual de Instalación (Requisitos previos, página 1) donde se explica el procedimiento para la instalación del lenguaje de programación en Windows y Linux.

Junto con el lenguaje de programación, es necesario utilizar un entorno de desarrollo integrado (IDE) si el usuario desea programar dinámicamente y eficientemente. Estos son los IDE más recomendados: Visual Studio Code, PyCharm, SublimeText y Eclipse. Todos los IDE ofrecen funcionalidades parecidas, se recomienda escoger al gusto del usuario.

A continuación se facilitarán los enlaces oficiales a las páginas de los IDE mencionados:

- « <https://code.visualstudio.com/> »
- « <https://www.jetbrains.com/es-es/pycharm/> »
- « <https://www.sublimetext.com/> »
- « <https://www.eclipse.org/ide/> »

La aplicación se ha diseñado para soportar varios dispositivos que envían información a la interfaz de forma simultánea. El uso de ciertas funcionalidades requieren dispositivos y softwares específicos. Todos los dispositivos y softwares se mencionan en el tercer apartado de la memoria del proyecto, “Recursos necesarios”. Se recomienda el uso de todos los dispositivos y softwares mencionados para una experiencia completa con la interfaz, pero no son

necesarios, sin ellos la interfaz aún seguirá siendo funcional, aunque no mostrará cierta información asociadas con los dispositivos.

## **Inicio de la aplicación**

Para llegar a este apartado se asume que se ha consultado el Manual de Instalación y que el usuario dispone del repositorio completo, junto con Python y una IDE de su preferencia.

Con estas herramientas y contando con la propia terminal del sistema operativo, se dará inicio a la experiencia con la aplicación. Para ello, disponemos de varias maneras:

Desde la terminal del sistema operativo:

1. Abrimos la terminal.
  2. Con el comando “cd”, nos dirigimos a la carpeta que contiene el repositorio.
- ```
C:\Users\Aner>cd C:\Users\Aner\Desktop\gui-industry-4.0
```
3. Una vez dentro, usamos el comando predefinido por Python para ejecutar los archivos .py.
- ```
C:\Users\Aner\Desktop\gui-industry-4.0> py main.py
```

En el caso de que la terminal no identifique el comando “py”, se recomienda probar con los siguientes:

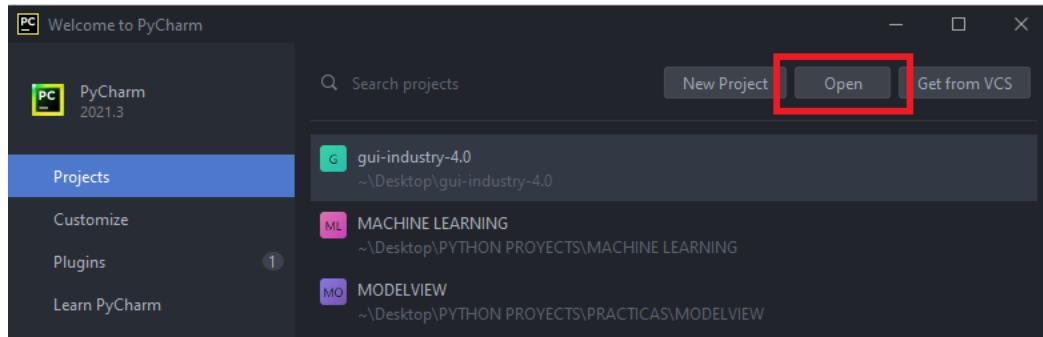
```
C:\Users\Aner\Desktop\gui-industry-4.0> python main.py
```

```
C:\Users\Aner\Desktop\gui-industry-4.0> python3 main.py
```

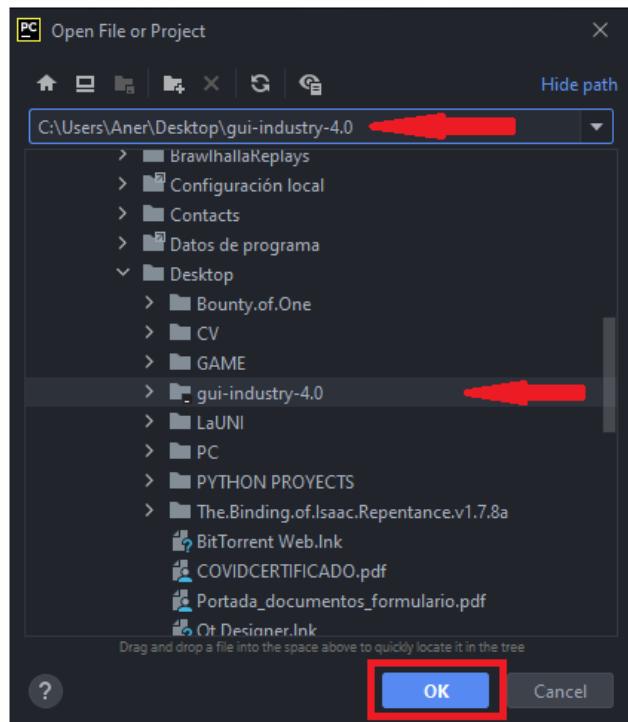
Esto puede depender del sistema operativo, de la versión de python, de la configuración en la instalación de python, etc... Para más información, acudir al enlace a continuación: «<https://stackoverflow.com/questions/50896496/what-is-the-difference-between-py-and-python-in-the-windows-terminal>»

Desde el IDE:

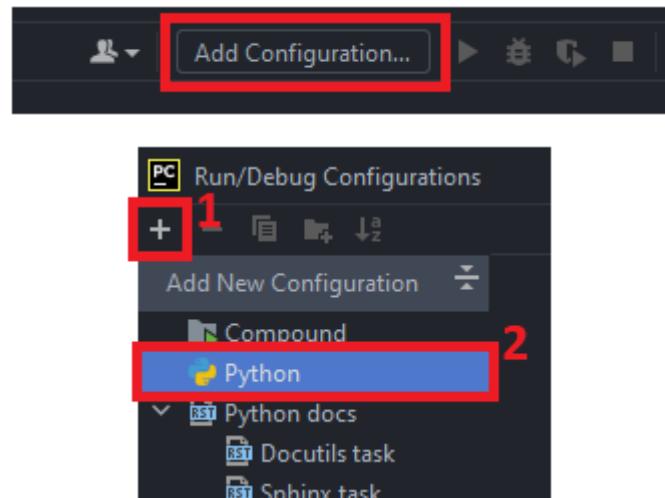
1. Abrir el IDE de preferencia (en este caso, PyCharm).
2. Abrimos el repositorio como proyecto.



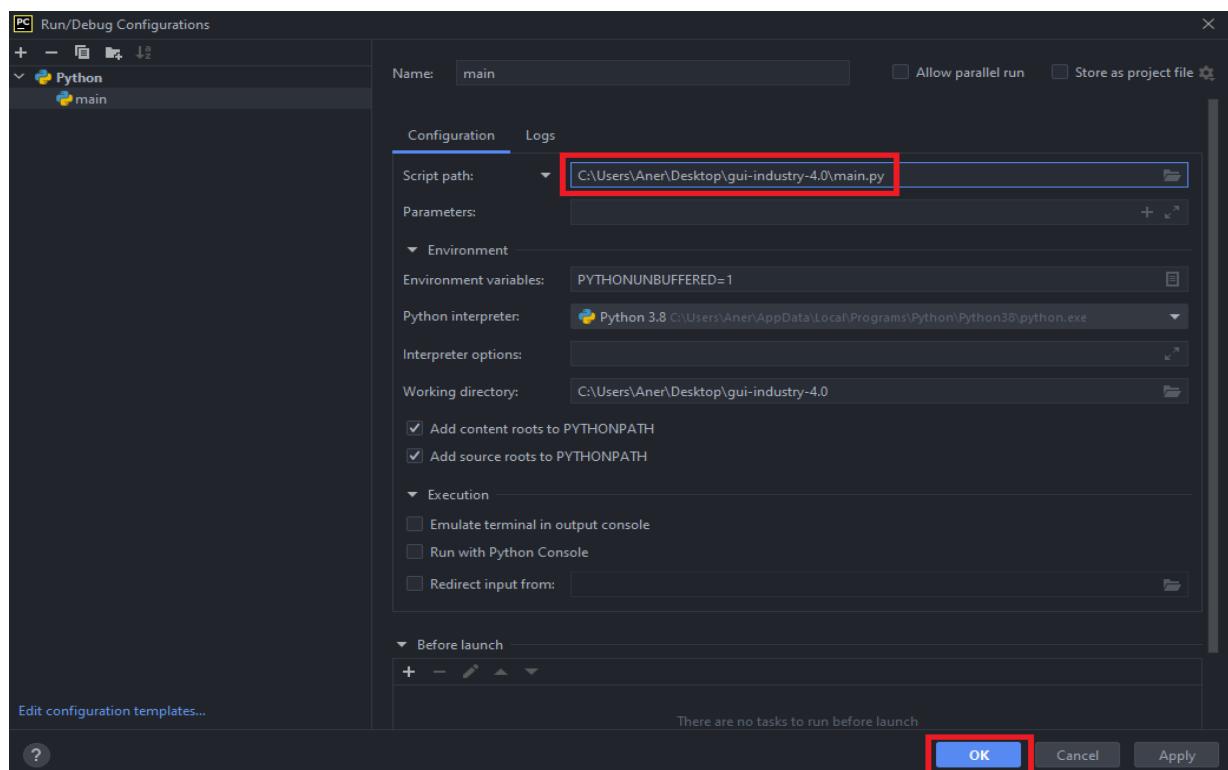
Podemos buscar el repositorio tanto por dirección o manualmente en la carpeta donde se descargó.



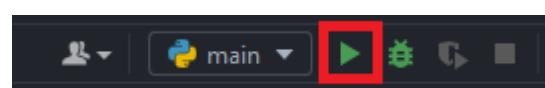
3. Creamos una configuración adecuada para la ejecución rápida de la aplicación.



Nos aseguramos de que el parámetro “Script path” identifica el archivo dentro del repositorio que se quiere ejecutar (en nuestro caso, “main.py”).



4. Una vez definida la configuración, nos dirigimos al icono que ahora se muestra en verde adyacente al ícono de configuración, el cual, ejecutará la aplicación.



## Layout

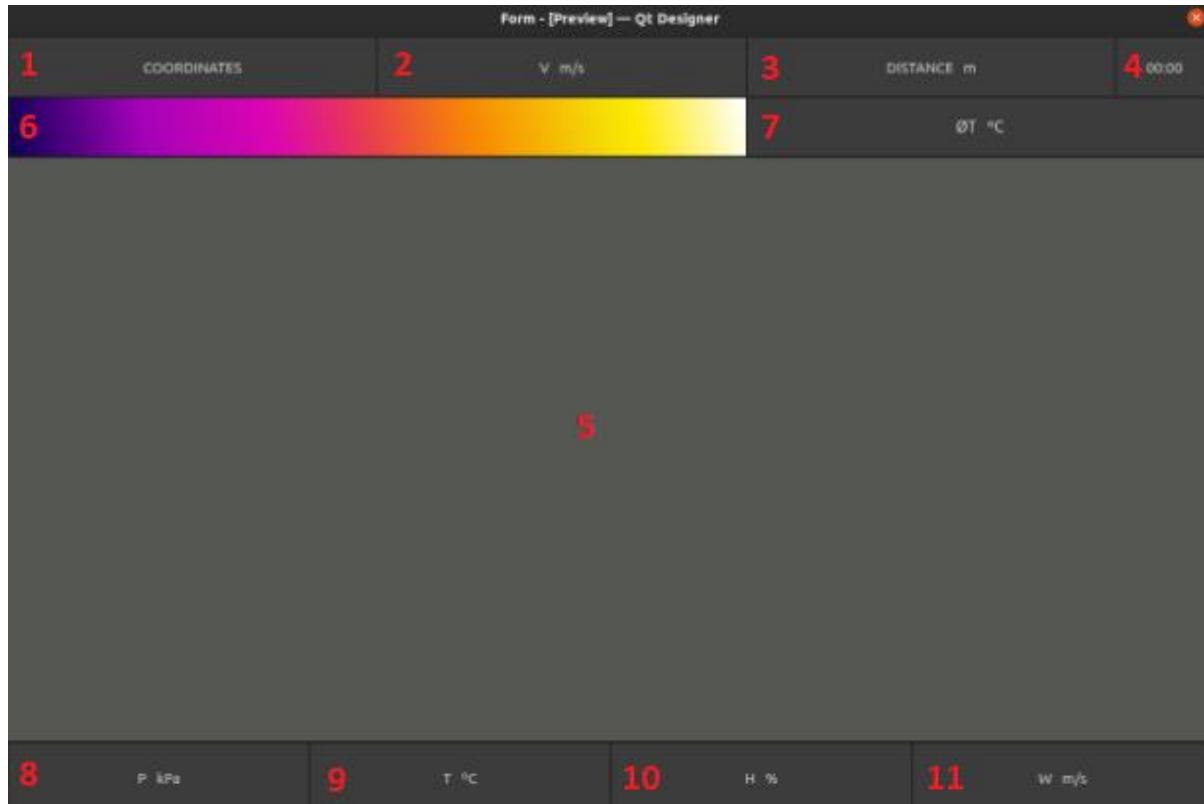
### MainWindow - Ventana principal/base



Esta GUI está diseñada específicamente para ser la ventana principal o ventana base. Diseñada a partir del concepto inicial. Posee un mecanismo de pestañas, el cual permite el cambio a diferentes páginas. Esta estructura facilita visualizar y gestionar diferentes GUIs en las pestañas seleccionadas.

El nombre de las pestañas actuales no suponen ningún tipo de información útil.

## Layout GUI



Información de geolocalización:

- 1. **"COORDINATES"**: Coordenadas del sensor. Latitud y Longitud
- 2. **"V m/s"**: Velocidad de movimiento del sensor.
- 3. **"Distance"**: Distancia recorrida.

Información Horaria:

- 4. **"Time"**: Hora y minutos actuales.

Información de video:

- 5. **Widget Central**: Implementación de imagen.
- 6. **Gradiente de color (Camara Termica)**: Específica para la camara termica, sirve para el analisis termico de la imagen.

- 7. “ $\bar{O}T \text{ } ^\circ C$ ” (*Cámara Térmica*): Específica para la cámara térmica, muestra la temperatura media de la medición.

Información meteorológica:

- 8. “ $P \text{ kPa}$ ”: Presión atmosférica, unidad de kiloPascales
- 9. “ $T \text{ } ^\circ C$ ”: Temperatura ambiente
- 10. “ $H \%$ ”: Humedad ambiente
- 11. “ $W \text{ m/s}$ ”: Velocidad de viento

## PLC GUI

VARIABLE NAME	VALUE	DATA TYPE	SOURCE TIMESTAMP	SERVER TIMESTAMP	STATUSCODE
1	2	3	4	5	6

- 1. **Variable name:** Nombre de la variable suscrita, permite ordenar la tabla alfabéticamente según el nombre. [ NAME ]
- 2. **Value:** Valor que poseen las variables. [ True / False / XXXX / ... ]
- 3. **Data type:** Tipo de dato de las variables. [ Int / String / Bool / ... ]
- 4. **Source TimeStamp:** Hora exacta de actualización en el cliente [ HH:MM:SS ]
- 5. **Server TimeStamp:** Hora exacta de actualización en el servidor [ HH:MM:SS ]

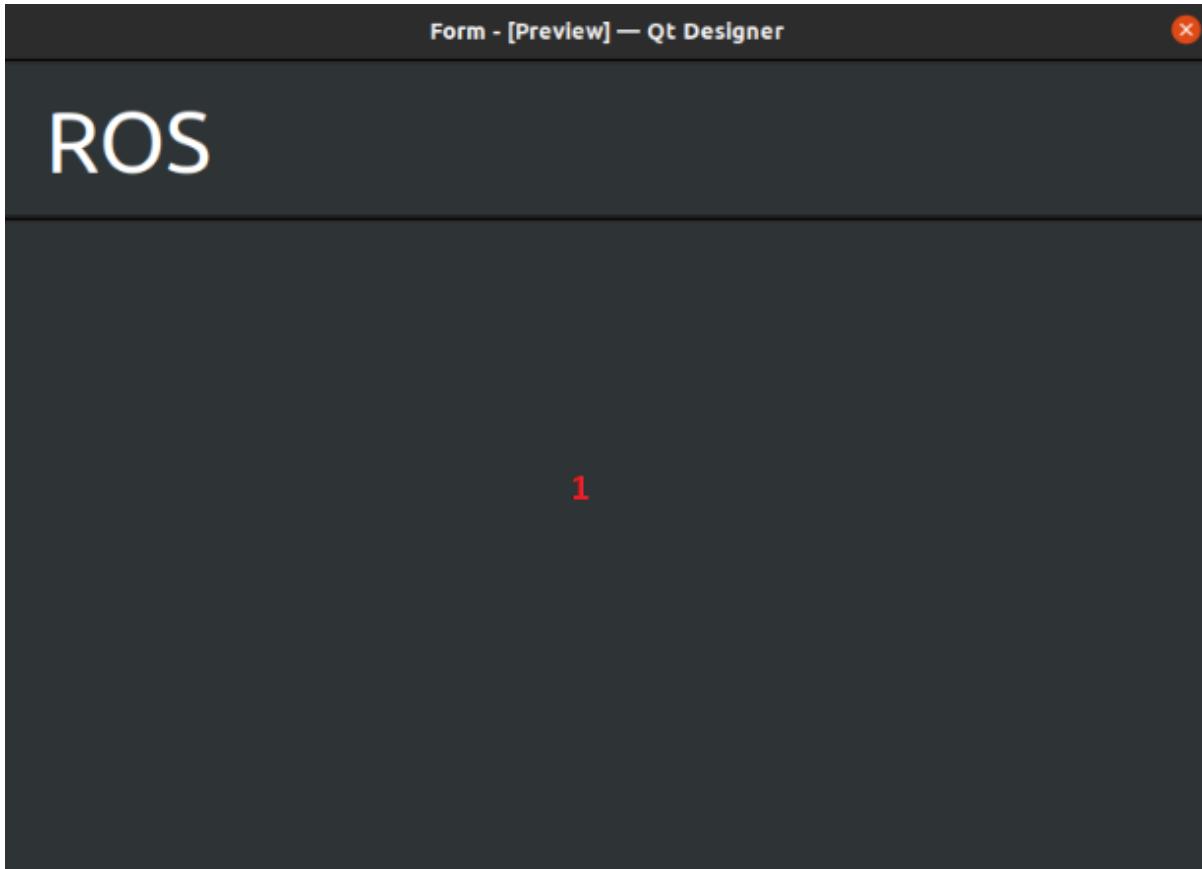
- **6. StatusCode:** Estado de la variable, representa la correcta suscripción a la variable.  
[ GOOD / BAD ]

## IDS GUI



- **1. Widget Central:** implementación de imagen, en blanco y negro. Una imagen por página. Útil para centralizar la atención en una imagen a tiempo real.

## ROS GUI



- **1. Widget Central:** Mensajes recibidos desde el protocolo ROS. Útil para la rápida y constante visualización del estado de la máquina.

## Configuración

Este apartado se centrará particularmente en el archivo de configuración que contiene el repositorio. Se recomienda consultar el apartado de la memoria “**5.2.1.1 Backend**” para facilitar el entendimiento y la estructuración de este archivo.

El archivo de configuración está basado en el lenguaje de programación XML, con estructura de árbol. Esta estructura supone una jerarquía, que se compone de nodos y atributos.

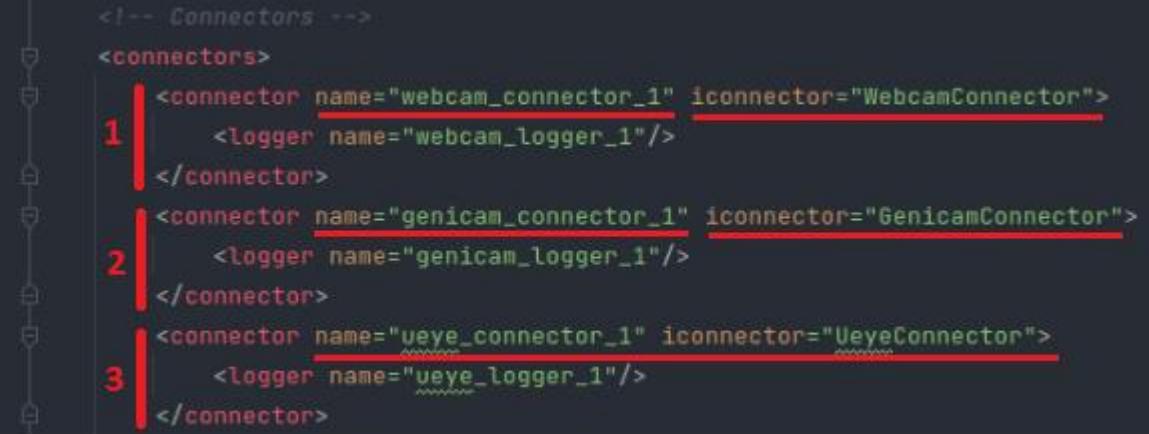
Los colores representan diferentes propiedades dentro de la estructura del lenguaje:

- Nodos
- Atributos
- Valores que definen los atributos
- Valores que definen los nodos

Para una edición común, donde el usuario quiera cambiar propiedades y atributos básicos, se deben editar únicamente los valores con colores **Verde** y **Blanco**.

Se recomienda conocer el funcionamiento y estructuración del lenguaje XML en caso de querer crear nuevas secciones en el archivo, también es recomendable revisar el código fuente para familiarizarse con los nombres de los atributos, y entender dónde se utilizan estas referencias.

## Edición del archivo - Conectores



```
<!-- Connectors -->
<connectors>
    <connector name="webcam_connector_1" iconnector="WebcamConnector">
        <logger name="webcam_logger_1"/>
    </connector>
    <connector name="genicam_connector_1" iconnector="GenicamConnector">
        <logger name="genicam_logger_1"/>
    </connector>
    <connector name="ueye_connector_1" iconnector="UeyeConnector">
        <logger name="ueye_logger_1"/>
    </connector>
```

La imagen muestra tres de los varios conectores que contiene el archivo.

Estas pequeñas secciones delimitadas por la franja vertical roja, definen el conector y sus propiedades, por lo que el usuario puede añadir ilimitados conectores sin tener que llegar a usarlos en la interfaz.

En caso de querer crear un nuevo conector, el usuario deberá seguir la estructura predefinida por el ejemplo (`<connector name=" ... " iconnector=" ... ">`) y terminar la estructura como se muestra (`</connector>`). En otras secciones del archivo se utilizaran estos mismo **Valores** referenciarse entre ellos, por lo que se recomiendan nombres que tengan alguna referencia a sus propiedades.

El valor “`iconnector`” tiene que coincidir con el nombre de la clase de los conectores en específico, por ejemplo: el valor “`iconnector`” del conector webcam, lo conseguiremos dirigiéndonos al script “`Webcam_connector.py`” y leyendo el nombre de la clase del conector.

```
20
21 class WebcamConnector(Connector):
22
```

A continuación copiaremos ese mismo nombre, y lo usaremos como valor “`iconnector`”, así, el programa identificará a que script hace referencia ese conector, y podrá utilizarlo a lo largo de la ejecución.

Si el conector en específico requiere de un “logger”, “parser” o un formato, se le aplicará al mismo conector en una jerarquía inferior, esto se consigue intentando, tal y como se muestra en la imagen.

```
15 <connector name="rs232_connector_1" iconnector="RS232Connector">
16   |<parser name="rs232_parser_1"/>
17   |<logger name="rs232_logger_1"/>
18   |<format string="COORDINATES : {} , {}" values="DS,TS"/>
19 </connector>
```

Al definir un “parser” o “logger”, estos se harán de la misma manera que se definen los conectores, asignándole el atributo “name” como el ejemplo (`<parser name=“...“ />`).

En el caso de querer asignarle un formato, el usuario deberá definir dos atributos:

- **String:** formato de texto en el que se mostrará el mensaje.
- **Values:** Datos del mensaje.

En el atributo “string” se definirá el formato de texto escrito que usará el Widget para mostrar los datos del mensaje recibido, en él, sustituimos la ubicación de los datos por “{}”, así el programa identifica donde añadir los datos. “Values” define los datos del mensaje que se insertarán en el formato de texto, en el mismo orden en el que se definen, se implementaran en el texto. El usuario debe conocer el protocolo que usa el conector en particular para enviar

datos, o al menos, tiene que conocer las palabras clave que usa el parser para transcribir esos datos y guardarlos, como por ejemplo: DS y TS.

## Edición del archivo - PLC y ROS

Algunos conectores no solo dependen de un “parser” o un “logger”, estos requieren de unos parámetros adicionales. El origen de estos parámetros depende del protocolo que usan ciertos dispositivos o softwares, por ejemplo: ROS y OPC UA, y el usuario deberá conocer los parámetros adicionales que requieren los nuevos conectores que se implementen.

### ROS

```

30      □
31      □
32      □
33      □
34      □
35      □
36      □
37      □

<connector name="ros_connector_1"  iconnector="ROSTopicConnector">
    <logger name="ros_logger_1"/>
    <parameters>
        <parameter name="topic_name" value="My_topic"/>
        <parameter name="comm_mode" value="subscriber"/>
        <parameter name="msg_type" value="JointState"/>
    </parameters>
</connector>
```

Para más información acerca de lo que representan los parámetros, se recomienda leer el apartado de la memoria “**2.1 ROS**” y “**5.2.1.1 Backend (Conector específico: Conector ROS)**”.

### OPC UA

```

38      □
39      □
40      □
41      □
42      □
43      □
44      □
45      □
46      □

<connector name="opcua_connector_1"  iconnector="OPCUAConnector">
    <logger name="opcua_logger_1"/>
    <parameters>
        <parameter name="url" value="opc.tcp://10.1.24.225:4840"/>
        <parameter name="read" value="True"/>
        <parameter name="variable_node" value="ns=3;i=54053"/>
        <parameter name="variables" value=" "/>
    </parameters>
</connector>
```

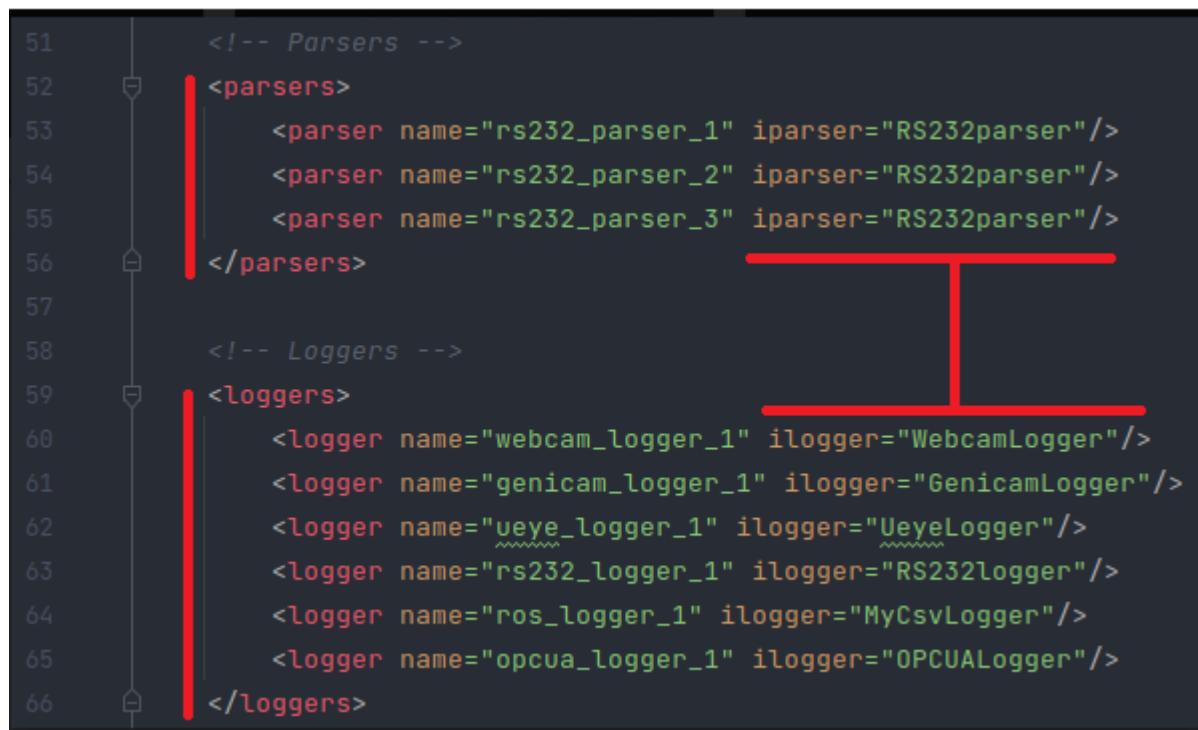
- **url:** Dirección IP del PLC, enlazada con el protocolo de OPC UA.
- **read:** Propiedad que habilita la lectura del PLC. [ TRUE / FALSE ]

- **variable\_node:** Dentro de la raíz del PLC, valor del nodo que almacena todas las variables dentro del PLC ( consultar vía UA Expert o u-create web ).
- **variables:** Nombres de la variables que el usuario quiere suscribirse [ ARRANQUE, PARO, SALIDA1, SALIDA2 ... ]. En caso de dejar un espacio en blanco, OPC UA se suscribirá a todas las variables almacenadas en el nodo.

Para más información acerca de lo que representan los parámetros, se recomienda leer el apartado de la memoria “**5.2.1.1 Backend (Conector específico: Conector OPC UA)**”.

## Edición del archivo - Parser y Logger

Continuando con el archivo, vemos que la sección de conectores se acaba, como se puede ver, cada sección en el documento xml se delimita por el comentario “`<!-- . . . -->`”.



```
51      <!-- Parsers -->
52      <parsers>
53          <parser name="rs232_parser_1" iparser="RS232parser"/>
54          <parser name="rs232_parser_2" iparser="RS232parser"/>
55          <parser name="rs232_parser_3" iparser="RS232parser"/>
56      </parsers>
57
58      <!-- Loggers -->
59      <loggers>
60          <logger name="webcam_logger_1" ilogger="WebcamLogger"/>
61          <logger name="genicam_logger_1" ilogger="GenicamLogger"/>
62          <logger name="ueye_logger_1" ilogger="UeyeLogger"/>
63          <logger name="rs232_logger_1" ilogger="RS232logger"/>
64          <logger name="ros_logger_1" ilogger="MyCsvLogger"/>
65          <logger name="opcua_logger_1" ilogger="OPCUALogger"/>
66      </loggers>
```

Estas secciones definen todos los parser y loggers que poseen los conectores.

Al igual que la sección de conectores, esta estructura sirve para definir un parser / logger y sus propiedades, aquí se crean las referencias que se utilizan al añadirles parser y logger a los conectores ya mencionados, por eso se aprecia que los nombres coinciden, eso debe de ser así para que la aplicación identifique que un logger específico se le implementa a un conector en específico.

Al igual que “**iconnector**”, los valores “**iparser**” e “**ilogger**” tienen que coincidir con el nombre de las clases a las que referencian ( nótese que al haber tres parser de rs232, se puede utilizar un solo **iparser**, esto es totalmente compatible ), por ejemplo:

```
 6
 7     class RS232parser(Parser):
 8         def __init__(self):
 9             super(RS232parser, self).__init__()
 5
 6     class RS232logger(Logger):
 7         def __init__(self):
 8             super(RS232logger, self).__init__()
```

Para cambiar estos valores y/o crear unos nuevos, el usuario tiene que ser totalmente consciente de los nombres de las nuevas clases creadas, o en caso de editarlos, cambiar manualmente el código para que coincidan individualmente con los nombres nuevamente elegidos.

## Edición del archivo - Configuración de ventanas

Por último, esta sección situada en el final del archivo, se utiliza para configurar la disposición de las ventanas.

```

68      <!-- Tabs -->
69      <tabs>
70          <tab name="Layout" page="0">
71              <widget iwidget="LayoutWidget">
72                  <gui>LayoutUI</gui>
73                  <connection connector="ueye_connector_1" widget="label"/>
74                  <connection connector="rs232_connector_1" widget="Coordinates"/>
75                  <connection connector="rs232_connector_2" widget="Velocity"/>
76                  <connection connector="rs232_connector_3" widget="Distance"/>
77                  <connection connector="time_connector_1" widget="Hora"/>
78          </widget>
79      </tab>
80      <tab name="PLC" page="1">
81          <widget iwidget = "PLCWidget">
82              <gui>PLC_TableView</gui>
83              <connection connector="opcua_connector_1" widget="" />
84          </widget>
85      </tab>
86      <tab name="IDS" page="2">
87          <widget iwidget = "IDSWidget">
88              <gui>ids</gui>
89              <connection connector="webcam_connector_1" widget="label"/>
90          </widget>
91      </tab>
92      <tab name="ROS" page="3">
93          <widget iwidget="ROSWidget">
94              <gui>ROS</gui>
95              <connection connector="ros_connector_1" widget="label"/>
96          </widget>
97      </tab>
98  </tabs>

```

En una jerarquía menor, se aprecian 4 sub-secciones, con **nodos** denominados como “**tab**”. Cada uno de estos “**tab**” representa una página dentro de la interfaz, y sus atributos “**name**” y “**page**” representan el nombre de la página y la posición de la página respectivamente.

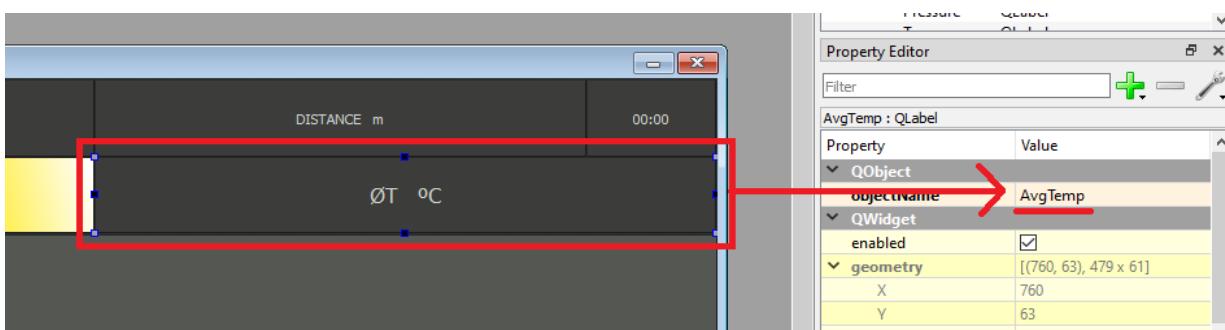
Por debajo se le asigna un “**widget**” a esa página, este “**widget**” sirve para representar que **iwidget** usará la página ( El cual, como en secciones anteriores, tiene que coincidir con la clase

exclusiva de su script propio), que se le asigna al **iwidget**, y que conectores debe incluir el **iwidget** en la guía, y cada conector a qué **widget** se implementara.

El valor incluido dentro del nodo “**gui**” debe coincidir con el nombre del archivo .ui que queramos utilizar, por ejemplo: LayoutUI hace referencia al archivo “LayoutUI.ui” dentro del repositorio.

El nombre del **connector**, tiene que coincidir con el anteriormente definido en la primera sección del archivo, así, el programa identificará la referencia.

El valor del atributo “**widget**” se utiliza para saber a qué widget de la gui aplicar el mensaje que el conector reciba. Para saber los nombres de los diferentes widget, el usuario deberá abrir el archivo .ui y mirar manualmente el nombre del widget en cuestión, tal y como se muestra en la siguiente imagen.



OPC UA no requiere de widget dado a que su IWidjet se ha desarrollado con el método de Modelo Vista Controlador, el cual ya genera un widget específico para el conector.

## **Excepciones, Errores y Soluciones**

Este apartado intenta recopilar todas las excepciones y errores que pueden surgir por una edición o uso inapropiado de la aplicación.

La aplicación, aunque terminada, se puede decir que está en una fase beta. Su continuo desarrollo hace que la primera versión pueda ser la más inestable. Se recomienda que para la edición del código fuente, el usuario tenga nociones básicas de programación en Python. Durante el desarrollo del código, se anotaron comentarios explicativos sobre el código y las razones por las que se desarrolló de esa manera. Junto con una estructuración simple y organizada, todo aquel que entre al código fuente podrá entender el programa y posteriormente, editarlo sin que cause ningún problema.

Teniendo esto en cuenta, estas son las excepciones y los errores más frecuentes durante el desarrollo y uso de la aplicación:

### **Utilizar el conector sin su dispositivo o software**

Este error ocurre cuando intentamos ejecutar la aplicación sin haber editado bien el archivo de configuración.

Intentamos ejecutar la aplicación sin disponer de todos los dispositivos y softwares configurados en el apartado de conectores, y el sistema falla al intentar establecer la conexión con los dispositivos.

### **¿Solución?**

En la sección de conectores del archivo de configuración, eliminamos o comentamos los nodos **connector** responsables de los dispositivos o protocolos que no tenemos conectados en el PC, este paso no es necesario con todos los conectores, pero si es recomendable ya que ocupan espacio útil en la memoria.

Y en la sección de ventanas, eliminamos el nodo **connection** correspondiente con ese conector

### **El conector no identifica el valor en el archivo de configuración**

Este error ocurre cuando los atributos **iconnector** en la sección de conectores, no son idénticos al nombre de la clase de su propio conector, especialmente en los conectores OPC UA y ROS, dado que estos conectores utilizan métodos distintos de conexión.

#### **¿Solución?**

Revisar el nombre de las clases de los conectores antes de editar o definir nuevos conectores en el archivo de configuración.

Se recomienda acudir al apartado **edición del archivo - Conectores** del manual, en este apartado se explica donde hay que mirar para conocer el nombre de la clase y referenciarla en el archivo.

### **Uso de dos cámaras con el mismo protocolo de conexión**

Este error es posible que se dé cuando se utilizan dos cámaras con el mismo protocolo de conexión, no es un error crítico, ya que la aplicación sigue su ejecución normal, pero ya no enseñara una de las dos cámaras. Además de no ser crítico, es un error que sucede con poca frecuencia, aun utilizando dos cámaras.

Sucede porque las dos cámaras pueden llegar a utilizar el mismo espacio en la memoria, es por eso que la aplicación a veces decide solo utilizar los fotogramas que llegan desde una de las cámaras.

### ¿Solución?

Intentar evitar el uso de cámaras con el mismo protocolo de conexión, por ejemplo: dos USB 3.0, dos USB 2.0 o dos Ethernet.

En caso de que se repita el error, intentar configurar desde la sdk de la cámara la frecuencia de captura de fotogramas a una que no coincida con la otra cámara. Esto debería evitar el solapamiento en la memoria.

### Error de compatibilidad ROS + OPC UA

Este no es un error crítico, la ejecución de la aplicación se efectúa con normalidad, pero los conectores de OPC UA y ROS no muestran información.

Esto sucede porque los primeros mensajes de OPC UA se solapan con los primeros mensajes de ROS.

### ¿Solución?

Este error está actualmente solucionado.

Si en algún caso llega a suceder este error, se debe cambiar el valor del periodo de suscripción de OPC UA a 150 o 200, en el script “opcua\_connector.py”, en el lugar donde indica la imagen:

```
76     def read(self, variables):
77
78         handler = Handler(self) # instance of the Handler class
79         sub = self.client.create_subscription(175, handler) # creates a sub creation instance using the handler
80
```

### Uso de IWidget repetidos

Este no es un error crítico, la ejecución de la aplicación se efectúa con normalidad.

Este error sucede cuando el usuario configura las ventanas de tal manera que dos ventanas utilizan el mismo IWidget. Esto conlleva que unos conectores deberán asociarse al IWidget de una ventana, y otros conectores deberán asociarse al IWidget de la otra ventana. Como el IWidget es idéntico, el sistema no tiene forma de distinguir entre uno y otro. Esta confusión genera que el sistema asocie todos los conectores al último de los IWidget repetidos.

### **¿Solución?**

Evitar el uso de repetidos IWidgets para diferentes ventanas. En caso de necesitar utilizar un IWidget idéntico, se recomienda clonar el IWidget y utilizar este nuevo IWidget con otro nombre, así el sistema podrá identificar la diferencia.

En caso de duda específica sobre el funcionamiento de alguna librería o alguna funcionalidad añadida en el código, así como la consulta de errores recurrentes, se recomienda acudir al enlace: « <https://es.stackoverflow.com/> » , y escribir el problema en el buscador de la página. Esta página se especializa en resolver dudas en el ámbito de la programación.

Si la duda es sobre el uso y edición del código fuente, se recomienda contactar con el desarrollador principal vía correo: a.villanueva0046@gmail.com

## **ANEXO III**

### **Actas de reunión**

#### **ÍNDICE**

Acta de reunión I	122
Acta de reunion II	123
Acta de reunión III	124
Acta de reunión IV	125
Acta de reunión V	126



# Acta de reunión I

FECHA:	11/02/2021	ASISTENTES:
LUGAR:	Sala de reuniones, Tekniker	Aner Villanueva Ander Ansuategui Carlos Tubio

## PUNTOS A TRATAR:

Diseño de un prototipo de archivo de configuración

Diseño y creación de la ventana principal

Diseño y creación de la estructura principal de la interfaz

## PUNTOS DE ACCIÓN:

Generar un prototipo de archivo de configuracion basado en el lenguaje XML.

Crear mediante Qt el diseño de la ventana principal y la estructuración misma de la interfaz.



# Acta de reunión II

FECHA:	4/03/2021	ASISTENTES:
LUGAR:	Sala de reuniones, Tekniker	Aner Villanueva Ander Ansuategui Carlos Tubio

## PUNTOS A TRATAR:

Crear e implementar un conector compatible con una camara webcam

Crear e implementar un conector compatible con el protocolo de ROS

Crear e implementar un conector compatible con una conexion rs232 de GPS

## PUNTOS DE ACCIÓN:

Haciendo uso de una camara webcam y un dispositivo GPS, junto con un servidor de ROS, se implementara a la interfaz una serie de conectores compatibles con los datos recibidos.

# Acta de reunión III

FECHA:	1/04/2021	ASISTENTES:	
LUGAR:	Sala de reuniones, Tekniker	Aner Villanueva Ander Ansueategui Carlos Tubio	

## PUNTOS A TRATAR:

- Crear e implementar un conector compatible con una camara de la empresa IDS
- Crear e implementar un conector compatible con el la interfaz de camaras GenICam
- Crear e implementar un conector compatible con la comunicacion bidireccional
- Usar un cliente-servidor de un PLC como comunicacion bidireccional

## PUNTOS DE ACCIÓN:

Avanzando con la compatibilidad con las camaras, se realizaran nuevas implementaciones con distintas camaras, para ello se hara uso de un nuevo conector para la interfaz GenICam.  
Se hara uso del metodo Modelo Vista Controlador para el diseño de una interfaz interactiva entre cliente y servidor para un PLC.

# Acta de reunión IV

FECHA:	2/05/2021	ASISTENTES:
LUGAR:	Sala de reuniones, Tekniker	Aner Villanueva Ander Ansuategui Carlos Tubio

## PUNTOS A TRATAR:

IWidgets, diseño y estructuración

Implementación de las GUIs

Test de conectores y dispositivos

Yapsy, implementación de plugins

## PUNTOS DE ACCIÓN:

Crear y diseñar los IWidgets, programar con una funcionalidad genérica para varios dispositivos y conectores.

Implementar todas las GUIs actuales, verificar el correcto funcionamiento de los conectores con sus respectivos dispositivos.

Utilizar Yapsy como método de añadir plugins al proyecto, con intención de hacerlo más dinámico.

# Acta de reunión V

<b>FECHA:</b> 27/06/2021	<b>ASISTENTES:</b> Aner Villanueva Ander Ansuategui Carlos Tubio
<b>LUGAR:</b> Sala de reuniones, Tekniker	

## PUNTOS A TRATAR:

Archivo de configuración, necesita una actualización de estructura

Documentación de las dependencias de la aplicación

TFG: estructura y contenido.

## PUNTOS DE ACCIÓN:

Actualizar el archivo de configuración en una estructura mas simple en XML, añadir nuevas funcionalidades acordes a las nuevas implementaciones.

Documentar dependencias y contenido de la aplicación en GitLab.

Revisar y corregir cualquier error e imperfección en la aplicación. Finalización del proyecto.

Comenzar a redactar la memoria del TFG, haciendo revisiones periodicas de la estructura y contenido del mismo.