Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

LABORATORIO REMOTO DE UNA PLANTA PILOTO DE DOS TANQUES COMUNICANTES

Autor:

Alonso Manrique, Gabriel

Tutor:

Zamarreño Cosme, Jesús María

Ingeniería de Sistemas y Automática

Valladolid, marzo 2025.

Agradecimientos

A mi familia, por su constante apoyo y confianza.

A mis compañeros y amigos, por creer en mí y hacer de mi paso por la universidad una experiencia inolvidable.

A mi tutor, Jesús María Zamarreño, por su increíble paciencia, disponibilidad y asistencia durante todo el desarrollo de este proyecto.

A la Universidad de Valladolid, por brindarme la oportunidad de estudiar esta carrera.

Resumen

Se ha desarrollado un laboratorio remoto para realizar experimentos sobre una planta piloto de dos tanques comunicados por la parte inferior a través de una tubería. Se ha buscado un comportamiento lo más parecido posible al software de control JavaRegula instalado en los ordenadores del laboratorio de Control de Procesos. En la planta, se actúa sobre el caudal de una bomba que lleva agua desde una bandeja situada en la base de ambos tanques hacia la parte superior del primero de estos; a su vez, mediante una válvula manual situada en el segundo tanque, se devuelve agua a la citada bandeja. Se puede realizar un control manual o con PID del nivel del segundo depósito. Se tiene una estructura con un servidor, que hace de intermediario entre la planta y el cliente, y un cliente donde se puede visualizar la planta y los datos recogidos en gráficas, así como manipular los parámetros necesarios para llevar a cabo el control de la planta de forma remota. Los datos de los experimentos realizados pueden ser exportados para un posterior análisis.

Palabras clave

Laboratorio remoto, simulación de sistemas, tanques comunicantes, control de procesos, fundamentos de automática.

Abstract

A remote laboratory has been developed to perform experiments on a pilot plant with two tanks connected at the bottom through a pipe. The aim was to achieve a behavior as similar as possible to the JavaRegula control software installed on the computers of the Process Control laboratory. In the plant, the flow rate of a pump that carries water from a tray located at the base of both tanks to the top of the first one is manipulated; in turn, by means of a manual valve located in the second tank, water is returned to the same tray. Manual or PID control of the level in the second tank is possible. There is a structure with a server, which acts as an intermediary between the plant and the client, and a client where the plant and the data collected in graphs can be visualized, as well as manipulating the necessary parameters to carry out the control of the plant

remotely. The data of the experiments performed can be exported for further analysis.

Keywords

Remote laboratory, system simulation, communicating tanks, process control, fundamentals of automation.

Índice del contenido

CAPÍTULO 1: Introducción y Objetivos 23

1.1. Introducción 23

1.2. Objetivos 24

CAPÍTULO 2: La planta real del laboratorio 25

2.1. Funcionamiento 25

2.2. Componentes de la planta 26

2.2.1. Depósitos 26

2.2.2. Fuentes de alimentación 26

2.2.3. Bomba 26

2.2.4. Sensor de nivel 28

2.2.4.1. Sensor de nivel mediante presión 28

2.2.4.2. Sensor de nivel capacitivo 29

2.2.5. Caudalímetro 30

2.2.6. Variador de frecuencia 30

2.2.7. Tarjeta de adquisición de datos 31

CAPÍTULO 3: Fundamentos y Herramientas 33

3.1. Estructura de control 33

3.1.1. Control manual 34

3.1.2. Control PID 34

3.1.2.1. Acciones del control de un PID 35

3.1.2.2. Ventajas y limitaciones de los controladores PID 37

3.1.2.3. Sintonización manual del PID 37

3.1.2.4. Efecto windup y limites en los actuadores 38
3.1.2.5. Controlador PID con filtro en la acción derivativa 39
3.1.2.6. Controlador PID de JavaRegula 41
3.2. Easy JavaScript Simulation y el protocolo RIP 43
3.2.1. Introducción 43
3.2.2. Protocolo RIP 43
3.2.3. Estructura de EJSS 44
3.2.3.1. Ventana Descripción 45
3.2.3.2. Ventana Modelo 45
3.2.3.3. Ventana HtmlView 47
3.2.3.4. Barra lateral de tareas 47
3.3. Yawcam. 48
CAPÍTULO 4: Implementación del laboratorio remoto 49
4.1. Estructura del laboratorio 49
4.2. Lado servidor 50
4.2.1. Servidor RIP de Python 50
4.2.1.1. RIPMatlab.py 51
4.2.1.2. RIPGeneric.py 51
4.2.1.3. AppConfig.py 51
4.2.1.4. App.py 54
4.2.2. Script de Matlab 54
4.2.2.1. Inicialización y manejo de variables 54
4.2.2.2. Inicio, manejo y cierre de las sesiones con los clientes 55
4.2.2.3. Fichero de configuración 56

4.2.2.4. Conexión OPC....57
4.2.2.5. Control de la planta....57
4.3. Lado cliente....60
4.3.1. Ventana modelo en EJSS del cliente....60
4.3.1.1. Variables....60
4.3.1.2. Inicialización....64
4.3.1.3. Relaciones fijas....65
4.3.1.4. Propio....72
4.3.1.5. Elemento RIP....73
4.3.2. Ventana HTML en EJSS del cliente....77
4.3.2.1. Panel barra de herramientas....77
4.3.2.2. Panel opciones interfaz y controlador....84
4.3.2.3. Panel visualización....94
4.3.3. Empaquetado y cambios adicionales....100
4.3.3.1. Empaquetado del cliente....100
4.3.3.2. Cambios adicionales....102
CAPÍTULO 5: Validación del laboratorio remoto....107
5.1. Validación del control manual....107
5.2. Validación del PID estilo JavaRegula....110
5.3. Validación del PID ideal....112
5.4. Conclusiones....113
CAPÍTULO 6: Instalación y puesta en marcha del servidor....115
6.1. Software necesario....115
6.1.1. Instalación de Matlab....115

6.1.2. Instalación de Python 115
6.1.3. Creación y preparación del entorno virtual de trabajo 119
6.1.4. Instalación de Yawcam. 120
6.2. Puesta en funcionamiento del servidor 122
6.2.1. Comprobaciones previas 122
6.2.2. Arranque del servidor RIP 125
6.2.2.1. Primera forma de arranque del servidor 125
6.2.2.2. Segunda forma de arranque del servidor 125
6.2.3. Arranque del servidor de la cámara 126
6.3. Posibles fallos y cómo solucionarlos 127
6.3.1. Error al instalar los módulos del entorno virtual 127
6.3.2. Errores al lanzar App.py 128
6.3.2.1. No Python at 128
6.3.2.2. ModuleNotFoundError 128
CAPÍTULO 7: Manual de usuario del cliente 129
7.1. Elementos de la interfaz 129
7.1.1. Menú superior 129
7.1.1.1. Botón Descripción 129
7.1.1.2. Selector del tipo de control 129
7.1.1.3. Botón Guardar 130
7.1.1.4. Botón Configuración 130
7.1.1.5. Botón Conectar 130
7.1.1.6. Botón Desconectar 130
7.1.1.7. Icono información 131

7.2. Menú de opciones 131
7.2.1. Panel opciones 131
7.2.2. Paneles parámetros de control 132
7.2.2.1. Panel de control manual 132
7.2.2.2. Panel de opciones PID 132
7.3. Visualización de la planta 132
7.3.1. Imagen de la planta real 133
7.3.2. Gráficas 133
7.4. Instrucciones de funcionamiento 134
7.4.1. Configuración 134
7.4.1.1. Google Chrome 135
7.4.1.2. Firefox 136
7.4.1.3. Microsoft Edge 137
7.4.2. Conectar 138
7.4.3. Desconectar 138
CAPÍTULO 8: Conclusiones y trabajo futuro 139
8.1. Conclusiones 139
8.2. Trabajo futuro 140
CAPÍTULO 9: Bibliografía 141
CAPÍTULO 10: Anexos 143

Índice de figuras

Figura 1. Montaje de una de las plantas del laboratorio 25

Figura 2. Fuentes de alimentación ML30 de Puls 26

Figura 3. Bomba modelo 4300-242 de FloJet 27

Figura 4. Modelo Cerabar T PMC131 de Endress+Hauser 28

Figura 5. Modelo NMC-T12G603 de Kobold 29

Figura 6. Caudalímetro de KOBLOD tipo MIK 30

Figura 7. Variador de frecuencia FVR0.4E9S-7EN de Fuji 31

Figura 8. Tarjeta DAQ USB-1408FS-Plus de MC 31

Figura 9. Conexiones de la tarjeta DAQ 32

Figura 10. Estructura de control en lazo cerrado 33

Figura 11. Estructura de control con controlador PID paralelo 35

Figura 12. Offset tras salto unitario en la variable de control 36

Figura 13. Esquema de la comunicación RIP 44

Figura 14. Ventana “Descripción” de EJSS 45

Figura 15. Ventana “Modelo” de EJSS 45

Figura 16. El elemento RIP de EJSS 46

Figura 17. Ventana “HtmlView” de EJSS 47

Figura 18. Estructura general del laboratorio remoto 49

Figura 19. Canal de YouTube UNILabs 50

Figura 20. Evolución del vector ids tras desconexión del maestro 55

Figura 21. Contenido del fichero de configuración 56

Figura 22. Código Velocidad_simulación del cliente en EJSS 64

Figura 23. Código Arranque_cliente del cliente en EJSS 65

Figura 24. Código Registro_datos del cliente en EJSS 65

Figura 25. Introducción de los valores actuales a los arrays alturas, controles y tiempos 66

Figura 26. Obtención del máximo y mínimo de la altura visibles en la gráfica 66

Figura 27. Obtención del máximo y mínimo del % de la bomba visibles en la gráfica 66

Figura 28. Obtención del máximo y mínimo de la referencia visibles en la gráfica 67

Figura 29. Asignación a los valores máximos y mínimos de las gráficas 67

Figura 30. Acciones realizadas si se activa el registro de datos 68

Figura 31. Acciones al iniciar la conexión RIP 69

Figura 32. Acciones realizadas tras desconectar con el servidor 69

Figura 33. Acción para correcto cambio de automático a manual 69

Figura 34. Actualización traza del valor de referencia 69

Figura 35. Actualización del eje de tiempo de las gráficas 70

Figura 36. Acciones para la correcta visualización en los selectores de modo 70

Figura 37. Código para mostrar los valores instantáneos en los displays en las gráficas 71

Figura 38. Acciones realizadas en la pestaña MaestroEsclavo del cliente..... 72

Figura 39. Código de la función propia guardar 73

Figura 40. Código de la función propia conectar 73

Figura 41. Elemento RIP añadido al proyecto 74

Figura 42. Pestaña Server Configuration una vez conectada con el servidor 74

Figura 43. Pestaña Experience en la configuración del elemento RIP 75

Figura 44. Pestaña Auto Update en la configuración del elemento RIP 75

Figura 45. Bloques principales de la interfaz del cliente 77

Figura 46. Contenido del panel barra herramientas 77

Figura 47. Propiedades del panel barra de herramientas 78

Figura 48. Propiedades del botón Descripción 78

Figura 49. Código CSS del botón Descripción 79

Figura 50. Propiedades del selector del tipo de control 79

Figura 51. Código CSS del selector del tipo de control 79

Figura 52. Código ejecutado al cambiar de opción en el selector de tipo de control 80

Figura 53. Propiedades del botón de guardado 81

Figura 54. Código CSS del panel RIP 81

Figura 55. Propiedades del botón de configuración 82

Figura 56. Propiedades del botón Conectar 82

Figura 57. Propiedades del botón Desconectar 83

Figura 58. Código CSS para los botones del panel RIP 83

Figura 59. Icono utilizado como botón de información 83

Figura 60. Contenido del Panel opciones interfaz y controlador 84

Figura 61. Código CSS del panel opciones interfaz y controlador 84

Figura 62. Código CSS para los paneles configuración interfaz, operación manual y controlador PID 84

Figura 63. Contenido del panel de configuración de la interfaz 85

Figura 64. Código CSS para los títulos de los paneles título configuración, título operación manual y título control PID 85

Figura 65. Código CSS del panel opciones 85

Figura 66. Código CSS para los paneles franja temporal visible y periodo muestreo 86

Figura 67. Código CSS para los paneles autoescalado y registro datos 86

Figura 68. Propiedades del elemento Valor franja visible 86

Figura 69. Propiedades del elemento Valor periodo muestreo 87

Figura 70. Código de la propiedad OnChange del elemento Valor periodo muestreo 87

Figura 71. Aspecto del panel opciones interfaz y controlador 88

Figura 72. Contenido del panel para el control manual 88

Figura 73. Código CSS del Panel potencia bomba 88

Figura 74. Código CSS de Slider potencia bomba 89

Figura 75. Propiedades de Slider potencia bomba 89

Figura 76. Propiedades del elemento Valor potencia bomba 89

Figura 77. Aspecto de Panel operación bomba 90

Figura 78. Contenido del panel controlador PID 90

Figura 79. Código CSS del Panel selección tipo PID 91

Figura 80. Propiedades del selector tipo controlador PID 91

Figura 81. Código del parámetro OnChange del Selector tipo controlador PID 91

Figura 82. Propiedades del elemento Valor referencia 92

Figura 83. Código CSS del Panel parámetros PID 92

Figura 84. Código CSS del panel título parámetros 93

Figura 85. Código CSS del Panel parámetros 93

Figura 86. Parámetros de las etiquetas Kp, Ti, Td y constante filtro 93

Figura 87. Parámetros de los elementos Valor Kp, Valor Ti, Valor Td y valor constante filtro 94

Figura 88. Aspecto del Panel controlador PID 94

Figura 89. Contenido del panel visualización 94

Figura 90. Código CSS del Panel visualización 95

Figura 91. Propiedades del panel CamPlanta 95

Figura 92. Código CSS del panel Gráficas 96

Figura 93. Parámetros del panel Leyenda 96

Figura 94. Código CSS del panel Leyenda 97

Figura 95. Propiedades de la etiqueta altura 97

Figura 96. Código CSS de las etiquetas altura(izq.) y referencia(dcha.) 97

Figura 97. Propiedades de la etiqueta referencia 98

Figura 98. Propiedades de los paneles Indicación altura e Indicación bomba 98

Figura 99. Código CSS de los paneles Indicación altura(izq.) e Indicación bomba(dcha.) 98

Figura 100. Propiedades de los campos valor altura y valor bomba 99

Figura 101. Código CSS de los campos valor altura(izq.) y valor bomba(dcha.) 99

Figura 102. Ajuste de los ejes de las gráficas altura deposito(izq.) y potencia bomba(dcha.) 99

Figura 103. Aspecto del panel visualización 100

Figura 104. Guardado al empaquetar el cliente EJSS 101

Figura 105. Archivos generados al empaquetar el cliente 101

Figura 106. Contenido del cliente tras eliminar los archivos irrelevantes ... 102

Figura 107. Panel de configuración de las URLs de la cámara y del servidor RIP 103

Figura 108. Código añadido para recibir las direcciones en la interfaz del cliente 103

Figura 109. Modificación del elemento RIP para recibir la dirección fijada en la configuración 104

Figura 110. Modificación de CamPlanta para recibir la dirección fijada en la configuración 104

Figura 111. Código de los créditos automáticos de EJSS 104

Figura 112. Contenido final de la carpeta del cliente 105

Figura 113. Cargar datos del laboratorio remoto en Excel 107

Figura 114. Selección del delimitador en la carga de datos de Excel 108

Figura 115. Evolución del sistema al cambiar la bomba del 25 al 28% en modo manual 108

Figura 116. Comparación del porcentaje de la bomba en ambos experimentos JavaRegula 111

Figura 117. Comparación de la altura del tanque de salida en ambos experimentos JavaRegula 111

Figura 118. Evolución temporal del nivel del tanque en el experimento con PID ideal 112

Figura 119. Evolución temporal del porcentaje de la bomba en el experimento con PID ideal 112

Figura 120. Descarga de Matlab para Windows 115

Figura 121. Apartado de descarga de Python en su página oficial 116

Figura 122. Ventana inicial en la instalación de Python 116

Figura 123. Características opcionales en la instalación de Python 117

Figura 124. Opciones avanzadas en la instalación de Python 117

Figura 125. Ventana al terminar la instalación de Python realizada correctamente 118

Figura 126. Comprobación de una instalación de Python correcta 118

Figura 127. Primer paso para abrir una ruta fácilmente en el Símbolo del sistema 119

Figura 128. Segundo paso para abrir una ruta fácilmente en el Símbolo del sistema 119

Figura 129. Página de descarga de Yawcam..... 120

Figura 130. Aceptar acuerdo de licencia Yawcam 121

Figura 131. Elección de ruta instalación Yawcam..... 121

Figura 132. Opciones de creación de accesos directos del instalador de Yawcam..... 122

Figura 133. Ventana inicial de Softing OPC Toolbox Demo Client..... 123

Figura 134. Añadir un servidor en Softing OPC Toolbox Demo Client..... 123

Figura 135. Etiqueta con las conexiones de la tarjeta de adquisición de datos 124

Figura 136. Entradas y salidas añadidas en Softing OPC Toolbox Demo Client 124

Figura 137. Modelo correcto del fichero de configuración del servidor 124

Figura 138. Resultado al arrancar el servidor. Servidor a la espera de cliente. 125

Figura 139. Elección del dispositivo en Yawcam..... 126

Figura 140. Opciones de trabajo de Yawcam 126

Figura 141. Menú superior de la interfaz del cliente 129

Figura 142. Contenido del fichero de guardado de datos del laboratorio.... 130

Figura 143. Panel de configuración de las URLs de los servidores RIP y de la cámara..... 130

Figura 144. Página con los créditos del proyecto..... 131

Figura 145. Panel de opciones de la interfaz del cliente..... 131

Figura 146. Panel de control manual en la interfaz del cliente..... 132

Figura 147. Panel de opciones del PID en la interfaz del cliente..... 132

Figura 148. Imagen de la planta real en la interfaz del cliente..... 133

Figura 149. Gráficas de la interfaz del cliente 134

Figura 150. Primer paso para activar el contenido mixto en Google Chrome 135

Figura 151. Segundo paso para activar el contenido mixto en Google Chrome 135

Figura 152. Primer paso para activar el contenido mixto en Firefox..... 136

Figura 153. Segundo paso para activar el contenido mixto en Firefox 136

Figura 154. Tercer paso para activar el contenido mixto en Firefox 136

Figura 155. Primer paso para activar el contenido mixto en Microsoft Edge 137

Figura 156. Segundo paso para activar el contenido mixto en Microsoft Edge 137

Índice de tablas

Tabla 1. Variables intercambiadas mediante el protocolo RIP.....53

Tabla 2. Variables Autoescalado del cliente 60

Tabla 3. Variables Control del cliente.....61

Tabla 4. Variables Modelo del cliente.....62

Tabla 5. Variables Interfaz del cliente 63

Tabla 6. Variables Registro_datos del cliente.....64

Tabla 7. Conexión de las variables intercambiadas mediante RIP 76

Tabla 8. Sintonía de Zieger-Nichols en lazo abierto 109

Tabla 9. Versiones de matlabengine compatibles con cada versión de Matlab 127

CAPÍTULO 1: Introducción y Objetivos

1.1. Introducción

Un laboratorio remoto permite realizar experimentos sobre plantas reales sin la necesidad de acudir presencialmente, en este caso, bastará con tener en funcionamiento el programa servidor en un ordenador conectado a la planta y lanzar un fichero xhtml con la interfaz necesaria para llevar a cabo los experimentos.

Este trabajo toma como referencia el proyecto UNILabs (UNED, s.f.) de la UNED (Universidad Nacional de Educación a Distancia), una iniciativa que permite a diversas universidades de todo el mundo desarrollar y compartir laboratorios virtuales y remotos en un repositorio público. Con el objetivo de ofrecer una formación flexible y accesible, en internet se han publicado numerosas simulaciones para cualquier persona interesada en el tema.

Este trabajo consiste en el desarrollo de un laboratorio remoto para el control del nivel del tanque de salida de una planta de dos tanques comunicantes. Para ello, se puede elegir entre dos implementaciones diferentes de un controlador PID. También se ofrece la opción de control manual con el que se elige directamente el caudal que entra al sistema, este modo permite una observación rápida del funcionamiento del sistema o la obtención de un punto de trabajo con el que iniciar el control automático.

La planta del laboratorio se compone de una bandeja rectangular sobre la que se sitúan dos tanques comunicados por la parte inferior a través de una tubería. En la bandeja se recoge el líquido de trabajo, en este caso agua, que se introducirá a uno de los tanques a través de una bomba. En el tanque opuesto al de entrada, se tiene una válvula manual y por eso, no podemos controlar el caudal de salida desde el ordenador. En el tanque de salida tenemos un sensor de nivel que nos permite conocer el llenado de este, así como realizar los cálculos necesarios para el control. Como intermediario entre la instrumentación y el ordenador, se ha instalado una tarjeta de adquisición de datos (TAD).

1.2. Objetivos

Se busca realizar un programa que trabaje de manera similar al software JavaRegula instalado en los ordenadores del laboratorio con el que se realizan los experimentos presencialmente, pero que permita realizarlo de manera remota, es decir, desde un ordenador distinto al del laboratorio. Con esto, los alumnos que no puedan acudir al laboratorio podrían realizar el experimento correspondiente desde sus casas y, al tener ambas aplicaciones comportamientos similares, no tendrían problemas para retomar los experimentos presencialmente.

Con este software se podrá además de realizar el control similar al software JavaRegula, un control con una implementación ideal de PID y un control manual que permita la manipulación directa sobre la bomba. En la interfaz del cliente se facilitarán una visualización de la planta y unas gráficas con los datos más relevantes, también se permitirá al usuario manipular los parámetros de los controladores para llevar a cabo los experimentos y, por último, se podrán exportar los datos obtenidos en un archivo de texto.

CAPÍTULO 2: La planta real del laboratorio

2.1. Funcionamiento

La planta se compone de dos tanques de agua abiertos por la parte superior que están comunicados por la parte inferior, una bomba, una bandeja situada en la base de los tanques, un transmisor de nivel y uno de caudal, un variador de frecuencia, una tarjeta de adquisición de datos, una válvula de apertura manual y las fuentes de alimentación. En el próximo apartado se explica detalladamente cada uno de estos elementos.

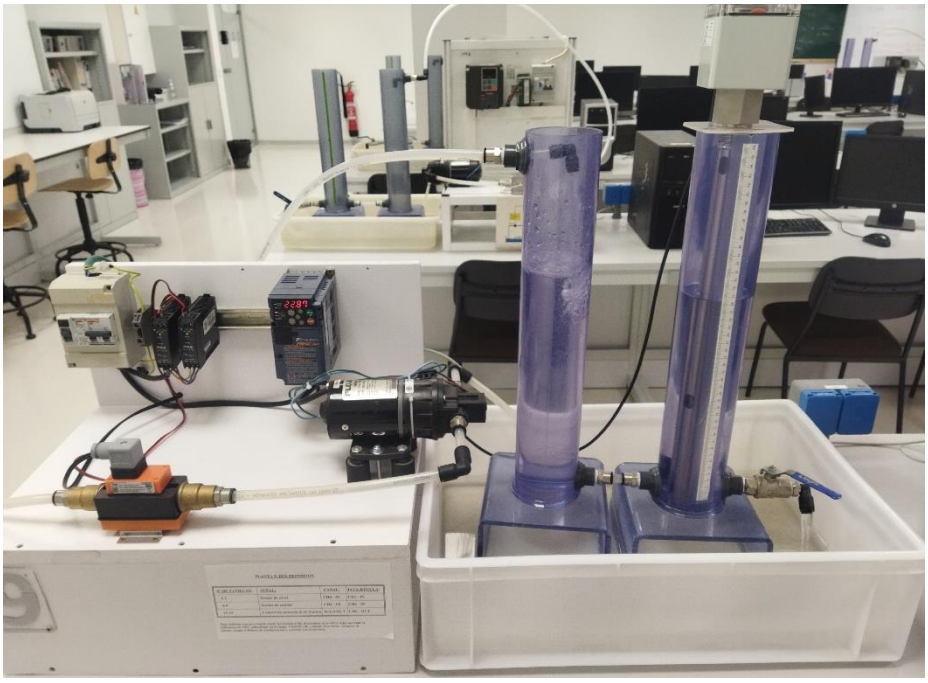


Figura 1. Montaje de una de las plantas del laboratorio

El funcionamiento es bastante simple, la bomba, cuya potencia se fija a través del variador de frecuencia, recoge agua de la bandeja y lo impulsa al tanque de entrada, este hará llegar el agua por la tubería que las conecta al tanque de salida donde, a través de la válvula manual, se devolverá a la bandeja.

2.2. Componentes de la planta

En este apartado se explicará con detalle cada uno de los elementos, ya mencionados anteriormente, de los que se compone la planta del laboratorio.

2.2.1. Depósitos

Los depósitos son cilíndricos con unas dimensiones de 50cm de alto y 8cm de diámetro, la tubería que los une tiene una longitud de 15cm y un diámetro de 1cm.

2.2.2. Fuentes de alimentación

En la planta se tiene un par de fuentes que transformarán la tensión de red a 24V para poder alimentar al sensor de nivel y al caudalímetro utilizados en el laboratorio.

Un modelo de estas fuentes utilizado es el mostrado a continuación:

Figura 2. Fuentes de alimentación ML30 de Puls

2.2.3. Bomba

La bomba es el actuador del sistema, se encarga de hacer llegar el agua de la bandeja al tanque de entrada. La bomba es alimentada por el variador de frecuencia.

De esta manera, en función de la señal que reciba bombeará más o menos caudal. La bomba utilizada requiere una tensión alterna de 220/240V y 50/60 Hz y 0.75 Amperios, el caudal máximo es de 18.2 l/min y la presión máxima 2.1 bar.

La bomba utilizada es de diafragma (desplazamiento positivo). Funciona de la siguiente manera:

- La bomba tiene una cámara de bombeo que contiene uno o varios diafragmas flexibles, el fluido entra en la cámara de bombeo a través de una entrada. La salida del fluido se controla mediante válvulas de entrada y salida.

- El diafragma se mueve hacia adentro y hacia afuera de la cámara de bombeo mediante un motor eléctrico, que será el que controlemos mediante el variador de frecuencia.

- Cuando el diafragma se mueve hacia adentro, se reduce el volumen dentro de la cámara de bombeo, lo que aumenta la presión del fluido y fuerza a la válvula de entrada a cerrarse para evitar que el fluido retroceda. Al mismo tiempo, la válvula de salida se abre para permitir que el fluido salga de la bomba.

- Cuando el diafragma se mueve hacia afuera, se expande el volumen dentro de la cámara de bombeo, lo que disminuye la presión y permite que la válvula de entrada se abra para permitir la entrada de más fluido. Al mismo tiempo, la válvula de salida se cierra para evitar que el fluido regrese hacia la entrada.

Figura 3. Bomba modelo 4300-242 de FloJet

2.2.4. Sensor de nivel

El sensor de nivel es el que nos permite conocer el llenado del tanque de salida, es decir, es fundamental para llevar a cabo el control automático de la planta.

En el mercado se encuentran múltiples tipos de sensores de nivel, pero en el laboratorio solo encontraremos de tipo capacitivo o de presión. Independiente del tipo de sensor ambos generan una señal normalizada de 4-20mA.

2.2.4.1. Sensor de nivel mediante presión

El transmisor de nivel mediante presión consiste en un sensor piezoeléctrico, que contiene la electrónica que permite convertir la señal del sensor en una señal normalizada de 4-20mA. Cuanta más presión se ejerza sobre el sensor, es decir, cuanto más lleno este el tanque, mayor será la señal obtenida.

Un sensor de nivel de este tipo presente en el laboratorio es el modelo Cerabar T PMC131 de Endress+Hauser con las siguientes características:

- Sistema directo sin diafragma separador, diafragma resistente a sobrecargas/vacío y conexión por cable mediante un conector de equipos IP65.
- Requiere una alimentación entre 11 y 30V.
- Proporciona una señal en el estándar 4-20 mA, por lo que se puede comunicar directamente con la tarjeta de adquisición de datos.
- El rango de medida del sensor es de 100mbar a 40bar.

Figura 4. Modelo Cerabar T PMC131 de Endress+Hauser

2.2.4.2. Sensor de nivel capacitivo

Este tipo de sensores miden la capacitancia del condensador formado por las placas del medidor, o entre una placa y la pared del recipiente. La capacidad del conjunto depende linealmente del nivel del líquido.

De esta forma, a medida que el tanque se vaya llenando de agua, aumentará su capacidad y con ello la señal que le llegue a la tarjeta.

Figura 5. Modelo NMC-T12G603 de Kobold

Un modelo que encontramos en el laboratorio es el Transmisor de Nivel Capacitivo NMC de KOBOLD, que presenta las siguientes características:

- Rango de Medición: máx. 4 m
- Conexión: G 1 rosca macho, acero inox, o G 2 rosca macho, PVDF
- Material Sensor: Acero Inox. con
- Revestimiento PTFE o PVDF
- $P_{máx}$: -1 a 30 bar
- $T_{máx}$: -20 a +90 °C / -20 a +200 °C
- Alimentación: 12 a 35 V_{DC} (ATEX 12...30 V_{DC})
- Salida Analógica 4-20 mA

2.2.5. Caudalímetro

Aunque para los experimentos que se van a realizar en este laboratorio remoto no se utilice, la planta contiene un caudalímetro de KOBLOD tipo MIK. En la imagen adjuntada podemos observar las características y conexiones del dispositivo.

Figura 6. Caudalímetro de KOBLOD tipo MIK

El modelo utilizado es un medidor para líquidos conductivos, ya que se basa en la ley de Faraday, es decir, el líquido conductor que pasa a través del campo magnético generado por el medidor de flujo crea un voltaje que es directamente proporcional al caudal del líquido. De esta forma se puede enviar la señal a la tarjeta de adquisición de datos y obtener el caudal en el ordenador para poder observar su evolución o realizar algún tipo de control.

2.2.6. Variador de frecuencia

El variador de frecuencia utilizado es el modelo FVR0.4E9S-7EN de Fuji. A través del nombre del producto podemos obtener la siguiente información:

- FVR: tipo del producto
- 0.4: potencia nominal en kW, es decir, 0.4 kW de potencia
- E9S: nombre de la serie de productos
- 7: indica que el sistema de tensión de alimentación es monofásico de 220V
- EN: extensión de la serie

Figura 7. Variador de frecuencia FVR0.4E9S-7EN de Fuji

El variador de frecuencia se encarga de alimentar a la bomba. En función del valor que obtenga por parte de la tarjeta de adquisición de datos podrá modificar la frecuencia de la alimentación de la bomba, haciendo que esta envíe más o menos caudal al tanque de entrada.

2.2.7. Tarjeta de adquisición de datos

La tarjeta de adquisición de datos se encargará de que podamos visualizar las señales de los dos medidores y además nos permitirá enviar una señal al variador de frecuencia.

Figura 8. Tarjeta DAQ USB-1408FS-Plus de MC

Como los transductores trabajan con señales en el estándar 4-20 mA y la tarjeta únicamente recibe entradas de tensión, se han situado resistencias de 250Ω a la entrada de los canales; de esta manera las señales de los transductores estarán en el rango de 1 a 5 V.

La tarjeta se encarga de recibir señales analógicas y mediante un conversor A/D se encarga de transmitirlas a un ordenador a través de un cable USB. La conversión conlleva a poder detectar únicamente un número de valores que vendrá dado por la resolución del sistema. Nuestra tarjeta tiene una resolución de 14 bits y podríamos obtener solamente 2^{14} valores, es decir, 16384 valores distintos. Esto nos permite saber cuánto debe cambiar la señal de entrada para que la tarjeta detecte que se ha producido un cambio en la medida, lo que se conoce como sensibilidad, cuanto menor sea el valor, más sensible será la tarjeta. En nuestro caso la tarjeta está configurada en el rango ±5V, con lo que obtenemos una sensibilidad de 0,61mV.

Las conexiones realizadas son las siguientes:

Nº DE PATILLAS:	SEÑAL:	CANAL.	JAVA-REGULA:
1-2	Sensor de nivel	CH0 - IN	CH1- IN
4-5	Sensor de caudal	CH1 - IN	CH2- IN
12-13	Salida a válvula	D/A 0-OUT	CH0 - OUT

Figura 9. Conexiones de la tarjeta DAQ

CAPÍTULO 3: Fundamentos y Herramientas

3.1. Estructura de control

La estructura de control se encarga de producir una señal de control en función de las señales de entrada para que la variable controlada difiera lo mínimo posible del valor de referencia deseado.

Si no se tiene realimentación de la salida hacia la entrada, hablamos de sistemas en lazo abierto, mientras que, los que presentan realimentación, se denominan sistemas de lazo cerrado.

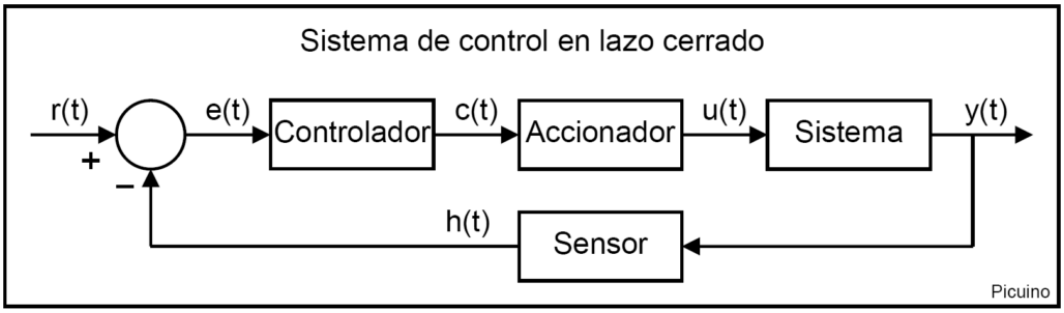


Figura 10. Estructura de control en lazo cerrado

Por otro lado, la estructura puede tomar dos modos, automático o manual, en función de si la señal de control la fija el propio sistema o un usuario, respectivamente.

En la figura anterior se puede observar la notación utilizada para las diferentes variables del sistema:

- r(t): es la referencia fijada por el usuario, es decir, el valor que se quiere obtener en la variable controlada.
- h(t): valor proveniente de un sensor encargado de obtener la medida de la variable que se está controlando.
- e(t): error del sistema, se corresponde con la diferencia entre la referencia y la variable controlada.
- c(t): valor calculado por el controlador en función de la señal de error.

- u(t): se conoce como la señal de control, se obtiene a partir del valor calculado por el controlador y, teniendo en cuenta la naturaleza de los actuadores y demás instrumentos de medida, este se puede ajustar entre un rango de valores.

- y(t): se corresponde con la variable controlada del sistema.

En resumen, el lazo de control se compone de un sensor encargado de obtener el valor de la variable controlada, un controlador que, en función del error entre la referencia y el valor medido, genera la señal de entrada al accionador a través del que se actuará sobre la planta en la que se encuentra la variable a controlar.

En este trabajo, se implementa un tipo de controlador muy simple, fácil de configurar y comúnmente utilizado en sistemas de control diversos, el conocido PID, además se facilitará la opción de experimentar en modo manual.

3.1.1. Control manual

En este caso se trabaja con el sistema en lazo abierto y no se tiene en cuenta el valor recibido por parte del sensor, será el propio usuario el que determine la acción del actuador. Este tipo de control permite realizar experimentos como encontrar un punto de equilibrio o entender el funcionamiento de la planta como paso previo a la experimentación con el control PID.

3.1.2. Control PID

El controlador PID es el encargado de calcular la entrada al actuador según el error obtenido de la diferencia entre el valor de referencia y la medida por el sensor. El PID es capaz de realizar tres acciones diferentes: Proporcional, Integral y Derivativa, dando lugar a su nombre. Estas acciones pueden situarse en serie o en paralelo.

El tipo de controlador que implementamos en este proyecto viene dado en paralelo, como se muestra en la figura y cuya ecuación se muestra en (3.1).

$$u(t) = K_p \cdot \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de_f}{dt} \right) \qquad (3.1)$$

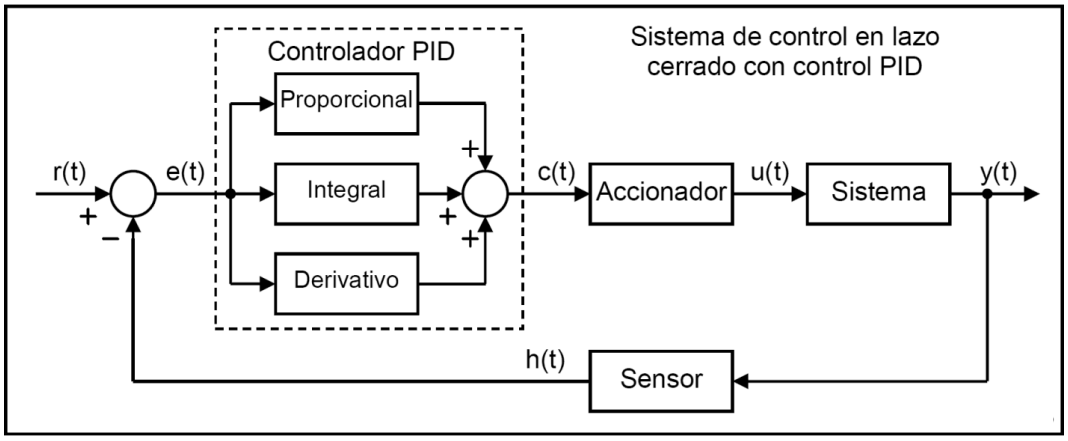


Figura 11. Estructura de control con controlador PID paralelo

En la planta de tanques comunicantes del laboratorio, el valor de referencia es el nivel deseado en el tanque de salida, mientras que la salida del sistema es el nivel real del tanque, es decir, la variable controlada será el nivel del tanque en la salida.

3.1.2.1. Acciones del control de un PID

Acción de control proporcional

Como su nombre indica, esta acción de control es proporcional a la señal de error. Internamente la acción proporcional multiplica la señal de error por una constante (Kp) e intenta minimizar el error del sistema. Cuando el error es grande, la acción de control es grande y tiende a minimizar este error.

Aumentar la acción proporcional tiene efectos deseables, tales como aumentar la velocidad de respuesta y disminuir el error del proceso, pero también tiene el efecto negativo de volver al sistema más inestable, razón por la cual esta acción debe buscar un punto de equilibrio en el que se consiga suficiente rapidez de respuesta del sistema y reducción del error, sin que el sistema sea demasiado inestable.

Aunque se fije un valor de Kp suficientemente bueno, en la mayoría de los sistemas nunca se alcanzará el valor de referencia en estacionario, produciéndose el error en estacionario conocido como offset. Este error es el que se encargará de eliminar la acción integral, explicada en el próximo apartado.

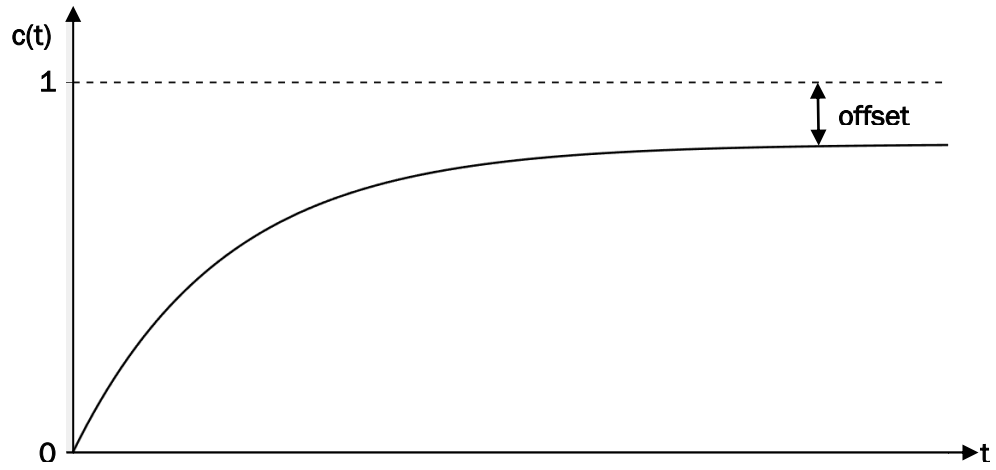


Figura 12. Offset tras salto unitario en la variable de control

Acción de control Integral

El objetivo principal de esta acción es eliminar el error en estado estacionario u offset. Para conseguirlo, esta acción acumula gradualmente el error y modifica la salida del regulador. El error se suma a lo largo del tiempo, es decir, realiza la integral del error a lo largo del tiempo y luego, lo multiplica por la ganancia integral (Ki = Kp/Ti).

Al aumentar la ganancia integral (o disminuir el tiempo integral Ti), se consigue alcanzar el error nulo en estacionario más rápidamente, pero también podría dar lugar a oscilaciones sobre la referencia.

Acción de control Derivativa

Esta acción de control es proporcional a la derivada de la señal de error o, en otras palabras, a la velocidad del error. Cuando el proceso que estamos controlando se mueve a alta velocidad hacia el punto deseado, el sistema sobrepasará el valor de la referencia debido a su inercia, produciendo sobrepicos y oscilaciones. Para evitar este problema, el controlador debe reconocer la velocidad a la que el sistema se acerca al valor de referencia deseado para poder atenuar el valor de la acción de control a medida que se acerque a la referencia deseada y evitar que la sobrepase.

Al aumentar la constante de control derivativa (Kd = Kp x Td), o el tiempo derivativo Td, se obtiene un aumento en la estabilidad del sistema controlado, disminuye un poco la velocidad del sistema, pero el error estacionario se mantiene constante. Esta acción de control sirve, por lo tanto, para estabilizar una respuesta que oscile demasiado.

3.1.2.2. Ventajas y limitaciones de los controladores PID

Ventajas

- Simplicidad: Fácil de comprender y poner en práctica.
- Eficacia: Capaz de gestionar diversos problemas de control.
- Robustez: Capaz de funcionar correctamente a pesar de los cambios en el sistema.
- Flexibilidad: Adaptable a distintos tipos de sistemas y procedimientos.

Limitaciones

- Ajuste de la complejidad: Determinar las ganancias ideales (Kp, Ki y Kd) puede resultar difícil y llevar mucho tiempo.
- Rendimiento en sistemas no lineales: Los sistemas altamente no lineales pueden ser difíciles de manejar para los controladores PID.
- Sensibilidad al ruido: El ruido de alta frecuencia de la señal de error puede amplificarse mediante el término derivativo.
- Capacidad predictiva limitada: Basándose en tendencias históricas, responden a los errores, pero son incapaces de anticiparse a otros nuevos.

3.1.2.3. Sintonización manual del PID

El ajuste manual de los parámetros de un PID es el método más simple de sintonización, sin embargo, este procedimiento requiere que el controlador PID esté encendido y conectado directamente al sistema. Este método se conoce también como método de prueba y error.

Lo primero que se ajusta es la ganancia proporcional. Para ello, se establecen en cero las ganancias integral y derivativa y se va aumentando la proporcional hasta obtener un comportamiento que se ajuste a la respuesta buscada. Si esta es suficientemente buena, no sería necesario seguir configurando.

A continuación, se puede aumentar Td para conseguir una mejor respuesta. Td hace referencia al tiempo derivativo, multiplicado a la constante proporcional, se consigue la constante derivativa Kd (Kd = Kp*Td).

Por último, se fijará el valor del tiempo integral Ti, que dividido a la constante proporcional da lugar a la constante integral (Ki=Kp/Ti), con lo que se obtendrá un error estacionario nulo. Se puede empezar con un valor elevado de Ti e ir disminuyendo hasta obtener el comportamiento deseado. Si el sistema se volviese inestable sería necesario modificar el valor de Td.

De esta manera se irían modificando los valores hasta obtener la sintonía deseada que dé lugar a una respuesta cumpliendo con las especificaciones requeridas por el sistema.

3.1.2.4. Efecto windup y limites en los actuadores

El fenómeno windup en los controladores PID se presenta cuando se tiene una planta con entrada acotada y la acción de control supera los límites del actuador; como consecuencia, la integral del error se acumula continuamente dando lugar a una saturación en el valor integral y obteniéndose una mala respuesta. Existen varios métodos anti-windup, algunos de ellos se comentan a continuación.

- **Método de la Integral condicional**: consiste en detener la acumulación del error en el integrador cuando se cumplan ciertas condiciones asignadas por el diseñador.

- **Método de reinicio de la integral**: Consiste en reiniciar el valor de la integral del error a un valor definido previamente, cuando se alcancen unas condiciones determinadas.

- **Recálculo y seguimiento.** La estrategia de recálculo y seguimiento tiene como principio básico de funcionamiento el recálculo de la integral a través de un lazo de realimentación, este tiene en cuenta la diferencia entre la salida del actuador y la acción de control, de tal manera que la salida del controlador no supera el límite de saturación del actuador.

Es necesario conocer los límites del actuador para llevar a cabo un correcto control, por ejemplo, la bomba con la que se trabaja en la planta de tanques comunicantes tiene un valor máximo del 100%, o sea que, si el integrador indica un valor del 110%, nunca podrá aplicarse.

3.1.2.5. Controlador PID con filtro en la acción derivativa

Uno de los controladores implementados en este proyecto es el PID con filtro en la acción derivativa. Para poder implementarlo en el software posteriormente, será necesario conocer las ecuaciones de este y discretizarlas.

Las ecuaciones que describen este tipo de controlador son las siguientes:

$$\begin{cases} u(t) = K_p \cdot \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de_f}{dt} \right) \\ e(t) = \alpha T_d \frac{de_f}{dt} + e_f \end{cases} \quad (3.2)$$

Para discretizar estas ecuaciones empezaremos por realizar sobre la ecuación (3.2), la aproximación de Riemann de sumatorio para la integral y la derivada de e_f como incremento por unidad de tiempo, obteniendo el resultado mostrado en (3.3).

$$\begin{cases} u(t) \approx K_p \cdot \left(e(t) + \frac{1}{T_i} \sum_{i=1}^{t} e(iT)T + T_d \frac{e_f(t) - e_f(t-T)}{T} \right) \\ \vdots \\ e(t) \approx \alpha T_d \frac{e_f(t) - e_f(t-T)}{T} + e_f \end{cases} \quad (3.3)$$

A continuación, obtenemos el valor de u para t= t-T sobre la ecuación superior de (3.3):

$$\begin{aligned} & u(t-T) \approx \\ & \approx K_p \cdot \left(e(t-T) + \frac{1}{T_i} \sum_{i=1}^{t-T} e(iT)T + T_d \frac{e_f(t-T) - e_f(t-2T)}{T} \right) \end{aligned} \quad (3.4)$$

Realizando la diferencia entre las ecuaciones (3.3) y (3.4) se llega a la ecuación (3.5) donde se consigue eliminar el sumatorio.

$$\begin{aligned} &u(t) - u(t-T) \approx \\ &\approx K_p e(t) - K_p e(t-T) + K_p \frac{T}{T_i} e(t) + \\ &+ K_p T_d \frac{e_f(t) - 2e_f(t-T) + e_f(t-2T)}{T} \end{aligned} \tag{3.5}$$

Aislamos el valor de u(t) y obtenemos la ecuación que nos interesa para el controlador, (3.6).

$$\begin{aligned} &u(t) \approx \\ &\approx u(t-T) + K_p\big(e(t) - e(t-T)\big) + K_p \frac{T}{T_i} e(t) + \\ &+ K_p \frac{T_d}{T}\big(e_f(t) - 2e_f(t-T) + e_f(t-2T)\big) \end{aligned} \tag{3.6}$$

En la ecuación (3.6) se pueden diferenciar las tres acciones diferentes del controlador, el segundo sumando se corresponde con la acción proporcional, el tercero con la integral y el cuarto con la derivativa con filtro. Este error filtrado se corresponde con la ecuación inferior de (3.2) y también tendremos que discretizarla. Si realizamos los mismos pasos que para la ecuación anterior, se llega a lo siguiente:

$$\frac{\alpha T_d}{T} e_f(t) + e_f(t) - \frac{\alpha T_d}{T} e_f(t-T) = e(t) \tag{3.7}$$

Se saca factor común el error de filtrado que es lo que nos interesa y se despeja ese valor obteniendo las ecuaciones (3.8) y (3.9), respectivamente.

$$\left(1 + \frac{\alpha T_d}{T}\right) e_f(t) - \frac{\alpha T_d}{T} e_f(t-T) = e(t) \tag{3.8}$$

$$e_f(t) = \frac{\frac{\alpha T_d}{T}}{\left(1 + \frac{\alpha T_d}{T}\right)} e_f(t-T) + \frac{1}{\left(1 + \frac{\alpha T_d}{T}\right)} e(t) \tag{3.9}$$

Con todo esto, hemos conseguido las ecuaciones discretizadas, (3.10), del controlador PID con filtro derivativo que se implementará en el proyecto.

$$\begin{cases} u(t) \approx u(t-T) + K_p\big(e(t) - e(t-T)\big) + K_p\dfrac{T}{T_i}e(t) + \\ \quad + K_p\dfrac{T_d}{T}\big(e_f(t) - 2e_f(t-T) + e_f(t-2T)\big) \\ \quad \vdots \\ e_f(t) = \dfrac{\frac{\alpha T_d}{T}}{\left(1+\frac{\alpha T_d}{T}\right)}e_f(t-T) + \dfrac{1}{\left(1+\frac{\alpha T_d}{T}\right)}e(t) \end{cases} \qquad (3.10)$$

Con el sistema (3.10), podemos implementar fácilmente el código del controlador en Matlab, como se explicará más detalladamente en el próximo capítulo.

3.1.2.6. Controlador PID de JavaRegula

El otro tipo de controlador que se va a implementar en este trabajo y que es el principal objetivo del trabajo, es el que utiliza el software JavaRegula instalado en los ordenadores del laboratorio.

Este tipo de controlador contiene un filtro de paso bajo, cuyas ecuaciones vienen definidas en el manual de usuario. Este nos dice que la constante de proporcionalidad viene dada como (3.11) y la acción proporcional puede verse en (3.12).

$$K = Kp \cdot \frac{\text{Valor máximo del control (en \%)}}{\text{Span de la medida}} \qquad (3.11)$$

$$P(t) = K \cdot \big(e(t) - e(t-1)\big) \qquad (3.12)$$

La acción integral implementada por el software JavaRegula es la siguiente, donde Ti es el tiempo integral expresado en minutos y que será uno de los parámetros que podrá configurar el usuario.

$$I(t) = \frac{K \cdot T}{T_i \cdot 60}e(t-1) \qquad (3.13)$$

Por último, se tiene la acción derivativa, que se obtiene mediante las diferencias entre la derivada de la salida del sistema que, en nuestro caso, se trata de la altura del tanque de salida, y la derivada de esa salida pasada por un filtro de primer orden. En este caso se podrá configurar el parámetro Td, correspondiente al tiempo derivativo en minutos y el valor del filtro (α), que estará comprendido entre 0 y 1.

$$D(t) = \frac{K \cdot T_d \cdot 60 \cdot \alpha}{T} (\_y(t) - \_y_f(t)) \qquad (3.14)$$

_y(t) será la diferencia entre el valor de la variable controlada actual y el valor anterior, (3.15).

$$\_y(t) = y(t) - y(t-1) \qquad (3.15)$$

Por su parte, $\_y_f(t)$, es la diferencia entre el valor de la variable controlada actual y el valor anterior filtrados, (3.16).

$$\_y_f(t) = y_f(t) - y_f(t-1) \qquad (3.16)$$

El valor filtrado se obtiene de la siguiente manera:

$$y_f(t) = y_f(t-1) + \alpha \cdot (y(t-1) - y_f(t-1)) \qquad (3.17)$$

Uniendo las acciones recogidas en (3.12), (3.13) y (3.14), se obtiene la ecuación para el cálculo de la variable de control, como se muestra a continuación:

$$\begin{aligned} u(t) = \\ = K \cdot \big(e(t) - e(t-1)\big) + \frac{K \cdot T}{T_i \cdot 60} e(t-1) + \\ + \frac{K \cdot T_d \cdot 60 \cdot \alpha}{T} (\_y(t) - \_y_f(t)) + u(t-1) \end{aligned} \qquad (3.18)$$

Estas ecuaciones serán las que implementaremos en el programa del servidor para realizar el control.

3.2. Easy JavaScript Simulation y el protocolo RIP

3.2.1. Introducción

El software Easy JavaScript Simulations (Esquembre, 2013), a partir de ahora EJSS para abreviar, es un entorno desarrollo por el catedrático de Análisis Matemático en el departamento de la Universidad de Murcia, Francisco Esquembre Martínez. Este software es de libre acceso y se puede modificar siempre que sea reconocida la autoría de su desarrollador. Con esto, me gustaría agradecer el poder realizar parte de este trabajo con este software, ya que facilita en gran parte su realización.

La principal ventaja que ofrece EJSS es poder crear una simulación o interfaz gráfica de un laboratorio sin necesidad de ser un experto programando. Su uso es muy sencillo, basta con arrastrar elementos a la zona de trabajo del programa y asignarle las propiedades deseadas. Además, el software permite añadir elementos externos, como puede ser una página descriptiva del uso de la interfaz o una página de créditos.

Este programa ha recibido numerosos reconocimientos y galardones a nivel mundial. Algunos de estos premios son el premio GOLC(Global Online del Consorcio de Laboratorios), el "Premio de la sociedad Física Americana a la Excelencia en la Educación Física" o el "Premio al contenido multimedia de Enseñanza y Aprendizaje en Física". Además, ha recibido dos premios de la Unesco, uno por la oficina de educación de Asia y de la Región del Pacífico y por el uso de nuevas tecnologías en la educación con el premio rey Hamad Bin Isa Al-Khalifa.

En este proyecto se utilizará este software para la realización de la interfaz del cliente y para llevar a cabo la comunicación con el servidor se usará el protocolo RIP, explicado en el próximo apartado.

3.2.2. Protocolo RIP

RIP ofrece una solución de comunicación simple, pero potente, utilizable desde clientes web. Como tal, RIP sólo utiliza protocolos estándar HTTP puros, soportados por los principales navegadores web. RIP está diseñado para comunicar clientes web con laboratorios online, ya sean virtuales o remotos. Cuando se utiliza para comunicarse con un laboratorio virtual, RIP expone metadatos y métodos de entrada y salida y variables relacionadas con un modelo de simulación que está alojado y se ejecuta en un ordenador.

Cuando se utiliza para comunicarse con un laboratorio remoto, RIP hace lo mismo con un programa de control definido en un ordenador (normalmente, un servidor remoto) para monitorizar y manipular el equipo del laboratorio, este es el caso llevado a cabo en este trabajo. Se tendrá un servidor RIP en el laboratorio que gracias a Python y Matlab permitirá comunicarse con la planta y a su vez implemente el protocolo RIP para intercambiar información con el cliente lanzado en el buscador.

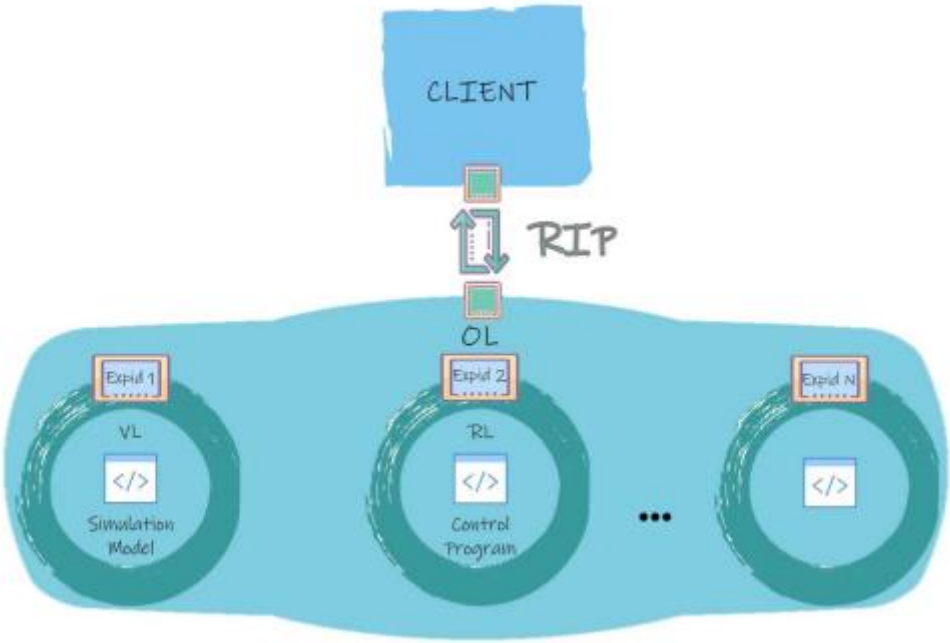


Figura 13. Esquema de la comunicación RIP

RIP proporciona un mecanismo sencillo para que los usuarios y las máquinas/programas cliente adquieran información sobre las experiencias de laboratorio definidas en el servidor y sobre las entradas y salidas de cada experiencia. El protocolo también define métodos para leer y escribir los valores de estas entradas y salidas, respectivamente.

3.2.3. Estructura de EJSS

El software EJSS se compone de 3 ventanas principales, la descripción, la del modelo y HtmlView, además de una barra lateral de tareas y de un panel de mensajes que hace las veces de consola.

En este apartado se comentarán los aspectos de mayor interés para la realización de este trabajo.

3.2.3.1. Ventana Descripción

En esta ventana se puede crear la ventana de descripción del laboratorio, no obstante, para este trabajo la realizaremos de manera independiente y la añadiremos una vez exportado el laboratorio, para tener un mayor control sobre su aspecto.

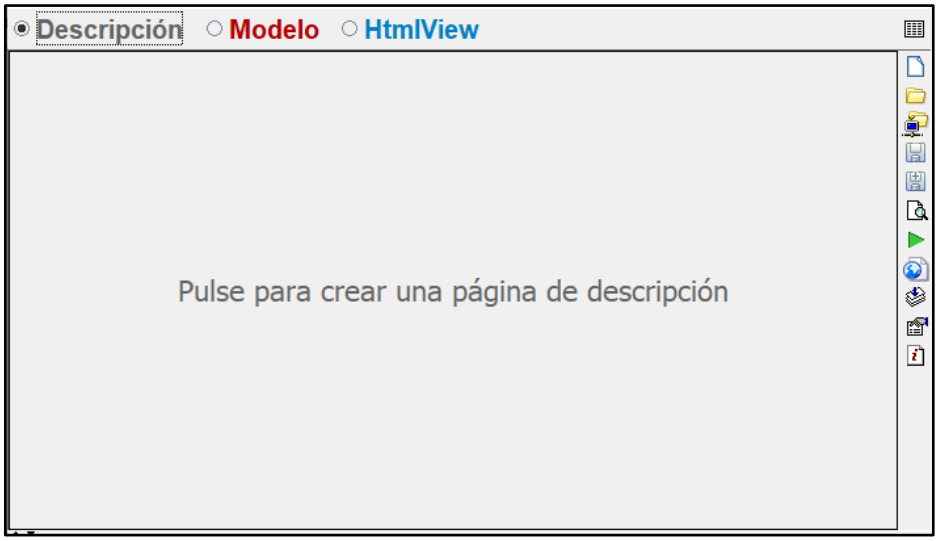


Figura 14. Ventana "Descripción" de EJSS

3.2.3.2. Ventana Modelo

Se utiliza este panel para crear variables que describen el modelo de la simulación, para inicializar estas variables y para escribir algoritmos que describen cómo este modelo cambia en el tiempo.

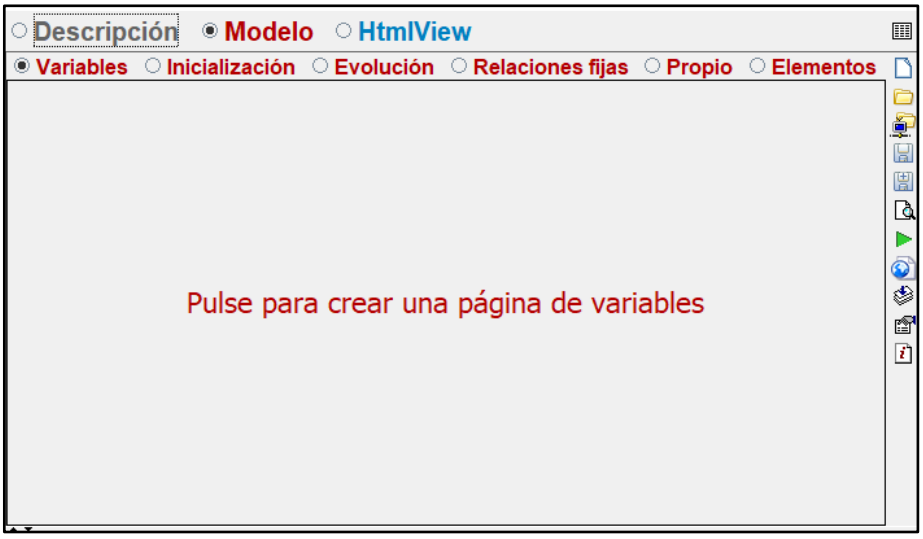


Figura 15. Ventana "Modelo" de EJSS

Como puede observarse en la Figura 15, a su vez se divide en 6 pestañas diferentes, que se describen a continuación:

- **Variables:** se utiliza para declarar e inicializar las variables del modelo
- **Inicialización:** permite ingresar algoritmos de inicialización adicionales
- **Evolución:** describe como avanza el modelo en el tiempo, en nuestro caso no será de interés ya que trabajamos sobre una planta real y no necesitamos modelar la planta, está enfocado a laboratorios virtuales en los que se tengan que introducir las ecuaciones del modelo.
- **Relaciones fijas:** establece relaciones fijas entre las variables y cálculos adicionales y necesarios para mantener el modelo actualizado. En esta ventana se introducirá la lógica necesaria para que la interfaz del cliente funcione correctamente.
- **Propio:** permite la creación de funciones para usar en otras partes del código o de la vista.
- **Elementos:** aquí se tienen objetos predefinidos y listos para usar que simplifican algunas tareas avanzadas. Entre estos elementos, el único que utilizaremos, y que es crucial para llevar a cabo el correcto funcionamiento del laboratorio remoto, es el elemento RIP, que implementa el protocolo RIP, ya comentado anteriormente, permitiendo realizar la comunicación con el servidor.

Figura 16. El elemento RIP de EJSS

3.2.3.3. Ventana HtmlView

Este panel de trabajo nos permite crear una interfaz gráfica que incluye visualización, interacción con el usuario y control del programa con un mínimo de programación.

La forma en que se visualizan los elementos refleja la forma en que se organizan los elementos de vista de las simulaciones: una estructura de árbol.

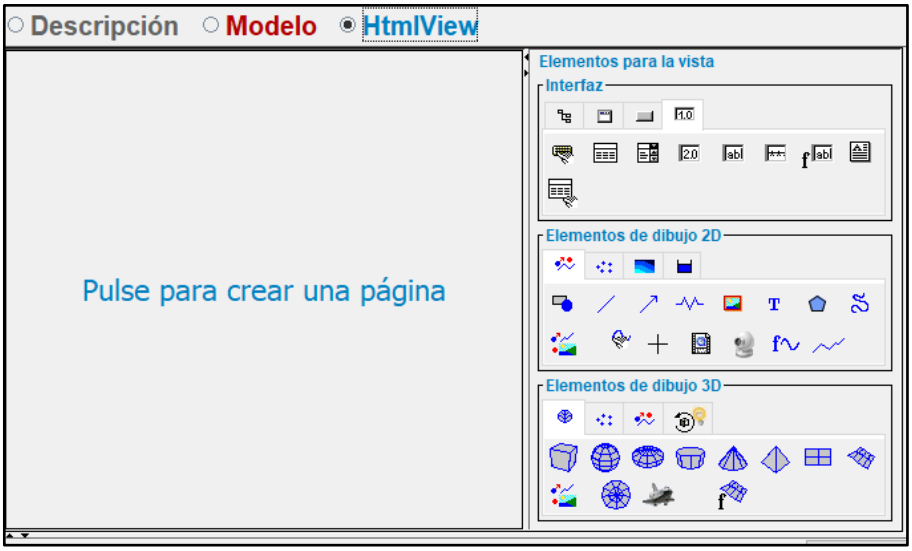


Figura 17. Ventana "HtmlView" de EJSS

Los elementos de la vista se agregan a la vista mediante la función *de arrastrar y soltar* . Al hacer clic en un elemento de la vista, se selecciona y el cursor se transforma en una *varita mágica*. Luego, el elemento de la vista se arrastra al árbol de elementos y se suelta sobre el elemento principal adecuado. De esta manera resulta muy sencillo realizar la interfaz del cliente, los elementos de dibujo nos permitirán introducir gráficas en la interfaz para visualizar la evolución en la altura del depósito de salida y de la apertura de la bomba.

3.2.3.4. Barra lateral de tareas

La barra de tareas se compone de iconos que, al clicar sobre ellos, realizan tareas como crear, abrir o buscar un nuevo archivo de simulación, guardar la simulación actual, guardar la simulación actual con otro nombre, también se puede buscar en el código, arrancar la simulación o traducir la interfaz del proyecto. Aparece también un icono donde se puede configurar algunas opciones del programa EJSS y otro que nos llevará a la ayuda de EJSS.

Pero, sin duda, la acción de mayor interés resulta la de empaquetar, que nos permite exportar el proyecto en una carpeta comprimida con los archivos necesarios para lanzar el modelo desde el navegador sin necesidad de tener instalado el software.

3.3. Yawcam.

De entre todas las opciones disponibles para mostrar una imagen en vivo del estado del nivel en el depósito, para este trabajo se ha optado por el uso de Yawcam (Yawcam - Yet Another Webcam Software, s.f.), ya que es la única opción gratuita que genera un servidor web con la imagen de una cámara USB. El resto del software encontrado no permitían generar una URL para cámaras USB o requerían algún coste adicional. Por otro lado, este software presenta un inconveniente relacionado con la seguridad: las URL donde aloja las imágenes sigue el protocolo HTTP, con lo que la información del streaming de vídeo se transmite sin cifrar.

El cliente remoto se podrá lanzar desde un servidor con la siguiente dirección: https://www2.eii.uva.es/jmzama/material.html. Nótese que este sitio utiliza HTTPS y, a su vez, tendrá alojada la dirección generada por Yawcam que es HTTP, obteniéndose contenido mixto, que significa que una página HTTPS tiene contenido HTTP. En este caso, "la parte HTTP puede ser leída o modificada por los atacantes, incluso aunque la página principal utilice HTTPS" (¿Cómo afecta a mi seguridad el contenido no seguro?, 2019). Este contenido será bloqueado por los navegadores por motivos de seguridad y para poder acceder a él necesitaremos modificar la configuración de seguridad asumiendo los riesgos que ello conlleva (véase cómo en el apartado 7.4.1). Esto no es una práctica recomendable, pero con los recursos que se disponen actualmente es la mejor opción posible. Lo ideal sería poder obtener un certificado SSL para las direcciones de Yawcam y tener un sitio completamente seguro.

Como alterativa a esto, se puede lanzar el cliente de manera local, es decir, teniendo todos los archivos alojados en la misma máquina que lo estamos utilizando, de esta manera los navegadores no bloquearían el contenido.

CAPÍTULO 4: Implementación del laboratorio remoto

4.1. Estructura del laboratorio

El laboratorio se basa en una estructura cliente-servidor utilizando el protocolo RIP. Además, la planta se comunicará con el ordenador a través de la tarjeta DAQ, empleando otra estructura cliente-servidor, esta vez con el protocolo OPC. Para mostrar la evolución a tiempo real en la interfaz del cliente, el servidor cuenta con una cámara enfocada al tanque de salida de la planta.

El laboratorio se divide en tres grandes elementos: por un lado, está la planta, descrita en el capítulo anterior, luego tenemos el lado del cliente desarrollado con JavaScript, HTML y CSS, y por último el servidor, que está programado en Python y Matlab. A continuación, puede verse la estructura general del laboratorio.

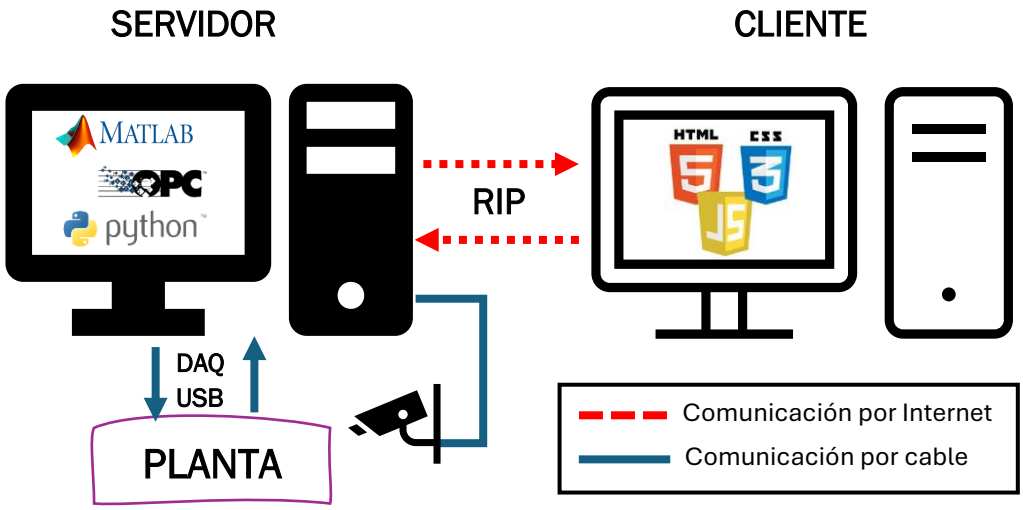


Figura 18. Estructura general del laboratorio remoto

El procedimiento es muy simple, en el ordenador servidor se ejecutará la aplicación de Python en el ordenador conectado a la planta (servidor), este a su vez arrancará Matlab en segundo plano y se mantendrá a la espera de que se conecte algún cliente. Matlab nos servirá para poder leer y escribir en la tarjeta DAQ, además de realizar el control correspondiente, que se explicará más adelante. Esta comunicación se puede realizar gracias al estándar OPC(Open Platform Communications).

"OPC es el estándar de interoperabilidad para el intercambio seguro y confiable de datos en el ámbito de la automatización industrial y en otras industrias. Es independiente de la plataforma y garantiza el flujo continuo de información entre dispositivos de múltiples proveedores" (OPC foundation, s.f.). Esto nos quiere decir que, gracias a OPC, podemos comunicar la tarjeta de adquisición de datos y Matlab siendo de distintos de fabricantes. Esto es fundamental para llevar a cabo un correcto funcionamiento del laboratorio.

Una vez se haya conectado algún cliente, el servidor pasa a estar activo e intercambia una serie de parámetros mediante el protocolo RIP. Estos parámetros se introducen desde Python en el espacio de trabajo de Matlab donde se llevan a cabo las acciones correspondientes en función de los valores fijados por el cliente. Una vez hecho esto, desde Python se recogen los nuevos valores del espacio de trabajo de Matlab y se envían al cliente.

Este es, a grandes rangos, el procedimiento del laboratorio, en los próximos apartados se profundizará en el desarrollo de cada una de las partes para un mejor entendimiento del funcionamiento de este.

4.2. Lado servidor

En este apartado se comentará el contenido del lado del servidor, esto implica la parte del código de Python y la parte implementada en Matlab.

4.2.1. Servidor RIP de Python

Para realizar el servidor del laboratorio remoto se ha utilizado como base un servidor RIP de Python realizado para el proyecto de UNILabs, este es de libre acceso bajo la licencia GNU-GLP (Chacón Sombria & de la Torre, 2018). Además, gracias a unos videos didácticos del canal de UNILabs de YouTube (UNEDLabs, s.f.), he podido familiarizarme rápidamente con el funcionamiento del servidor.

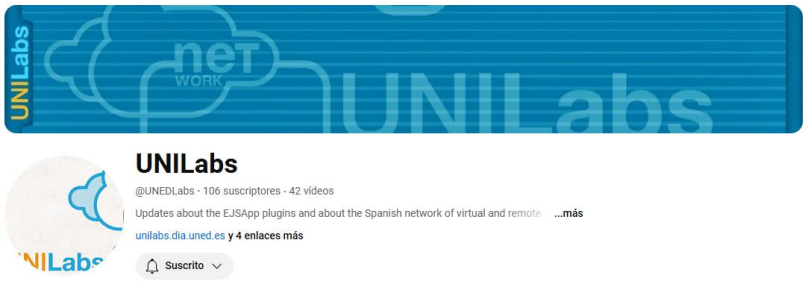


Figura 19. Canal de YouTube UNILabs

Este servidor contiene varios ficheros que no se utilizarán en este proyecto, ya que implementa configuraciones para Arduino u Octave. En este caso, como ya se ha comentado anteriormente, se utilizará Matlab; este fichero de configuración será el que se modifique con las variables del laboratorio. A continuación, comentaré con más detalle este fichero además de otros de interés.

4.2.1.1. RIPMatlab.py

Este script define la clase *RIPMatlab* que actúa como puente de comunicación entre Matlab y el cliente RIP. Para ello, hereda la clase *RIPGeneric*, comentada en el próximo punto, y también el paquete *matlab.engine*, que permite el uso de Matlab desde Python. Además, implementa métodos con los que se pueden leer y escribir valores en el workspace de Matlab.

El método *preGetValuesToNotify* se llama antes de obtener los valores que se notificarán, asegurando que cualquier preparación necesaria en MATLAB se realice antes de la notificación de los datos de *RIPGeneric*. Por ello desde este método será desde el que ejecutaremos el script de Matlab que realiza el control y que se comunica con la planta. Este archivo se comentará en el apartado 4.2.2.

4.2.1.2. RIPGeneric.py

Este archivo contiene todas las funciones para la comunicación RIP, algunas de están pueden ser la creación del elemento RIP para comunicarse u obtener las variables de lectura y de escritura con el cliente. En este archivo también se establece el periodo de actualización de los datos, que se ha fijado en un segundo.

4.2.1.3. AppConfig.py

Este fichero contiene una macroestructura con la configuración del servidor RIP, para ello se define el diccionario *config* que contiene dos secciones, *server* y *control*.

Sección server

En esta sección se recoge la información del servidor, en dos campos, host y port, como se puede ver a continuación:

```
'server': {
    'host': getIP(),
    'port': 8080,
  }
```

Nótese que el host se obtiene mediante la función *getIP*, esta función se encarga de extraer la dirección IP local, que será a la que se conecte el cliente. Esta función se ha añadido con el objetivo de no tener que modificar el código del servidor si la dirección IP se ve modificada en algún momento, o si se instala el servidor en otro ordenador del laboratorio.

Sección control

Esta sección se compone también de dos subsecciones, en primer lugar, se tendrá que definir el módulo de implementación, en este caso *RIPMatlab*, como se muestra a continuación:

```
'impl_module': 'RIPMatlab',
```

La segunda subsección será *info*, que contiene los detalles del servidor, incluyendo:

- name: Nombre del servidor.
- description: Descripción del servidor.
- authors: Autores del servidor.
- keywords: Palabras clave relacionadas con el servidor.

Los valores anteriores son puramente informativos, los campos realmente importantes serán las listas *readables* y *writables* que definen las variables que se intercambiarán con el cliente, estas variables serán conectadas a las variables del cliente fácilmente gracias al elemento RIP de EJSS.

Al declarar estos objetos se pueden definir los siguientes parámetros:

- name: nombre de la variable
- description: breve descripción del objeto
- type: tipo de una variable (int, float, boolean…)
- min, max y precisión: valor mínimo, máximo y precisión del objeto

En la siguiente tabla se recoge cada uno de los objetos que se intercambiarán con el cliente:

Readables			Writables		
Nombre	**Descripción**	**Tipo**	**Nombre**	**Descripción**	**Tipo**
u	Señal de control de la bomba	float	**Kp**	Constante de proporcionalidad PID	float
h	Altura del tanque de salida	float	**Ti**	Tiempo integral del PID	float
t	Tiempo del servidor	float	**Td**	Tiempo derivativo del PID	float
idMaestro	Identificador del maestro	string	**Sp**	Valor de nivel deseado	float
kpMaestro	Constante de proporcionalidad PID del maestro	float	**bomba**	Potencia de la bomba	float
tiMaestro	Tiempo integral del PID del maestro	float	**modo**	Tipo de control	int
tdMaestro	Tiempo derivativo del PID del maestro	float	**alfa**	Filtro del PID	float
spMaestro	Valor de nivel deseado del maestro	float	**Ts**	Tiempo de muestreo del PID	float
bombaMaestro	Potencia de la bomba del maestro	float	**sesion**	Identificador de la sesion	string
modoMaestro	Tipo de control del maestro	int	**cerrar**	Aviso desconexión	boolean
alfaMaestro	Filtro del PID fijado por el maestro	float	**limpiar**	Aviso limpiar gráfica	boolean
tsMaestro	Tiempo de muestreo fijado por el maestro	float			
idCerrado	Identificador de una sesión cerrada	string			
idNuevo	Identificador de una sesión nueva	string			

Tabla 1. Variables intercambiadas mediante el protocolo RIP

Esta lista de variables también se introducirá en el espacio de trabajo de Matlab. Cada segundo, periodo en el que se actualizan los valores, se escribirán los valores de la lista *writables* en el workspace y se ejecutará la función de Matlab con el control correspondiente. Después de realizar el control correspondiente se recogerán los valores de la lista *readables* del workspace para enviárselos al cliente.

4.2.1.4. App.py

Este archivo es el que se tendrá que ejecutar para arrancar el servidor. En este se define la función *load_control* que carga dinámicamente el módulo y la clase de control especificados en la configuración.

En el bloque principal, carga el control desde la configuración, obtiene la configuración SSL y arranca el servidor HTTP (*HttpServer*) con los parámetros de host, puerto y control obtenidos de *AppConfig.*

4.2.2. Script de Matlab

El funcionamiento de este script consiste en leer los valores que el servidor de Python escribe en el workspace, realizar las acciones de control necesarias y, por último, escribir los nuevos valores en el workspace para que el servidor de Python pueda recogerlos y mandárselos al cliente. La función de control se realizará de forma paralela, ya que el cliente podrá modificar el tiempo de muestreo, esta función será la encargada de leer el valor del sensor de la tarjeta y de escribir el valor calculado de la bomba.

Este script se compone de varias secciones y funciones que se explican en los siguientes apartados.

4.2.2.1. Inicialización y manejo de variables

En este script las variables se tienen que fijar en el espacio de trabajo, para que el servidor de Python las reciba y además se puedan utilizar desde cualquier función del script. Para ello, Matlab ofrece las funciones *evalin* y *assignin*.

La función *evalin* permite leer los valores del workspace, para ello utiliza dos parámetros, el primero es el workspace del que leemos las variables, en nuestro caso será siempre base, el segundo parámetro es el nombre de la variable.

La función *assignin,* permite escribir un valor en una variable del espacio de trabajo, para ello requiere 3 parámetros, los dos primeros son igual que en *evalin* y el tercero es el valor que se quiere escribir.

A la hora de inicializar variables, con *evalin* podemos evaluar si existe una variable en el workspace y en caso de que no exista introducirla con un valor inicial con la variable *assignin*, como se muestra a continuación:

```
for k = 1:size(variables, 1)
    if evalin('base', ['exist("' variables{k, 1} '", "var") == 0'])
        assignin('base', variables{k, 1}, variables{k, 2});
    end
end
```

En el bucle for se recorre *variables* que es una matriz de celdas compuesta por parejas con el nombre de la variable y el valor inicial, de esta forma nos ahorramos muchas líneas de código que si se hiciese para cada variable por separado.

El valor se asigna únicamente si la variable no existe ya que, para el resto de las ejecuciones, los valores utilizados serán los obtenidos y calculados por el servidor. Además, es necesario inicializar las variables para evitar fallos en las primeras ejecuciones de algunas funciones.

4.2.2.2. Inicio, manejo y cierre de las sesiones con los clientes

Cada cliente tiene asignado un identificador que envía al servidor dentro de la variable *sesion.* Los identificadores(ids) de cada cliente se irán almacenando en un vector denominado *ids*, el cliente maestro es el que se ha conectado en primer lugar, es decir, el que tenga su id en la primera posición del vector. . Al desconectarse el maestro el que esté en la siguiente posición tomará el rol de maestro, como puede verse en la *Figura 20*.

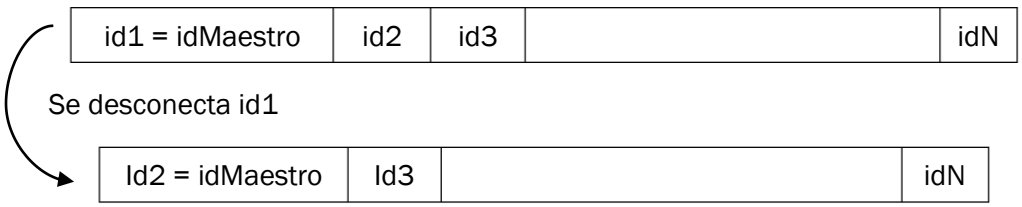


Figura 20. Evolución del vector ids tras desconexión del maestro

Se asigna a la variable *idMaestro* el valor del primer elemento de *ids*, esto nos permite comparar el *idMaestro* con el identificador de cada sesión y hacer caso únicamente de los valores fijados por dicho cliente.

Por esta razón se utilizan en el servidor de Python variables con y sin la palabra maestro, las primeras son variables que el servidor cogerá del workspace y las hará llegar a los clientes, mientras que las otras son los valores que llegan de cada cliente. Con estos valores más el de la variable maestro, se podrá incluir la lógica necesaria en el código del cliente, explicado en el apartado 4.3, para que solamente el maestro pueda actuar sobre la planta.

Para saber si una sesión es nueva se ha implementado la función *contiene_valor* que comprobará si el vector ids contiene el identificador de la sesión y devolverá true en caso de que sí y falso en caso contrario.

A la hora de cerrar la sesión, cuando se reciba que un cliente se quiere desconectar, es decir, cuando la variable cerrar sea true, se eliminará el identificador de esa sesión del vector ids y se actualizará en el workspace. Por último, se actualizará la variable *idCerrado* para que el cliente sepa que se ha desconectado.

4.2.2.3. Fichero de configuración

Al inicio de la función se obtendrán una serie de parámetros de un fichero de configuración, los parámetros serán el nombre del servidor de la tarjeta OPC, el de las variables de la tarjeta correspondientes al sensor de nivel y del actuador y, por último, los valores de la pendiente y constante de la recta que relaciona la tensión con el porcentaje de altura del tanque de salida. A continuación, se muestra el aspecto del fichero de configuración:

```
IMPORTANTE:
--> Dejar un espacio antes y después del igual
--> Para los decimales utilizar punto
A0 = -25.4
A1 = 25.4
sesionOPC = Servidor OPC USB-1408FS-Plus
idBomba = ENTRADAS ANALOGICAS.In_Volts00
idAltura = SALIDAS ANALOGICAS.Out_Volts00
```

Figura 21. Contenido del fichero de configuración

Para obtener los datos se hace uso de la función *leerConfig*. Esta función abre el archivo de configuración, lee su contenido línea por línea, ignora las líneas de cabecera, extrae los valores de las variables especificadas y asigna estos valores al workspace de MATLAB. Finalmente, cierra el archivo.

4.2.2.4. Conexión OPC

La primera vez que se llame a la función se realizará la conexión con el servidor OPC de la tarjeta. Para hacerlo una sola vez, se hace uso de la variable de tipo persistent bajo el nombre *opc*, de modo que, si *opc* está vacía, lo que ocurrirá en la primera ejecución, se ejecuta la función *conectar_opc,* después de eso se fija el valor de *opc* a true para que no se vuelva a conectar en las ejecuciones posteriores.

La función conectar_opc, en primer lugar, obtendrá los valores leídos del fichero de configuración que se almacenaron en el workspace, estos son:

- servidorOPC: nombre del servidor OPC de la tarjeta de adquisición de datos
- idAltura: nombre de la entrada correspondiente al sensor de nivel
- idBomba: nombre de la salida correspondiente con la bomba

Después, se hace uso de la función *opcda* para crear un objeto en el host local para *servidorOPC*, este objeto se almacena bajo el nombre *da.* A continuación, se establece la conexión al servidor OPC de la tarjeta.

Por último, con la función *addgroup* se crea un grupo de ítems en un objeto OPC Data Access. Un grupo de ítems es una colección de variables OPC que se gestionan juntas. Al tener las variables en un grupo se gestionan de manera más eficiente y ordenada. Para añadir las variables al grupo creado se utiliza la función *additem*, en la que indicaremos el nombre del grupo y el de la variable como parámetros, en este caso tenemos dos ítems, *IObj1* e *IObj2*, el primero se corresponde con *idAltura* y el segundo con *idBomba*.

Para posteriormente, en la función de control, poder acceder a IObj1 e IObj2 y leer y escribir, respectivamente datos en la tarjeta, será necesario almacenar ambos ítems en el workspace.

4.2.2.5. Control de la planta

Es importante conocer que, el servidor de Python estará actualizando los datos con el cliente cada segundo, es decir, ejecuta el script *servidorRemoto* cada segundo. Al mismo tiempo, se estará ejecutando la función *control* con un periodo fijado por el cliente en la variable Ts. Para poder llevar a cabo esto, se hace uso de temporizadores, al lanzar el script principal por primera vez, se iniciará el temporizador con un tiempo de Ts.

En el código se estará leyendo en cada ejecución *Ts* y guardando ese valor en *Ts_ant* para realizar una comparación de ambos, en el caso de que el cliente cambie el valor del tiempo de muestreo del PID, tendrá distinto valor y el temporizador actual no nos servirá. Por eso, si *Ts* y *Ts_ant* son distintos, se pararán y eliminarán todos los temporizadores con las funciones *stop* y *delete.* Como parámetro se les introducirá la variable interna de Matlab *timerfindall,* que contiene la información de todos los temporizadores.

Una vez hecho esto, se creará el nuevo objeto del temporizador con el nuevo valor de *Ts*, la inicialización del temporizador se hará mediante la función *timer* de la siguiente manera:

```
temp = timer('ExecutionMode', 'fixedRate', 'Period', double(Ts),
'TimerFcn', @Control)
```

Como puede observarse se introducen 6 parámetros, que en realidad van en grupos de dos, estos son:

- ExecutionMode: establece el modo de ejecución del temporizador. En este caso, es 'fixedRate', lo que significa que el temporizador se ejecutará a intervalos regulares.
- Period: establece el período del temporizador, que es el tiempo de muestreo actual.
- TimerFcn: especifica la función que se ejecutará cuando el temporizador se dispare. En este caso, es la función *Control*.

Por último, se inicia el temporizador recién creado con *start(temp).*

Función Control

Para entender por completo el funcionamiento del servidor, solo falta por explicar la función *control*, que es la encargada de realizar el control de la planta, así como de actualizar los valores obtenidos de la tarjeta en el workspace para hacérselos llegar al cliente.

Lo primero que se hace es obtener los parámetros de control fijados por el cliente del workspace de matlab. Estos parámetros son *modo*, *Kp*, *Ti*, *Td*, *Sp*, *alfa*, *Ts* y *bomba*, lo que representa cada valor que ya se explicó en la *Tabla 1.* Variables intercambiadas mediante el protocolo RIP

A continuación, todos estos valores excepto el valor de la bomba, se asignarán a las variables correspondientes al maestro. También se lee el vector *ids* que contiene los identificadores de las sesiones con los clientes.

El siguiente paso es inicializar las variables necesarias para el cálculo del controlador PID si no existen en el espacio de trabajo base.

Una vez obtenidos todos los parámetros necesarios para el cálculo del PID, pasamos a obtener el valor de la medida del sensor de nivel que, como ya se explicó en el apartado anterior, se corresponde con *IObj1*. Hay que tener en cuenta que este valor viene dado en voltios y a la entrada del PID necesitamos el valor en porcentaje. Es aquí donde entran en juego los valores A0 y A1 del fichero de configuración, que nos permite realizar la aproximación lineal de voltios a porcentaje de la siguiente manera:

$$\mathrm{h}(\%) = \mathrm{h}(\mathrm{V}) * \mathrm{A1}\left(\frac{\%}{\mathrm{V}}\right) + \mathrm{A0} \qquad (4.1)$$

Con el valor de la altura convertido en porcentaje, lo siguiente es obtener la señal del controlador, esto depende del valor almacenado en la variable *modo,* correspondiente al tipo de control fijado por el cliente. En función del modo escogido se utilizarán unas u otras ecuaciones de las obtenidas en el apartado 3.1.

Si se está trabajando en modo manual, el valor de la señal del controlador se corresponderá con el valor almacenado en la variable *bomba*.

En el caso de que se active el modo automático, se tiene la opción del PID ideal y del PID estilo JavaRegula. Para ello, se han implementado en Matlab las ecuaciones (3.10) y (3.15) a (3.18), respectivamente.

Una vez obtenido el valor del controlador, se tiene que realizar una conversión de porcentaje a voltios y también se debe tener en cuenta que el valor ha de estar comprendido entre los 0 y 5 voltios. Por esta razón, si el valor calculado con las ecuaciones es negativo se asignarán 0 voltios, si supera el 100, se le dará un valor de 5 voltios y en otro caso se dividirá el valor entre 20 para convertir la señal de porcentaje a voltios. Para escribir este valor en la tarjeta, se utilizará *IObj2*, como ya se dijo anteriormente y se hará uso de la función *write.* Por último, se escriben los valores de la altura y de la señal de control obtenidos en las variables del workspace para hacérselas llegar al cliente.

4.3. Lado cliente

En este apartado se explica cómo se ha realizado la parte del cliente del laboratorio remoto. Cómo se adelantó en apartados anteriores, se ha hecho uso del software EJSS y de los lenguajes HTML, CSS y JavaScript para completar el cliente.

4.3.1. Ventana modelo en EJSS del cliente

4.3.1.1. Variables

Autoescalado

Nombre	Valor inicial	Tipo	Descripción
autoescalado	false	boolean	Flag de autoescalado
ymaxalt	100	double	Valor máximo altura gráficas
yminalt	0	double	Valor mínimo altura gráficas
ymaxbomba	100	double	Valor máximo bomba gráficas
yminbomba	0	double	Valor mínimo bomba gráficas
hmax	0	double	Valor máximo % altura visible
hmin	0	double	Valor mínimo % altura visible
alturas	[]	double	Array con las alturas
umax	0	double	Valor máximo % bomba visible
umin	0	double	Valor mínimo % bomba visible
controles	[]	double	Array con los % bomba
refmax	0	double	Valor máximo ref visible
refmin	0	double	Valor mínimo ref visible
referencias	[]	double	Array con las referencias
tiempos	[]	double	Array con los tiempos
i	0	int	Iterador para bucles for

Tabla 2. Variables Autoescalado del cliente

Control

Nombre	Valor inicial	Tipo	Descripción
bomba	0	double	Valor de la bomba en modo manual en %
m_bomba	0	double	Valor de *bomba* fijado por el maestro
g_bomba	0	double	Valor de *bomba* usado en la interfaz
ref	0	double	Altura deseada en el tanque de salida en %
m_ref	0	double	Valor de *bomba* fijado por el maestro
g_ref	0	double	Valor de *ref* usado en la interfaz
kp	5	double	Constante proporcional del PID en %/%
m_kp	5	double	Valor de *kp* fijado por el maestro
g_kp	5	double	Valor de *kp* usado en la interfaz
td	0	double	Tiempo derivativo del PID en minutos
m_td	0	double	Valor de *td* fijado por el maestro
g_td	0	double	Valor de *td* usado en la interfaz
ti	0.3	double	Tiempo integral del PID en minutos
m_ti	0.3	double	Valor de *ti* fijado por el maestro
g_ti	0.3	double	Valor de *ti* usado en la interfaz
alfa	0	double	Filtro derivativo del PID
m_alfa	0	double	Valor de *alfa* fijado por el maestro
g_alfa	0	double	Valor de *alfa* usado en la interfaz
u	0	double	Variable de control del PID en %
T	0.5	double	Tiempo de muestreo del PID en segs
m_T	0.5	double	Valor de *T* fijado por el maestro
g_T	0.5	double	Valor de *T* usado en la interfaz

Tabla 3. Variables Control del cliente

Como puede observarse en la Tabla 3, se tienen tres variables diferentes para el mismo elemento, esto es necesario porque los primeros valores, los que tiene el nombre a secas, se conectan a las variables del servidor mediante el elemento RIP. Las variables precedidas por la letra *m*, son las que recibimos del servidor y se corresponden a las del maestro, esto es necesario hacerlo para que los clientes esclavos envíen los mismos valores del maestro en las variables de escritura de la conexión RIP y no modifiquen los valores del servidor. Por último, se tiene una tercera variable que se utilizará para los elementos de la interfaz.

Modelo

Nombre	Valor inicial	Tipo	Descripción
h1	0	double	Valor de la altura en %
t	0	double	Tiempo de ejecución en s
control	1	int	Modo de control
m_modo	1	int	Modo de control del maestro
sesion	""	String	Identificador de esta sesión
idCerrado	"c"	String	Último identificador cerrado
idNuevo	"n"	String	Último identificado nuevo
maestro	false	boolean	Flag de maestro
idMaestro	"0"	String	Identificador del maestro
id	0	double	Id numérico de esta sesión
cerrar	false	boolean	Flag para cerrar sesión
display_bomba	""	String	Display valor actual de la bomba
display_altura	""	String	Display valor actual de la altura
dif_modo		int	Diferencia entre modo de control actual y el anterior
cont_ant		int	Almacena el modo de control anterior

Tabla 4. Variables Modelo del cliente

Interfaz

Nombre	Valor inicial	Tipo	Descripción
opccontrol	["Operación manual", "Control PID"]	String	Opciones de control
elcontrol	"Operación manual"	String	Elección de control
dispopManual	"inline-block"	String	Visibilidad opciones manual
dispControlador	"none"	String	Visibilidad opciones PID
tiempo	0	double	Tiempo de las gráficas
tmin	0	double	Tiempo mínimo gráficas
tmax	30	double	Tiempo máximo gráficas
refx	0	double	Coordenada x referencia
refy	0	double	Coordenada y referencia
opcPID	["PID ideal","PID JavaRegula"]	String	Opciones del PID
elPID	"PID ideal"	String	Elección del PID
tvisible	30	double	Tiempo visible gráficas
bgcolor	"rgb(58, 5, 139,0.7)"	String	Fondo botón habilitado
limpiar	true	boolean	Flag limpiar gráficas
tDelay	0	double	Tiempo de delay con servidor
ripON	false	boolean	Flag conexión RIP
bgOff	"rgb(58, 5, 139,0.2)"	String	Fondo botón deshabilitado
bgDesconectado	bgOff	String	Fondo botón Des*conectar*
bgConectado	bgcolor	String	Fondo botones *Conectar* y *Configuración*
bgGuardar	bgOff	String	Fondo botón *Guardar*

Tabla 5. Variables Interfaz del cliente

Registro_datos

Nombre	Valor inicial	Tipo	Descripción
arraydatos		String	Vector donde se almacenarán los datos necesarios para la función guardar
registro	false	boolean	Flag del registro de datos
espacio	" "	String	Espacio entre datos
fecha	""	String	Fecha del registro de datos
separador	"---------- ---------- ---------- ---------- ---------- ---------- \n"	String	Separador utilizado en la cabecera del archivo de registro de datos

Tabla 6. Variables Registro_datos del cliente

4.3.1.2. Inicialización

En la ventana de inicialización se han generado las siguientes tres pestañas:

Velocidad_simulación

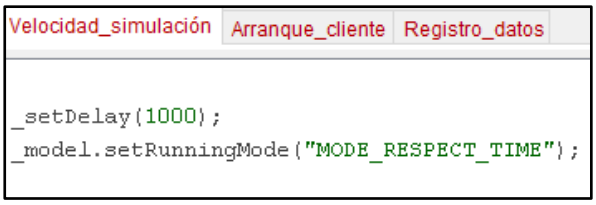


Figura 22. Código Velocidad_simulación del cliente en EJSS

En esta pestaña se especifica que el tiempo de la simulación tome el tiempo real y, se ha especificado un retraso de 1 segundo para que en el registro de datos almacene solamente un dato por segundo.

Arranque_cliente

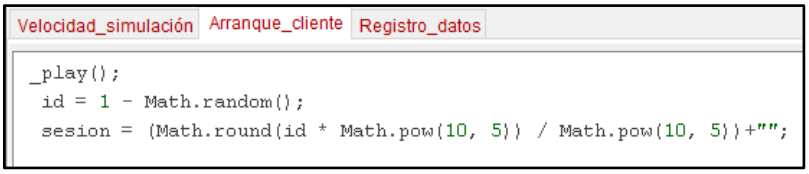


Figura 23. Código Arranque_cliente del cliente en EJSS

En esta pestaña se arranca el cliente con *_play* y además se genera el identificador de este cliente.

Registro_datos

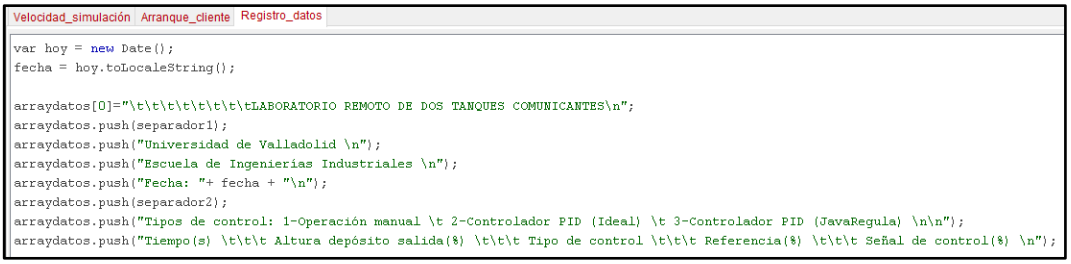


Figura 24. Código Registro_datos del cliente en EJSS

Los resultados de la simulación se guardarán en un archivo de texto. Este archivo incluirá una cabecera con el título del experimento, la universidad y la escuela, así como la fecha y hora de ejecución. Debajo de esta, se añade una explicación del significado de los diferentes valores que puede tener el campo *Tipo de Control* y el nombre de las columnas para los diferentes datos.

4.3.1.3. Relaciones fijas

El código fijado en esta ventana se ejecutará de manera continua con cada actualización de los valores de RIP o cualquier modificación en la interfaz.

Se han generado tres pestañas: Autoescalado, Interfaz y MaestroEsclavo.

Autoescalado

La opción de escalado automático de EJSS escoge el valor máximo y mínimo históricos de la gráfica, esto es un inconveniente cuando cambiamos el tiempo visible en las gráficas ya que puede ser que el máximo histórico sea 89% y actualmente estemos visualizando valores entorno al 40%.

Por esta razón se ha optado por crear un sistema de autoescalado que seleccione unos valores máximo y mínimo entre los mostrados en la gráfica en cada instante. En los próximos párrafos se comenta el sistema de autoescalado desarrollado.

En primer lugar, mediante la función *push* se almacenan los nuevos valores de la altura, del porcentaje de la bomba y el instante de tiempo actual en los arrays *alturas*, *controles, referencias* y *tiempos*, respectivamente.

```
alturas.push(h1);
controles.push(u);
referencias.push(ref);
tiempos.push(t - tDelay);
```

Figura 25. Introducción de los valores actuales a los arrays alturas, controles y tiempos

El siguiente paso consiste en buscar los máximos y mínimos visibles en la gráfica. Para ello, en un bucle for se va recorriendo el vector *alturas* y comprobando si el valor correspondiente en *tiempos* está dentro del rango [*tmin,tmax*], si cumple esta condición quiere decir que ese valor está visible en la gráfica y se realizará la comprobación para ver si es máximo o mínimo. Para comprobar fácilmente si un valor es máximo o mínimo se utilizan las funciones *min* y *max* de la librería *Math* que selecciona el valor menor o mayor entre el valor evaluado en *alturas* y *hmin* o *hmax* (Figura 26).

```
for (i = 0; i < alturas.length; i++) {
  if (tiempos[i] > tmin && tiempos[i] <= tmax) {
    hmax = Math.max(hmax, alturas[i]);
    hmin = Math.min(hmin, alturas[i]);
  }
}
```

Figura 26. Obtención del máximo y mínimo de la altura visibles en la gráfica

Se procede de la misma manera para el vector *controles* almacenando el valor mínimo y máximo en las variables *umin* y *umax*.

```
for (i = 0; i < controles.length; i++) {
  if (tiempos[i] > tmin && tiempos[i] <= tmax) {
    umax = Math.max(umax, controles[i]);
    umin = Math.min(umin, controles[i]);
  }
}
```

Figura 27. Obtención del máximo y mínimo del % de la bomba visibles en la gráfica

Se repite para el vector *referencias* almacenando el valor mínimo y máximo en las variables *refmin* y *refmax*, respectivamente.

```
for (i = 0; i < referencias.length; i++) {
  if (tiempos[i] > tmin && tiempos[i] <= tmax) {
    refmax = Math.max(refmax, referencias[i]);
    refmin = Math.min(refmin, referencias[i]);
  }
}
```

Figura 28. Obtención del máximo y mínimo de la referencia visibles en la gráfica

Por último, solamente si se activa la opción de autoescalado, se asignarán los valores máximos y mínimos a los ejes Y de las gráficas.

Si estamos en modo automático, se escoge el máximo valor entre refmax y hmax y el mínimo valor entre refmin y hmin, mientras que en modo manual solamente se tienen en cuenta hmax y hmin. Para una visualización más clara en las gráficas, se otorga a los valores de las gráficas un margen del 5%. De nuevo, se hará uso de las funciones *max* y *min* de *Math* para que los valores no superen ni el 100 por arriba ni el 0 por abajo.Si la opción autoescalado se tiene desactivada los valores asignados serán el máximo y mínimo posibles, 100 y 0, respectivamente.

```
if (autoescalado) {
  if (control == 1){
    ymaxalt = Math.min(hmax + 5, 100);
    yminalt = Math.max(hmin - 5, 0);
  }else{
    ymaxalt = Math.min(Math.max(hmax, refmax) + 5, 100);
    yminalt = Math.max(Math.min(hmin, refmin) - 5, 0);
  }
  ymaxbomba = Math.min(umax + 5, 100);
  yminbomba = Math.max(umin - 5, 0);
} else {
  ymaxalt = 100;
  yminalt = 0;
  ymaxbomba = 100;
  yminbomba = 0;
}
```

Figura 29. Asignación a los valores máximos y mínimos de las gráficas

Interfaz

En esta pestaña se realizan varias condiciones y acciones en función de los valores de la interfaz. Si se tiene la casilla *Registro de datos* marcada en la interfaz, los valores se almacenarán en el array *arraydatos*. Los valores que se almacenan son el tiempo de ejecución, la altura del tanque de salida, el tipo de control, la referencia fijada y la potencia de la bomba. Además, se modifica el fondo del botón *Guardar*, dando el color activo al botón únicamente si se están registrado datos, ya que no tiene sentido guardar datos si no se está registrando nada.

```
if(registro){
  bgGuardar = bgcolor;
  arraydatos.push("    "+Number.parseFloat(t-tDelay).toFixed(2)+espacio + "    "+
                  Number.parseFloat(h1).toFixed(2)+espacio+ "      "+
                  control+espacio +"   "+Number.parseFloat(ref).toFixed(2)+
                  espacio+ "         "+Number.parseFloat(u).toFixed(2)+"\n");
}
else{
  bgGuardar = bgOff;
}
```

Figura 30. Acciones realizadas si se activa el registro de datos

A continuación, se realizan una serie de depuraciones para que la interfaz funcione de manera correcta. El tiempo que recibimos de ejecución del servidor será mayor que el de ejecución del cliente, por lo que, para que en las gráficas se empiece a contar desde 0, se guarda en *tDelay* los segundos que lleve el servidor en ejecución al lanzar el cliente y en ese momento se limpiarán las gráficas.

Además, como *idNuevo* se recibe mediante RIP, en el momento que se cumpla la condición *idNuevo==sesion,* sabremos que se ha establecido la conexión RIP y se fija *ripON* en true. La variable *ripON* nos permite desactivar y activar los botones *Configuración*, *Conectar* y *Desconectar* según sea necesario.

Una vez establecida la conexión con el servidor no tiene sentido volver a pulsar los botones *Configuración* y *Conectar* de la interfaz, por eso, cuando *ripON* es verdadero, se desactivan estos botones y se cambia su color de fondo a un color apagado. Con el botón *Desconectar* ocurre lo contrario, si no se ha realizado la conexión, este botón permanece con un color apagado y desactivado, cuando *ripON* se activa, se habilita y toma un color vivo.

```
if(idNuevo == sesion){
  limpiar = true;
  tDelay = t;
  ripON = true;
}
else{
  limpiar = false;
}
if (ripON) {
  bgDesconectado = bgcolor;
  bgConectado = bgOff;
}
```

Figura 31. Acciones al iniciar la conexión RIP

Cuando se reciba la confirmación de que se ha eliminado la conexión con el servidor, es decir, cuando se cumpla la condición de la *Figura 32*, se reinicia el botón cerrar y se refresca el cliente por si se quiere cambiar algún parámetro de la configuración o volver a conectar.

```
if (idCerrado==sesion) {
  cerrar = false;
  window.location.reload();
}
```

Figura 32. Acciones realizadas tras desconectar con el servidor

Si el control seleccionado es alguno de los controles automáticos, se tiene que asignar al valor *g_bomba* el valor calculado para la bomba en todo momento, de esta forma, cuando se vuelva al valor manual, se mantendrá la bomba en el último valor calculado.

```
if(control!=1){
  g_bomba = u;
}
```

Figura 33. Acción para correcto cambio de automático a manual

Para la representación del valor de referencia en las gráficas se tienen que actualizar los valores refx y refy como se observa en la *Figura 34*.

```
refy=ref;
refx=t-tDelay;
```

Figura 34. Actualización traza del valor de referencia

Por otro lado, se actualizan los valores del eje de tiempo de las gráficas, si el tiempo del último valor representado supera al tiempo visible seleccionado por el usuario, se modifican los valores mínimo y máximo del eje de tiempo.

```
if( t - tDelay > tvisible){
  tmin=t-tDelay-tvisible;
  tmax=t-tDelay;

}else{
  tmin = 0;
  tmax=tvisible;
}
```

Figura 35. Actualización del eje de tiempo de las gráficas

A continuación, se llevan a cabo las acciones necesarias para la correcta visualización en los selectores del tipo de control de la interfaz.

```
if (!maestro){
  control = m_modo;
  if (m_modo != undefined){
    if (m_modo == 1){
      opccontrol = ["Operación manual"];
      dispopManual = "inline-block";
      dispControlador = "none";

    }
    else{
      if (m_modo == 2){
        opcPID = ["PID ideal"];
      }else{
        opcPID = ["PID JavaRegula"];
      }
      opccontrol = ["Control PID"];
      dispopManual = "none";
      dispControlador = "inline-block";
    }
  }
}
else {
  opccontrol = ["Operación manual","Control PID"];
  opcPID = ["PID ideal","PID JavaRegula"];
}
```

Figura 36. Acciones para la correcta visualización en los selectores de modo

Cuando el cliente no es maestro, los selectores de la interfaz deberán mostrar el valor seleccionado por el maestro, esto se consigue gracias a la variable *m_modo* que recibimos del servidor con el modo de control escogido por el maestro. Por otro lado, si el cliente es el maestro, no es necesario modificar los selectores ya que el valor mostrado se corresponderá directamente con el del maestro.

Por último, se actualiza el valor de los displays situados en las gráficas con los valores instantáneos de la altura del tanque de salida y de la bomba. Para mostrar los valores únicamente son necesarios dos decimales, por eso se tiene que ajustar el valor. Para ello, se multiplican los valores por 100, ya que queremos dos decimales, luego se redondea ese valor para eliminar los decimales y, para terminar, se vuelve a dividir entre 100 para obtener el valor con los decimales de nuevo.

```
display_bomba =  "  Bomba = " + (Math.round(u * Math.pow(10, 2)) / Math.pow(10, 2)) + "%";
display_altura = "  Altura = " + (Math.round(h1 * Math.pow(10, 2)) / Math.pow(10, 2)) + "%";
```

Figura 37. Código para mostrar los valores instantáneos en los displays en las gráficas

MaestroEsclavo

En esta pestaña se actualizan los valores que se envían al servidor (Figura 38).

Por un lado, si el valor de *idMaestro* coincide con el identificador *sesion*, quiere decir que el cliente es el maestro, por eso los valores enviados al servidor son directamente los de la interfaz, es decir, los que selecciona el usuario en la interfaz.

Por otro lado, si el cliente es esclavo los elementos de la interfaz toman los valores que se reciben del servidor correspondientes a los datos del maestro. También se asignan estos valores a los enviados al cliente para no modificar el control en el servidor y tener un correcto funcionamiento.

Además, si el cliente es maestro o no, es aquí donde se modifica el valor de la variable *maestro* que se utiliza para una correcta visualización de la interfaz y para deshabilitar o no alguno de los elementos de la interfaz.

```
if (idMaestro !== undefined){

  if(sesion == idMaestro){
    bomba = g_bomba;
    ref = g_ref;
    kp = g_kp;
    ti = g_ti;
    td = g_td;
    alfa = g_alfa;
    T = g_T;
    maestro = true;
  }
  else{
    bomba = m_bomba;
    g_bomba = m_bomba;
    ref = m_ref;
    g_ref = m_ref;
    kp = m_kp;
    g_kp = m_kp;
    ti = m_ti;
    g_ti = m_ti;
    td = m_td;
    g_td = m_td;
    alfa = m_alfa;
    g_alfa = m_alfa;
    T = m_T;
    g_T = m_T;
    maestro = false;
  }
}
else{
  g_bomba = m_bomba;
  bomba = m_bomba;
}
```

Figura 38. Acciones realizadas en la pestaña MaestroEsclavo del cliente

4.3.1.4. Propio

En esta ventana se pueden crear funciones nuevas, para este proyecto se han generado las funciones *guardar* y *conectar.*

Guardar

La función guardar se ejecutará al pulsar el botón *Guardar* en la interfaz, su código se muestra y explica justo abajo.

```
function guardar () {
    const a = document.createElement("a");
    const archivo = new Blob(arraydatos, { type: 'text/plain' });
    const url = URL.createObjectURL(archivo);
    a.href = url;
    a.download = "Datos Laboratorio Tanques Comunicantes.txt";
    a.click();
    URL.revokeObjectURL(url);
}
```

Figura 39. Código de la función propia guardar

La función crea un objeto tipo documento y un objeto tipo blob con los datos. Una vez hecho esto, se descarga el archivo en el navegador bajo el nombre *Datos Laboratorio Tanques Comunicantes.txt.*

Conectar

Esta función se ejecuta cuando se pulsa el botón *Conectar.* Únicamente realiza dos acciones: iniciar todas las funciones del cliente y conectarse con el servidor RIP como se muestra a continuación:

```
function conectar(){
  _play();
  RIP.connect();
}
```

Figura 40. Código de la función propia conectar

4.3.1.5. Elemento RIP

Como se ha comentado en apartados anteriores de esta memoria, para llevar a cabo la comunicación con el servidor se utiliza el elemento RIP implementado en el software EJSS(Figura 41). Para añadir el elemento al proyecto, basta con arrastrarlo a la lista de elementos. Una vez añadido se le da el nombre que se quiera, en este caso se ha llamado RIP, nótese que este nombre es el utilizado en la función conectar de la *Figura 40*. Esto es importante para establecer la conexión.

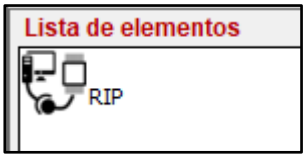


Figura 41. Elemento RIP añadido al proyecto

Para configurar el elemento RIP se clica dos veces sobre este para que aparezca la ventana de configuración compuesta por tres pestañas: *Server Configuration*, *Experience* y *Auto Update*

Para realizar la configuración del elemento, es mejor arrancar el servidor de RIP en python de manera local, es decir, desde la misma máquina, ya que solamente se van a establecer las relaciones entre las variables del cliente con las del servidor. Cuando se tenga el laboratorio en funcionamiento, sí que será necesario introducir la dirección IP del ordenador de la planta de laboratorio donde se tenga el servidor, pero antes se necesita realizar esta configuración para relacionar las variables en ambos extremos de la comunicación RIP.

Por esta razón, se introduce la URL *http://localhost:8080/* en la pestaña *Server Configuration* para conectarse al servidor. En el apartado 4.3.3. se explica el método desarrollado para poder modificar la dirección del servidor desde fuera de EJSS, es decir, desde la interfaz del cliente en el navegador.

Al pulsar en *Get Experiences,* si se ha podido conectar, nos aparecerá la información del servidor de la siguiente manera:

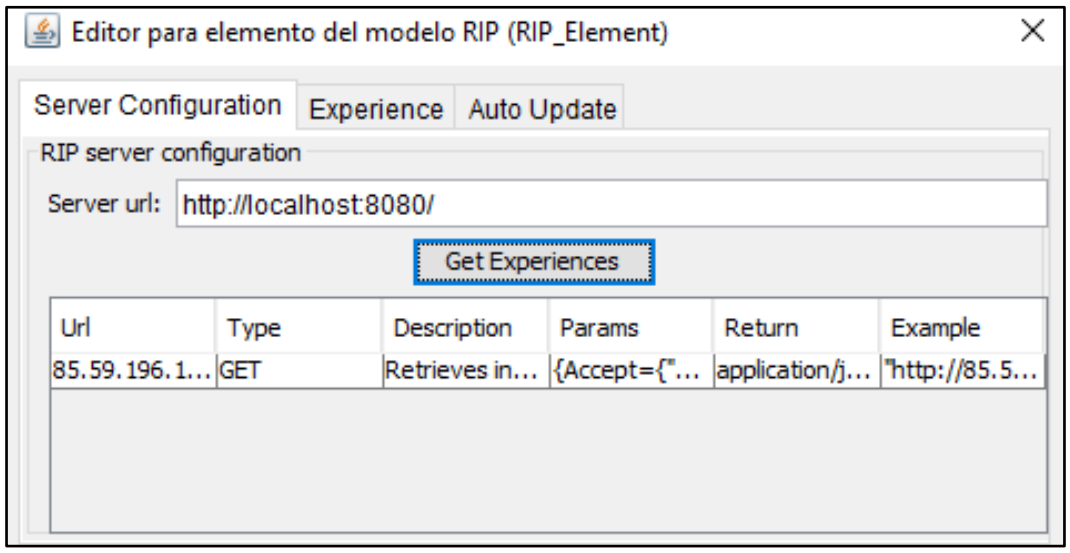


Figura 42. Pestaña Server Configuration una vez conectada con el servidor

Una vez conectado con el servidor, hemos conseguido las experiencias de este, que se pueden ver en la pestaña *Experience*:

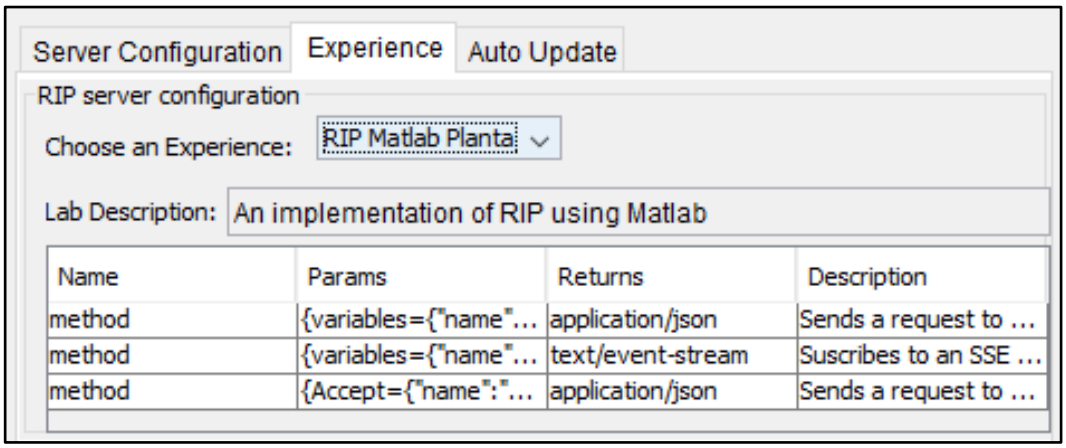


Figura 43. Pestaña Experience en la configuración del elemento RIP

Por último, hay que hacer lo más importante, conectar las variables cliente con las del servidor. Al haber conectado con el servidor, en la pestaña *Auto Update* aparecen las variables del servidor.

Para conectar una variable hay que hacer clic derecho sobre la variable que se quiera conectar y elegir la variable del cliente a la que conectarse. Una vez hecho esto, la conexión establecida se recoge en una tabla inferior.

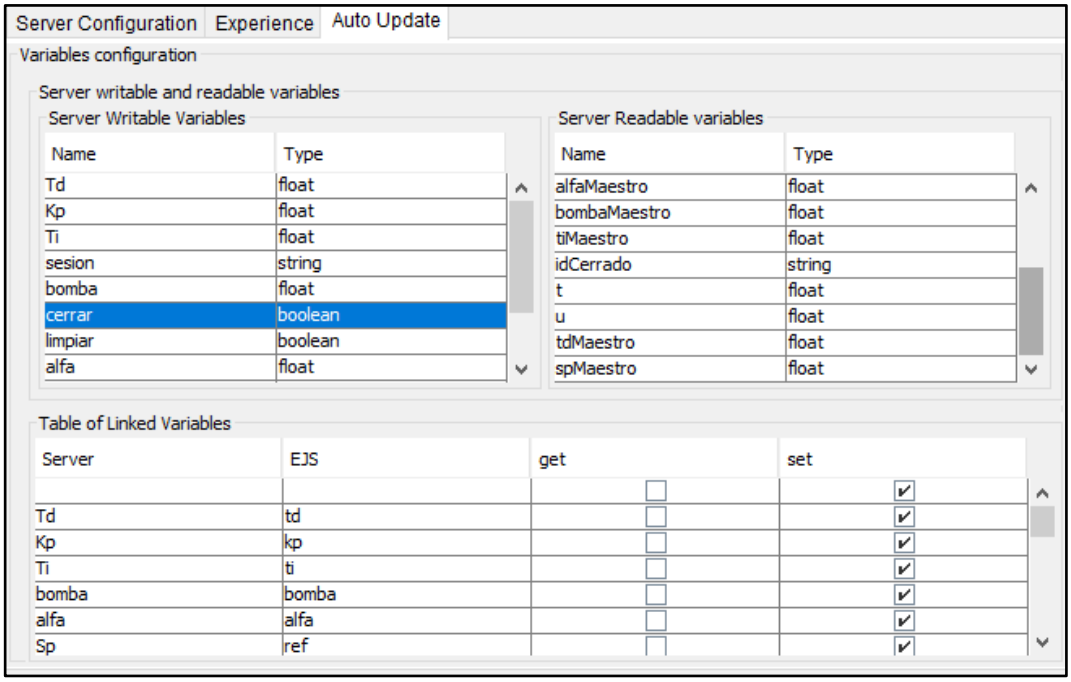


Figura 44. Pestaña Auto Update en la configuración del elemento RIP

Nombre variable en el servidor	Nombre variable en el cliente	Tipo
Td	td	set
Kp	kp	set
Ti	ti	set
bomba	bomba	set
alfa	alfa	set
Sp	ref	set
modo	control	set
Ts	T	set
t	t	get
u	u	get
h	h1	get
idMaestro	idMaestro	get
sesion	sesion	set
kpMaestro	m_kp	get
alfamaestro	m_alfa	get
bombaMaestro	m_bomba	get
tiMaestro	m_ti	get
tdMaestro	m_td	get
spMaestro	m_ref	set
tsMaestro	m_T	get
modoMaestro	m_modo	get
cerrar	cerrar	set
idCerrado	idCerrado	get
idNuevo	idNuevo	get
limpiar	limpiar	set

Tabla 7. Conexión de las variables intercambiadas mediante RIP

Las variables con la casilla *set* marcada son variables que modifica el cliente y las escribe en el servidor, mientras que, si tienen la casilla *get* marcada, su valor es modificado por el servidor. En la Tabla 7 se recogen todas las conexiones realizadas.

4.3.2. Ventana HTML en EJSS del cliente

En esta ventana es donde se crea la interfaz del cliente y se configura cada elemento. La interfaz está compuesta por tres bloques principales:

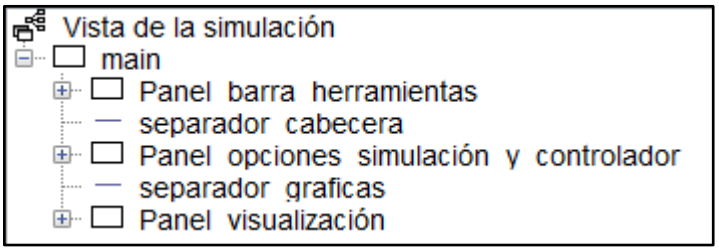


Figura 45. Bloques principales de la interfaz del cliente

Estos tres paneles están dentro de un bloque principal, su función principal es dar color al fondo y ajustar todo el contenido dentro de él.

4.3.2.1. Panel barra de herramientas

Este bloque contiene la barra de herramientas superior cuyo contenido se muestra en la Figura 46.

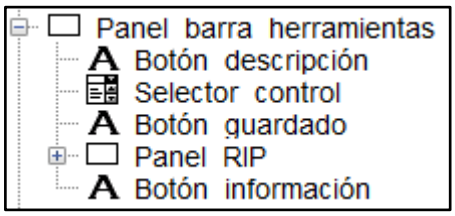


Figura 46. Contenido del panel barra herramientas

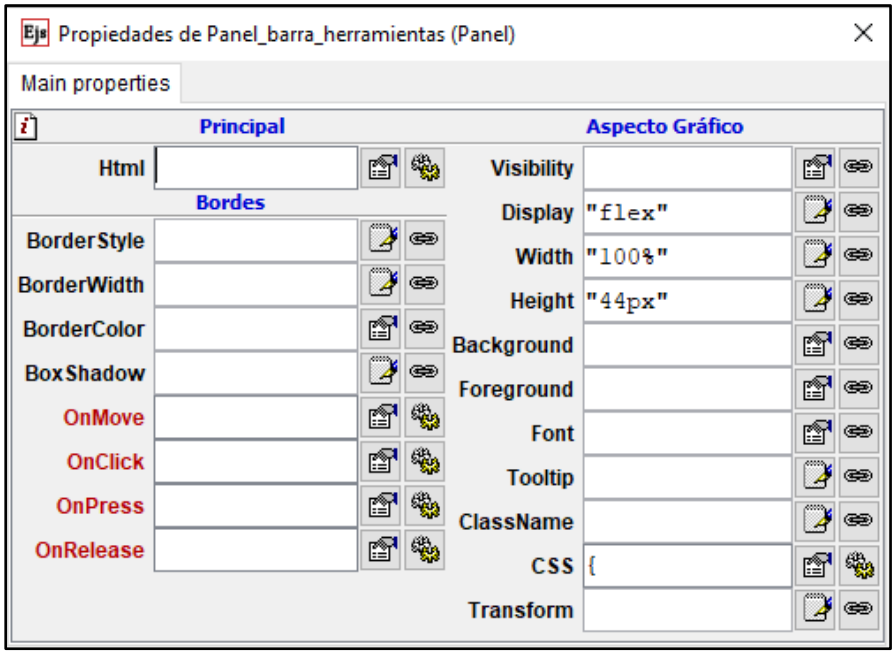


Figura 47. Propiedades del panel barra de herramientas

En el código CSS de la barra de herramientas se ha añadido un padding de 0.5rem.

Botón descripción

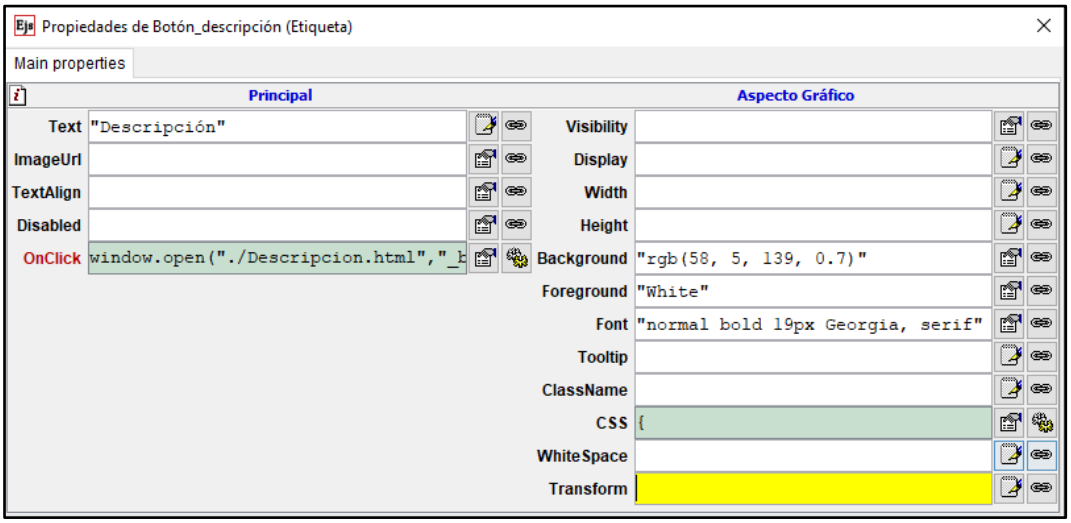


Figura 48. Propiedades del botón Descripción

Al hacer clic en el botón descripción se abrirá la página con la descripción del proyecto en una nueva pestaña, esto se consigue con el siguiente código:

```
window.open("./Descripcion.html","_blank");
```

Y el código CSS del elemento es:

```
"border": "2px solid black",
"border-radius": "5px",
"padding": "0.5rem",
"margin-right": "0.5rem"
```

Figura 49. Código CSS del botón Descripción

Selector control

Este elemento contiene el código necesario para el correcto funcionamiento del selector de control. Véase que la variable *maestro* nos sirve para deshabilitar el selector cuando el cliente sea esclavo, anulando de esta manera el control de este sobre la planta.

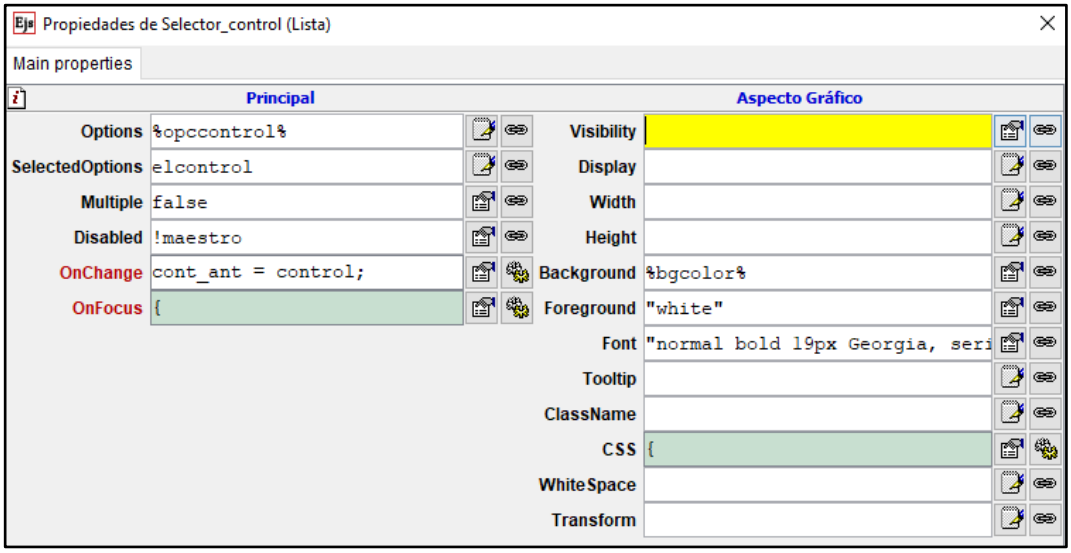


Figura 50. Propiedades del selector del tipo de control

La propiedad *OnFocus* realiza acciones cuando el elemento está enfocado. En este caso se cambiará el fondo del elemento introduciendo el siguiente código:

```
bgcolor = "rgb(58, 5, 139, 0.5)"
```

En el apartado CSS del selector se ha introducido solamente el borde del botón de la siguiente manera:

```
"border": "2px solid black",
"border-radius" : "5px"
```

Figura 51. Código CSS del selector del tipo de control

En código de la propiedad *OnChange* se ejecuta cada vez que se cambie de selección. Esto nos permite asignar el valor correspondiente a la variable *control* en función de la selección del cliente. Además, en función de si la elección de control es manual o automática, se debe ocultar o mostrar el panel correspondiente a cada tipo de control.

Por último, para un correcto cambio del modo manual al automático, se almacena en la variable *cont_ant,* el modo de control escogido anteriormente, y en *dif_modo*, se guardará la diferencia entre el control actual y el anterior. De esta manera, si la diferencia vale 1 o 2 y el control anterior es manual, se le asignará al valor de referencia del PID el valor actual de la altura del tanque de salida. Haciendo esto, se mantendrá el nivel obtenido en el modo manual en el tanque (cambios de manual a automático sin saltos – bumpless).

```
cont_ant = control;
if(elcontrol=="Operación manual"){
  control = 1;
  dispopManual = "inline-block";
  dispControlador = "none";
}

else if(elcontrol=="Control PID"){
  if(elPID=="PID ideal"){
    control=2;
  }else if (elPID=="PID JavaRegula"){
    control=3;
  }
  dispopManual = "none";
  dispControlador = "inline-block";
}
dif_modo = control-cont_ant;
if ((dif_modo==1||dif_modo==2) && cont_ant==1){
  ref = h1;
  g_ref = h1;
  m_ref = h1;
}
```

Figura 52. Código ejecutado al cambiar de opción en el selector de tipo de control

Botón guardado

El botón de guardado, como ya se ha comentado anteriormente, ejecutará la función guardar al hacer clic sobre él, y solamente estará habilitado si se está realizando un registro de datos.

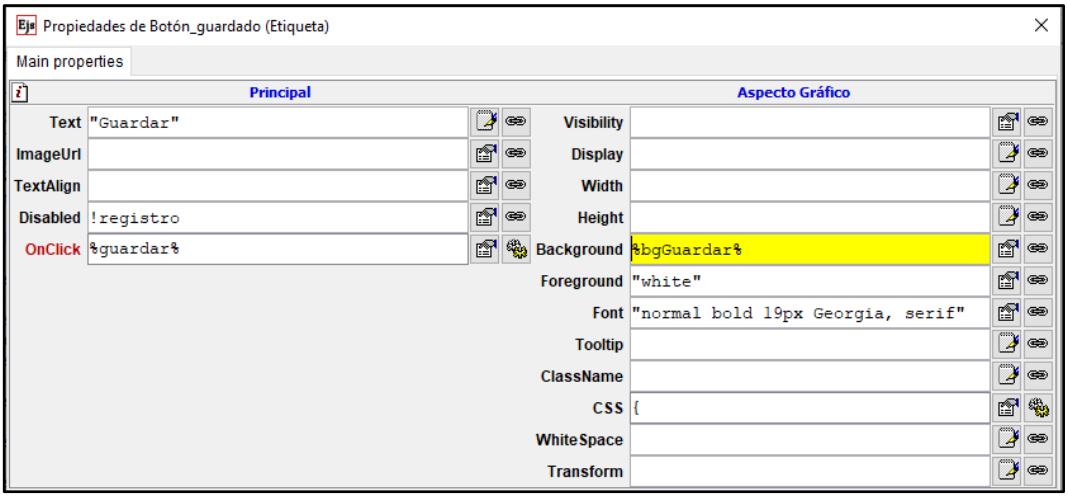


Figura 53. Propiedades del botón de guardado

El código CSS de este elemento consiste en dar formato a los bordes como se hace en todos los elementos de la cabecera y además se ha añadido un margen a la izquierda del elemento para separarlo del selector de control.

Panel RIP

En este panel se han introducido los tres botones relacionados con la conexión con el servidor, estos son: *Configuración, Conectar* y *Desconectar.* El tener los tres botones dentro del mismo panel facilita la configuración para situarlos en la parte superior derecha. Para hacer esto, basta con asignar a la propiedad *Display* del panel RIP el tipo *flex* y el siguiente CSS:

```
"align-items": "center",
"float":"right",
"margin-top":"5px",
"margin-right":"5px",
"margin-bottom":"5px"
```

Figura 54. Código CSS del panel RIP

Como se comentó en apartados anteriores, la variable *ripON* representa que la conexión con el cliente se ha establecido y por eso, si es true se deshabilitan los botones de configuración y conexión y se habilita el botón de desconexión.

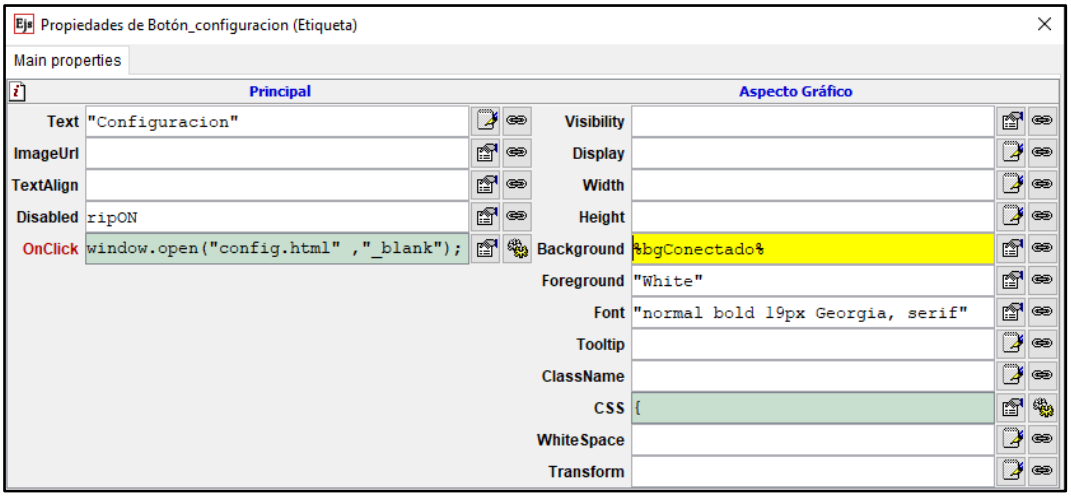


Figura 55. Propiedades del botón de configuración

La acción llevada a cabo al pulsar el botón de configuración es abrir la página con la configuración para la comunicación RIP como indica la Figura 55.

Al clicar sobre el botón *Conectar*, se ejecutará la función *conectar*.

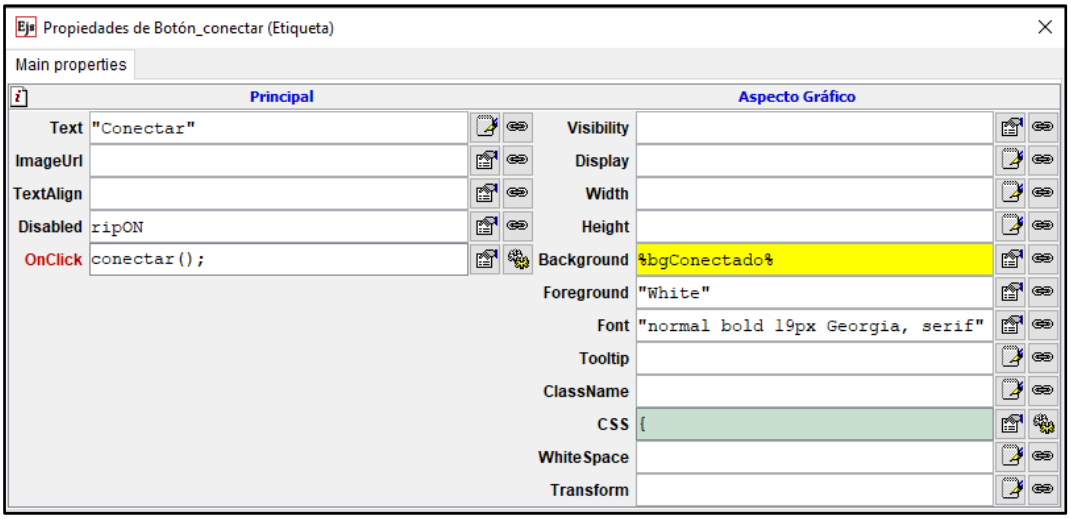


Figura 56. Propiedades del botón Conectar

Con el botón *Desconectar,* solamente se modificará el valor de la variable cerrar a true para notificárselo al servidor.

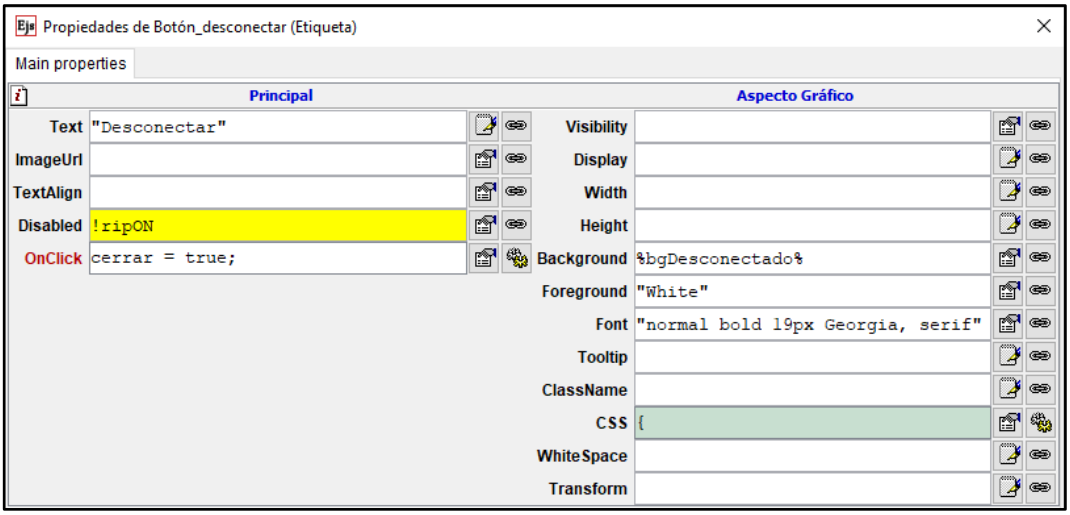


Figura 57. Propiedades del botón Desconectar

El código CSS que se ha incluido es el mismo para los tres botones:

```
"border": "2px solid black",
"border-radius": "5px",
"padding": "0.5rem",
"margin-right": "0.5rem"
```

Figura 58. Código CSS para los botones del panel RIP

Botón información

El botón de información abre la página con los créditos de proyecto al introducir el siguiente comando en el parámetro *OnClick*:

```
window.open("./Creditos.html","_blank");
```

Además, en lugar de introducir texto, se ha optado por utilizar el icono de la Figura 59. Para conseguir esto, se introduce la ruta con el archivo del icono en el parámetro *ImageUrl.*

Figura 59. Icono utilizado como botón de información

4.3.2.2. Panel opciones interfaz y controlador

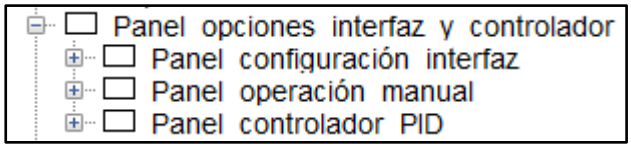


Figura 60. Contenido del Panel opciones interfaz y controlador

Este panel contiene los elementos con los parámetros para la configuración de la interfaz gráfica y también los parámetros del control que se esté realizando ya sea manual o automático. Para situar los elementos correctamente, se introduce el tipo *flex* en el parámetro *Display* de las opciones del panel. Además, se utiliza el siguiente código CSS:

```
"flex-direction":"row",
"align-self": "center",
"padding" : "0.5rem"
```

Figura 61. Código CSS del panel opciones interfaz y controlador

Panel configuración interfaz

En este panel también se ha introducido la característica *flex* en el parámetro Display y además al fondo el color *rgb(58, 5, 139, 0.3)*, dónde el último valor se corresponde con la transparencia. El código CSS, que será el mismo para el *Panel operación manual* y para el *Panel controlador PID* puede verse en la Figura 62.

```
"float":"left",
"flex-direction":"column",
"border-radius" : "10px",
"border" : "2px solid black",
"align-self" : "center"
```

Figura 62. Código CSS para los paneles configuración interfaz, operación manual y controlador PID

El contenido del *Panel configuración interfaz* es el siguiente:

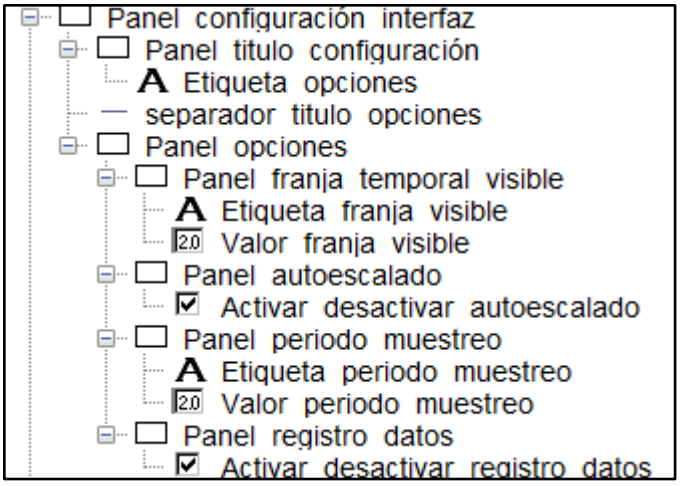


Figura 63. Contenido del panel de configuración de la interfaz

Al *Panel título configuración* se le ha asignado el color de fondo *rgb(58, 5, 139)* y las siguientes características CSS:

```
"padding-top":"1rem",
"padding-bottom":"0px",
"border-radius": "10px 10px 0px 0px"
```

Figura 64. Código CSS para los títulos de los paneles título configuración, título operación manual y título control PID

A la *Etiqueta opciones*, se le da el color el blanco y una tipografía *normal bold 22px Georgia, serif*. En el *Panel opciones,* el contenido se distribuye como si fuese una tabla, esto se consigue con *grid* en la propiedad *Display* del panel. En el código CSS del panel opciones se introducen las características del *grid*.

```
"float":"left",
"grid-template-columns":"300px 200px",
"grid-template-rows":"50px 50px"
```

Figura 65. Código CSS del panel opciones

Dentro de *Panel opciones*, se tienen cuatro parámetros como puede observarse en la *Figura 63.* A los cuatro paneles se les ha asignado el tipo *flex* en la propiedad *Display* y el código CSS mostrado en Figura 66 y Figura 67.

```
"padding" : "0rem",
"flex-direction":"row",
"margin":"0.5rem 0.5rem 0.5rem 1.5rem"
```

Figura 66. Código CSS para los paneles franja temporal visible y periodo muestreo

```
"padding" : "0rem",
"flex-direction" : "row",
"margin": "0.5rem 0.5rem 0.5rem 0.5rem"
```

Figura 67. Código CSS para los paneles autoescalado y registro datos

En cuanto a las etiquetas *Etiqueta franja visible* y *Etiqueta periodo muestreo* se les ha asignado como única propiedad, obviando el texto de la etiqueta, la fuente *normal normal 16px*.

Ya solamente falta por comentar los elementos del panel opciones que conectan con las variables del laboratorio.

En primer lugar, al elemento *Valor franja visible* se le han asignado las propiedades de la *Figura 68* y cómo único parámetro CSS se ha introducido la propiedad *aling-self* con un valor *center* para que se sitúe en el centro del panel que lo contiene.

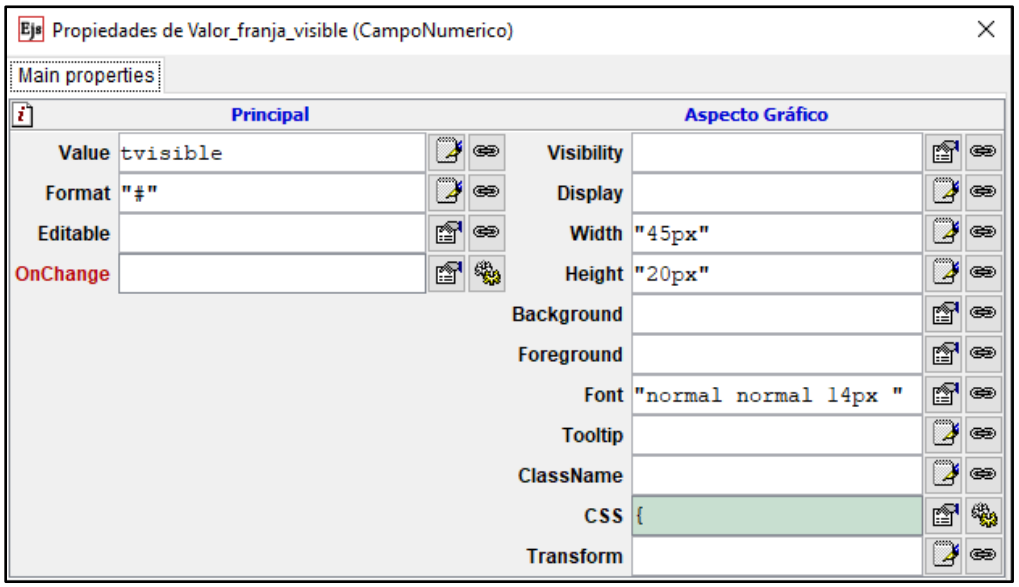


Figura 68. Propiedades del elemento Valor franja visible

Al elemento *Activar desactivar autoescalado*, se le asigna la variable *autoescalado* a la propiedad *Checked* y el texto *Autoescalado eje Y*.

Al elemento *Valor periodo muestreo* se le han asignado las siguientes propiedades:

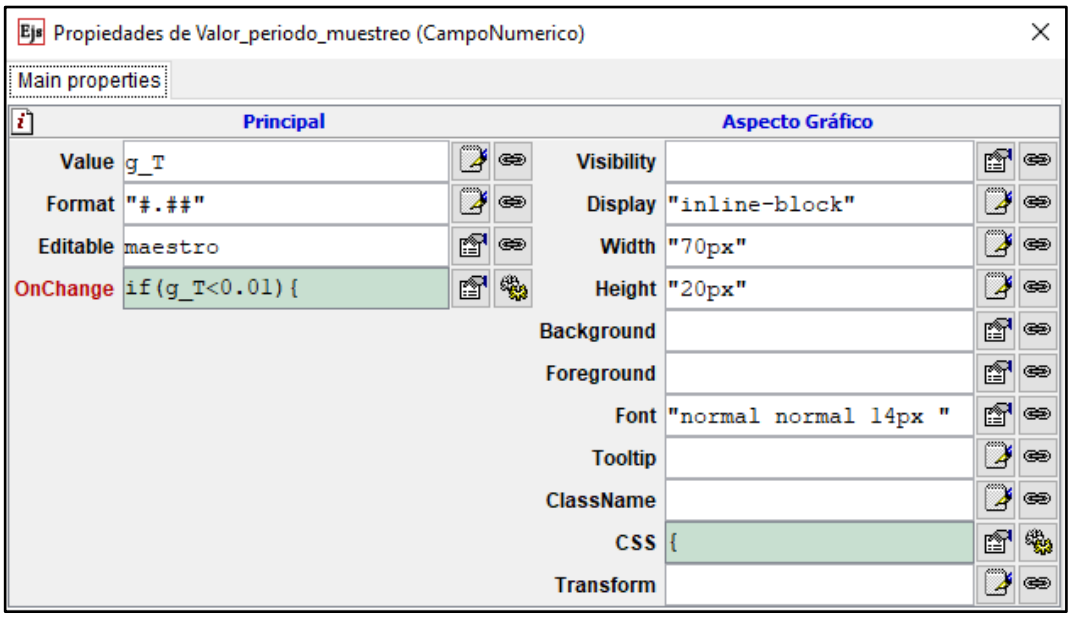


Figura 69. Propiedades del elemento Valor periodo muestreo

Como se observa en la figura superior, este elemento solamente es editable si el cliente es maestro del control, además se ha introducido un código evaluando el valor introducido con el que se limita el valor del periodo de muestreo a un mínimo de 0.01 segundos.

```
if(g_T<0.01){
  g_T=0.01;
  }

_resetSolvers();
```

Figura 70. Código de la propiedad OnChange del elemento Valor periodo muestreo

Por último, al selector *Activar desactivar registro datos*, se le ha conectado con la variable *registro*.

Con todo esto, se obtiene el siguiente resultado en la interfaz(Figura 71).

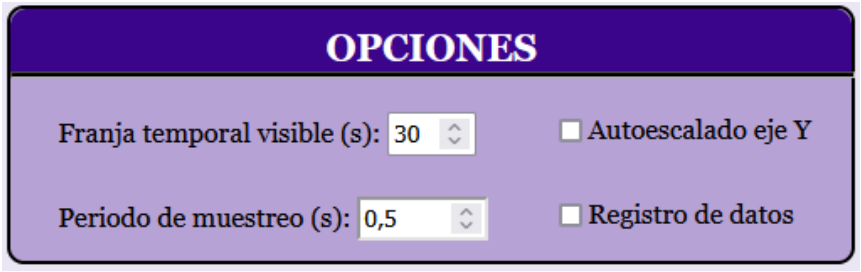


Figura 71. Aspecto del panel opciones interfaz y controlador

Panel operación manual

Las propiedades del *Panel operación manual* son las mismas que para el *Panel configuración interfaz* con la única diferencia en el parámetro *Display*, donde se utiliza la variable *dispopManual*.

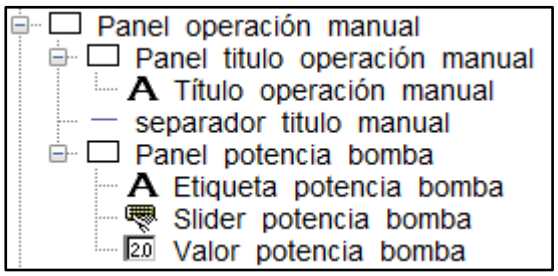


Figura 72. Contenido del panel para el control manual

El *Panel título operación manual* y la etiqueta *Título operación manual* poseen las mismas características que el *Panel título configuración* y la *Etiqueta opciones*, respectivamente. El único cambio, lógicamente, es el texto de la etiqueta.

En la operación manual solamente se manipula la potencia de la bomba, por eso únicamente se tiene el *Panel potencia bomba* con la propiedad *Display* en *flex* y el siguiente CSS:

```
"padding" : "1rem",
"flex-align":"row"
```

Figura 73. Código CSS del Panel potencia bomba

La etiqueta *Etiqueta potencia bomba* tiene una fuente *normal normal 16px.* El valor de la potencia de la bomba se puede modificar con una slider o introduciendo directamente el valor.

La slider tiene las propiedades y código CSS mostradas en la Figura 74.

```
"margin-left": "0.5rem",
"margin-right" : "0.5rem",
"align-self" : "center"
```

Figura 74. Código CSS de Slider potencia bomba

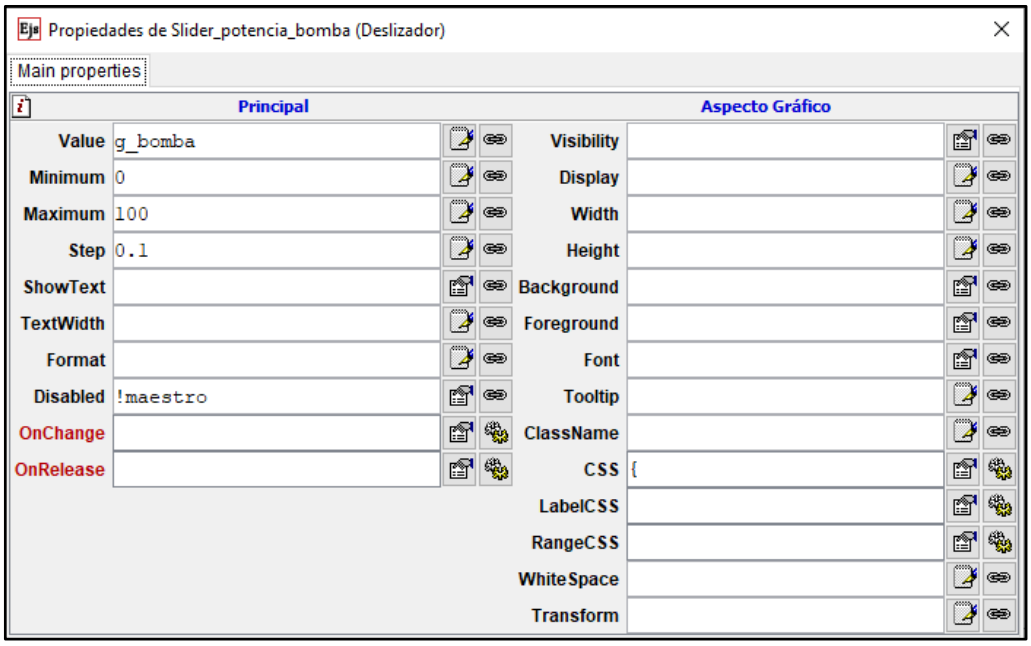


Figura 75. Propiedades de Slider potencia bomba

Las propiedades de *Valor potencia bomba* son las siguientes:

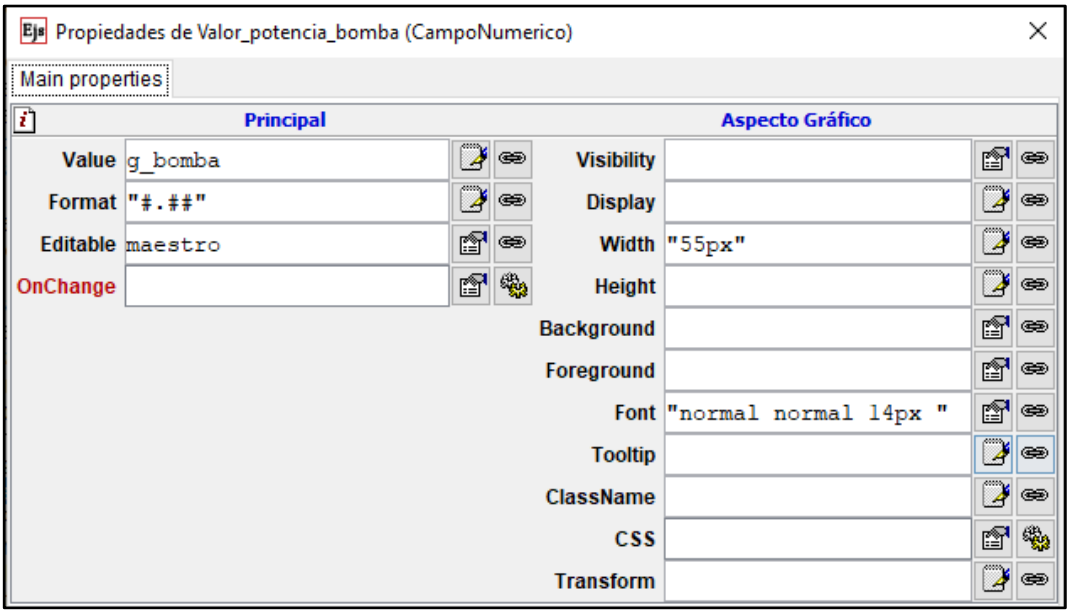


Figura 76. Propiedades del elemento Valor potencia bomba

Nótese el uso de la variable *maestro* para inhabilitar los elementos. En el caso de la slider, se introduce el valor opuesto, ya que la propiedad *Disabled* necesita un valor false para que sea manipulable, a diferencia del valor numérico, ya que para la propiedad *Editable* es en el caso false cuando no se podrá modificar.

Con todo esto, el aspecto del *Panel operación manual* se ve así:

Figura 77. Aspecto de Panel operación bomba

Panel controlador PID

Las propiedades del *Panel controlador PID* son las mismas que para el *Panel configuración interfaz* a excepción del parámetro *Display*, donde se utiliza la variable *dispopControlador*.

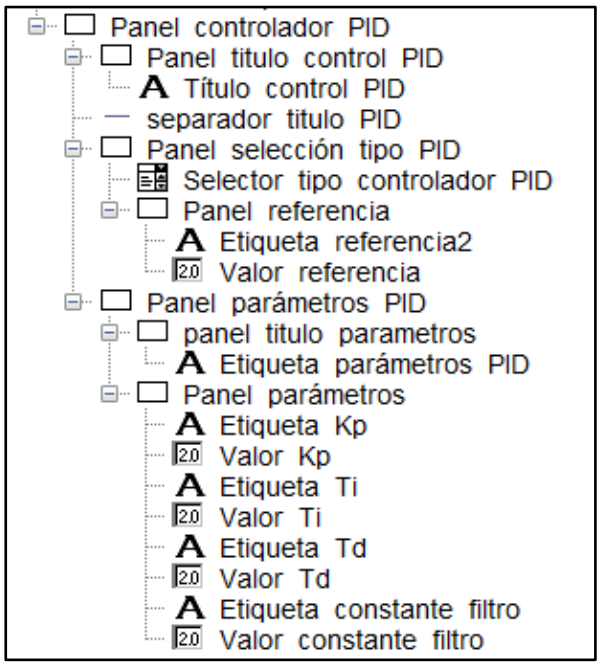


Figura 78. Contenido del panel controlador PID

El *Panel título control PID* y la etiqueta *Título control PID* tiene las mismas propiedades que el *Panel título configuración* y la *Etiqueta opciones*, respectivamente. Cambiando únicamente el texto de la etiqueta.

Al *Panel selección tipo PID* se le ha asignado el tipo flex al parámetro Display y el siguiente CSS:

```
"flex-direction" : "row",
"padding" : "0.25rem 1rem 0.75rem 1rem",
"align-self" : "center",
"justify-content": "center"
```

Figura 79. Código CSS del Panel selección tipo PID

El *Selector tipo controlador PID* tiene las siguientes propiedades:

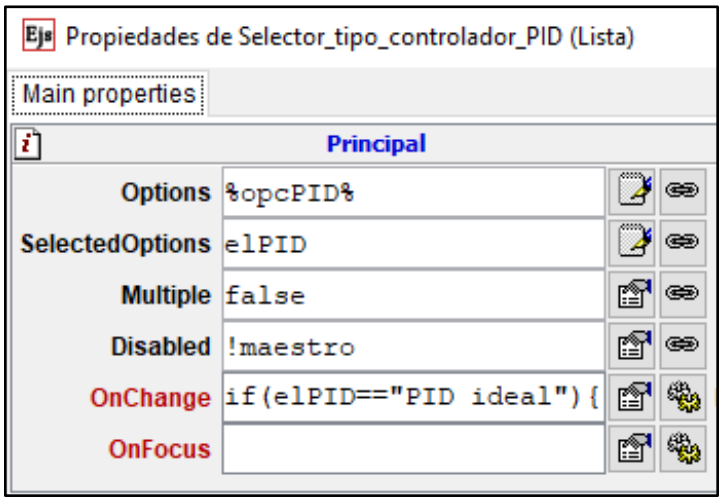


Figura 80. Propiedades del selector tipo controlador PID

Al cambiar de elección del tipo de PID se tiene que actualizar la variable *control* para eso se introduce el siguiente código en el parámetro *OnChange:*

```
if(elPID=="PID ideal"){
    control=2;
}else if (elPID=="PID JavaRegula"){
    control=3;
}
_resetSolvers;
```

Figura 81. Código del parámetro OnChange del Selector tipo controlador PID

El panel referencia se compone de una etiqueta con el nombre de la variable(*Etiqueta referencia2*) y de un campo numérico donde se introduce el valor de referencia del controlador(*Valor referencia*). Al primer elemento se le han dado solamente el color negro y la fuente: *normal normal 16px*. Al campo numérico se le han asignado las propiedades de la Figura 82.

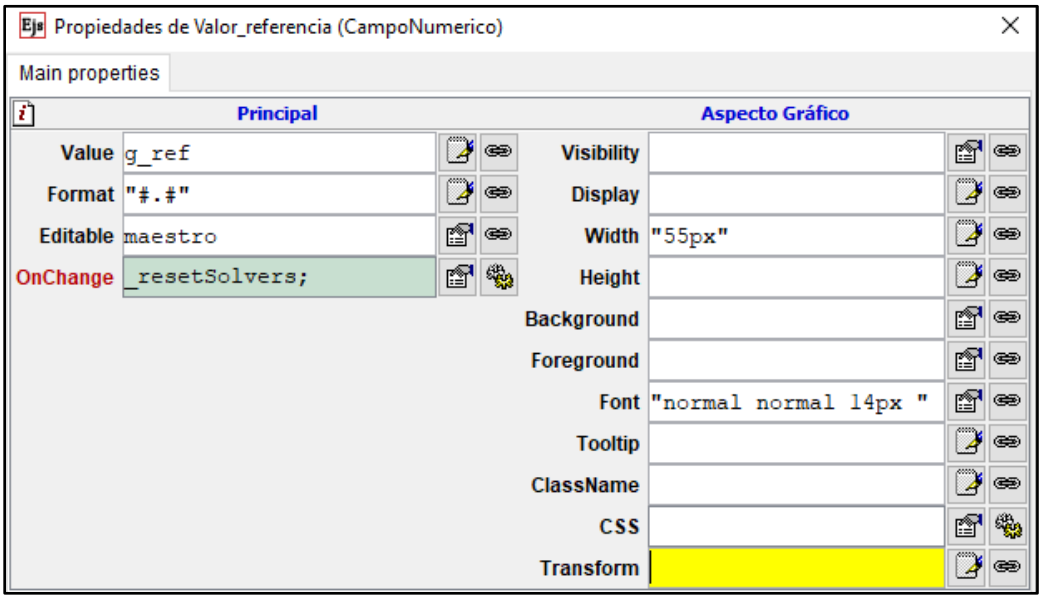


Figura 82. Propiedades del elemento Valor referencia

Por otro lado, se ha creado el *Panel parámetros PID* donde se incluyen la constante proporcional(*Kp*), el tiempo integral(*Ti*), el tiempo derivativo(*Td*) y el filtro derivativo(*α*). Este panel tiene el tipo *flex* en el parámetro *Display* y el siguiente CSS:

```
"margin" : "0px 2px 0px -2px",
"flex-direction":"column"
```

Figura 83. Código CSS del Panel parámetros PID

Este panel se compone de dos subpaneles uno para el título y otro para cada uno de los valores.

El panel del título tiene el color de fondo *rgb(58, 5, 139, 0.7)* y el CSS de la *Figura 84*, y la *Etiqueta parámetros PID* tiene color blanco y una fuente *normal normal 18px*.

```
"border-width" : "2px",
"border-color" : "black",
"border-style" : "solid",
"border-radius" : "10px 10px 0px 0px"
```

Figura 84. Código CSS del panel título parámetros

El segundo subpanel es *Panel parámetros* al que se le ha dado el tipo *grid* a su parámetro *Display* y la configuración en CSS de la Figura 85.

```
"padding":"0.5rem",
"grid-template-columns":"100px 80px 100px 80px",
"grid-template-rows":"40px 40px",
"align-self" : "center",
```

Figura 85. Código CSS del Panel parámetros

Por último, las cuatro etiquetas y campos numéricos de los valores tienen las mismas propiedades, cambiando únicamente el parámetro *text* o *value* según corresponda.

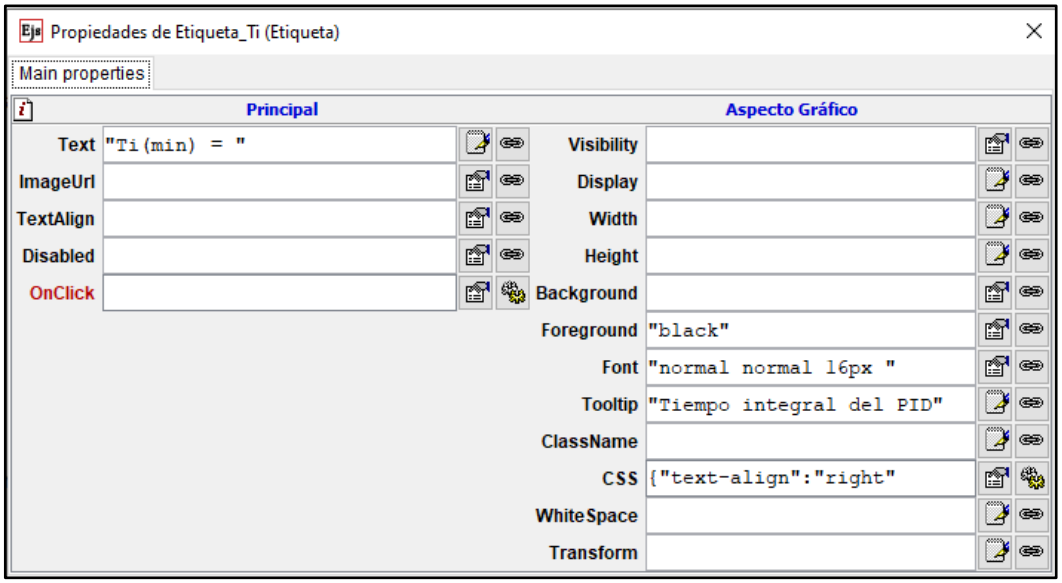


Figura 86. Parámetros de las etiquetas Kp, Ti, Td y constante filtro

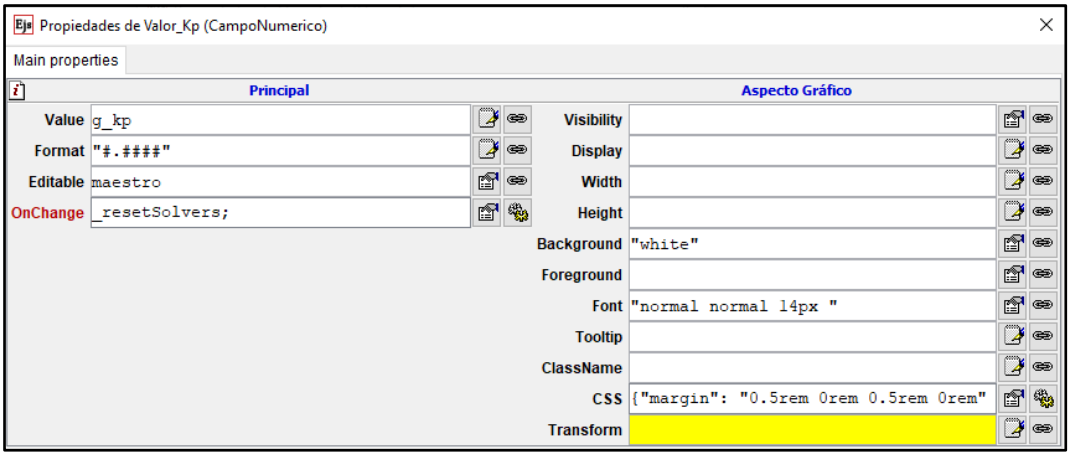


Figura 87. Parámetros de los elementos Valor Kp, Valor Ti, Valor Td y valor constante filtro

Finalmente, *Panel controlador PID* tiene el aspecto de la Figura 88.

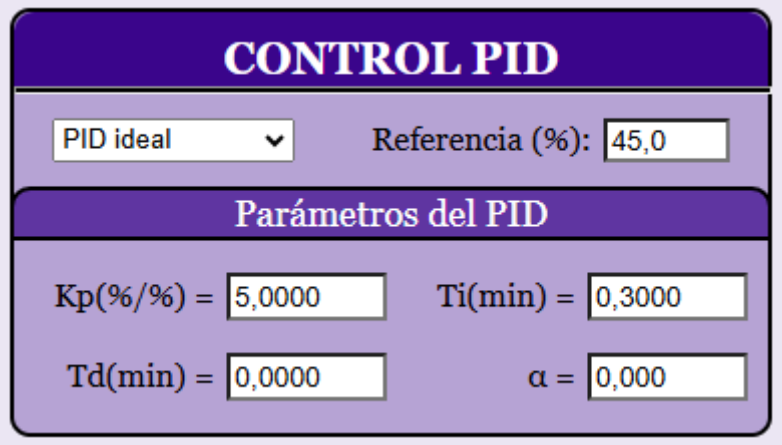


Figura 88. Aspecto del Panel controlador PID

4.3.2.3. Panel visualización

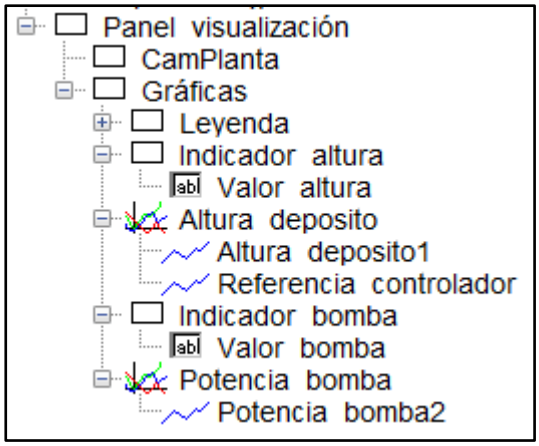


Figura 89. Contenido del panel visualización

Este panel contiene la interfaz gráfica del laboratorio remoto y está dividida en dos partes, por un lado, encontramos el panel *CamPlanta* donde se puede visualizar la planta real del laboratorio y, por otro lado, el panel *Gráficas* donde se dibujan la evolución temporal de la altura del tanque de salida y de la potencia de la bomba, además de su valor en cada instante.

Al parámetro *Display* de *Panel visualización* se le ha asignado el tipo *flex* y el CSS siguiente:

```
"margin-top":"8px",
"margin-bottom":"8px",
"justify-content:":"space-between",
"align-items":"left",
"align-content":"left"
```

Figura 90. Código CSS del Panel visualización

El panel *CamPlanta* tiene las siguientes propiedades:

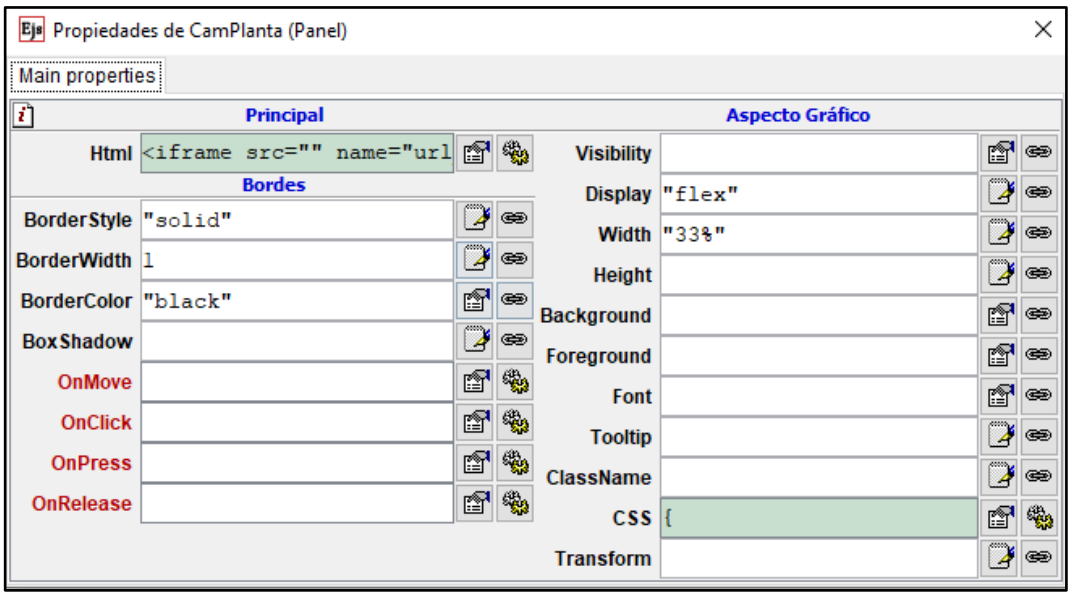


Figura 91. Propiedades del panel CamPlanta

Para incrustar un video en una página web se utiliza la etiqueta *iframe* de HTML con la dirección IP de la cámara de la siguiente manera:

```
<iframe src="" name="url_cam" title="Camara planta" id="urlCam"
style="width:100%"></iframe>
```

La IP con la imagen de la cámara debe ir en el campo *src*, en este caso no se ha introducido nada, ya que al igual que la dirección del servidor RIP, una vez empaquetado el cliente en EJSS, los parámetros permanecen estáticos, es decir, no se puede modificar. Por esta razón se modificará el código generado para que las direcciones se puedan cambiar desde el botón *Configuración.* Estos cambios son comentados en el apartado 4.3.3.

Por otro lado, el panel Gráficas tiene el parámetro *Display* de tipo *flex*, un ancho del 67%, es decir, dos terceras partes de la pantalla (el otro tercio se ha reservado para *CamPlanta*) y el código CSS mostrado a continuación:

```
"flex-direction":"column",
"justify-content:":"center",
"align-items":"center",
"align-content":"center",
"flex-shrink":"8",
"flex-grow":"8",
"margin-right":"8px",
"padding":"0.5rem"
```

Figura 92. Código CSS del panel Gráficas

La primera gráfica incluye una leyenda que muestra tanto la altura del tanque de salida como la referencia. Esto se encuentra en el panel *Leyenda*, el cual tiene solo las siguientes propiedades en el campo *Aspecto Gráfico*:

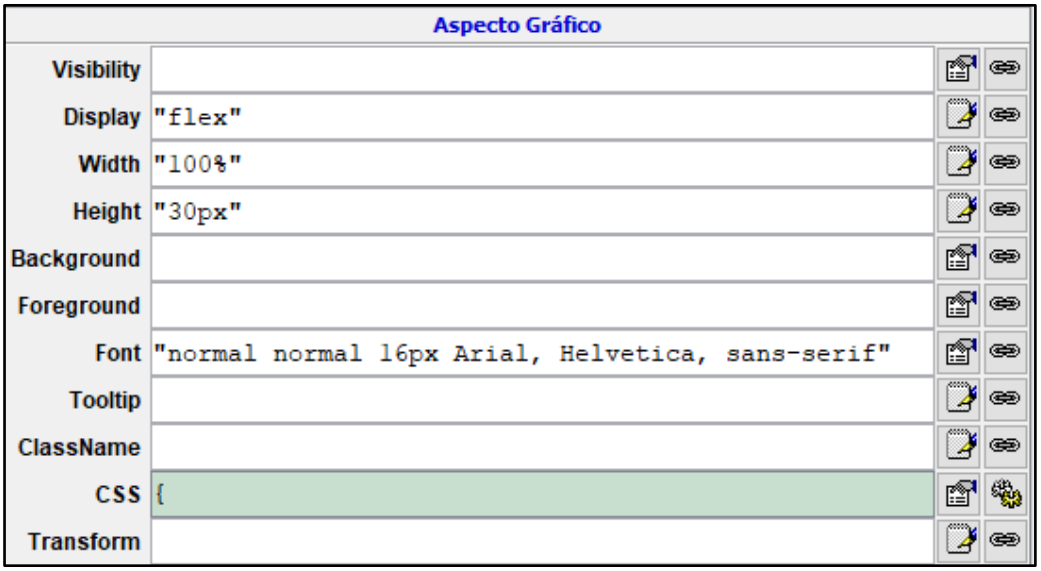


Figura 93. Parámetros del panel Leyenda

En la *Figura 94*, se presenta el CSS correspondiente al panel *Leyenda*.

```
{
  "float":"right",
  "flex-direction":"row",
  "z-index":"3",
  "padding-bottom":"50px",
  "margin-left" : "100%",
  "padding-right" : "-800px"
 }
```

Figura 94. Código CSS del panel Leyenda

Las etiquetas de altura y referencia de la leyenda tienen las siguientes propiedades. Nótese en la *Figura 97* que la referencia solamente es visible cuando la variable *control* tenga un valor distinto de 1, es decir, cuando el tipo de control seleccionado por el maestro sea automático.

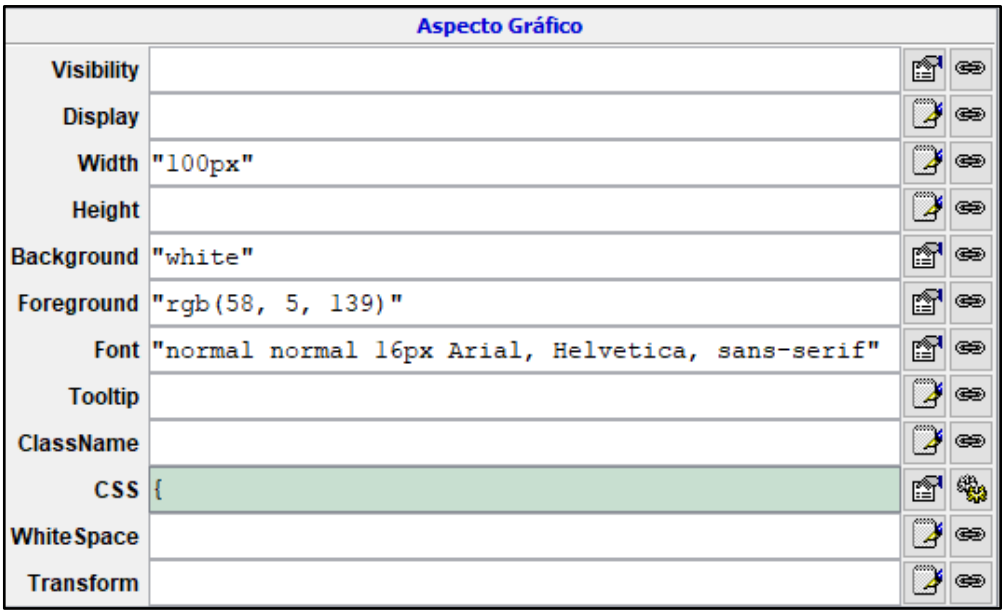


Figura 95. Propiedades de la etiqueta altura

```
"border": "2px solid black",
"margin-left" : "-98px"
```

```
"border": "2px solid black",
"text-align" : "rigth",
"margin-left" : "-198px"
```

Figura 96. Código CSS de las etiquetas altura(izq.) y referencia(dcha.)

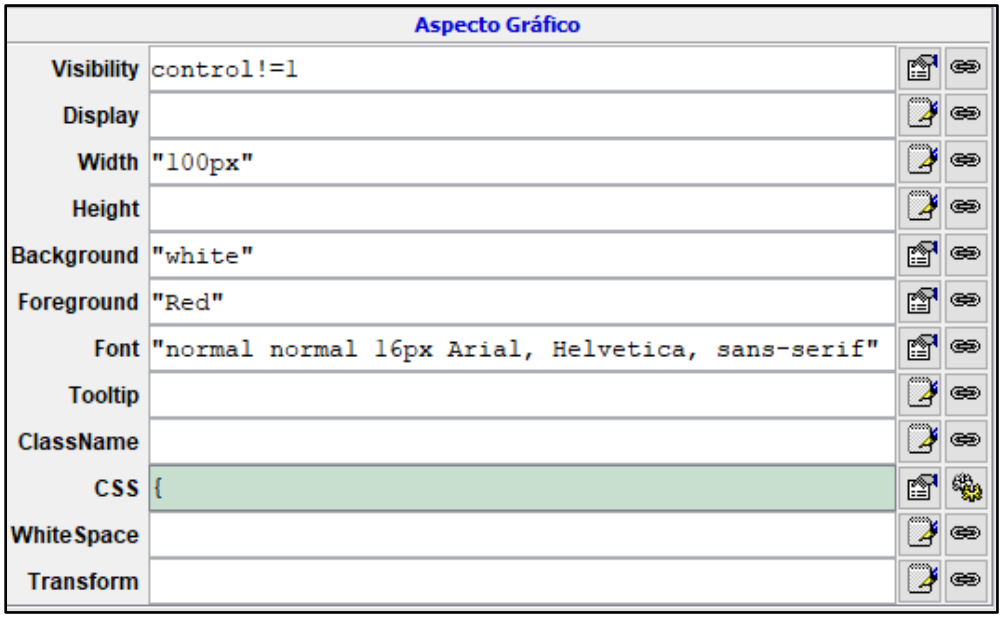


Figura 97. Propiedades de la etiqueta referencia

Los paneles Indicador altura e Indicador bomba contienen las siguientes propiedades y CSS:

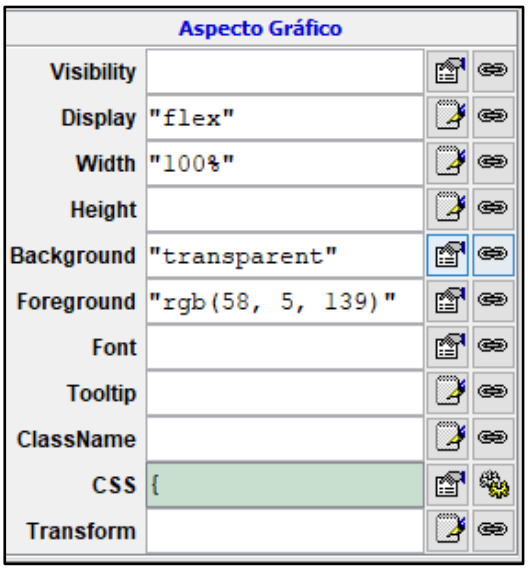


Figura 98. Propiedades de los paneles Indicación altura e Indicación bomba

```
"flex-direction" : "row",
"text-align": "right",
"margin-bottom" : "-83px",
"z-index" : "2",
"margin-left" : "100%"
```

```
"flex-direction" : "row",
"text-align": "right",
"z-index" : "2",
"margin-left" : "100%",
"margin-top" : "60px"
```

Figura 99. Código CSS de los paneles Indicación altura(izq.) e Indicación bomba(dcha.)

A continuación, se pueden ver las propiedades y CSS de los campos numéricos con los valores instantáneos de la altura del tanque de salida y de la bomba.

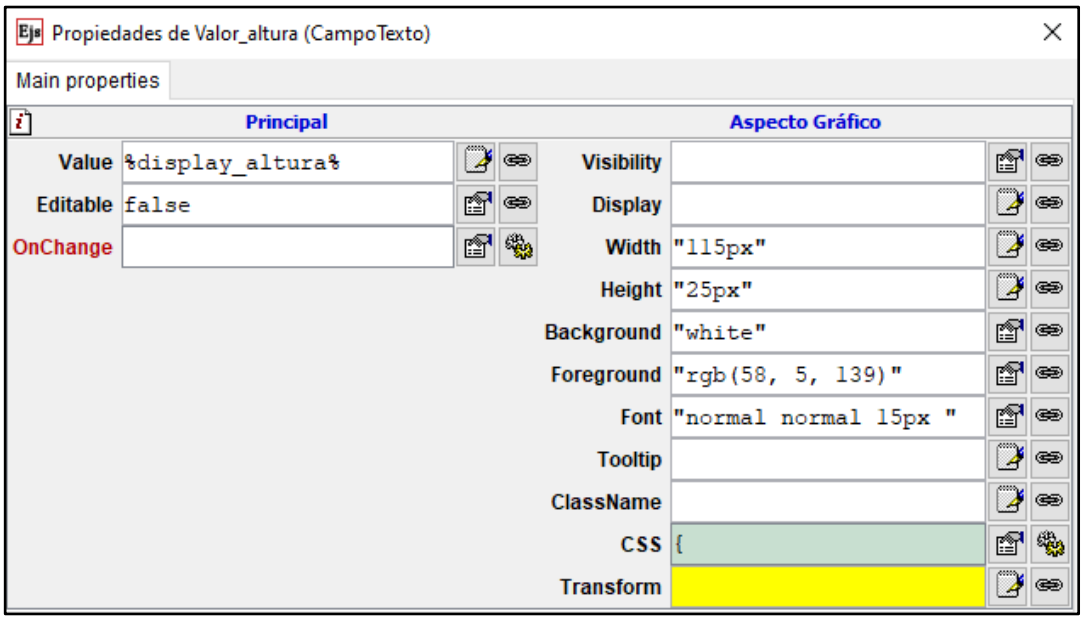


Figura 100. Propiedades de los campos valor altura y valor bomba

```
"border" : "2px solid black",
"margin-left" : "-170px"
```

```
"border" : "2px solid black",
"margin-left" : "-180px",
"z-index" : "2"
```

Figura 101. Código CSS de los campos valor altura(izq.) y valor bomba(dcha.)

Por último, faltan por conocer las propiedades de las gráficas *Altura deposito* y *Potencia bomba.* Del gran número de parámetros que se pueden configurar, en este caso solamente se ha dado nombre a los ejes, un título a las gráficas y definido los valores máximo y mínimo tanto del eje temporal como del eje y de ambas gráficas.

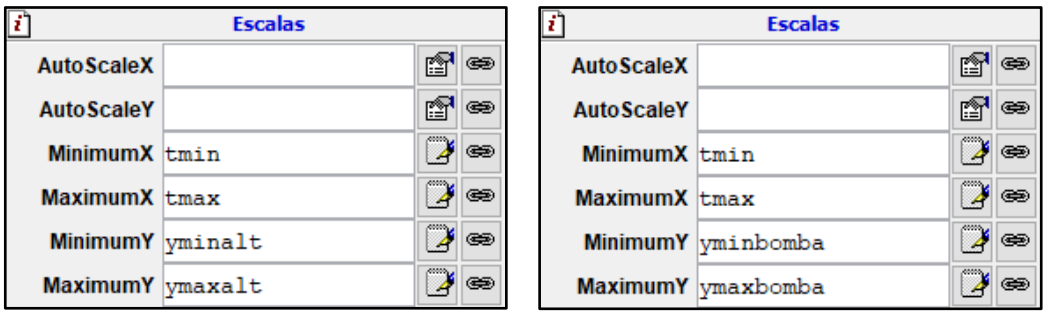


Figura 102. Ajuste de los ejes de las gráficas altura deposito(izq.) y potencia bomba(dcha.)

Por último, para configurar las trazas de las gráficas se ha añadido en el parámetro *InputX* el valor *t-tDelay* y en el parámetro *InputY* la variable que contiene el valor a representar en cada traza, estas variables son *h1*, *refx* y *u* para las trazas *Altura deposito1, referencia controlador y Potencia bomba*, respectivamente. Además, para la traza *referencia controlador* se ha utilizado el color rojo y para las otras dos el color *rgb(58, 5, 139).* El parámetro *ClearAtInput* de las trazas se ha conectado con la variable *limpiar*.

El resultado final del *Panel visualización* en la interfaz del cliente es el siguiente:

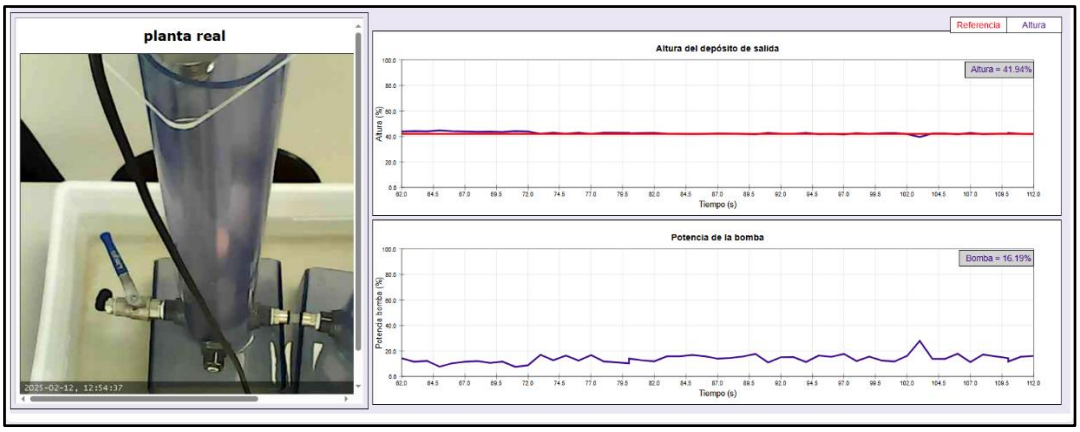


Figura 103. Aspecto del panel visualización

4.3.3. Empaquetado y cambios adicionales

En este apartado se comentará cómo se ha empaquetado el cliente desde EJSS, los cambios realizados y las ventanas adicionales realizadas.

4.3.3.1. Empaquetado del cliente

Una vez se ha realizado el cliente de la manera descrita en el apartado 4.3.1 y 4.3.2, se realiza el empaquetado del cliente que nos genera los ficheros necesarios para ejecutar el cliente desde el navegador.

Al hacer clic en el botón de empaquetar aparecerá una ventana como la siguiente donde podemos escoger dónde guardar la carpeta comprimida generada.

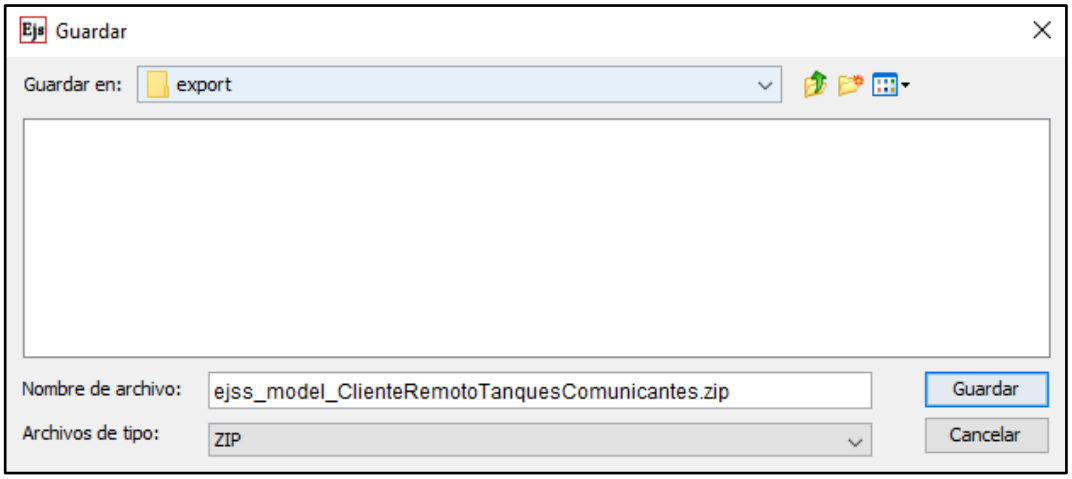


Figura 104. Guardado al empaquetar el cliente EJSS

En este caso hemos modificado el nombre de la carpeta a *ClienteRemotoTanquesComunicantes*. Una vez guardada, descomprimimos la carpeta y vemos que se han generado los siguientes archivos:

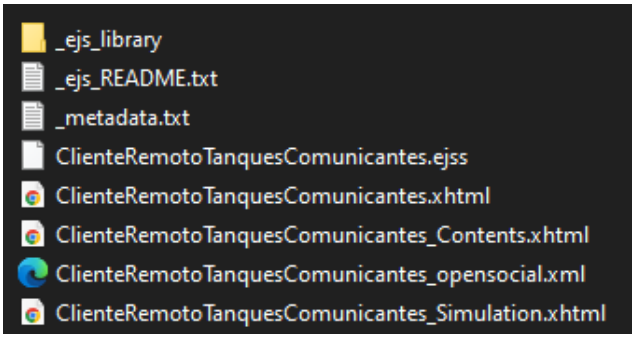


Figura 105. Archivos generados al empaquetar el cliente

De todos los archivos con extensión *xhtml* generados, el archivo relevante es aquel que tiene el sufijo *_Simulation*, ya que contiene el código correspondiente a la interfaz del cliente. Por eso, el resto de los archivos *xhtml* se eliminarán y se suprimirá el sufijo *_Simulation* del archivo para convertirlo en el archivo principal.

Por otro lado, tenemos la carpeta *_ejs_library* que contiene archivos adicionales con código CSS o imágenes, el fichero *_ejs_README* que contiene una breve descripción con el cometido de los archivos generados en el empaquetado y el fichero *_metadata* que contiene información como el autor o cual es el archivo principal.

Tanto *_ejs_README* como *_metadata* se han eliminado ya que, el primero no contiene ninguna información de interés una vez eliminados todos los archivos *xhtml* menos el principal y el segundo se sustituirá por la ventana Créditos.

Por último, el empaquetado ha guardado el fichero comprimido del entorno EJSS, este archivo contiene toda la información recogida en los apartados 4.3.1 y 4.3.2, y es de utilidad si se quisiera realizar futuros cambios.

El contenido de la carpeta una vez hechas estas modificaciones es el siguiente:

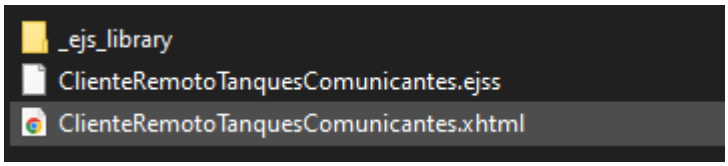


Figura 106. Contenido del cliente tras eliminar los archivos irrelevantes

4.3.3.2. Cambios adicionales

Además del contenido de la *Figura 106*, se han añadido los archivos con la descripción del cliente y los créditos del proyecto que se abrirán mediante el botón *Descripción* y el icono de información de la interfaz, respectivamente. Adicionalmente, se han creado las carpetas *fotos* y *css*; la primera contiene las imágenes utilizadas en el cliente y la segunda los estilos *CSS*.

Aparte de esto, ha sido necesario el desarrollo de un método para poder modificar las direcciones del servidor RIP y de la cámara del laboratorio, como se adelantó en apartados anteriores.

Esto se debe a que, una vez empaquetado el cliente, se guarda la última dirección introducida en el elemento RIP de EJSS y no se puede modificar desde la interfaz. Cada vez que quisiéramos conectarnos a un servidor con otra dirección tendríamos que modificarlo desde EJSS y volver a empaquetar el cliente o hacer uso de un editor de código y modificar el fichero principal. Esto mismo ocurriría si quisiéramos modificar la dirección de la cámara del laboratorio.

Para solucionar este problema se ha optado por utilizar el botón de configuración para abrir una ventana donde se puedan modificar las direcciones.

El funcionamiento es muy simple, se pulsa el botón configuración y se abre la ventana siguiente donde se introducen las direcciones:

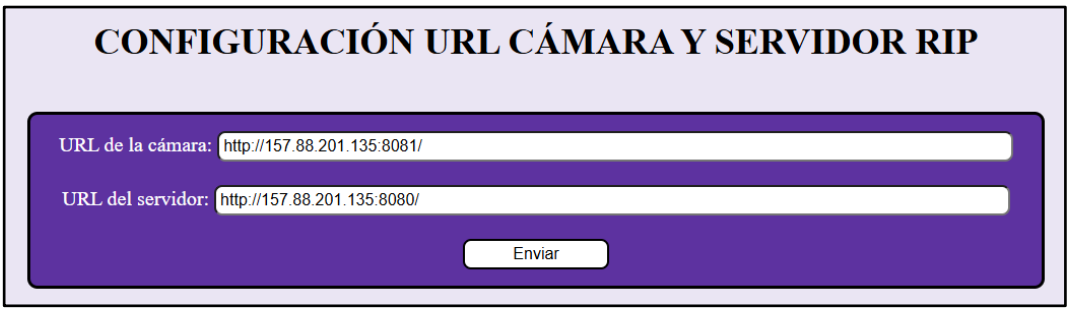


Figura 107. Panel de configuración de las URLs de la cámara y del servidor RIP

Una vez se introducen las direcciones se pulsa el botón *Enviar* que nos llevará de nuevo a la interfaz. Para poder mover los valores de las direcciones de una a otra ventana se utiliza el almacén local del navegador(localStorage) y una serie de funciones para escribir y leer variables en este. Además, se utilizará un método paralelo ya que algunos navegadores no funcionan correctamente con *localStorage*, este consiste en añadir las variables con los valores de las direcciones a continuación de la URL de la interfaz del cliente en el buscador.

Una vez abierta de nuevo la interfaz del cliente, se reciben los valores para introducirlos en el elemento RIP y en el panel de la cámara. Para eso, con ayuda del editor de código *Visual Studio Code,* se modifica el código del archivo *ClienteRemotoTanquesComunicantes.xhtml,* de la siguiente manera:

1. De la línea 30 a la 45 se añade el siguiente código que permite recibir los valores introducidos en la ventana de configuración. Estos valores son guardados en las variables *urlserv* y *urlcamara*.

```
30  var urlServDef = "http://157.88.201.135:8080/";
31  var urlCamDef = "http://157.88.201.135:8081/";
32
33  function getQueryParams() {
34    const params = {};
35    const queryString = window.location.search.substring(1);
36    const regex = /([^&=]+)=([^&]*)/g;
37    let m;
38    while (m = regex.exec(queryString)) {
39      params[decodeURIComponent(m[1])] = decodeURIComponent(m[2]);
40    }
41    return params;
42  }
43    const params = getQueryParams();
44    const urlserv = params.url_rip || urlServDef;
45    const urlcamara = params.url_cam || urlCamDef;
```

Figura 108. Código añadido para recibir las direcciones en la interfaz del cliente

2. Se intercambia la variable *urlserv* por la dirección utilizada por el elemento RIP.

Inicial:

```
1144    var __conf = {'host':'http://157.88.201.135:8080/','port':'8080'};
```

Modificado:

```
1144    var __conf = {'host':urlserv,'port':'8080'};
```

Figura 109. Modificación del elemento RIP para recibir la dirección fijada en la configuración

3. En el panel *CamPlanta* se introduce la variable *urlcamara*.

Inicial:

```
2349            .setProperty("Html","<iframe src=\"\" name=\"url_cam\
```

Modificado:

```
2349            .setProperty("Html","<iframe src=\"" + urlcamara + "\" name=\"
```

Figura 110. Modificación de CamPlanta para recibir la dirección fijada en la configuración

Con todos estos cambios se le permite al usuario modificar fácilmente las direcciones desde la interfaz sin necesidad de modificar el código del laboratorio.

El código que genera EJSS añade unos créditos en la parte inferior de la interfaz, pero al haber realizado una ventana propia, carece de sentido su utilización. Para eliminar los créditos se suprime el siguiente código del archivo principal:

```
2487    <div id="metadata" class="metadata">
2488      <br />
2489      <div id="title_author">
2490        <hr />
2491        <b>Title and author:</b>
2492        <p>
2493        Tanques Comunicantes Remoto<br />
2494        </p>
2495        <p>
2496    Gabriel Alonso Manrique
2497        </p>
2498      </div>
2499      <hr />
2500      <p></p>
2501      <div id="copyright_message">
2502        <div class="cc_left"  style="float:left">&#169; 2025, Gabriel Alonso Manrique.</div>
2503        <div class="cc_right" style="float:right">    Released under a <a rel="license" targ
2504      </div>
2505    </div>
```

Figura 111. Código de los créditos automáticos de EJSS

Nótese que para los argumentos *id* y *class* se utiliza *metadata* y es uno de los archivos de texto que hemos eliminado al realizar una ventana de créditos personalizada.

Una vez hecho esto, el cliente queda terminado y comprimido en la carpeta y el contenido de la carpeta *ClienteRemotoTanquesComunicantes* es el siguiente:

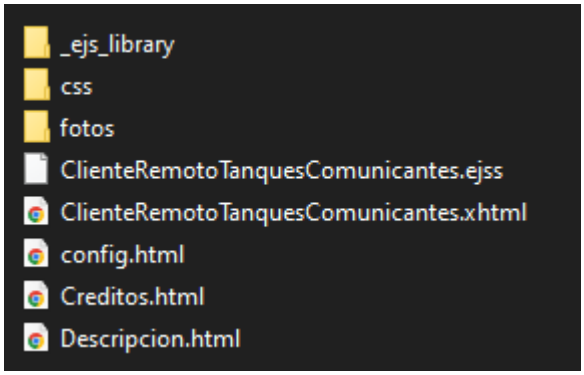


Figura 112. Contenido final de la carpeta del cliente

CAPÍTULO 5: Validación del laboratorio remoto

En este apartado se comprobará si el laboratorio remoto cumple con lo esperado para poder utilizarse como alternativa al trabajo presencial. Para validar el laboratorio remoto basta con obtener un comportamiento esperado en cada uno de los modos de control implementados, por esta razón este apartado se divide en tres partes: la comprobación del modo manual, la comprobación del PID al estilo JavaRegula donde se realizará el mismo experimento con el software JavaRegula y con el software realizado en este proyecto y, por último, la validación del PID ideal.

En los siguientes apartados, se comentan con más detalle los experimentos realizados para cada una de las validaciones.

5.1. Validación del control manual

Para este modo de control, se hicieron cambios en la potencia de la bomba y se observó si la evolución era correcta. Se comenzó con una potencia del 20% y, una vez alcanzado el estado estacionario, se aumentó al 25%. Los datos fueron registrados y guardados mediante las opciones de la interfaz.

Para realizar un análisis adecuado, se utiliza Excel. Para cargar el archivo del experimento correctamente, desde la pestaña *Datos*, se accede a *Obtener datos*, luego a *De un archivo* y, finalmente, a *De texto/CSV* para cargar el archivo de datos del laboratorio remoto.

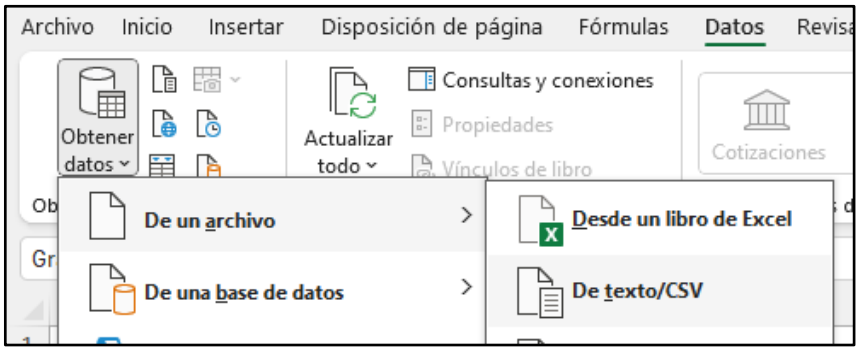


Figura 113. Cargar datos del laboratorio remoto en Excel

Una vez seleccionado el archivo aparece una previsualización, para cargarlos correctamente es necesario seleccionar la opción *–Personalizado–* como delimitador.

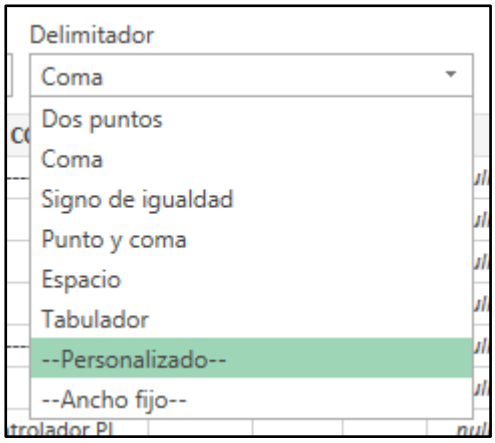


Figura 114. Selección del delimitador en la carga de datos de Excel

De esta forma, hemos importado los datos correctamente a un Excel y ahora podemos realizar un mejor análisis de los datos. El resultado del experimento es el siguiente:

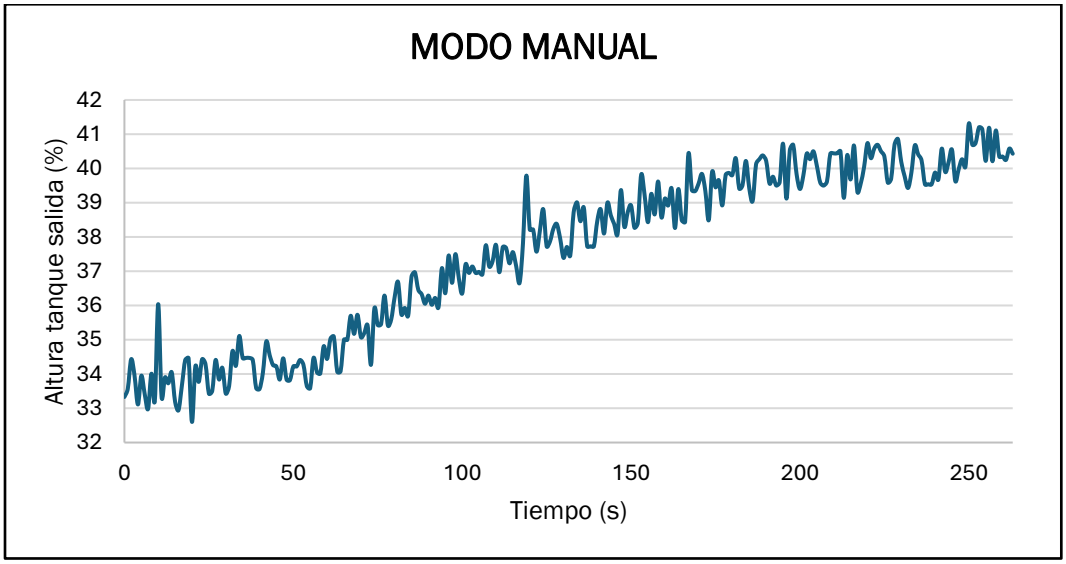


Figura 115. Evolución del sistema al cambiar la bomba del 25 al 28% en modo manual

Este experimento nos permite obtener un modelo aproximado del sistema para calcular una posible sintonía del PID. Para ello, necesitamos obtener una serie de valores sobre la gráfica de la *Figura 115*.

En primer lugar, obtendremos los valores de la constante de tiempo (τ) y el retardo (d) a partir de los tiempos para los valores al 28,3% (t1) y 63,2% (t2) de alcanzar el valor estacionario. Los valores obtenidos son los siguientes:

$$\Delta h = h_f - h_i = 40\% - 34\% = 6\% \tag{5.1}$$

$$h(28{,}3\%) = 6 * 0{,}283 + 34 = 35{,}7\% \rightarrow t1 = 28\text{seg} \tag{5.2}$$

$$h(63{,}2\%) = 6 * 0{,}632 + 34 = 37{,}8\% \rightarrow t2 = 64\text{seg} \tag{5.3}$$

$$\tau(\text{min}) = 1{,}5 \cdot (t2 - t1)/60 = 0{,}025 \cdot (64 - 28) = 0{,}9\text{min} \tag{5.4}$$

$$d(\text{min}) = t2 \ - \ \tau = 64/60 - 0{,}9 = \frac{1}{6}\text{min} \approx 0{,}167\text{min} \tag{5.5}$$

Por último, obtenemos la ganancia *K* de la siguiente manera:

$$K = \frac{\Delta h}{\Delta u} = \frac{6\%}{3\%} = 2\frac{\%}{\%} \tag{5.6}$$

Para obtener la sintonía del PID se han utilizado los valores de Ziegler-Nichols recogidos en la siguiente tabla:

Tipo	Ganancia Kp	Tiempo integral	Tiempo derivativo
P	τ / (K d)		
PI	0.9 τ / (K d)	3.33 d	
PID serie	1.2 τ / (K d)	2 d	0.5 d

Tabla 8. Sintonía de Zieger-Nichols en lazo abierto

Introduciendo los valores obtenidos en las ecuaciones *(5.4), (5.5) y (5.6)* en la tabla anterior podremos obtener una sintonía para el modo automático.

5.2. Validación del PID estilo JavaRegula

En este apartado se llevará a cabo el mismo experimento tanto en el software JavaRegula como en el laboratorio remoto para verificar el funcionamiento de este último.

Para ello, es necesario obtener la sintonía que utilizaremos haciendo uso de la *Tabla 8* y las ecuaciones *(5.4), (5.5) y (5.6).* Se ha optado por un controlador de tipo PI y los valores obtenidos son los siguientes:

$$\mathrm{Kp} = \frac{0{,}9 \cdot \tau}{\mathrm{K} \cdot \mathrm{d}} = \frac{0{,}9 \cdot 0{,}9\mathrm{min}}{2\frac{\%}{\%} \cdot 0{,}167\mathrm{min}} = 2{,}43\frac{\%}{\%} \tag{5.7}$$

$$\mathrm{Ti} = 3{,}33 \cdot \mathrm{d} = 3{,}33 \cdot 0{,}167 = 0{,}555\mathrm{min} \tag{5.8}$$

Una vez obtenidos los valores del controlador, es importante iniciar desde el mismo estado en ambos casos: un nivel del tanque del 40%. Desde ese punto se realizará un cambio en la referencia al 45% y una vez se alcance ese valor se volverá al 40%. En ambos experimentos se ha mantenido la referencia al 45% durante el mismo tiempo para una mejor comparación de los resultados.

Se registran y se introducen los datos de los dos experimentos a un Excel de la misma manera que hicimos con los resultados del modo manual y obtenemos los resultados mostrados en la *Figura 116* y la *Figura 117*.

En la *Figura 116* se observa la evolución del porcentaje de la bomba en ambos casos. Se puede ver cómo el PID del software JavaRegula ha calculado esfuerzos en torno a un 3% más bajos que el PID del laboratorio remoto, lo que podemos considerar como un buen funcionamiento por parte del laboratorio remoto.

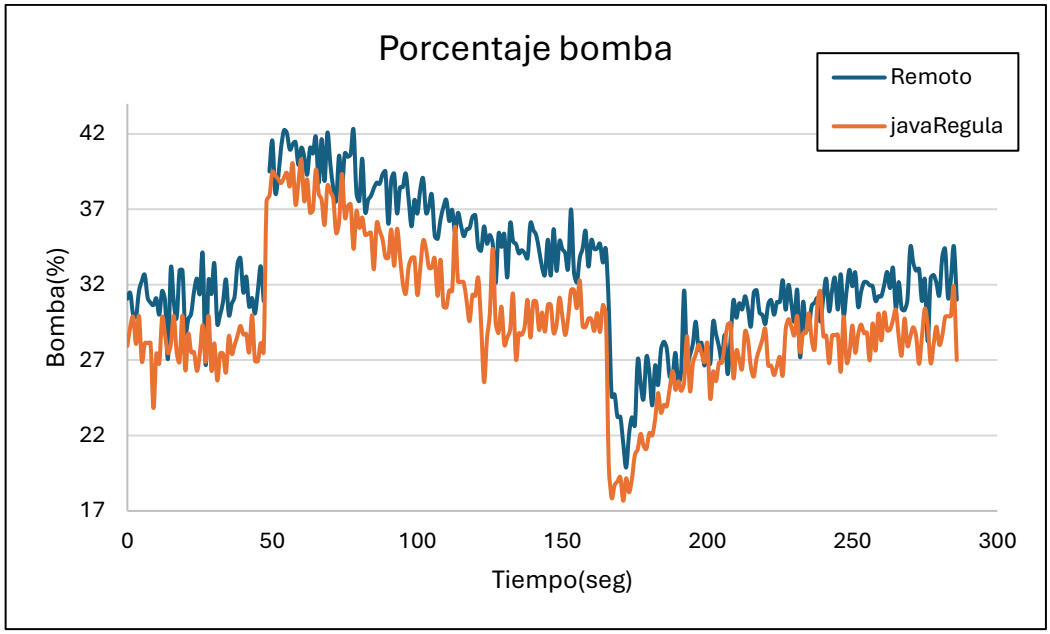


Figura 116. Comparación del porcentaje de la bomba en ambos experimentos JavaRegula

Por otro lado, en la *Figura 117* se recoge la evolución del nivel en el tanque de salida del sistema. Como se dijo anteriormente, se ha mantenido durante el mismo tiempo la referencia al 45%, lo que nos permite comparar directamente en la misma gráfica los resultados. Se observa cómo ambos resultados están prácticamente superpuestos; esto nos indica que el comportamiento del control del laboratorio remoto es correcto en comparación con el de JavaRegula.

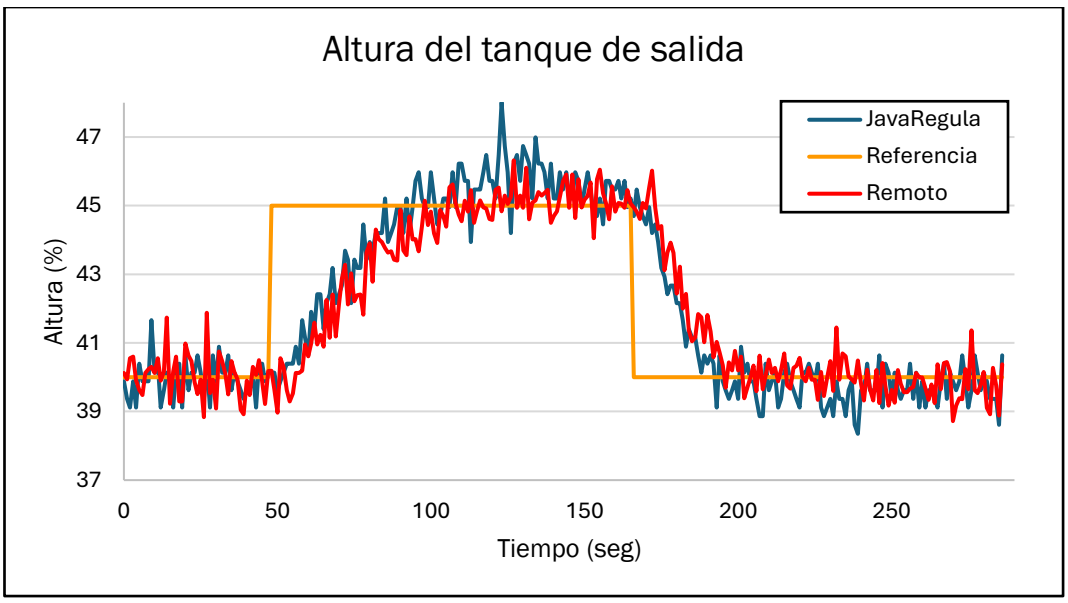


Figura 117. Comparación de la altura del tanque de salida en ambos experimentos JavaRegula

5.3. Validación del PID ideal

En este apartado se ha realizado el mismo experimento que en el caso anterior para el PID ideal con el objetivo de verificar el comportamiento de este.

Partiendo de un nivel de referencia del 40% se introducen los valores de la sintonía para el PI calculados en las ecuaciones (5.7) y (5.8) y se modifica el valor de referencia al 45%, una vez se alcance este valor se vuelve a reducir la referencia al 40%. El resultado del experimento es el siguiente:

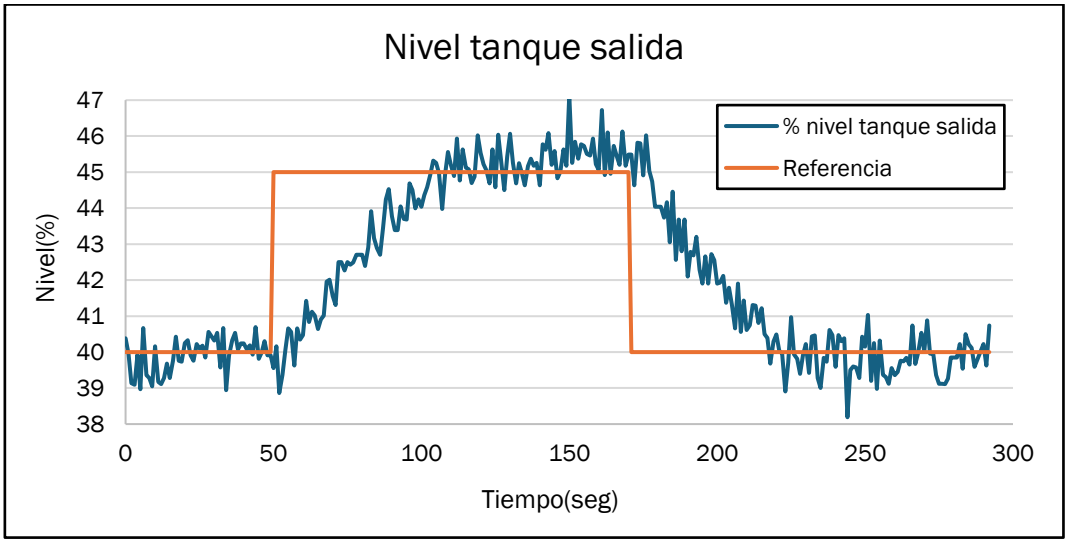


Figura 118. Evolución temporal del nivel del tanque en el experimento con PID ideal

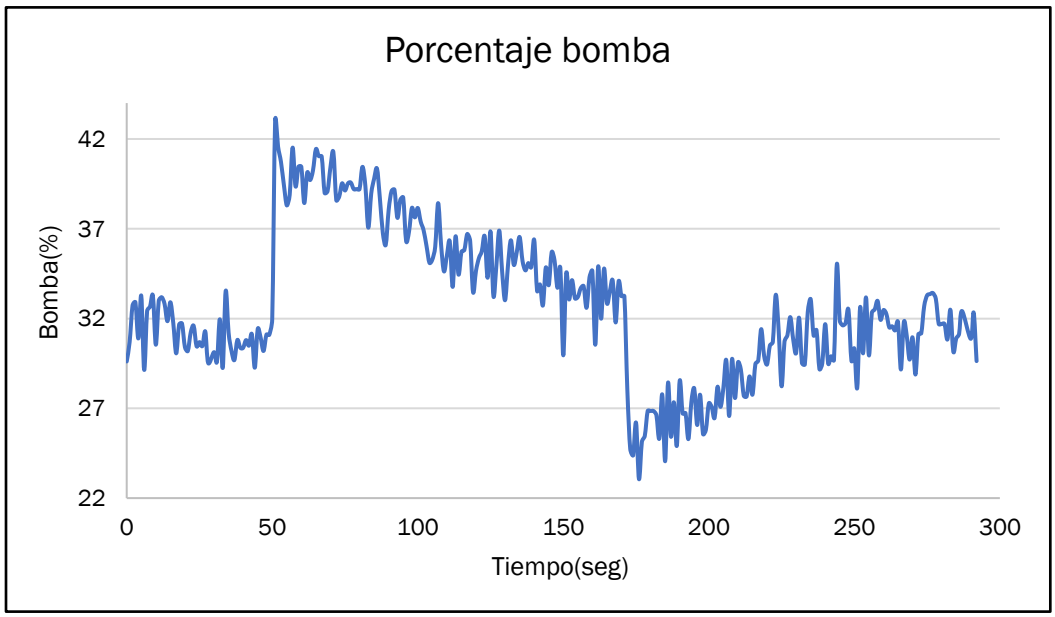


Figura 119. Evolución temporal del porcentaje de la bomba en el experimento con PID ideal

En los resultados del experimento se observa una buena respuesta ante el cambio de referencia y unos esfuerzos en la bomba similares a los experimentos realizados en el apartado anterior. Con esto se puede concluir que el PID desarrollado funciona correctamente.

5.4. Conclusiones

Con los experimentos llevados a cabo queda validado el laboratorio remoto debido a que se comporta correctamente en cada uno de los modos de control implementados y, además, se ha comprobado que el comportamiento del modo JavaRegula es similar al del software homólogo utilizado de manera presencial en el laboratorio.

CAPÍTULO 6: Instalación y puesta en marcha del servidor

6.1. Software necesario

Para el funcionamiento del servidor es necesario Python y Matlab. Además, para mostrar la imagen de la cámara en el cliente, se ha optado por utilizar el software Yawcam. En este apartado se indican los pasos para la instalación de cada uno de ellos.

6.1.1. Instalación de Matlab

En los ordenadores del laboratorio Matlab suele estar instalado, pero si no fuese así, tendríamos que ir al enlace de descarga de Matlab (The Mathworks - Downloads, s.f.) y habría que iniciar sesión con la cuenta de correo de la universidad. Una vez hecho esto, podemos seleccionar la versión de Matlab que queremos y seleccionar *Descargar*. Durante la descarga habrá que seleccionar la opción licencia individual y los productos que queramos.

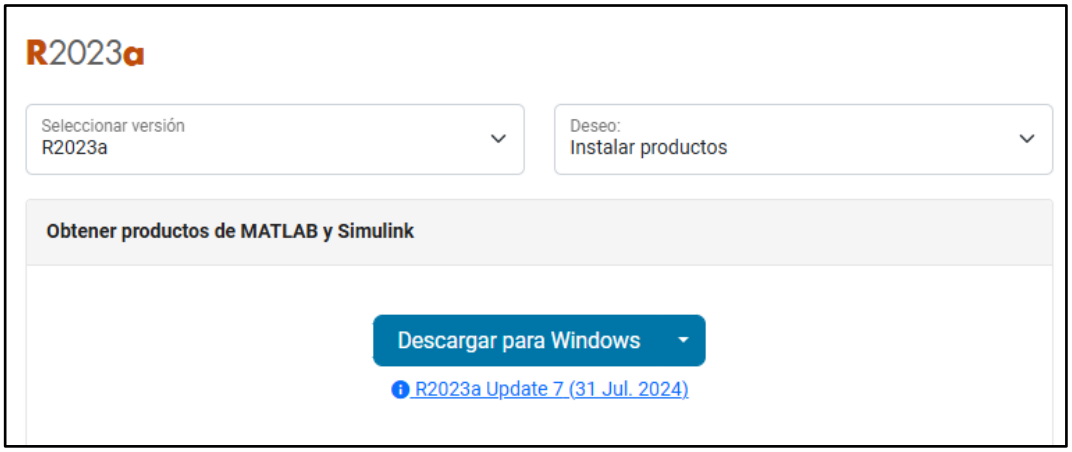


Figura 120. Descarga de Matlab para Windows

6.1.2. Instalación de Python

Para la instalación de Python nos dirigimos a su página oficial (Python, s.f.) y seleccionamos en la pestaña downloads la opción para el sistema operativo que estemos trabajando. En este caso se ha seleccionado Windows (Python Releases for Windows, s.f.), ya que los ordenadores del laboratorio utilizan este S.O.

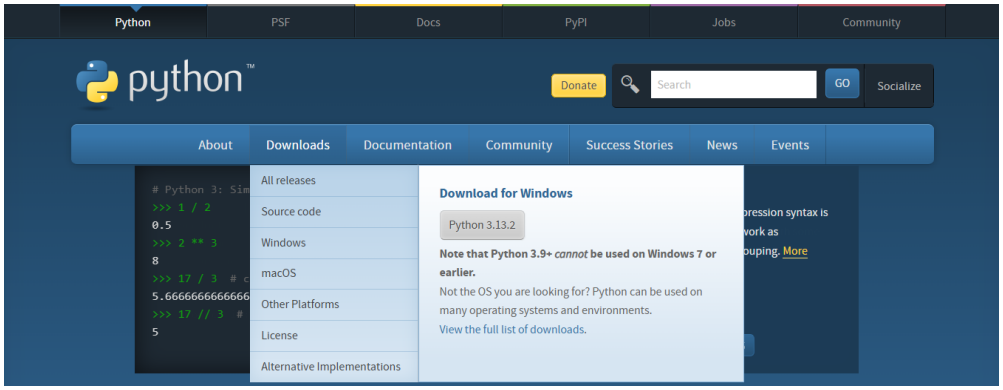


Figura 121. Apartado de descarga de Python en su página oficial

Se escoge la versión que queremos; en este caso se ha optado por la 3.10.11 ya que la API de Matlab para Python (matlabengine) es compatible con Python 3.10 en todas las versiones recientes de Matlab.

Una vez descargado el instalador, se abre, y es importante marcar la opción *Add python.exe to PATH*; si esto no se realiza se tendrá que añadir la variable de entorno a mano. Una vez seleccionada esta opción, como se muestra en la Figura 122, clicamos en la opción *Customize installation*.

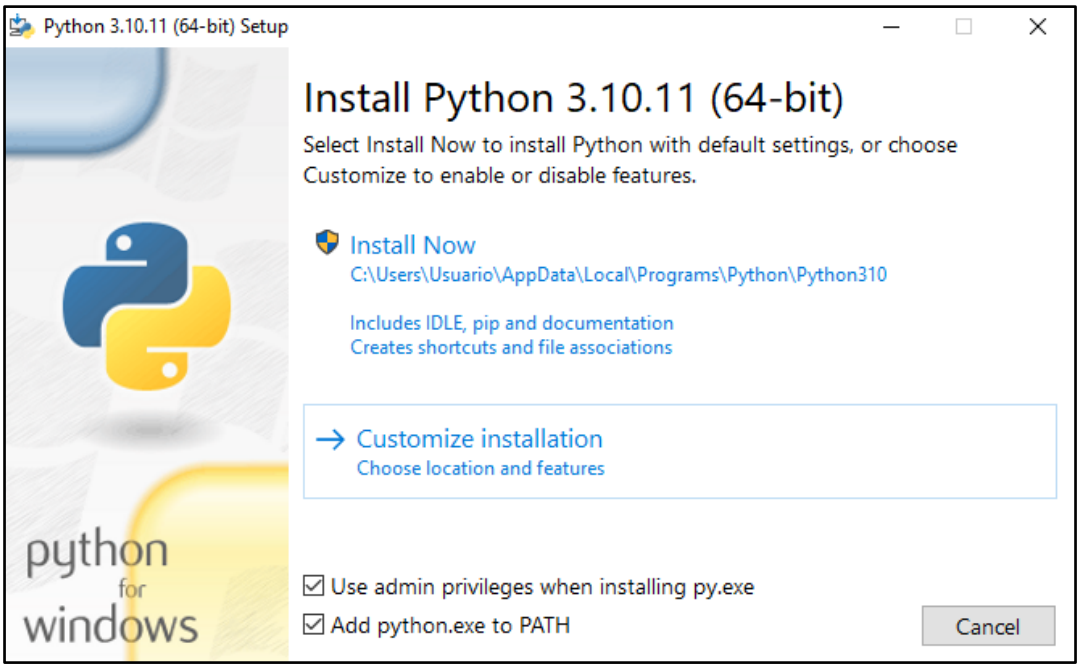


Figura 122. Ventana inicial en la instalación de Python

Esta opción de instalación nos permite seleccionar las características que queremos en la instalación de Python y escoger la ruta donde se ubicará el programa. En esta ventana (Figura 123) se marcan todas las opciones para evitar cualquier tipo de problema posterior y se clica en Next.

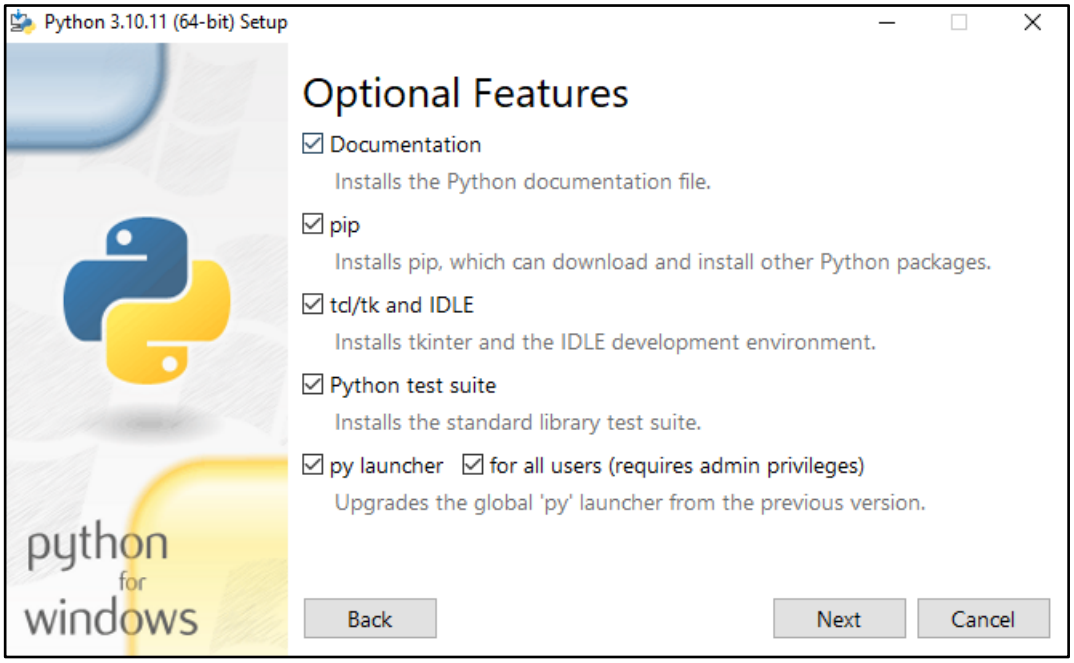


Figura 123. Características opcionales en la instalación de Python

En esta última ventana (Figura 124) es importante comprobar que la opción *Add Python to enviroment* variables esté marcada; si no fuera así, la marcamos. También se ha marcado la opción *Install Python 3.10 for all users* para que se pueda arrancar el servidor desde cualquier usuario.

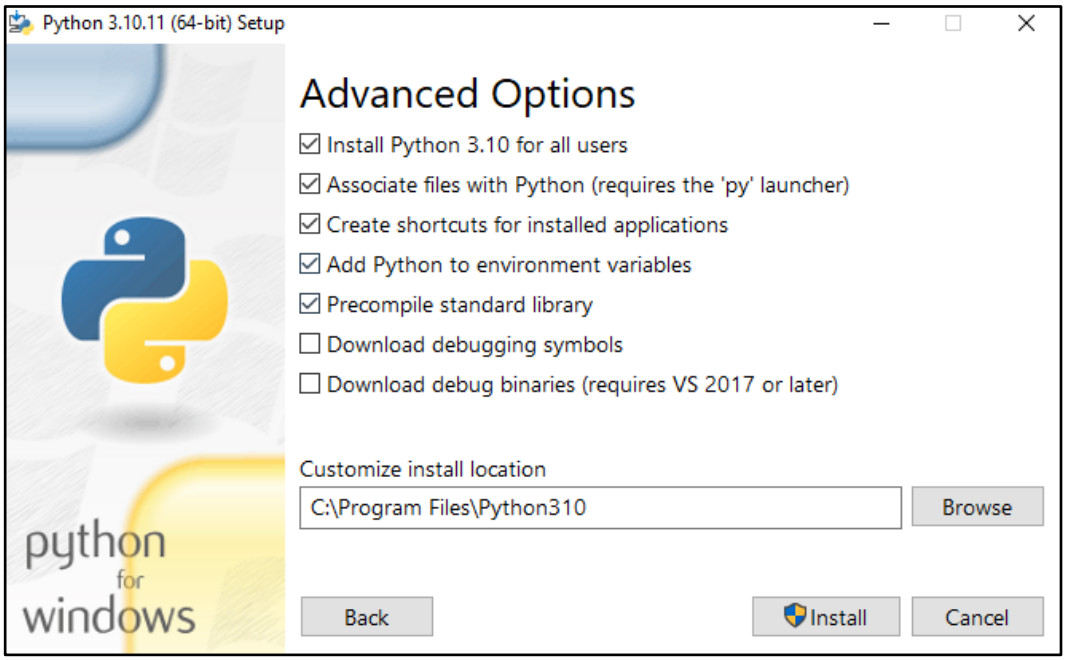


Figura 124. Opciones avanzadas en la instalación de Python

Una vez todas las casillas estén marcadas como en la figura superior, clicamos en *Install* y comienza la instalación. Una vez finalizada la instalación deberíamos obtener lo siguiente:

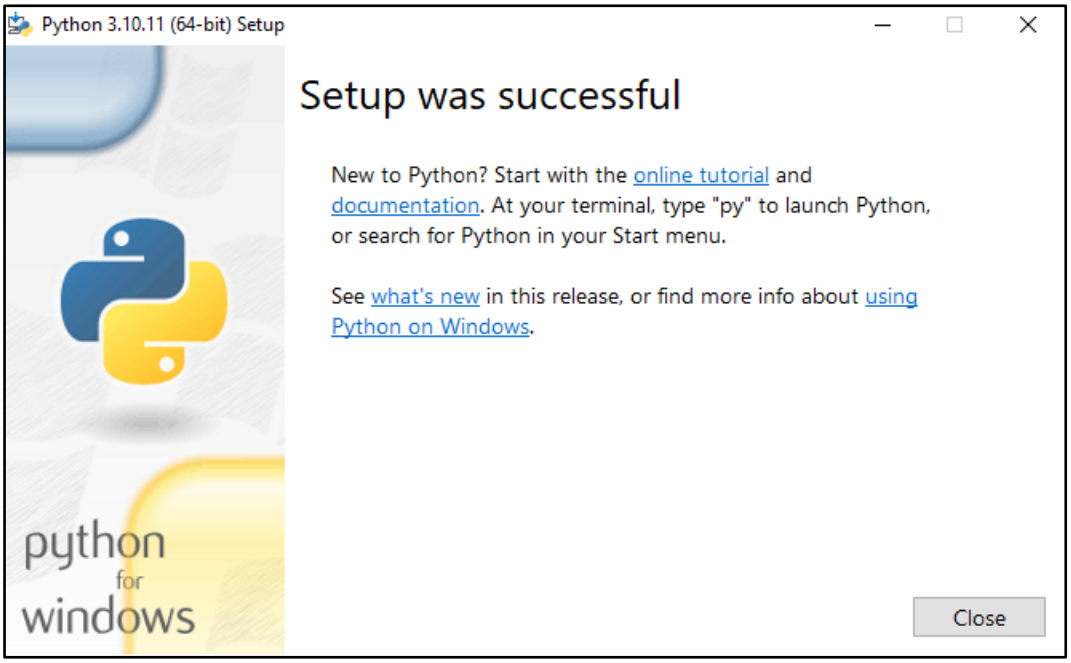


Figura 125. Ventana al terminar la instalación de Python realizada correctamente

Para comprobar si Python se ha instalado de manera correcta, abrimos el Símbolo del sistema y al lanzar el comando *python* debería de entrar en la consola de Python de la siguiente forma:

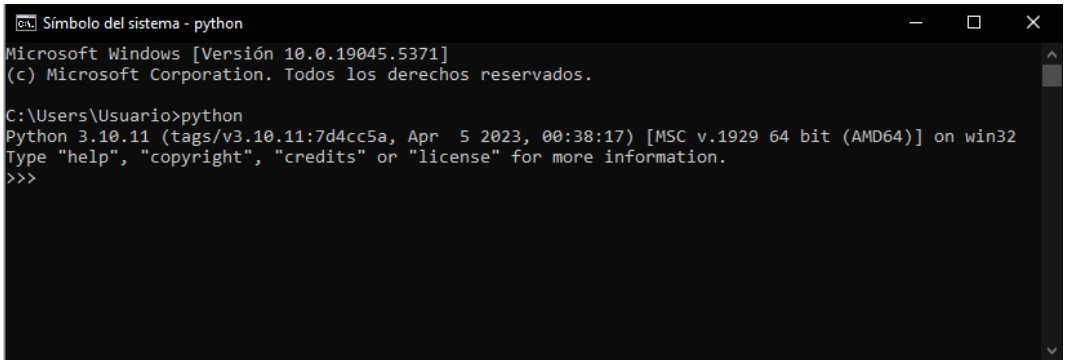


Figura 126. Comprobación de una instalación de Python correcta

Si no se ha conseguido el resultado de la imagen anterior, véase el apartado 6.3.

6.1.3. Creación y preparación del entorno virtual de trabajo

Una vez hecho esto, es necesario crear y preparar el entorno virtual con los módulos necesarios para que el servidor funcione correctamente.

Lo primero es descomprimir la carpeta rip-python-server que contiene todos los ficheros del servidor; esta carpeta se encuentra en los anejos del proyecto. Una vez descomprimida, necesitamos abrir *cmd* (consola de Windows) y situarnos dentro de esta. Para abrir fácilmente la ruta en la consola, la abrimos en el explorador de archivos y clicamos en la barra superior (no sobre el texto), de esta forma se selecciona la ruta (Figura 127).

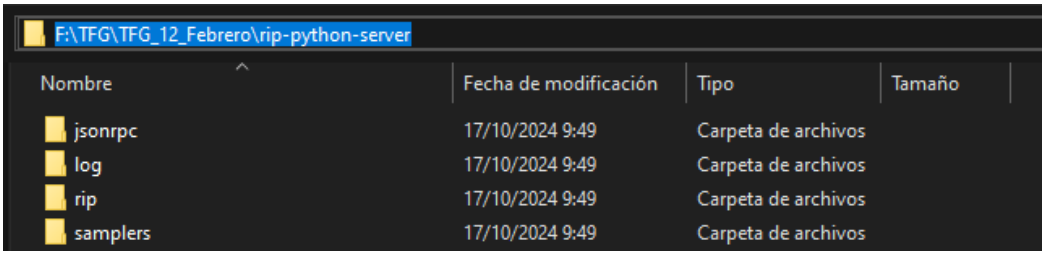


Figura 127. Primer paso para abrir una ruta fácilmente en el Símbolo del sistema

Una vez veamos la ruta marcada en azul, escribimos cmd y pulsamos la tecla Intro, de esta manera se nos abrirá la ruta en el Símbolo del sistema de manera muy sencilla.

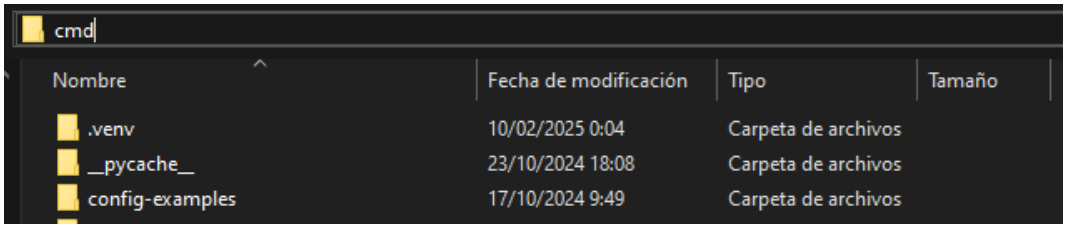


Figura 128. Segundo paso para abrir una ruta fácilmente en el Símbolo del sistema

Para crear el entorno virtual, lanzamos el siguiente comando:

```
python -m venv .venv
```

Esperamos a que termine de crearlo y lo activamos mediante el siguiente comando:

```
.venv\Scripts\activate
```

Si se ha activado correctamente, debe de salir ***.venv*** al inicio de la siguiente línea. Una vez el entorno virtual se ha creado y activado, se instalarán los módulos con el siguiente comando:

```
python -m pip install -r requirements.txt
```

La instalación llevará un poco de tiempo, pero una vez haya terminado, el servidor estará listo para funcionar.

6.1.4. Instalación de Yawcam.

Para la visualización de la cámara se ha optado por utilizar Yawcam ya que genera de manera muy sencilla una dirección URL con la señal de la cámara y es gratuito.

Para instalar Yawcam basta con ir hasta su sitio web (Yawcam - Yet Another Webcam Software, s.f.) y en el apartado Download clicar sobre yawcam_install.exe para descargar el instalador.

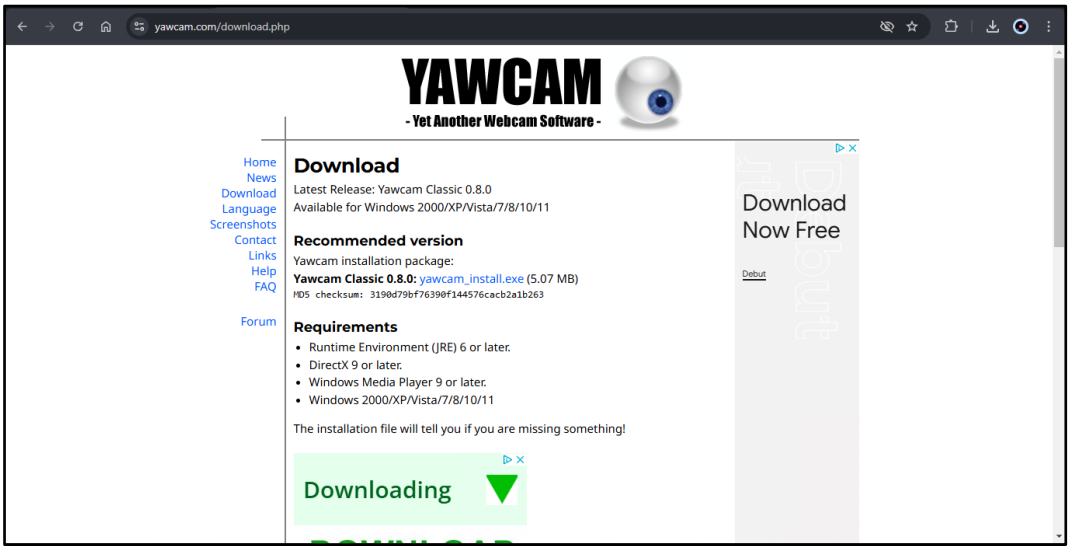


Figura 129. Página de descarga de Yawcam

Ejecutamos el instalador y damos sí a todo lo que nos pida hasta llegar a una ventana donde tendremos que aceptar el acuerdo de licencia (Figura 130).

Pasamos a la siguiente ventana con Next y nos da la opción de escoger la ruta donde se instalará el programa (Figura 131). Con la ruta seleccionada pasamos al siguiente paso con el botón *Next*.

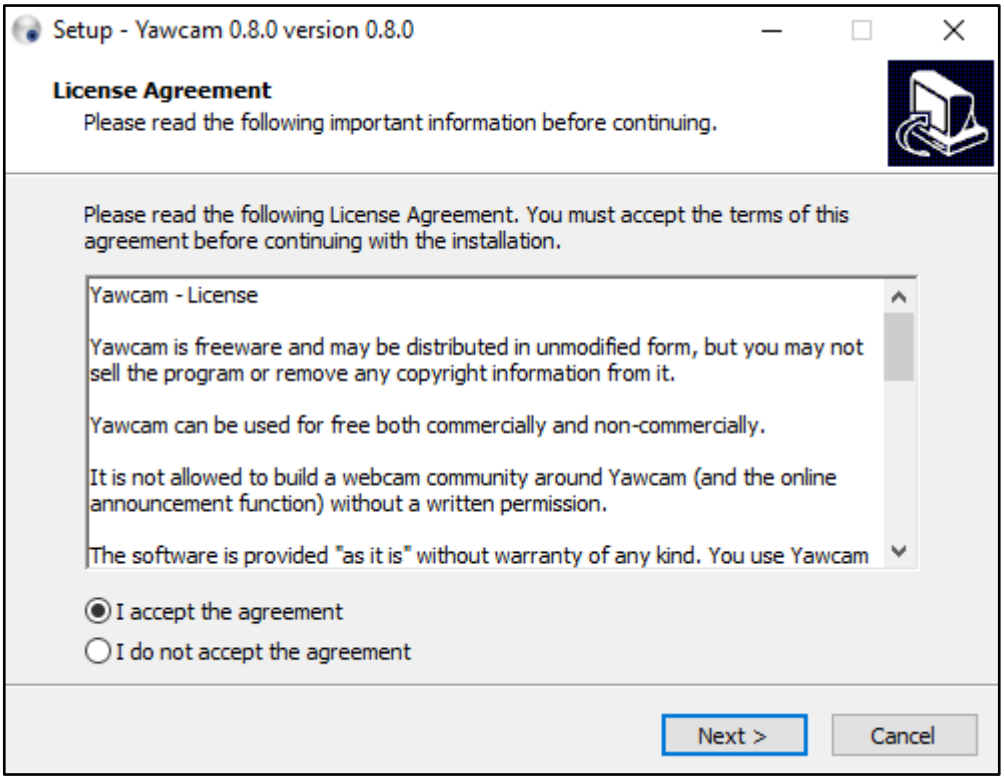


Figura 130. Aceptar acuerdo de licencia Yawcam

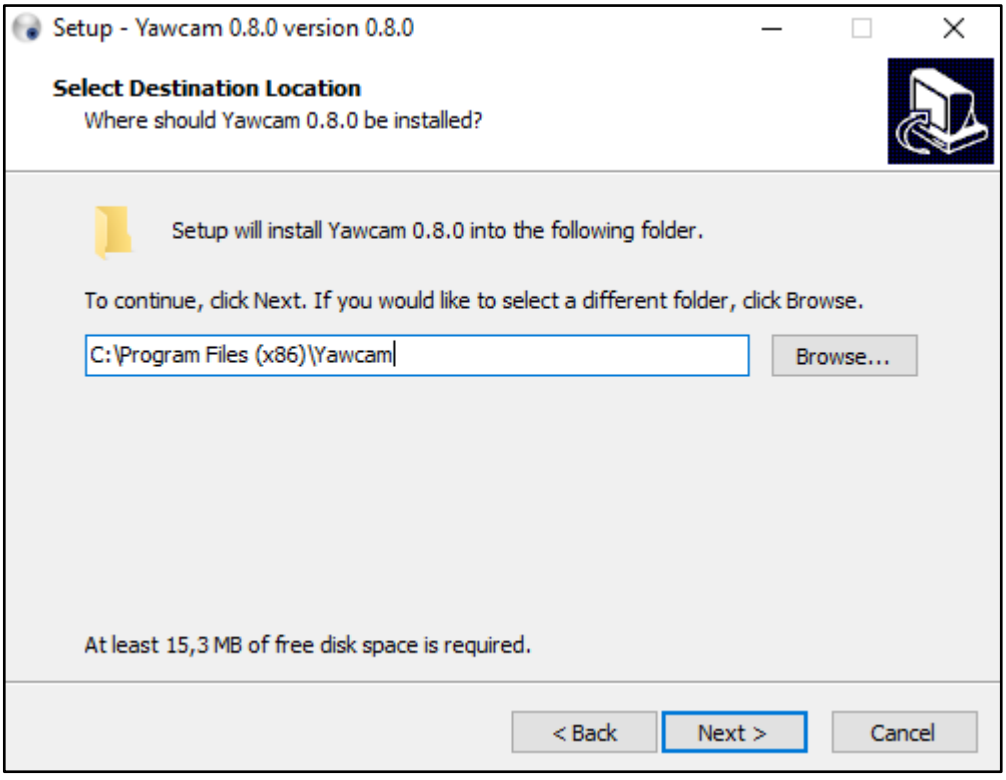


Figura 131. Elección de ruta instalación Yawcam

En la nueva ventana se puede cambiar el nombre de la carpeta donde se aloja el programa. Volvemos a pulsar *Next* y llegamos a la siguiente ventana donde podemos aceptar que se cree un icono en el escritorio y otro para rápido acceso.

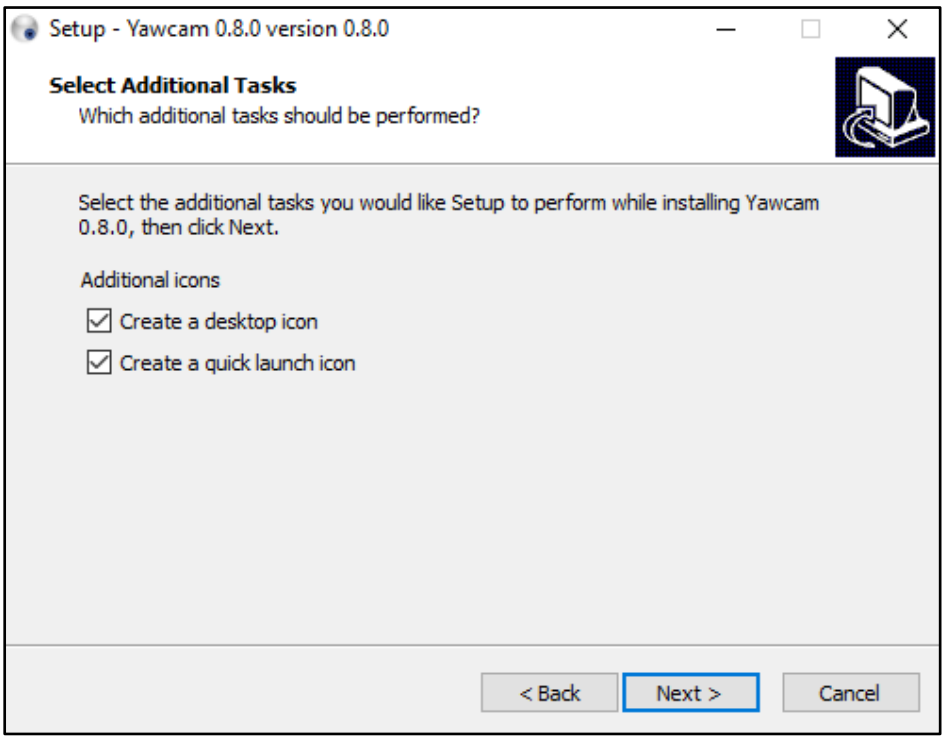


Figura 132. Opciones de creación de accesos directos del instalador de Yawcam

Por último, nos mostrará un resumen con las opciones seleccionadas. Si se quiere cambiar alguna, se puede volver mediante el botón *Back*. Una vez esté todo configurado a nuestro gusto, iniciamos la instalación mediante el botón *Install*.

6.2. Puesta en funcionamiento del servidor

6.2.1. Comprobaciones previas

La primera vez que se arranque el servidor es importante comprobar que el fichero de configuración situado en la carpeta rip-python-server contiene los valores correctos, ya que la información puede variar de una planta a otra. Estos valores incluyen el nombre del servidor OPC de la tarjeta de adquisición de datos y las conexiones de la bomba y el sensor de nivel. Para ello podemos abrir el programa Softing OPC Toolbox Demo Client, que vendrá instalado en los ordenadores del laboratorio, si no fuese así, se puede instalar desde los anejos.

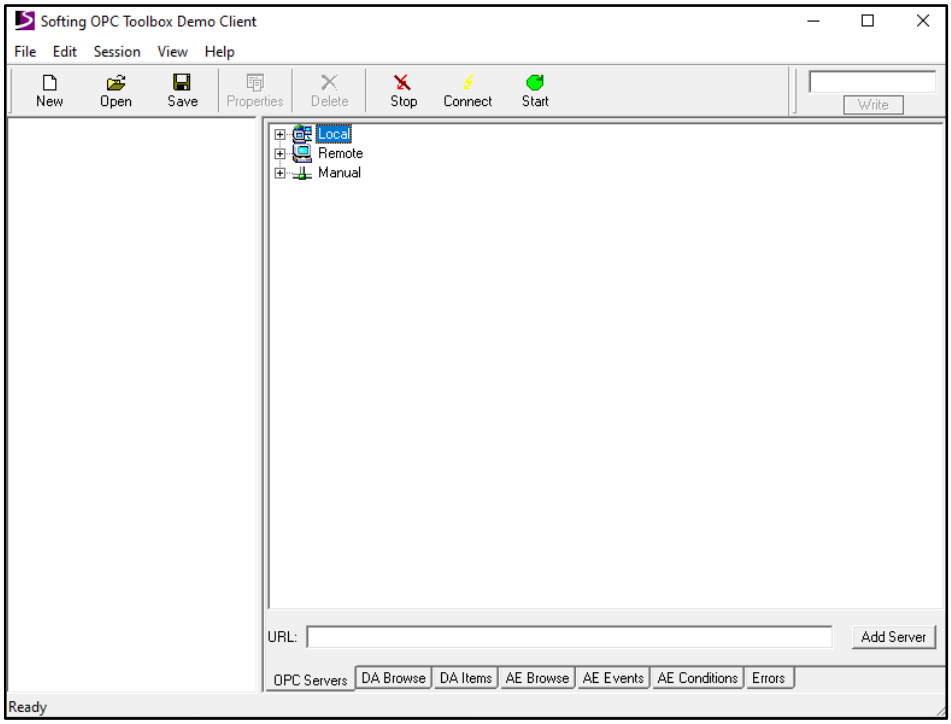


Figura 133. Ventana inicial de Softing OPC Toolbox Demo Client

Una vez ahí entramos en *Local* y buscamos el nombre de la tarjeta; este será el que deba asignarse a *SesionOPC* en el fichero de configuración. Pinchamos en este y le damos al botón *Add Server*.

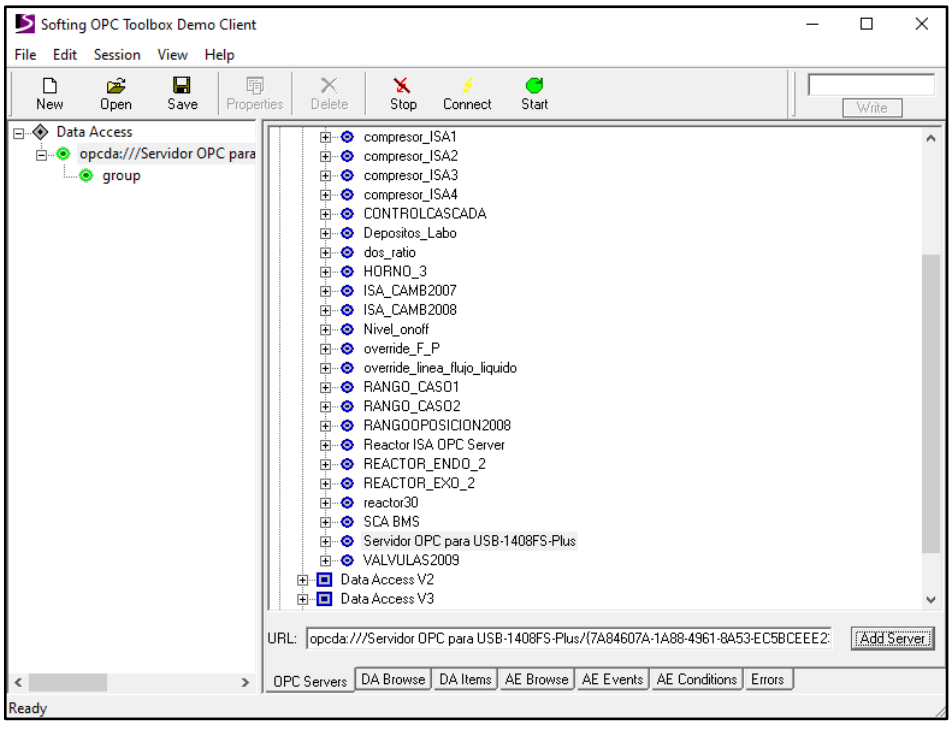
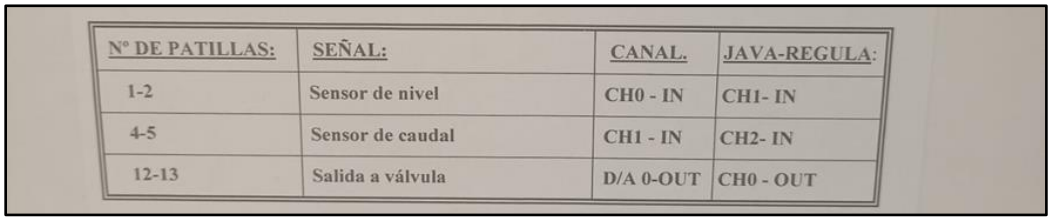


Figura 134. Añadir un servidor en Softing OPC Toolbox Demo Client

A continuación, nos desplazamos a la pestaña *DA Browse* y buscamos la entrada y salida analógica correspondiente; los canales se encuentran en la etiqueta pegada a la planta como la siguiente:



N° DE PATILLAS:	SEÑAL:	CANAL.	JAVA-REGULA:
1-2	Sensor de nivel	CH0 - IN	CH1- IN
4-5	Sensor de caudal	CH1 - IN	CH2- IN
12-13	Salida a válvula	D/A 0-OUT	CH0 - OUT

Figura 135. Etiqueta con las conexiones de la tarjeta de adquisición de datos

Una vez encontrados los canales clicamos en la opción correspondiente en voltios y a la izquierda nos aparecerá el nombre que debemos incluir en el fichero de configuración. Un modelo correcto del contenido del fichero de configuración puede verse en la Figura 137.

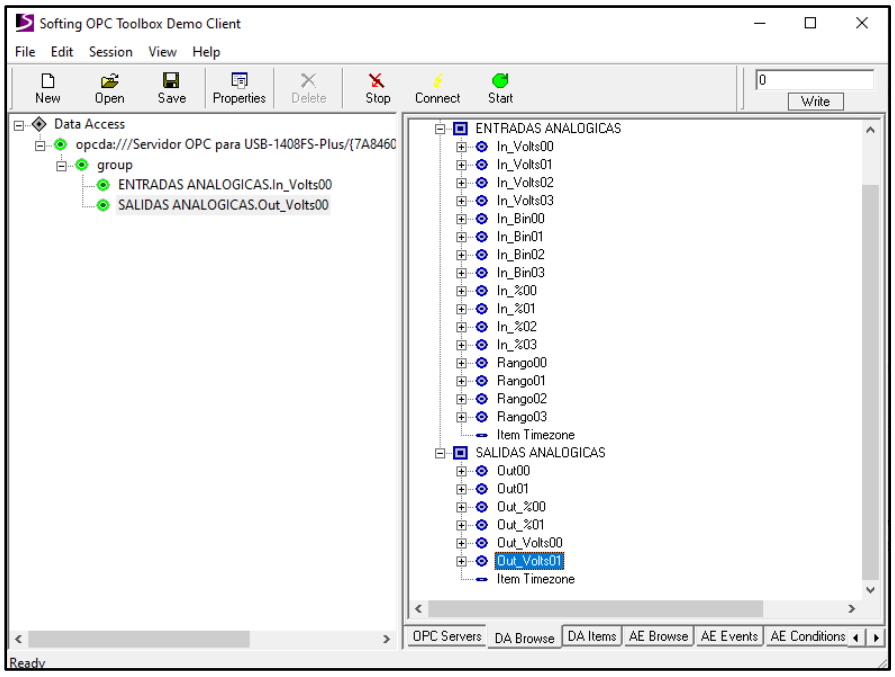


Figura 136. Entradas y salidas añadidas en Softing OPC Toolbox Demo Client

```
IMPORTANTE:
--> Dejar un espacio antes y después del igual
--> Para los decimales utilizar punto
A0 = -25.4
A1 = 25.4
sesionOPC = Servidor OPC USB-1408FS-Plus
idBomba = ENTRADAS ANALOGICAS.In_Volts00
idAltura = SALIDAS ANALOGICAS.Out_Volts00
```

Figura 137. Modelo correcto del fichero de configuración del servidor

6.2.2. Arranque del servidor RIP

Una vez hecho esto, debemos situarnos en la carpeta rip-python-server en el símbolo del sistema; para hacerlo de manera sencilla, seguir las indicaciones que acompañan a la *Figura 127* y a la *Figura 128.*

El servidor se puede arrancar de dos maneras diferentes, ambas son igual de válidas.

6.2.2.1. Primera forma de arranque del servidor

En este caso, activamos el entorno virtual con el siguiente comando:

```
.venv\Scripts\activate
```

Una vez activado el entorno virtual, lanzamos el comando siguiente para poner el servidor en funcionamiento:

```
python App.py
```

6.2.2.2. Segunda forma de arranque del servidor

En este método se arranca el servidor mediante un único comando.

```
.venv\Scripts\python App.py
```

Se use uno u otro método, se debería observar en la pantalla algo similar a lo mostrado en la imagen inferior. Esto nos indica que el servidor está arrancado a la espera de que algún cliente se conecte.

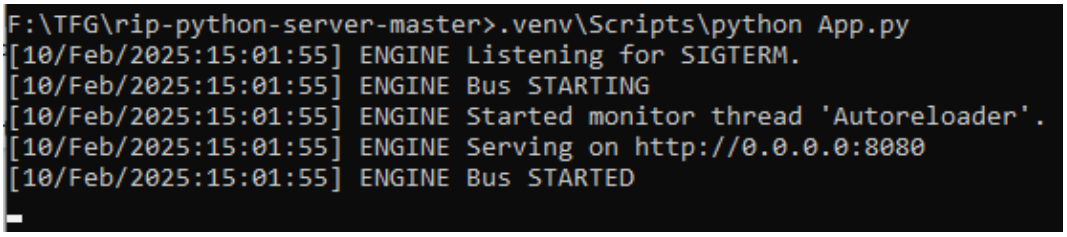


Figura 138. Resultado al arrancar el servidor. Servidor a la espera de cliente.

Cuando se termine de utilizar el servidor, para cerrarlo se tendrá que pulsar la combinación de teclas *control+C* para terminar todos los procesos correctamente. Si se cierra directamente la consola es posible que algún proceso no se termine y siga consumiendo memoria.

Si durante alguno de estos pasos aparece algún tipo de error, en el apartado 6.3 se revisan algunas de las soluciones más habituales.

6.2.3. Arranque del servidor de la cámara

También es necesario iniciar el programa Yawcam, seleccionar la cámara y activar el servidor de imagen en directo. En primer lugar, abrimos el programa y seleccionamos la cámara de la siguiente manera:

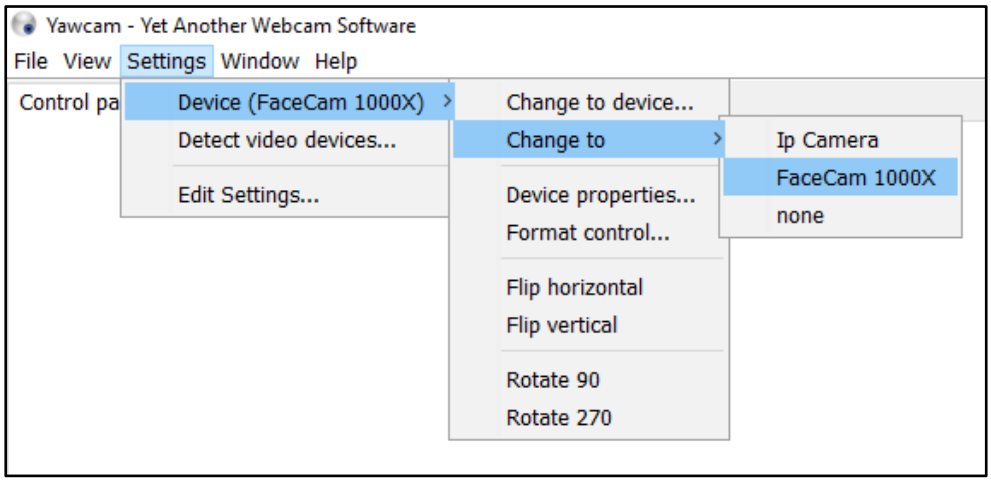


Figura 139. Elección del dispositivo en Yawcam

Una vez hecho esto, activamos la opción *Stream* y con esto ya quedaría arrancado el servidor con la imagen en directo de la planta.

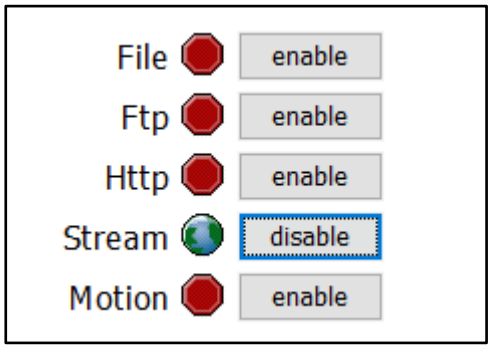


Figura 140. Opciones de trabajo de Yawcam

6.3. Posibles fallos y cómo solucionarlos

6.3.1. Error al instalar los módulos del entorno virtual

Al instalar los módulos del entorno virtual puede aparecer:

```
RuntimeError: No compatible MATLAB installation found in
Windows Registry. This release of MATLAB Engine API for
Python is compatible with version 23.2. The found versions
were 9.14.
```

Esto se soluciona modificando el fichero *requeriments.txt* con una versión adecuada de matlabengine para la versión de Matlab que tengamos instalada. El fichero *requirements* se encuentra en la carpeta *rip-python-server*.

Una vez abierto el fichero se busca matlabengine y se modifica la versión por una adecuada.

Versión de Matlab	Versiones matlabengine
2022a	9.12, 9.12.10, 9.12, 9.12.14, 9.12.15, 9.12.16, 9.12.17, 9.12.19, 9.12.20, 9.12.21
2022b	9.13.1, 9.13.4, 9.13.5, 9.13.6, 9.13.7, 9.13.8, 9.13.9, 9.13.10, 9.13.11
2023a	9.14.2, 9.14.3, 9.14.4, 9.14.6, 9.14.7
2023b	9.15.2, 23.2.1, 23.2.2, 23.2.3
2024a	24.1.1, 24.1.2, 24.1.3, 24.1.4
2024b	24.2.1, 24.2.2
2025a	25.1.1, 25.1.2

Tabla 9. Versiones de matlabengine compatibles con cada versión de Matlab

En la Tabla 9 se muestran las versiones de matlabengine correspondientes a cada versión de Matlab desde la 2022a a la 2025a. Si se dispone de otra versión se puede consultar el historial de versiones de matlabengine (Historico de versiones matlabengine, s.f.), donde al clicar en una versión nos aparece la versión requerida de Matlab. Se recomienda utilizar la versión más reciente.

6.3.2. Errores al lanzar App.py

6.3.2.1. No Python at

```
No Python at 'C:\Program Files\Python39\python.exe’
```

Debemos abrir el archivo *pyvenv.cfg* de la carpeta *.venv* y modificar la ruta de home con la ruta donde se ubique python.exe en esa máquina.

6.3.2.2. ModuleNotFoundError

```
ModuleNotFoundError: No module named 'ujson'
```

Esto quiere decir que no encuentra el módulo en el entorno virtual, para solucionarlo bastará con instalarlo individualmente mediante

```
 pip install ujson
```

CAPÍTULO 7: Manual de usuario del cliente

7.1. Elementos de la interfaz

7.1.1. Menú superior

En la parte superior de la interfaz del cliente del laboratorio se dispone de una serie de botones cuyas funciones se comentan en este apartado. El color del fondo de los botones nos indica si un botón está activo o no. En la Figura 141, por ejemplo, los botones *Descripción, Guardar, Configuración* e *Información* están activos mientras que los botones *Desconectar* y *Guardar* están desactivados.

Figura 141. Menú superior de la interfaz del cliente

7.1.1.1. Botón Descripción

Este botón abre la pestaña con la descripción del laboratorio. En la parte superior derecha de esta encontramos un botón para volver a la interfaz.

7.1.1.2. Selector del tipo de control

Este selector únicamente estará activo cuando el cliente esté conectado. Permite seleccionar entre las siguientes opciones:

- **Operación manual**: el usuario puede experimentar controlando la potencia de la bomba o, visto de otra forma, se fija el caudal de entrada al tanque.

- **Control PID**: el usuario introduce los parámetros del controlador y este trata de alcanzar el nivel de agua deseado. Se indicarán la referencia, la constante proporcional, el tiempo derivativo, el tiempo integral y la constante del filtro del término derivativo. Se podrá aplicar tanto un controlador PID con filtro en el término derivativo como el controlador PID empleado por el software JavaRegula.

7.1.1.3. Botón Guardar

Mediante este botón los datos registrados se almacenarán en un archivo de texto plano organizados en columnas separadas por espacios.

LABORATORIO REMOTO DE DOS TANQUES COMUNICANTES

Universidad de Valladolid
Escuela de Ingenierías Industriales
Fecha: 4/2/2025, 11:06:10

Tipos de control: 1-Operación manual 2-Controlador PID (Ideal) 3-Controlador PID (JavaRegula)

Tiempo(s)	Altura depósito salida(%)	Tipo de control	Referencia(%)	Señal de control(%)
856,00	40,24	3	40,00	14,84
857,00	39,53	3	40,00	15,42
858,00	39,87	3	40,00	15,15
859,00	39,79	3	40,00	15,22
860,00	39,00	3	40,00	15,86
861,00	39,68	3	40,00	15,31
862,00	40,18	3	40,00	14,91
863,00	40,05	3	40,00	15,01
864,00	39,62	3	40,00	15,37
865,00	40,24	3	40,00	14,87

Figura 142. Contenido del fichero de guardado de datos del laboratorio

7.1.1.4. Botón Configuración

Este botón abre una página para configurar la URL del servidor RIP y de la cámara.

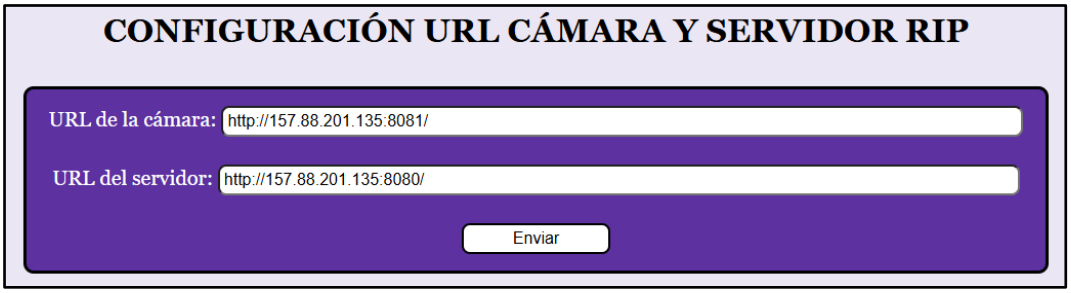


Figura 143. Panel de configuración de las URLs de los servidores RIP y de la cámara

7.1.1.5. Botón Conectar

Este botón nos permite conectar con el servidor RIP introducido en la configuración. Únicamente se conectará si el servidor RIP con esa URL ha sido arrancado previamente. Mientras se permanezca conectado con el servidor, este botón permanecerá desactivado.

7.1.1.6. Botón Desconectar

Mediante este botón nos desconectamos del servidor RIP al que esté conectado. Si el botón Conectar se encuentra activo, este botón estará desactivado y viceversa.

7.1.1.7. Icono información

Mediante este icono se muestran los créditos del trabajo.

Figura 144. Página con los créditos del proyecto

7.2. Menú de opciones

Justo debajo del menú superior encontramos los paneles donde podemos modificar los parámetros del laboratorio. Se dispone de dos paneles diferentes: uno con las opciones generales del laboratorio y otro con las opciones del tipo de control.

7.2.1. Panel opciones

Mediante este panel podemos modificar los siguientes cuatro parámetros.

- La franja temporal visible en las gráficas.
- Modificar el periodo de muestreo del PID.
- Activar o desactivar la opción de autoescalado en el eje Y.
- Activar el registro de datos. Si no se selecciona esta opción el botón Guardar permanece inactivo.

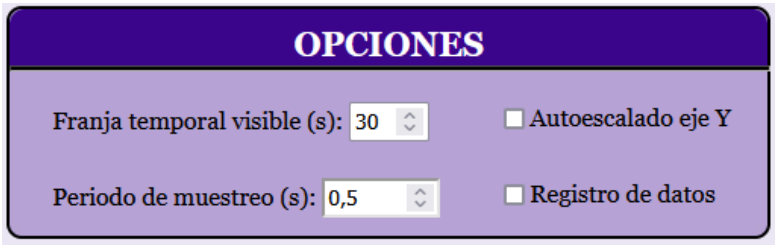


Figura 145. Panel de opciones de la interfaz del cliente

7.2.2. Paneles parámetros de control

Como se adelantó anteriormente, en función del tipo de control escogido se mostrará uno u otro panel de control.

7.2.2.1. Panel de control manual

En este caso solamente se puede modificar la potencia de la bomba; esto se puede hacer mediante una slider o introduciendo directamente el valor. El valor está regulado entre 0 y 100.

Figura 146. Panel de control manual en la interfaz del cliente

7.2.2.2. Panel de opciones PID

En este panel disponemos de un selector para el tipo de PID: ideal o JavaRegula. También se podrán modificar el valor de la referencia y los parámetros del PID.

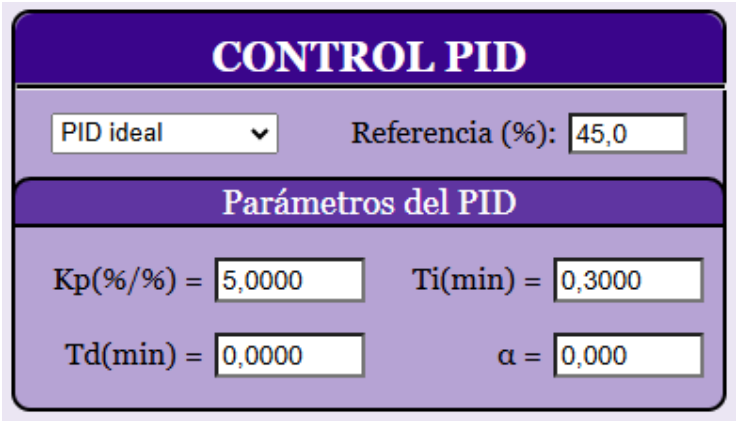


Figura 147. Panel de opciones del PID en la interfaz del cliente

7.3. Visualización de la planta

Para visualizar la evolución temporal de la planta disponemos de dos elementos: la imagen de la cámara y las gráficas.

7.3.1. Imagen de la planta real

Si se ha introducido una URL correcta de un servidor con la imagen de la cámara enfocada a la planta del laboratorio, se observará la imagen de la siguiente forma:

Figura 148. Imagen de la planta real en la interfaz del cliente

Si acercamos el ratón a la parte superior de la imagen podremos redimensionar la imagen para ajustarla mejor al espacio disponible.

7.3.2. Gráficas

Justo al lado de la imagen de la planta real tenemos las gráficas con los datos del laboratorio.

En la gráfica superior se representa la altura del tanque de salida en %; además, si se ha seleccionado el modo de control PID, aparecerá una segunda traza con el valor de la referencia a seguir por el controlador. Por otro lado, en la gráfica inferior se representa la potencia de la bomba en %.

En la parte superior derecha de ambas gráficas se dispone de un visualizador con el valor instantáneo tanto del nivel del tanque de salida como de la potencia de la bomba.

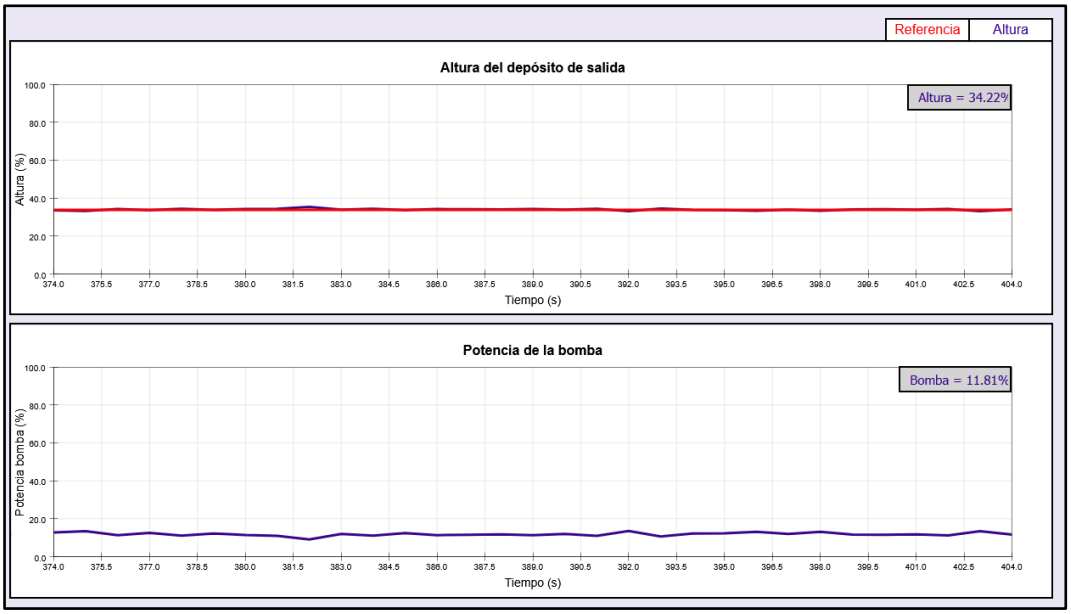


Figura 149. Gráficas de la interfaz del cliente

7.4. Instrucciones de funcionamiento

El orden de utilización del servidor es el siguiente: configuración, conexión y desconexión.

7.4.1. Configuración

Para abrir la interfaz del cliente se puede hacer de manera local, es decir, con el archivo alojado en el dispositivo o desde el servidor externo, accesible desde https://www2.eii.uva.es/jmzama/material.html.

En primer lugar, se pulsa el botón *Configuración*; ahí nos encontraremos con el panel para introducir las direcciones del servidor RIP y del servidor de la cámara. Justo debajo del panel encontraremos unas instrucciones para activar el contenido mixto. Si hemos abierto el cliente de manera local, podemos omitir la lectura de esas instrucciones, pero si abrimos el cliente desde el servidor, tendremos que seguir las instrucciones, ya que las URLs del servidor son HTTP y el servidor es HTTPS, como se explicó en el apartado 3.3.

A continuación, se muestran las indicaciones que encontraremos en la página de configuración para activar el contenido mixto en diferentes buscadores.

7.4.1.1. Google Chrome

1. Clique sobre icono situado justo a la izquierda de la barra del buscador y abra la *Configuración de sitios*.

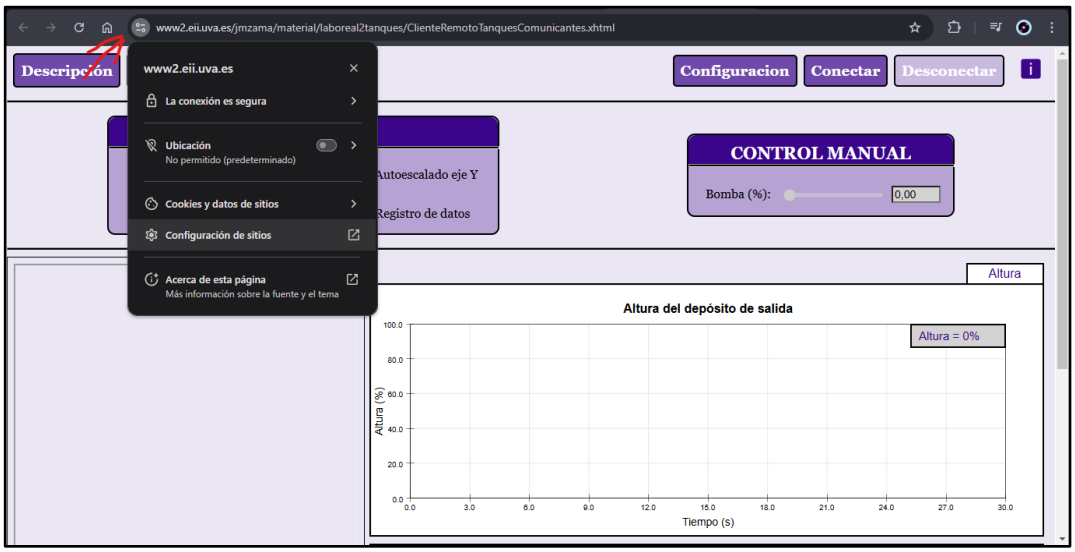


Figura 150. Primer paso para activar el contenido mixto en Google Chrome

2. Cambie la opción *Contenido no seguro* a Permitir. Al hacer esto ya puede conectarse al servidor.

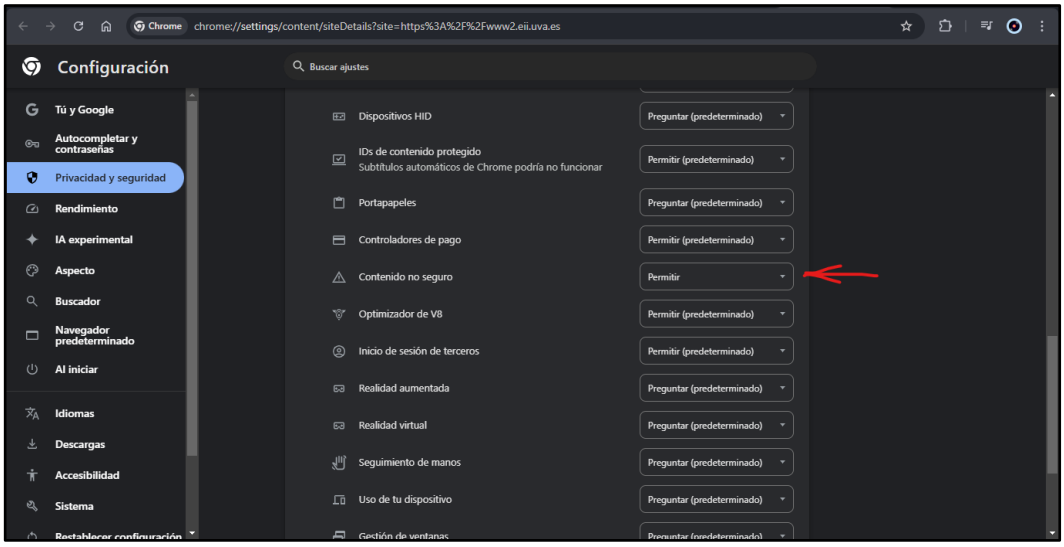


Figura 151. Segundo paso para activar el contenido mixto en Google Chrome

7.4.1.2. Firefox

1. Introduzca en el buscador about:config y pulse en *Aceptar el riesgo y continuar*.

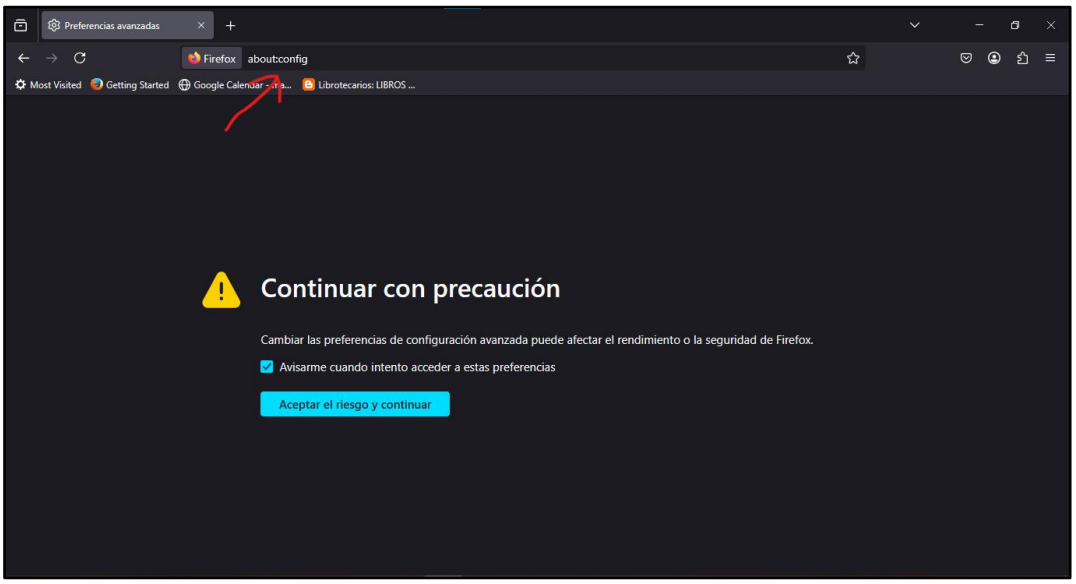


Figura 152. Primer paso para activar el contenido mixto en Firefox

2. En la barra de búsqueda, escriba *block_active*.

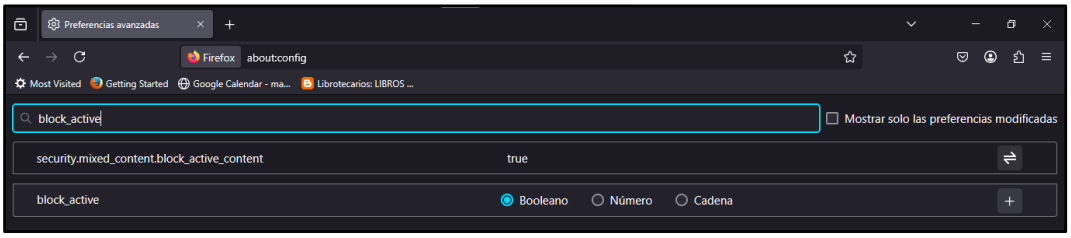


Figura 153. Segundo paso para activar el contenido mixto en Firefox

3. Cambie la opción *security.mixed_content.block_active_content* a false. Con esto ya puede conectarse al servidor.

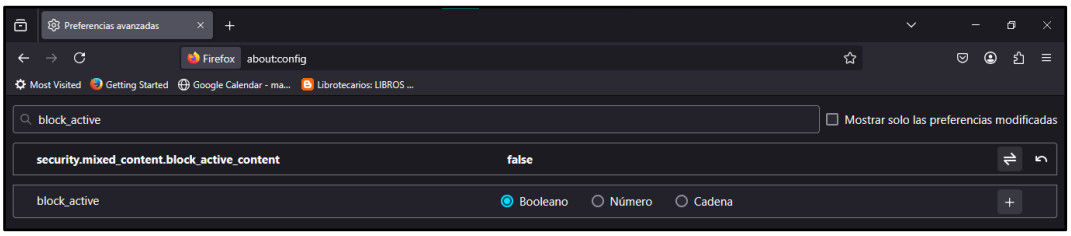


Figura 154. Tercer paso para activar el contenido mixto en Firefox

7.4.1.3. Microsoft Edge

1. Clique sobre el candado situado justo a la izquierda de la barra del buscador y vaya a la opción *Permisos para este sitio*.

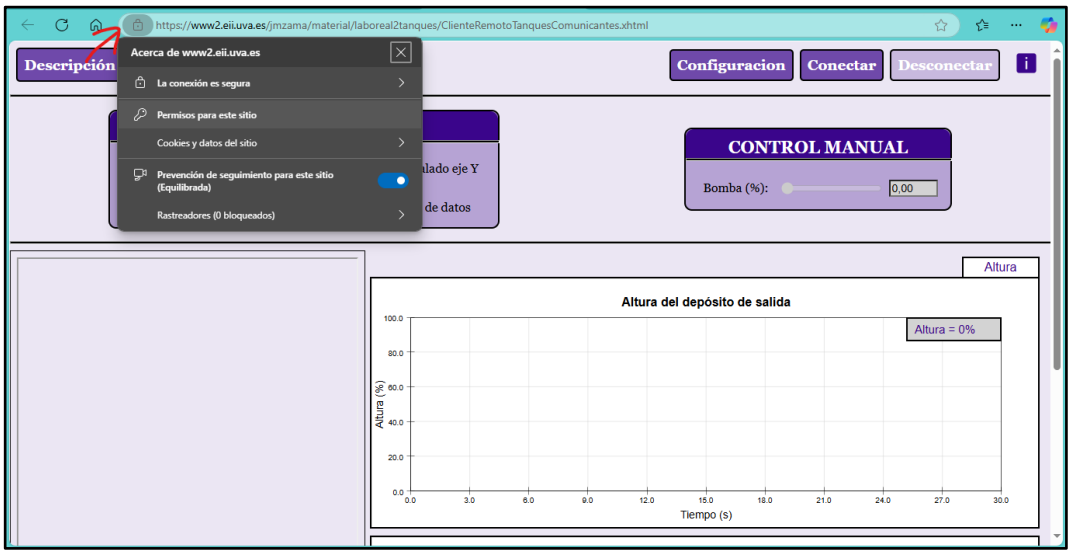


Figura 155. Primer paso para activar el contenido mixto en Microsoft Edge

2. Cambie la opción *Contenido no seguro* a Permitir. Al hacer esto ya puede conectarse al servidor.

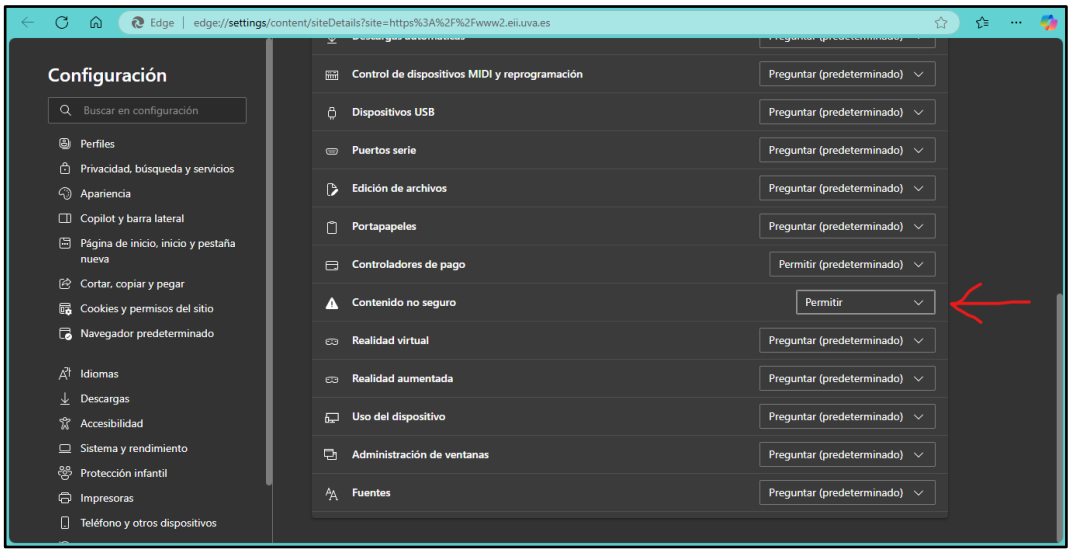


Figura 156. Segundo paso para activar el contenido mixto en Microsoft Edge

Una vez se tengan las URLs incluidas, pulsamos el botón *Enviar* o la tecla Intro y volveremos a la interfaz con los nuevos valores actualizados.

Al volver a la interfaz, nos tendría que aparecer la señal de la cámara; si no fuese así podría ser que la URL sea incorrecta o que el servidor de la cámara no esté arrancado.

7.4.2. Conectar

Una vez realizada la configuración pulsamos el botón *Conectar* para empezar a obtener datos del servidor de la planta; este proceso puede tardar unos segundos.

Si no se observa ningún cambio en la interfaz, puede ser por tres motivos diferentes: que la URL del servidor sea incorrecta, que el servidor no esté arrancado o, si se ha abierto la interfaz del cliente desde el servidor web, que no hayamos activado correctamente el contenido mixto.

Si al conectarse con el servidor no podemos editar ningún parámetro de control y solamente podemos observar los datos, quiere decir que ya hay un maestro conectado a esa planta. En ese caso deberemos esperar a que se cierre la sesión del maestro y del resto de clientes que se hayan conectado antes de nuestra sesión para obtener el rol de maestro.

7.4.3. Desconectar

Por último, una vez se termine de usar el cliente, es importante pulsar el botón *Desconectar* antes de cerrar la ventana para avisar al servidor del cierre de sesión. Si se cierra o se refresca la ventana con el cliente, la sesión con ese cliente va a seguir abierta y el resto de los clientes que se conecten lo harán como observadores. Si esto ocurriera se tendría que reiniciar el servidor.

CAPÍTULO 8: Conclusiones y trabajo futuro

8.1. Conclusiones

El principal objetivo de este proyecto consistía en el desarrollo de un laboratorio remoto para las plantas piloto de dos tanques comunicantes del laboratorio de Control de Procesos, emulando el comportamiento del software de control JavaRegula utilizado presencialmente. En vista de los resultados obtenidos y la comparación con el programa presencial, puede afirmarse que el laboratorio remoto realizado cumple con las expectativas y necesidades buscadas.

El procedimiento se ha basado en la programación tanto del servidor como del cliente del laboratorio remoto, la validación de su comportamiento y la optimización del programa. Durante el desarrollo no se han encontrado problemas significativos ni alteraciones relevantes.

Para la realización del proyecto han sido aplicados conocimientos adquiridos en asignaturas como: Fundamentos de Informática, Fundamentos de Automática, Informática Industrial o Control de Procesos. Además, ha sido necesario ampliar los conocimientos de Matlab y Python adquiridos en diversas asignaturas. Por otro lado, se han estudiado nuevos lenguajes de programación como HTML, CSS y JavaScript y profundizado en el programa Easy JavaScript Simulations (EJSS).

La mayor dificultad encontrada durante el desarrollo de este proyecto ha sido el entendimiento del funcionamiento de cada uno de los elementos utilizados en el laboratorio remoto, sobre todo del servidor RIP de Python utilizado, ya que no se ha encontrado mucha información sobre él, y del programa EJSS que, aunque sí que presenta un sitio web específico con bibliografía, muchas propiedades de los elementos no se encuentran o están mal traducidas o sin explicar.

Por otro lado, una vez empaquetado el cliente de EJSS, es necesario realizar una serie de modificaciones ya que genera archivos innecesarios. A esto hay que sumarle que el código generado es complicado de entender y no permite modificar fácilmente la dirección del servidor RIP introducida dentro del programa antes del empaquetado, lo que hizo necesario el desarrollo de una nueva página para su modificación.

A pesar de lo comentado en el párrafo anterior, EJSS resulta muy práctico a la hora de crear la interfaz del cliente debido a su fácil utilización basada en arrastrar objetos; además, gracias a la implementación del protocolo RIP, utilizando el servidor RIP de Python en el otro extremo, es de gran utilidad a la hora de crear laboratorios remotos como este.

El principal inconveniente del proyecto realizado es las escasas opciones de software disponible para generar un servidor web con la imagen de una cámara USB. De hecho, el único que cumple con estas condiciones es Yawcam. A esto hay que sumarle que este programa genera direcciones no seguras (http) para alojar las imágenes; esto conlleva a un problema de contenido mixto cuando utilizamos el cliente del laboratorio desde el servidor web en el que se aloja, ya que todos los buscadores bloquean el acceso a páginas no seguras desde una página segura. Aquí es donde encontramos el gran inconveniente: tener que activar el contenido mixto para utilizar todas las funciones del laboratorio, asumiendo el riesgo de ataques o modificaciones de los datos mostrados en la interfaz.

Para terminar, el laboratorio remoto realizado se comporta de manera similar al software JavaRegula, presenta una fácil utilización ya que contiene una interfaz bastante intuitiva y, además, utiliza una gama de colores cuidada desde un punto de vista estético. En definitiva, se ha obtenido una opción completamente sustituible al trabajo presencial.

8.2. Trabajo futuro

Como continuación de este trabajo, se propone dar solución a la problemática del contenido mixto en el servidor donde se aloja el cliente. Para ello, se podría realizar un programa que genere una dirección segura (HTTPS) donde se aloje la imagen de la cámara conectada al laboratorio; este programa sustituiría a Yawcam en el servidor del laboratorio remoto.

Por otro lado, se propone modificar el servidor para que directamente pueda obtener los valores de la tarjeta de adquisición de datos, es decir, integrar el protocolo OPC en el servidor RIP de Python; de esta manera se prescindiría de Matlab y todo el laboratorio estaría compuesto por software libre. También se tendrían que implementar las ecuaciones de los PIDs y el resto de las funciones del fichero *servidorRemoto.m* en Python.

CAPÍTULO 9: Bibliografía

¿Cómo afecta a mi seguridad el contenido no seguro? (27 de octubre de 2019). Recuperado el 10 de febrero de 2025, de https://support.mozilla.org/es/kb/como-afecta-mi-seguridad-el-contenido-no-seguro

Chacón Sombria, J., & de la Torre, L. (15 de Noviembre de 2018). *UNEDLabs/rip-python-server*. Recuperado el 2 de diciembre de 2024, de https://github.com/UNEDLabs/rip-python-server

de la Torre, L., & UNED. (18 de Noviembre de 2018). *UNEDLabs/rip-spec*. Recuperado el 2 de diciembre de 2024, de https://github.com/UNEDLabs/rip-spec

Esquembre, F. (2013). *Easy Java Simulations Wiki*. Recuperado el 26 de noviembre de 2024, de https://www.um.es/fem/EjsWiki/

Historico de versiones matlabengine. (s.f.). Recuperado el 12 de febrero de 2025, de https://pypi.org/project/matlabengine/#history

MATLAB. (s.f.). *MATLAB - MathWorks*. Recuperado el 5 de diciembre de 2024, de https://es.mathworks.com/help/matlab/index.html

OPC foundation. (s.f.). *What is OPC?* Recuperado el 2 de diciembre de 2024, de https://opcfoundation.org/about/what-is-opc/

Python. (s.f.). Recuperado el 10 de febrero de 2025, de Python: https://www.python.org/

Python Releases for Windows. (s.f.). Recuperado el 10 de febrero de 2025, de https://www.python.org/downloads/windows/

The Mathworks - Downloads. (s.f.). Recuperado el 10 de febrero de 2025, de Mathworks: https://es.mathworks.com/downloads/

UNED. (s.f.). *University Network of Interactive Laboratories*. Recuperado el 26 de noviembre de 2024, de https://unilabs.dia.uned.es/

UNEDLabs. (s.f.). *UNILabs*. Recuperado el 26 de noviembre de 2024, de https://www.youtube.com/@UNEDLabs

W3Schools. (s.f.). *HTML Tutorial*. Recuperado el 3 de diciembre de 2024, de https://www.w3schools.com/Html/

Yawcam - Yet Another Webcam Software. (s.f.). Recuperado el 12 de febrero de 2025, de yawcam: https://www.yawcam.com/

CAPÍTULO 10: Anexos

Dentro de los anexos del proyecto encontramos todos los archivos del laboratorio remoto. Este contenido se divide en dos carpetas, una con el cliente y otra con el servidor.

La carpeta del cliente se denomina ClienteRemotoTanquesComunicantes y contiene lo siguiente:

- **_ejs_library**: esta carpeta contiene las librerías y estilos utilizados por la página principal ClienteRemotoTanquesComunicantes.xhtml
- **css**: carpeta que contiene los estilos del resto de páginas.
- **fotos**: carpeta que contiene las imágenes utilizadas en las diferentes páginas del cliente.
- **ClienteRemotoTanquesComunicantes.ejss:** contiene el código de la interfaz del cliente realizado en EJSS.
- **ClienteRemotoTanquesComunicantes.xhtml:** archivo principal que contiene la interfaz del cliente.
- **Config.html:** página web para la configuración del cliente
- **Creditos.html:** página web con los créditos del laboratorio remoto.
- **Descripción.html:** página web con la descripción del cliente del laboratorio remoto.
- **Manual_Clientes_Remoto_Tanques_Comunicantes.pdf:** un documento que incluye la descripción de cada elemento de la interfaz del cliente y las instrucciones para un correcto funcionamiento.

La carpeta del servidor se denomina rip-python-server y se compone de los siguientes elementos:

- **jsonrpc**: carpeta con archivos que implementan el protocolo JSON-RPC para la comunicación con el cliente.
- **log**: directorio que contiene un archivo con el histórico de conexiones realizadas con el servidor.

- **rip**: directorio que contiene los archivos RIPGeneric y RIPMatlab, así como otros archivos que hacen capaces la comunicación del servidor con Matlab y con los clientes.

- **samplers**: carpeta que contiene la clase de un muestreador periódico utilizado para la actualización de datos.

- **__init__.py**: archivo necesario para que Python reconozca este directorio como un paquete y funcione correctamente.

- **App.py**: fichero principal del servidor; es el ejecutado para arrancar el servidor.

- **AppConfig.py**: fichero que contiene la dirección ip y las variables utilizadas por el servidor.

- **ClientE.zip**: contiene el software Softing OPC Toolbox Demo Client.

- **CODE_OF_CONDUCT.md**: archivo con el código de conducta creado por los desarrolladores del servidor.

- **configuracion.txt**: este archivo tiene configuración utilizada por servidorRemoto.m en la comunicación OPC.

- **CONTRIBUTING.md**: archivo de contribución creado por los desarrolladores del servidor.

- **HttpServer.py**: archivo encargado de generar la URL del servidor.

- **LICENSE.txt**: fichero con la licencia del laboratorio.

- **Manual_Servidor_Remoto_Tanques_Comunicantes.pdf**: un documento que incluye los pasos a seguir para la instalación y el arranque del servidor.

- **README.md:** archivo con información generada por los desarrolladores del servidor.

- **requirements.txt**: fichero que contiene la configuración del entorno virtual que se cree para el funcionamiento del servidor.

- **servidorRemoto.m**: contiene el código de Matlab para el control y comunicación con la planta.