

Escola Universitaria Politécnica



UNIVERSIDADE DA CORUÑA

Máster en Informática Industrial y Robótica

TRABAJO DE FIN DE MÁSTER

TFM Nº: 4523MI1A10

**TÍTULO: EMULACIÓN DE PLANTA DE NIVEL BASADA EN SISTEMA
EMBEBIDO**

AUTOR: DANIEL MÉNDEZ BUSTO

**TUTOR: FRANCISCO ZAYAS GATO
HÉCTOR QUINTIÁN PARDO**

FECHA: JUNIO DE 2022

Fdo.: EL/LA AUTOR/A

Fdo.: EL/LA TUTOR/A

TÍTULO: **EMULACIÓN DE PLANTA DE NIVEL BASADA EN SISTEMA
EMBEBIDO**

ÍNDICE

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: **JUNIO DE 2022**

AUTOR: **EL/LA ALUMNO/A**

Fdo.: **DANIEL MÉNDEZ BUSTO**

I ÍNDICE	3
Contenidos del TFG	5
Listado de figuras	9
Listado de tablas	13
II MEMORIA	15
Índice del documento Memoria	17
1 INTRODUCCIÓN	19
2 REFERENCIAS	23
2.1 Bibliografía	23
3 ESTRUCTURA GENERAL	25
4 IDENTIFICACIÓN	27
4.1 Introducción a los métodos de identificación	27
4.2 Aplicación del método de identificación	33
4.3 Regulador PID	35
4.4 Generar Señal PRBS	36
4.5 Respuesta Sistema a señal PRBS	38
4.6 Obtención de parámetros	39
4.7 Comprobación	45
5 DESPLIEGUE DEL MODELO	47
5.1 Función Main	48
5.2 Función Entradas recibidas desde node-red	49
5.3 Función Parámetros Función de Transferencia Planta	49
5.4 Función PID	49
5.5 Función Respuesta Planta	49
5.6 Función Display Variador	50
5.7 Función Paso a formato JSON	50
6 INTERFACE EN NODE-RED	51
6.1 Motor de la aplicación	54
6.1.1 Nodo Cíclico Main	54
6.1.2 Nodo Arranque	54
6.1.3 Nodo Función Inicialización	55
6.1.4 Nodo Función Parámetros de Entrada	55
6.1.5 Nodo Función String a CMD	56
6.1.6 Nodo Función Mensaje Llamada Planta Emulada	57
6.1.7 Nodo Función Comando Llamada Planta Emulada	58

6.1.8 Nodo Función de String a JSON	58
6.1.9 Nodo función Memoria Estados Anteriores	59
6.1.10 Datos para HMI	60
6.1.11 Datos desde HMI	61
6.2 Interface Gráfico	62
6.2.1 Pantalla de inicio	64
6.2.2 Pantalla Sinóptico	65
6.2.3 Pantalla Ajustes	69
6.2.4 Pantalla Gráficos	73
6.3 Arquitectura en Node-Red	74
 III PRESUPUESTO	 75
Índice del documento Presupuesto	77
 7 PRESUPUESTO MATERIALES	 79
 8 PRESUPUESTO MANO DE OBRA	 79
 9 PRESUPUESTO	 79
 IV PLIEGO DE CONDICIONES	 81
Índice del documento Pliego de Condiciones	83
 10 CONDICIONES DE TRABAJO	 85
 11 Hardware y Software	 86
11.1 Hardware	86
11.2 Software	86
 V ANEXOS	 87
Índice del documento Anexos	89
 12 Manual de funcionamiento	 91
12.1 Instalaciones previas	91
12.1.1 Instalación del sistema operativo	91
12.1.2 Iniciando Raspberry-Pi	94
12.1.3 Instalación de Node-Red	95
12.1.4 Configuración en Raspberry-Pi	96
12.2 Funcionamiento de la emulación	104
 13 Códigos de Programación	 109

13.1 Planta.py	109
13.1.1 Main	109
13.1.2 Entradas	109
13.1.3 Parámetros Planta	110
13.1.4 Regulador PID	111
13.1.5 Planta	111
13.1.6 Frecuencia Variador	112
13.1.7 Paso a JSON	112
14 Variables Scripts	113
14.1 Identificación	113
14.2 Comprobación	114

Listado de figuras

1.1	Planta pequeña de nivel	19
1.2	Sinóptico Planta pequeña de nivel	20
1.3	Acceso a la emulación	21
3.1	Estructura General	25
4.1	Tipos de métodos de identificación	27
4.2	Barrido Senoidal	28
4.3	Señal RBS	28
4.4	Señal PRBS	29
4.5	Método identificación offline	29
4.6	Método identificación online	30
4.7	Modelo general-lineal	30
4.8	Modelo ARX	31
4.9	Modelo ARMAX	31
4.10	Modelo Output-Error	31
4.11	Modelo Box-Jenkins	32
4.12	Métodos recursivos	32
4.13	Sistema en lazo cerrado	33
4.14	Señal PRBS introducida al sistema	34
4.15	Esquema Identificación	34
4.16	Esquema Entrada Escalón	37
4.17	Señal PRBS multiplicada por cinco	37
4.18	Esquema Script Identificación	38
4.19	Respuesta a PRBS en punto de operación 50	39
4.20	Ident: Ventana principal	39
4.21	Ident: Configuración de datos para análisis	40
4.22	Ident: Selección de método de estimación	40
4.23	Ident: Ventana principal modelo polinómico	41
4.24	Ident: Estimación	41
4.25	Ident: Resultado estimación	42
4.26	Ident: Estimación a espacio de trabajo	43
4.27	Variable arx221	43
4.28	Estructura modelo arx	43
4.29	Esquema Script Comprobación	45

4.30	Respuesta Sistema Emulado	46
5.1	Llamada cíclica a planta emulada	47
5.2	Intercambio de señales	47
5.3	Estructura fichero	48
6.1	Apariencia node-red	51
6.2	Estructura básica node-red	52
6.3	Arquitectura planta de nivel emulada	53
6.4	Node-Red: Nodo Main	54
6.5	Node-Red: Nodo Arranque	55
6.6	Node-Red: Nodo Inicialización	55
6.7	Node-Red: Nodo Parámetros de entrada	56
6.8	Node-Red: Nodo String a CMD	57
6.9	Node-Red: Mensaje llamada a planta emulada	57
6.10	Node-Red: Comando llamada a planta emulada	58
6.11	Node-Red: Paso de formato string a json	58
6.12	Node-Red: Memoria estados anteriores	59
6.13	Node-Red: Valores a HMI	60
6.14	Node-Red: Consigna a HMI	61
6.15	Node-Red: Consignas/Comandos desde HMI	62
6.16	Node-Red: Ejemplo Composición HMI	63
6.17	Node-Red: Ejemplo Composición Dashboard HMI	63
6.18	Node-Red: Código inicio	64
6.19	Node-Red: Inicio HMI	65
6.20	Node-Red: Distribución Sinóptico	65
6.21	Node-Red: Código sinóptico	66
6.22	Node-Red: Nivel sinóptico	67
6.23	Node-Red: Representación nivel	67
6.24	Node-Red: Frecuencia sinóptico	68
6.25	Node-Red: Sinoptico HMI	68
6.26	Node-Red: Introducción numérica	69
6.27	Node-Red: Representación campo introducción numérica	69
6.28	Node-Red: Comando desde HMI	70
6.29	Node-Red: Representación comando	70
6.30	Node-Red: Frecuencia Ajustes	71
6.31	Node-Red: Representación frecuencia (Gauge)	71
6.32	Node-Red: Grafico Ajustes	72
6.33	Node-Red: Representación grafica nivel	72
6.34	Node-Red: Ajustes HMI	73
6.35	Node-Red: Graficas HMI	73
6.36	Node-Red: Arquitectura	74
10.1	Posición válvula manual de vaciado	85

12.1	Descarga de Sistema Operativo	91
12.2	Ejecutar instalador	92
12.3	Finalizar instalación instalador	92
12.4	Raspberry-Pi Imager	92
12.5	Raspberry-Pi Imager - Selección Sistema Operativo	93
12.6	Raspberry-Pi Imager 2	93
12.7	Selección de medio de almacenamiento	94
12.8	Raspberry-Pi Imager 2 - Write	94
12.9	Descarga intalador Node-Red PC	95
12.10	Comprobar versión de node-red	96
12.11	Carpeta de configuración e imagenes	96
12.12	Carpeta de configuración	97
12.13	Carpeta de Imagenes	97
12.14	Modificación de la configuración	98
12.15	Editor de node-red	98
12.16	Menú - Importar	99
12.17	Seleccionar archivo planta emulada	99
12.18	Menú - Importar JSON	100
12.19	Nodos	100
12.20	Seleccionar instalación de paleta	101
12.21	Búsqueda de paquete	101
12.22	Instalación de paleta	102
12.23	Nodos reconocidos	102
12.24	Aceptar configuración nodos no configurados	103
12.25	Configuración correcta	103
12.26	Manual Funcionamiento: Pantalla de inicio	104
12.27	Manual Funcionamiento: Menú de navegación	105
12.28	Manual Funcionamiento: Ajuste de consigna y parámetros regulador	105
12.29	Manual Funcionamiento: Órden de marcha	106
12.30	Manual Funcionamiento: Sinóptico	106
12.31	Manual Funcionamiento: Gráficas	107

Listado de tablas

4.1	Parámetros regulador PID en cada punto de operación	36
4.2	Estimación obtenida en cada punto de operación	42
4.3	Función de Transferencia en cada punto de operación	44
7.1	Presupuesto materiales	79
8.1	Presupuesto mano de obra	79
9.1	Presupuesto	79

TÍTULO: **EMULACIÓN DE PLANTA DE NIVEL BASADA EN SISTEMA
EMBEBIDO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2022

AUTOR: EL/LA ALUMNO/A

Fdo.: DANIEL MÉNDEZ BUSTO

Índice del documento MEMORIA

1 INTRODUCCIÓN	19
2 REFERENCIAS	23
2.1 Bibliografía	23
3 ESTRUCTURA GENERAL	25
4 IDENTIFICACIÓN	27
4.1 Introducción a los métodos de identificación	27
4.2 Aplicación del método de identificación	33
4.3 Regulador PID	35
4.4 Generar Señal PRBS	36
4.5 Respuesta Sistema a señal PRBS	38
4.6 Obtención de parámetros	39
4.7 Comprobación	45
5 DESPLIEGUE DEL MODELO	47
5.1 Función Main	48
5.2 Función Entradas recibidas desde node-red	49
5.3 Función Parámetros Función de Transferencia Planta	49
5.4 Función PID	49
5.5 Función Respuesta Planta	49
5.6 Función Display Variador	50
5.7 Función Paso a formato JSON	50
6 INTERFACE EN NODE-RED	51
6.1 Motor de la aplicación	54
6.1.1 Nodo Cíclico Main	54
6.1.2 Nodo Arranque	54
6.1.3 Nodo Función Inicialización	55
6.1.4 Nodo Función Parámetros de Entrada	55
6.1.5 Nodo Función String a CMD	56
6.1.6 Nodo Función Mensaje Llamada Planta Emulada	57
6.1.7 Nodo Función Comando Llamada Planta Emulada	58
6.1.8 Nodo Función de String a JSON	58
6.1.9 Nodo función Memoria Estados Anteriores	59
6.1.10 Datos para HMI	60
6.1.11 Datos desde HMI	61
6.2 Interface Gráfico	62
6.2.1 Pantalla de inicio	64

6.2.2 Pantalla Sinóptico	65
6.2.3 Pantalla Ajustes	69
6.2.4 Pantalla Gráficos	73
6.3 Arquitectura en Node-Red	74

1 INTRODUCCIÓN

El objetivo del presente Trabajo fin de Máster es realizar el desarrollo de una herramienta para la emulación del comportamiento de la planta pequeña de nivel, ubicada en el laboratorio de optimización y control de la Escuela Universitaria Politécnica de Ferrol. Para ello, se hace uso de una Raspberry-Pi dotada con un interface gráfico desarrollado en Node-Red.

Esta planta es utilizada por los alumnos de la universidad para desarrollar las prácticas de diferentes materias. En este sentido, esta emulación permite que los alumnos puedan realizar estos trabajos sin necesidad de desplazarse al centro de estudio, ya que pueden hacer uso de la planta emulada. La emulación la pueden tener en sus propias casas en una Raspberry-Pi o también existe la posibilidad de conectarse remotamente vía internet a una Raspberry-Pi que contiene la emulación.

El funcionamiento de la planta se basa en mantener un nivel determinado en un depósito haciendo uso de reguladores programados desde un sistema de control. En este caso, el sistema de control está implementado en un ordenador que tiene instalado el software Matlab.

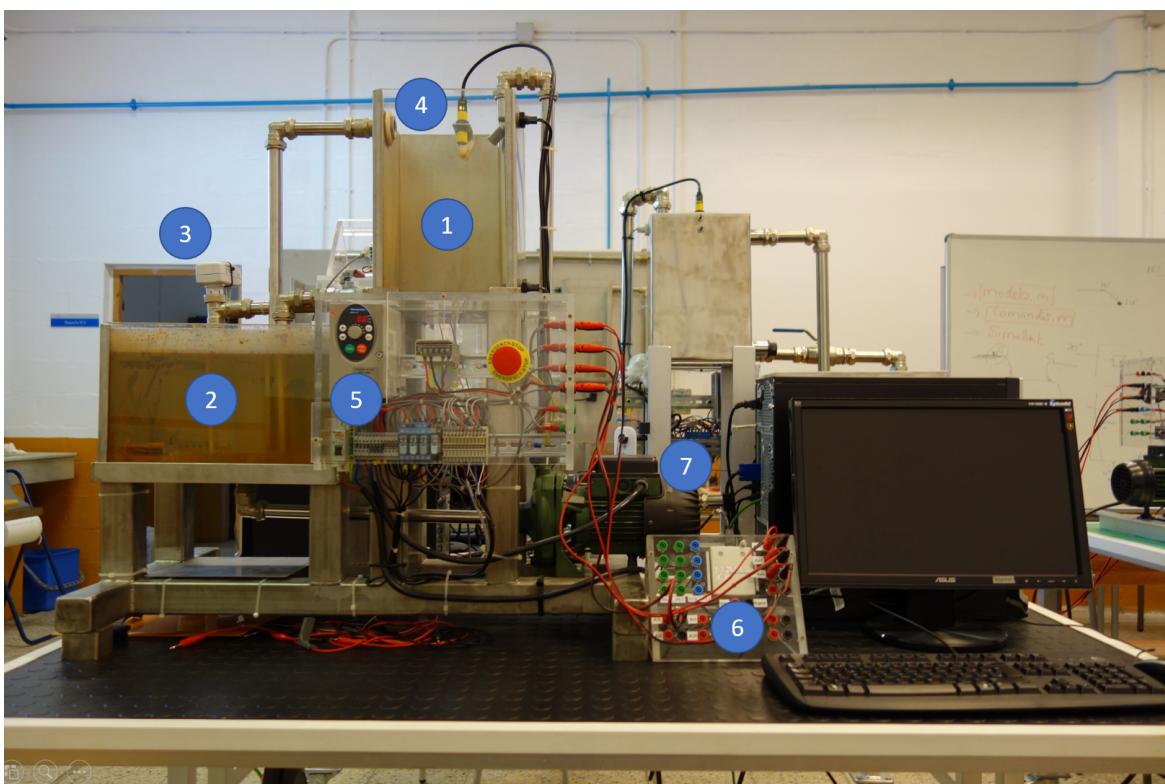


Figura 1.1 – Planta pequeña de nivel

Cómo se muestra en el imagen 1.1, la planta de nivel consta de dos depósitos. El depósito superior (1) es en el que se mantiene el nivel de agua, mientras que el depósito inferior (2) es el que lo alimenta. El nivel del depósito superior (1) es medido por el sensor de ultrasonidos (4) instalado en la parte superior del depósito. Para proteger la bomba y para evitar que el depósito inferior (2) se quede sin agua, es de mayor tamaño que el depósito superior (1). Para llenar

el depósito superior (1) se hace uso de una bomba (7). El régimen de giro de la bomba lo determina un variador de velocidad (5). El vaciado de éste depósito se puede realizar por tres métodos: a través del tubo superior, si el nivel del líquido es igual o superior a la capacidad del depósito, haciendo uso de la válvula manual o de la electroválvula (3) o mediante la propia bomba.

El intercambio de señales entre la planta y el sistema de control es realizado a través de una tarjeta de adquisición de datos (6). Las señales que se intercambian son:

- **Nivel medido por el sensor de ultrasonidos:** Planta a Sistema de control
- **Consigna de nivel:** Sistema de control a Planta
- **Señal de control para el variador:** Sistema de control a Planta
- **Señal de control para la electroválvula de vaciado:** Sistema de control a Planta

Por otra parte, el sistema emulado debe de ser una réplica del sistema real en comportamiento y en apariencia. Para replicar el comportamiento es necesario obtener la función de transferencia del sistema en los diferentes puntos de operación haciendo uso de algún método de identificación de sistemas. La apariencia es replicada haciendo uso del software Node-Red.

Se realiza un sinóptico de la planta real en el interface gráfico Node-Red. El usuario puede ver cuál es el nivel del depósito supeiror (1) y la frecuencia del variador (5), que se representan sobre el sinóptico.

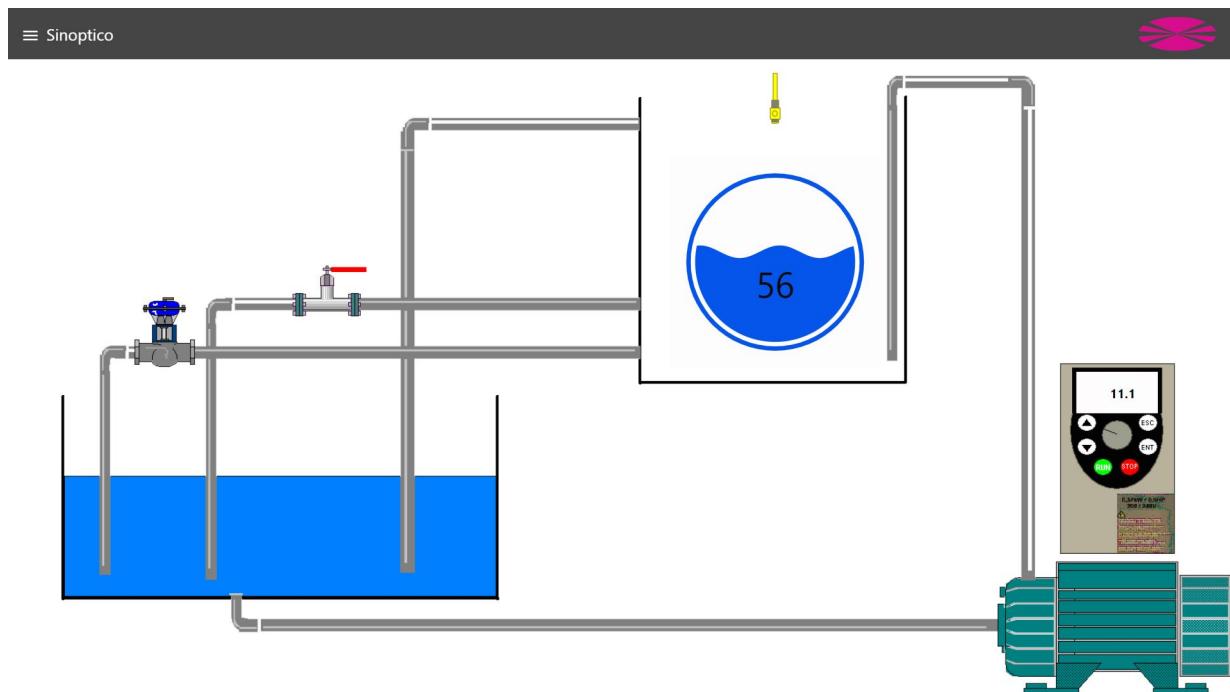


Figura 1.2 – Sinóptico Planta pequeña de nivel

Además, desde el interface gráfico se permite la introducción de la consigna (SP) de nivel y los parámetros típicos de un regulador PID: constante proporcional (K_p), tiempo de integración

(Ti) y tiempo derivativo (Td). El sistema emulado es sometido a la acción del regulador PID ajustado por el usuario. El interface gráfico tiene una pantalla de gráficas que permiten ver en el tiempo el nivel, el error, la señal de control, la consigna y la frecuencia del variador.

El interface gráfico se encuentra alojado en la Raspberry-Pi y se muestra en un servidor web que se accede mediante el navegador. El acceso a este servidor web, puede ser mediante un ordenador externo conectado en la misma red que la Raspberry-Pi mediante un cable RJ45 o mediante WiFi y también desde el navegador del propio sistema embebido.

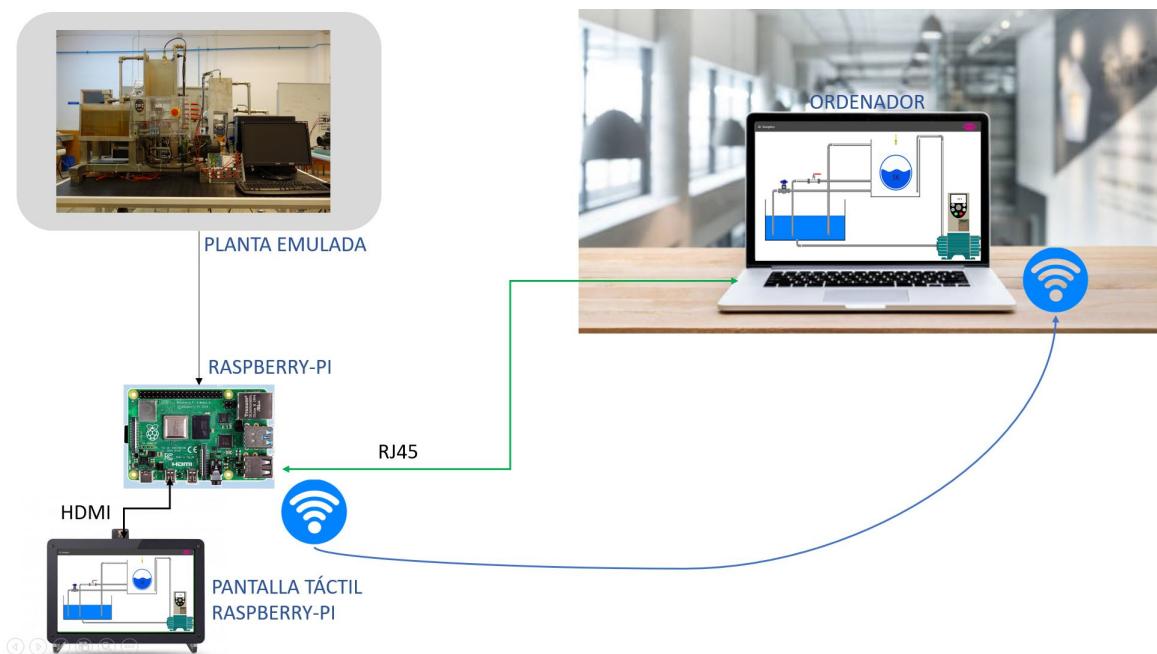


Figura 1.3 – Acceso a la emulación

Para replicar el funcionamiento de la planta real en la Raspberry-Pi es necesario seguir el siguiente procedimiento que se explica en las secciones siguientes:

- Estudiar el comportamiento de la planta real
- Comparar los diferentes métodos de identificación de sistemas
- Elegir el método de identificación que más se adecúa al sistema
- Obtener los parámetros de la función de transferencia del sistema a través del método elegido
- Comprobar que bajo la acción de un regulador PID, la planta real y la función de transferencia obtenida muestran una respuesta similar
- Implementar la función de transferencia en la Raspberry-Pi
- Desarrollar el interface gráfico en el Node-Red instalado sobre la Raspberry-Pi

2 REFERENCIAS

2.1. Bibliografía

- [1] DÍAZ DÍAZ, MIGUEL EDUARDO; *TFG 770G01A211: Identificación de la planta de laboratorio utilizando técnicas inteligentes*, Primera Edición, Ferrol, UDC, (2022).
- [2] MÉNDEZ BUSTO, DANIEL; *TFG 770G01A123: Control remoto sobre la planta pequeña de nivel del laboratorio*, Primera Edición, Ferrol, UDC, (2017).
- [3] OSCAR CALDAS FLAUTERO, SEBASTIÁN JIMÉNEZ GÓMEZ, EDILBERTO MEJÍA RUDA, JUAN HERNÁNDEZ MEJÍA, OSCARAVILÉS SÁNCHEZ *Identificación paramétrica en lazo cerrado de sistema de accionamiento neumático para cilindro de doble efecto*.

3 ESTRUCTURA GENERAL

El presente proyecto se estructura en tres bloques principales. Un bloque de identificación del sistema, un bloque de programación en lenguaje Python y un bloque de Node-Red.

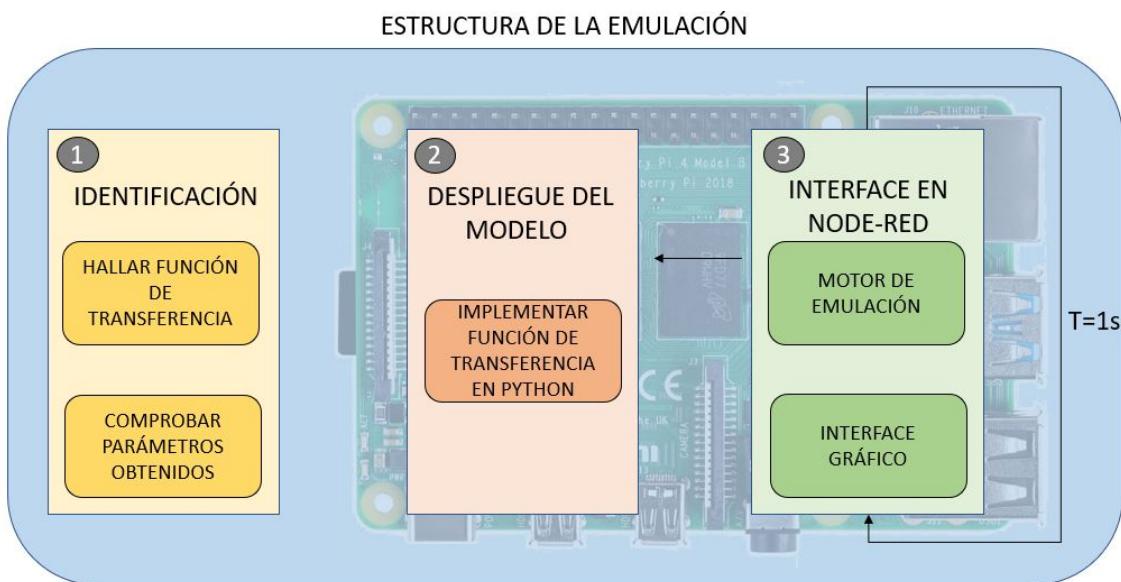


Figura 3.1 – Estructura General

El bloque de identificación del sistema se desarrolla en el software Matlab y tiene como objetivo obtener los parámetros que definen la función de transferencia del sistema en cada punto de operación. Esta planta tiene un rango de consigna de nivel que varía entre el 0 % y el 100 %. Por ello se obtienen funciones de transferencia para consignas del 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100 %.

Una vez obtenidos los parámetros, se comprueba que la respuesta de la función obtenida tiene el mismo comportamiento que la planta de nivel real. Para ello, se genera un script en Matlab que introduce la misma señal de control a la planta real y a la función de transferencia obtenida. Se grafican estas respuestas para comprobar que no hay diferencias entre ellas.

Tras comprobar que los parámetros obtenidos son correctos, se implementan las diez funciones de transferencia en el lenguaje de programación Python. Dependiendo de la consigna introducida por el usuario, se accede a una función de transferencia o a otra. Este fichero .py es llamado cíclicamente desde el motor de la emulación que está implementado con Node - Red sobre la Raspberry - Pi.

El software Node - Red tiene dos funciones en la emulación. Por una parte, es el motor de ejecución cíclica donde cada segundo se llama al fichero .py que contiene las funciones de transferencia del sistema y que devuelve el nivel de la planta emulada y la salida en frecuencia del variador emulado. Por otro lado, contiene el interface gráfico que permite mostrar el estado del sistema emulado, realizar gráficas e introducir consignas.

En los siguientes capítulos, se expone de forma más detallada cada uno de los elementos que componen la estructura general del presente trabajo.

4 IDENTIFICACIÓN

4.1. Introducción a los métodos de identificación

La identificación se basa en la obtención de un modelo matemático de un sistema dinámico haciendo uso de métodos empíricos. [1]

Dependiendo de la información que se conoce de la metodología seguida por el sistema de identificación para hallar las ecuaciones matemáticas que definen el comportamiento del sistema, se pueden clasificar en:

- **Black-box:** No se conoce la estructura interna del sistema de identificación. Se analiza el comportamiento de reacción de la salida con respecto a la entrada.
- **White-box:** El sistema de identificación es definido mediante leyes físicas. Los parámetros que lo componen son conocidos.
- **Gray-box:** Es una combinación de las dos anteriores. Contiene una parte conocida (white-box) y una parte desconocida (black-box).



Figura 4.1 – Tipos de métodos de identificación

Para realizar la identificación de un sistema tipo black-box, se necesita de una señal de excitación que se aplica en la entrada del sistema para provocar una respuesta en la salida y posteriormente analizar ambas señales.

- **Barrido senoidal:** Señal de tipo determinista de fácil generación y control. Se realiza un barrido de frecuencias en un rango determinado y durante un tiempo específico.

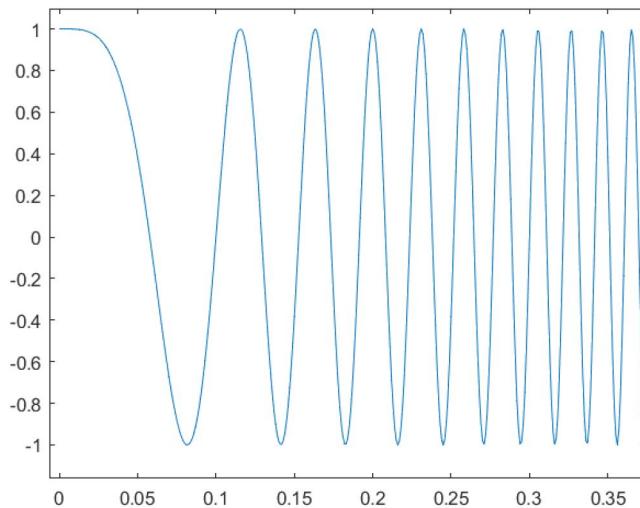


Figura 4.2 – Barido Senoidal

- **RBS:** Señal aleatoria de amplitud ajustable que asume dos valores. Es recomendada para sistemas lineales. En sistemas no lineales suele ser necesario trabajar con más de dos valores de entrada.

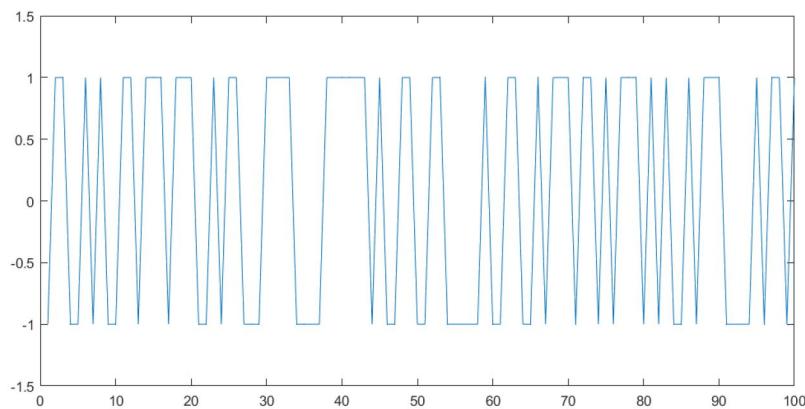


Figura 4.3 – Señal RBS

- **PRBS:** Señal periódica y determinista que posee características similares al ruido blanco. Varía entre dos valores teóricos (-1,1) que permiten ser escalados a valores deseados.

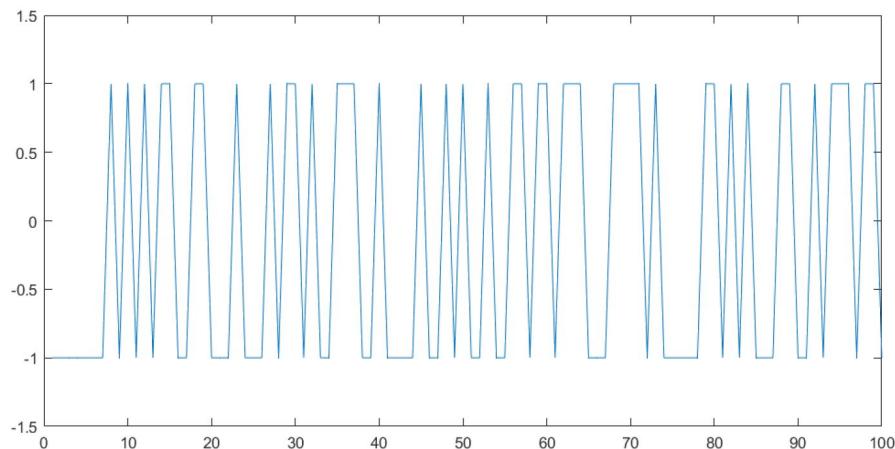


Figura 4.4 – Señal PRBS

Atendiendo al procedimiento empírico que se sigue para obtener las ecuaciones matemáticas que definen el comportamiento del sistema, se pueden clasificar en dos métodos:

- **Offline:** los datos de entrada y de salida son captados en el sistema real y posteriormente procesados para obtener los parámetros que definen el comportamiento del sistema.

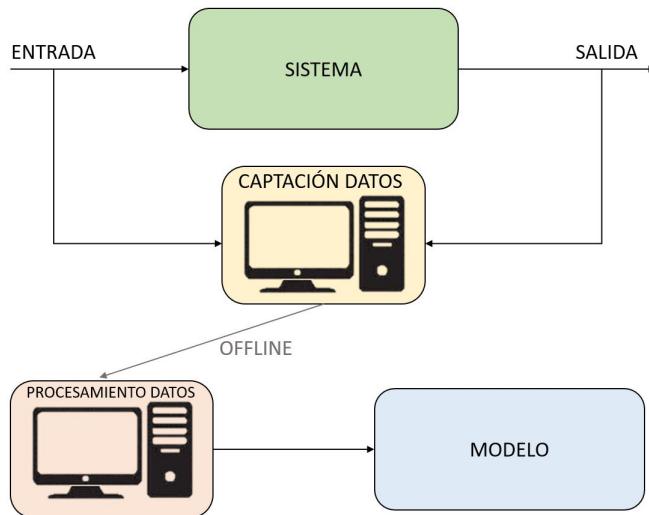
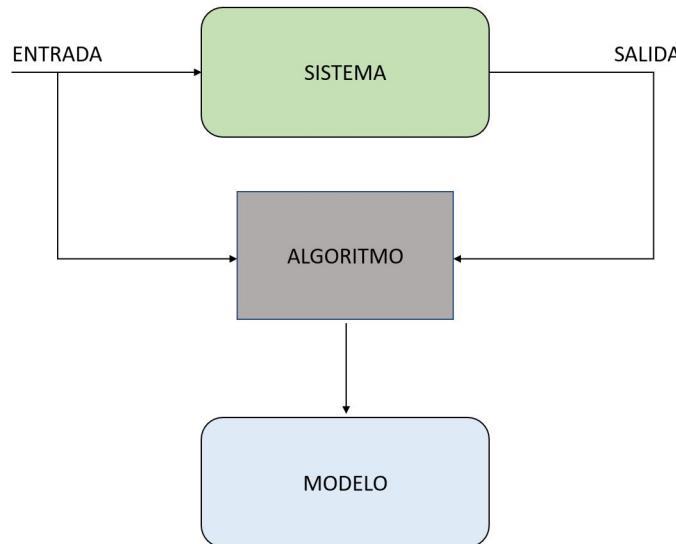


Figura 4.5 – Método identificación offline

- **Online:** a medida que se recogen los datos del funcionamiento del proceso, se evalúan y se obtienen los parámetros que definen el comportamiento del sistema

**Figura 4.6 – Método identificación online**

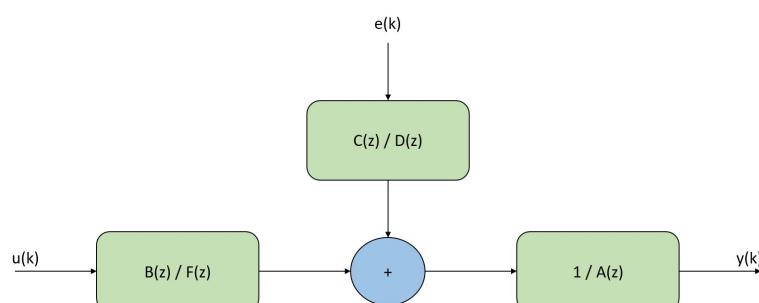
Los métodos offline se clasifican en:

- **Modelos paramétricos:** Emplean ecuaciones diferenciales y funciones de transferencia para describir el funcionamiento de los sistemas.
- **Modelos no parámetricos:** No emplean ecuaciones diferenciales ni funciones de transferencia para describir el funcionamiento del sistema. Suelen ser útiles para obtener información del sistema antes de aplicar la estimación paramétrica. Son más eficaces que los modelos paramétricos, pero son menos precisos.

Dentro de los modelos paramétricos destacan los modelos general-lineal. Para determinar la función de transferencia se emplea un modelo polinómico general-lineal que puede ser:

- **Determinista:** Especifica la relación entre la señal de entrada y de salida.
- **Estocástica:** Detalla cómo la perturbación aleatoria afecta a la señal de salida.

El flujo de señales que compone la función de transferencia se muestra en la imagen siguiente:

**Figura 4.7 – Modelo general-lineal**

Los métodos más representativos de los modelos general-lineal son:

- **ARX:** Este modelo incorpora una señal de estímulo. La función de transferencia estocástica y la determinista comparten el mismo conjunto de polos.

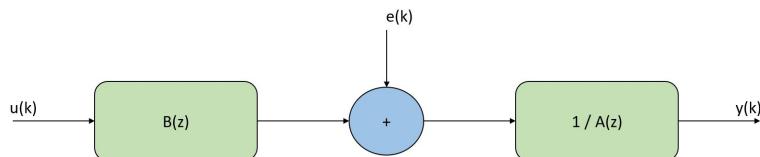


Figura 4.8 – Modelo ARX

- **ARMAX:** Es más flexible que el método ARX en el momento de modelar las perturbaciones.

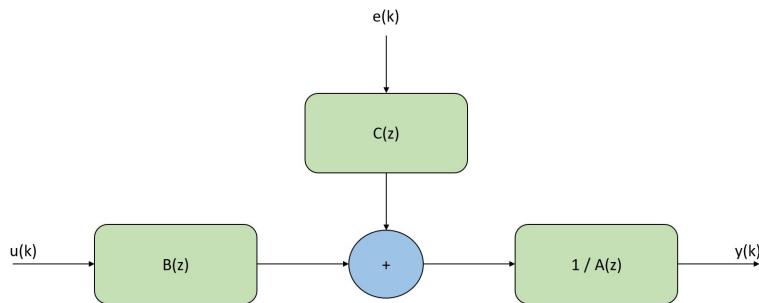


Figura 4.9 – Modelo ARMAX

- **Output-Error:** Describe la dinámica del sistema por separado de la dinámica estocástica. No emplea ningún parámetro para modelar las características de perturbación del sistema.

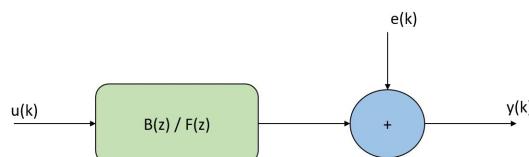


Figura 4.10 – Modelo Output-Error

- **Box-Jenkins:** Gran utilidad para perturbaciones tardías. Las características de las perturbaciones se encuentran separadas de la dinámica del sistema.

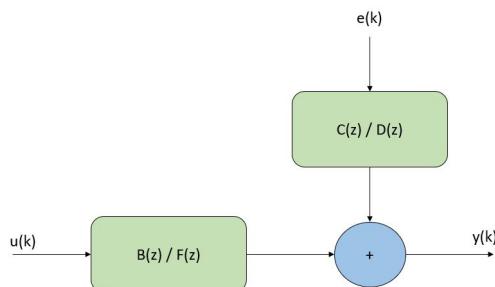


Figura 4.11 – Modelo Box-Jenkins

- **State-Space:** Proporciona una representación más completa del sistema. Es adecuado para sistemas con múltiples entradas y salidas. Tiene una simple configuración de parámetros (orden del modelo o número de estados del mismo).
- **Transfer Function:** Sólo se detalla la parte determinística de sistemas continuos o discretos.

Dentro de los modelos no paramétricos destacan:

- **Respuesta al impulso:** Se aplica una función impulso como entrada al sistema y se evalúa su respuesta.
- **Respuesta en frecuencia:** Se aplican señales senoidales de distintas frecuencias al sistema y se evalúa su respuesta.

En la identificación online destacan los métodos recursivos. Estos métodos se caracterizan por utilizarse en sistemas de detección de fallos y por ser una parte central de los sistemas adaptativos.

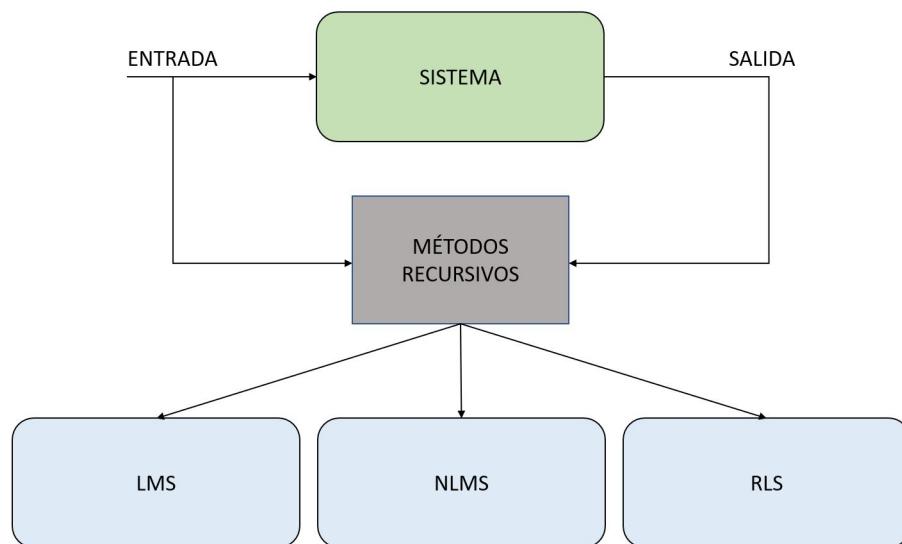


Figura 4.12 – Métodos recursivos

Los métodos recursivos más comunes son:

- **Least Mean Square (LMS):** Algoritmo de búsqueda ampliamente empleado debido a su baja complejidad computacional. Se divide en filtrado y adaptación. En la etapa de filtrado se realiza el cálculo de la salida y el error existente entre la salida deseada y la calculada. En la etapa de adaptación se ajustan los pesos del filtro empleando el error obtenido en la primera etapa.
- **Normalized Least Mean Square (NLMS):** Soluciona la lenta convergencia del LMS. En este caso el vector de entrada se normaliza. Requiere de complejidad computacional.
- **Recursive Least Square (RLS):** Tasa de convergencia más rápida que los modelos basados en LMS. Requiere de complejidad computacional. Necesita de datos de inicialización que pueden estimarse con k medidas o pueden ser inicializados.

4.2. Aplicación del método de identificación

Para realizar la identificación de la planta se hace uso de un método de identificación en lazo cerrado. Al ser en lazo cerrado el sistema presenta realimentación de la variable del proceso y se encuentra sometido a la acción de un controlador.

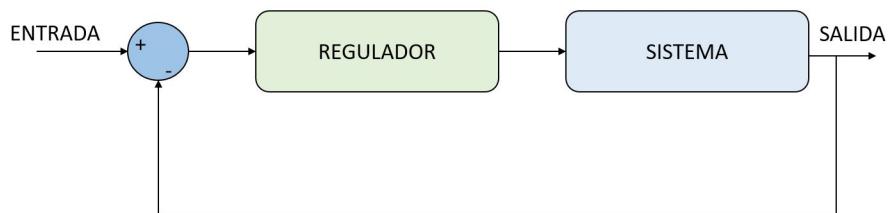


Figura 4.13 – Sistema en lazo cerrado

El método seleccionado es un método offline de tipo black-box. La señal de excitación seleccionada es la señal PRBS. Esta señal es una señal periódica y determinista que varía entre los valores -1 y 1. Esta señal es introducida en el sistema sumándola a la señal de control producida por el controlador. Esta señal es multiplicada por 5 para que cause un mayor efecto en el sistema.

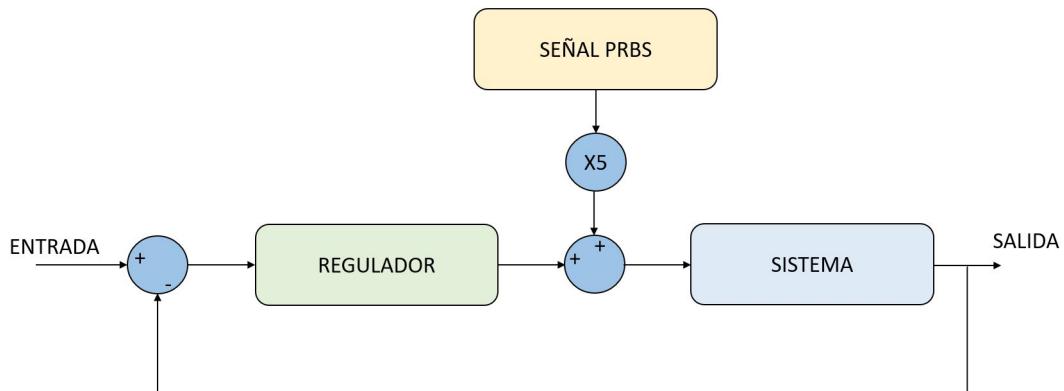


Figura 4.14 – Señal PRBS introducida al sistema

El sistema se somete a la acción de la suma de la señal de control y de la señal PRBS y se almacenan los datos de entrada (señal de control + señal PRBS) y de salida (nivel) en una matriz de Matlab. Haciendo uso de la instrucción ident de Matlab se realiza la aproximación al modelo paramétrico lineal ARX de segundo grado. El modelo devuelve los parámetros de la función de transferencia que define el sistema. Éste procedimiento se repite para cada punto de operación, obteniendo así 10 funciones de transferencia.

En el siguiente esquema se muestran los pasos que se siguen para obtener los parámetros de la función de transferencia.

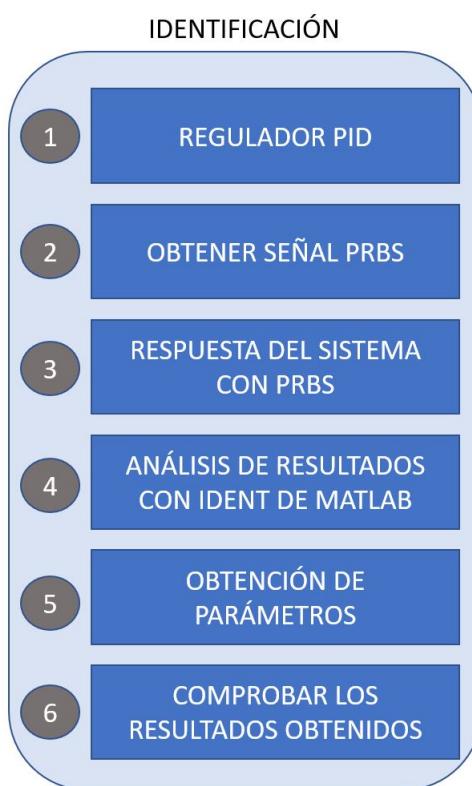


Figura 4.15 – Esquema Identificación

4.3. Regulador PID

La función del regulador PID en la presente aplicación es la de estabilizar el sistema en el punto de operación en el que se desea obtener la función de transferencia.

El regulador PID discretizado se obtiene analíticamente mediante la resolución de la ecuación diferencial del PID estándar.

$$u(t) = K_p \left(e(t) + \frac{1}{Ti} \int e(t) dt + Td \frac{de(t)}{dt} \right) \quad (4.1)$$

Dónde:

- $u(t)$: Señal de control expresada en tanto por ciento.
- K_p : Constante proporcional
- $e(t)$: Error
- Ti : Tiempo de integración
- Td : Tiempo derivativo

La ecuación en diferencias del algoritmo PID tiene la siguiente forma:

$$u(k) = K_p \left(e(k) + \frac{T}{Ti} \sum_{k=1}^{\infty} \frac{e(k-1) + e(k)}{2} + \frac{Td}{T} (e(k) - e(k-1)) \right) \quad (4.2)$$

Dónde:

- $u(k)$: Señal de control expresada en tanto por ciento.
- K_p : Constante proporcional
- $e(k)$: Error
- $e(k-1)$: Error en el instante $k-1$
- Ti : Tiempo de integración
- Td : Tiempo derivativo

El regulador PID se compone de la acción proporcional, de la acción integral y de la acción derivativa. [2]

La acción proporcional se encarga de modificar principalmente la ganancia del sistema. Se emplea para disminuir el error, sin embargo ésto puede aumentar las oscilaciones. Esta acción modifica el régimen transitorio y permanente de la respuesta del sistema.

La acción integral se encarga de anular el error en el régimen permanente de la respuesta del sistema. El uso de esta acción aumenta en 1 grado el tipo del sistema, por lo que aumenta la precisión del mismo con el riesgo de hacerlo inestable. Esta acción introduce un polo en el

origen, eliminando el error de posición. Modifica principalmente el régimen permanente de la respuesta del sistema.

La acción derivativa se encarga de modificar el comportamiento transitorio de la respuesta del sistema con el fin de modificar la velocidad y las oscilaciones de la respuesta. Esta acción introduce un cero en el sistema. Sin embargo, presenta inconvenientes como que amplifica las señales de alta frecuencia (ruido) que pueda presentar la señal, además de producir saturaciones en los actuadores cuando la señal de error varía bruscamente.

En la tabla 4.1 se muestran los parámetros que han sido utilizados para configurar el regulador PID en cada punto de operación. Estos parámetros han sido calculados en el desarrollo del TFG [2] y se han tomado como referencia. El tiempo de muestreo en todos los procedimientos es de un segundo ($T = 1s$). Se selecciona este tiempo de muestreo porque se garantiza que no es grande para perder información, ni es pequeño para no saturar los recursos del procesamiento.

SP (%)	Kp	Ti	Td
10	1.25	15	3.75
20	1.25	15	3.75
30	1.25	15	3.75
40	1.25	15	3.75
50	1.25	15	3.75
60	1.25	15	3.75
70	0.3	20	1
80	0.3	20	1
90	0.3	20	1

Tabla 4.1 – Parámetros regulador PID en cada punto de operación

4.4. Generar Señal PRBS

Para obtener la señal PRBS en Matlab, se hace uso de la instrucción *idinput*. Esta instrucción genera una señal PRBS con el número de elementos que configura el usuario. El número de elementos mínimo que se deben de configurar para obtener una señal PRBS óptima viene dada por la siguiente ecuación. [3]

$$T_s < T_m * N \quad (4.3)$$

Dónde:

- **Ts:** Tiempo de subida del sistema
- **Tm:** Tiempo de muestreo
- **N:** Número de bits para generar la señal

Para obtener el tiempo de subida del sistema, se introduce una entrada escalón de máxima amplitud (100 %). El tiempo de muestreo empleado es de 1 segundo.

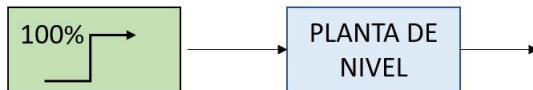


Figura 4.16 – Esquema Entrada Escalón

Se despeja la N de la ecuación 4.4:

$$N = \frac{T_s}{T_m} = \frac{10}{1} = 10 \quad (4.4)$$

Por lo que el número de elementos mínimo que se deben de utilizar son:

$$n = 2^{N-1} = 2^9 = 512 \quad (4.5)$$

En el presente proyecto se hace uso de señales PRBS de 3600 elementos.

El script de Matlab para almacenar la respuesta del sistema a la acción de la presente señal tiene un periodo de ejecución de un segundo y se ejecuta hasta evaluar todos los elementos que componen la señal PRBS. La duración de cada análisis en cada punto de operación es de una hora.

```
Original_PRBS=idinput(3600);
```

La señal PRBS obtenida multiplicada por cinco se muestra en la figura 4.17.

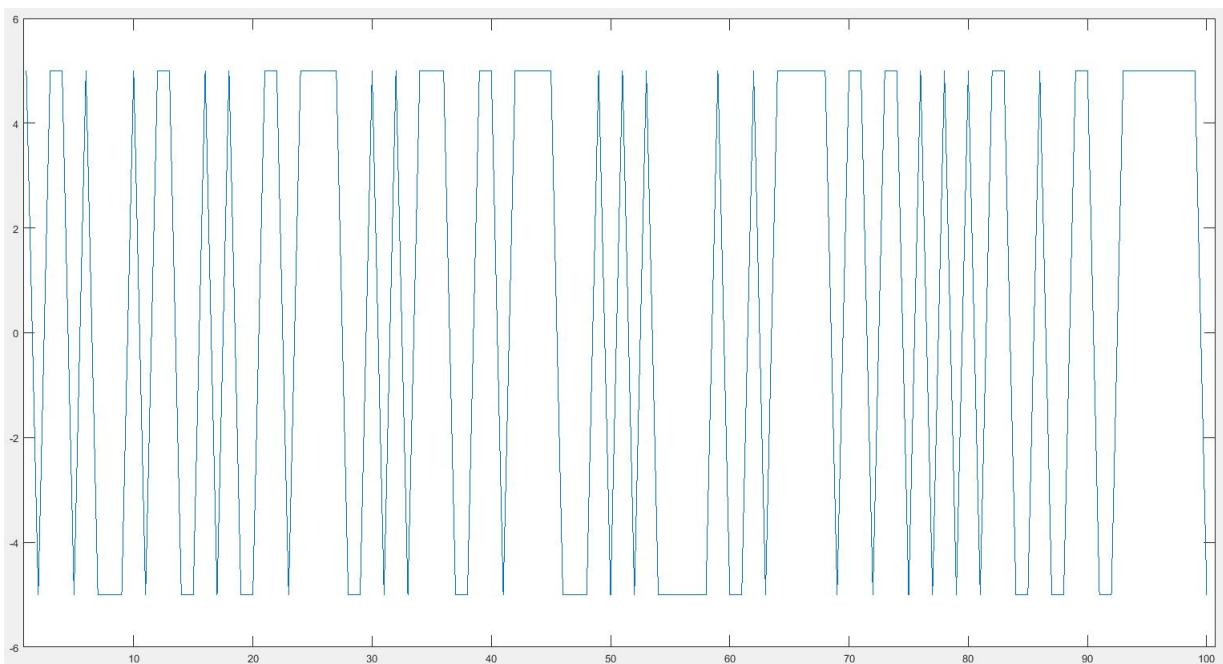


Figura 4.17 – Señal PRBS multiplicada por cinco

4.5. Respuesta Sistema a señal PRBS

Durante una hora se somete la planta real a la acción de la señal del control del regulador PID con la suma de la señal PRBS multiplicada por cinco y se guarda en una matriz de Matlab la señal que se aplica al sistema y la respuesta del mismo.

Para realizar esto se realiza un scrpit de Matlab para cada punto de operación y que presenta la siguiente estructura:

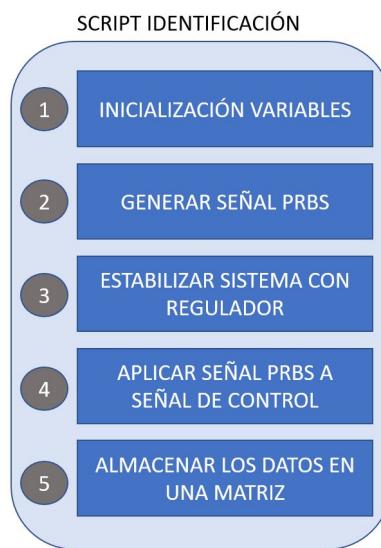


Figura 4.18 – Esquema Script Identificación

En primer lugar se inicializan las variables que son utilizadas en la ejecución del script. Estas variables se pueden clasificar en dos grupos. Un grupo de variables de proceso y otro grupo de variables de configuración. Estas variables se pueden encontrar en el anexo de Variables Scripts/Identificación (página 113).

A continuación se genera la señal PRBS haciendo uso de la instrucción explicada en el apartado Señal PRBS (página 36) y se multiplica por cinco. Tras esto se genera un bucle que realiza un número de iteracciones igual al número de elementos de la señal PRBS sumado al tiempo de estabilización. En este bucle se ejecuta el regulador PID y cuando el sistema está estabilizado se le suma la señal PRBS multiplicada por cinco a la señal de control del regulador. Se determina que el sistema está estabilizado cuando pasa el tiempo configurado en el parámetro de tiempo de estabilización. Cuando el número de iteracciones supera el valor del parámetro de configuración de tiempo para guardado de datos, se comienza con el almacenamiento de los valores de entrada y salida del sistema en la ruta especificada.

Tras realizar esto se grafican los valores almacenados en la matriz. Para un punto de operación de 50 se obtienen los resultados mostrados en la figura 4.19.

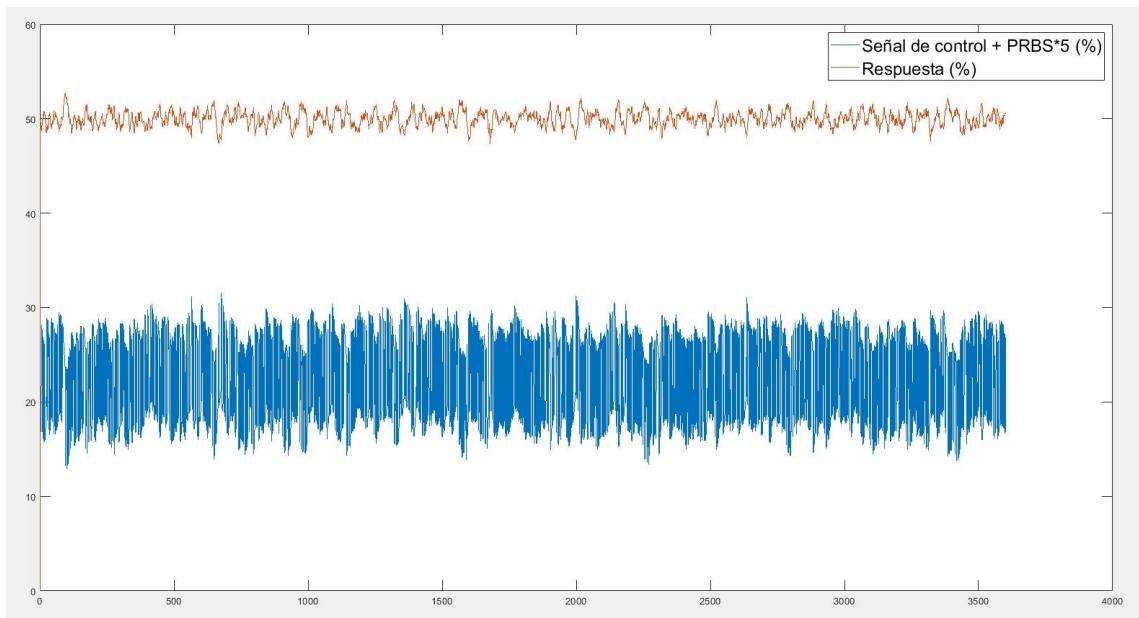


Figura 4.19 – Respuesta a PRBS en punto de operación 50

NOTA: Los resultados obtenidos en el resto de puntos de operación se pueden encontrar en https://github.com/Danielmendezb/TFM_Planta_Emulada/tree/main/2_Resultados/2_0_Identificación_Datos_PRBS

4.6. Obtención de parámetros

Para obtener los parámetros que forman la ecuación que describe la función de transferencia del sistema se hace uso de la Toolbox *ident* de Matlab. Esta herramienta permite hallar los parámetros discretos partiendo de la respuesta del sistema al someterlo a la señal PRBS.

Para ejecutar esta Toolbox se debe de escribir *ident* en la consola de comandos de Matlab. Al ejecutarla se muestra la ventana que se muestra en 4.20.

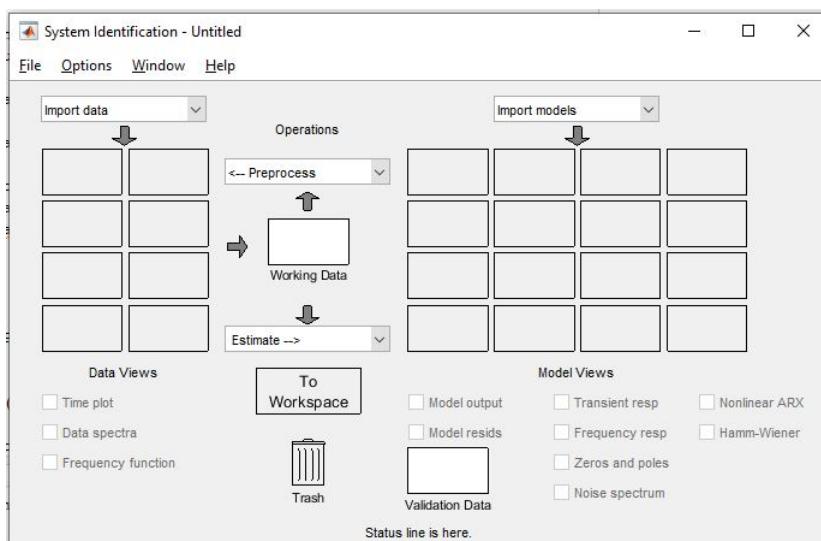


Figura 4.20 – Ident: Ventana principal

En el desplegable de la esquina superior derecha se selecciona *Time domain data* y se muestra la ventana de configuración de la imagen 4.21.

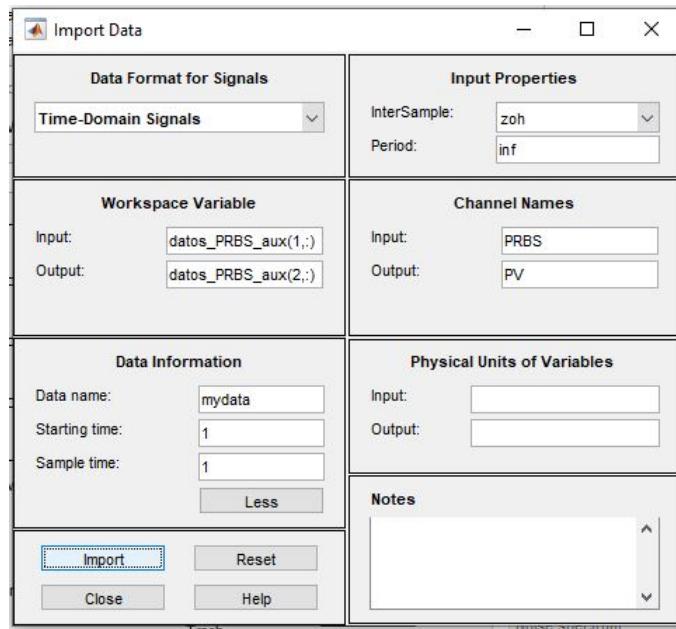


Figura 4.21 – Ident: Configuración de datos para análisis

En esta ventana, en el apartado de *Workspace Variable* se tienen que seleccionar los datos de entrada y salida. En este caso, los datos de entrada es la señal PRBS multiplicada por 5 y sumada a la señal de control del regulador PID y los datos de salida es la respuesta del sistema. Tras configurar estos datos se debe de pulsar en *Import*.

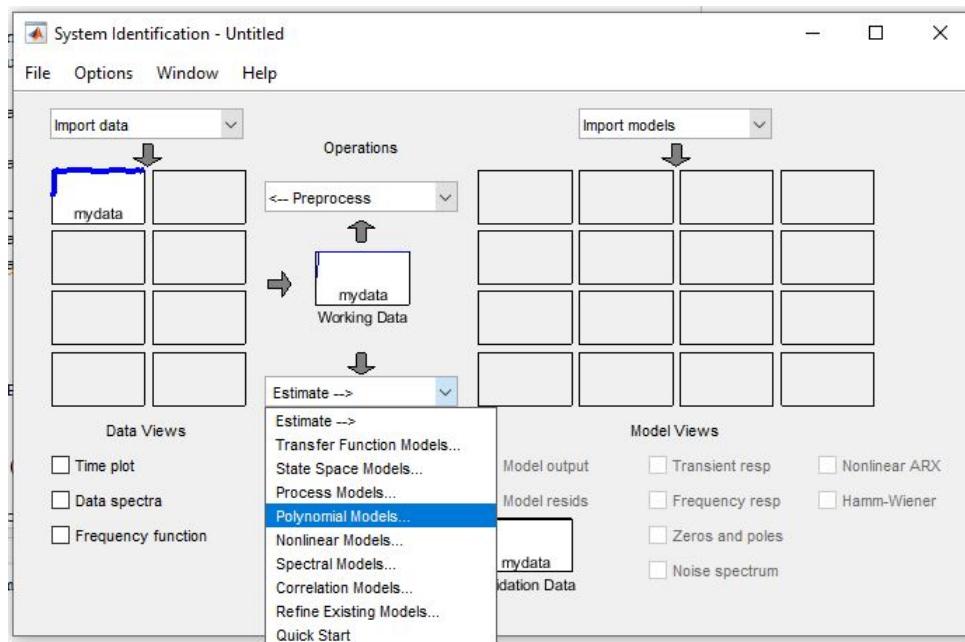


Figura 4.22 – Ident: Selección de método de estimación

Al seleccionar "Polynomial Models" se muestra la ventana de la figura 4.23.

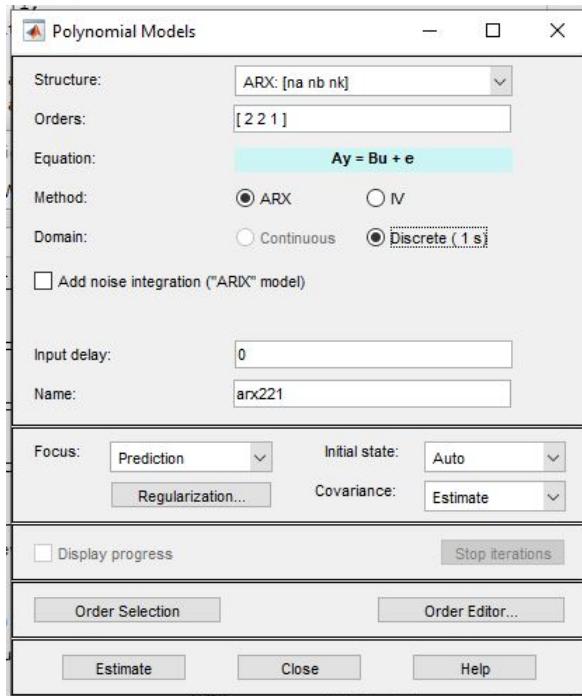


Figura 4.23 – Ident: Ventana principal modelo polinómico

En el apartado de *Orders* se configura un polinomio de orden dos y se pulsa en *Estimate*.

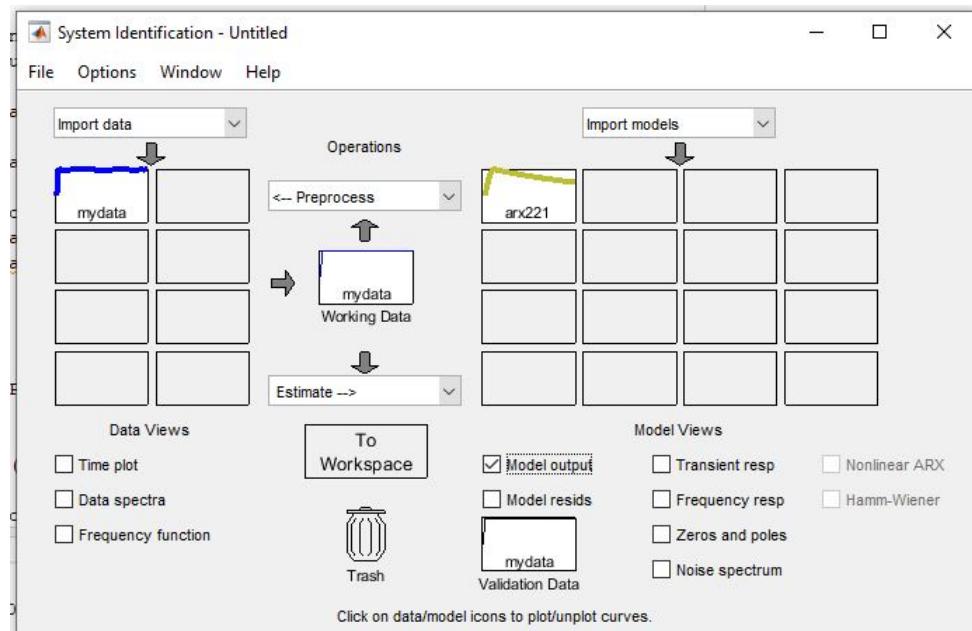


Figura 4.24 – Ident: Estimación

Al seleccionar en *Model Output* se muestra el resultado de comparar la estimación del sistema con los datos reales.

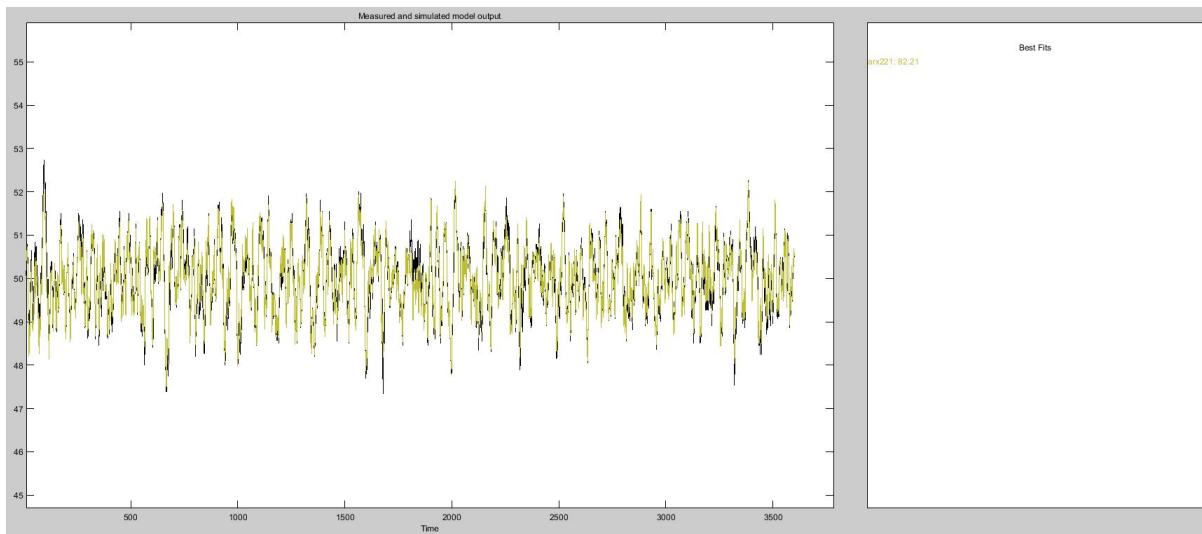


Figura 4.25 – Ident: Resultado estimación

Este proceso se repite para todos los puntos de operación y se obtienen los siguientes resultados de estimación.

SP (%)	Estimación (%)
10	18.67
20	23.29
30	41.69
40	67.70
50	82.21
60	87.26
70	87.89
80	90.60
90	-200.23

Tabla 4.2 – Estimación obtenida en cada punto de operación

NOTA: Los resultados obtenidos en el resto de puntos de operación se pueden encontrar en https://github.com/Danielmendezb/TFM_Planta_Emulada/tree/main/2_Resultados/2_0_Identificación

Como se puede observar en la tabla anterior, en valores de consigna inferior al 30 % se obtienen resultados bajos debido a las inercias del sistema. Para una consigna del 90 % se obtiene un valor erróneo debido a que el sistema satura. El rango funcional de la planta emulada será para consignas comprendidas en el rango del 30 % al 90 %.

Para exportar los resultados obtenidos con esta aproximación al espacio de trabajo de Matlab, se tiene que arrastrar el resultado a la ventana de *Workspace*.

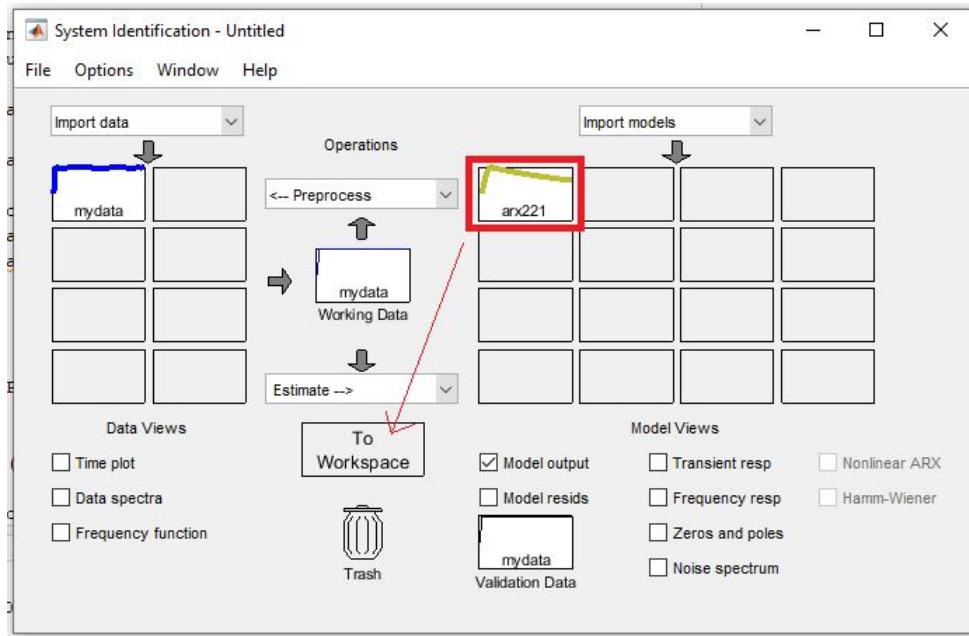


Figura 4.26 – Ident: Estimación a espacio de trabajo

Al realizar esto, se añade al espacio de trabajo una variable denominada *arx221*.

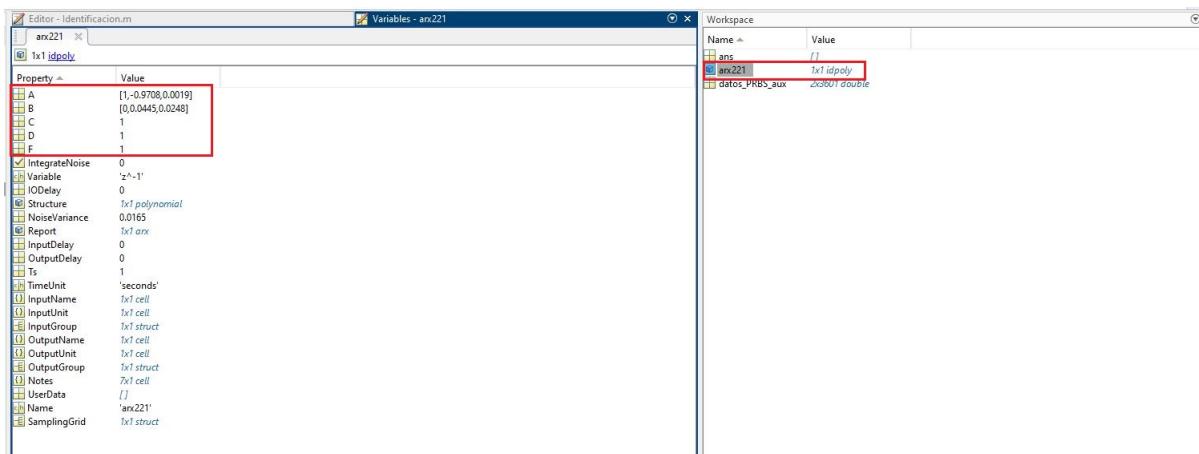


Figura 4.27 – Variable arx221

En esta variable hay cuatro elementos que forman parte de la función de transferencia del sistema A, B, C y D. El modelo arx que se ha utilizado sigue la estructura mostrada en la imagen 4.28 para formar la función de transferencia.

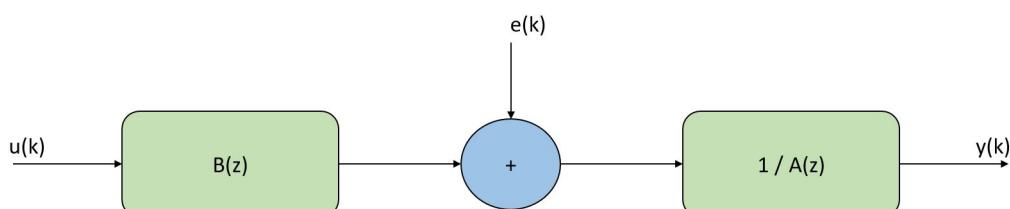


Figura 4.28 – Estructura modelo arx

Los polos de la función de transferencia se encuentran en la componente A y los ceros en la componente B, por lo tanto la función de transferencia discreta es definida por la siguiente ecuación.

$$F.T(k) = \frac{B}{A} = \frac{B_1 z^{-2} + B_2 z^{-1} + B_3}{A_1 z^{-2} + A_2 z^{-1} + A_3} \quad (4.6)$$

Las funciones de transferencia discretas obtenidas para los diferentes puntos de operación haciendo uso de la ecuación 4.6 han sido:

SP (%)	Función Transferencia (z)
30	$\frac{0,0445z^{-1}+0,0248}{z^{-2}-0,9708z^{-1}+0,0019}$
40	$\frac{0,0444z^{-1}+0,0261}{z^{-2}-0,9601z^{-1}+0,0029}$
50	$\frac{0,0445z^{-1}+0,0248}{z^{-2}-0,9708z^{-1}+0,0019}$
60	$\frac{0,045z^{-1}+0,0229}{z^{-2}-0,9765z^{-1}+0,0035}$
70	$\frac{0,0455z^{-1}+0,0226}{z^{-2}-0,9772z^{-1}+0,0018}$
80	$\frac{0,0452z^{-1}+0,0215}{z^{-2}-0,9815z^{-1}+0,0038}$

Tabla 4.3 – Función de Transferencia en cada punto de operación

4.7. Comprobación

Una vez obtenidas las funciones de transferencia en los diferentes puntos de operación, se debe de comprobar si la respuesta que muestran es similar a la respuesta de la planta real. Para ello se realiza un script de Matlab en el que se estabilizan el sistema real y el sistema emulado haciendo uso de un regulador PID. Tras la estabilización, el sistema se somete a cambios en la consigna y se grafican las respuestas de ambos sistemas. Este análisis se realiza para cada punto de operación. El script de Matlab de comprobación, tiene la estructura mostrada en la imagen 4.29.



Figura 4.29 – Esquema Script Comprobación

En primer lugar se inicializan las variables que son utilizadas en la ejecución del script. Estas variables se pueden clasificar en dos grupos. Un grupo de variables de proceso y otro grupo de variables de configuración. Estas variables se pueden encontrar en el anexo de Variables Scripts/Comprobación(página 114).

A continuación se genera un bucle de iteracciones infinitas y que se ejecuta hasta que el usuario realiza la salida por teclado (tecla q). En el interior de este bucle se implementa el sistema emulado siguiendo la ecuación 4.6 con los parámetros que se muestran en la tabla 4.3 y se somete a la acción del regulador PID. El sistema real es sometido a la acción de un regulador PID de las mismas características. Pasados dos minutos el sistema se encuentra estabilizado y se generan escalones de valor mas/menos 5 sobre la consigna con una duración de 40 segundos por escalón. Se grafica la respuesta de ambos sistemas y se obtiene el siguiente resultado para un punto de operación de 50.

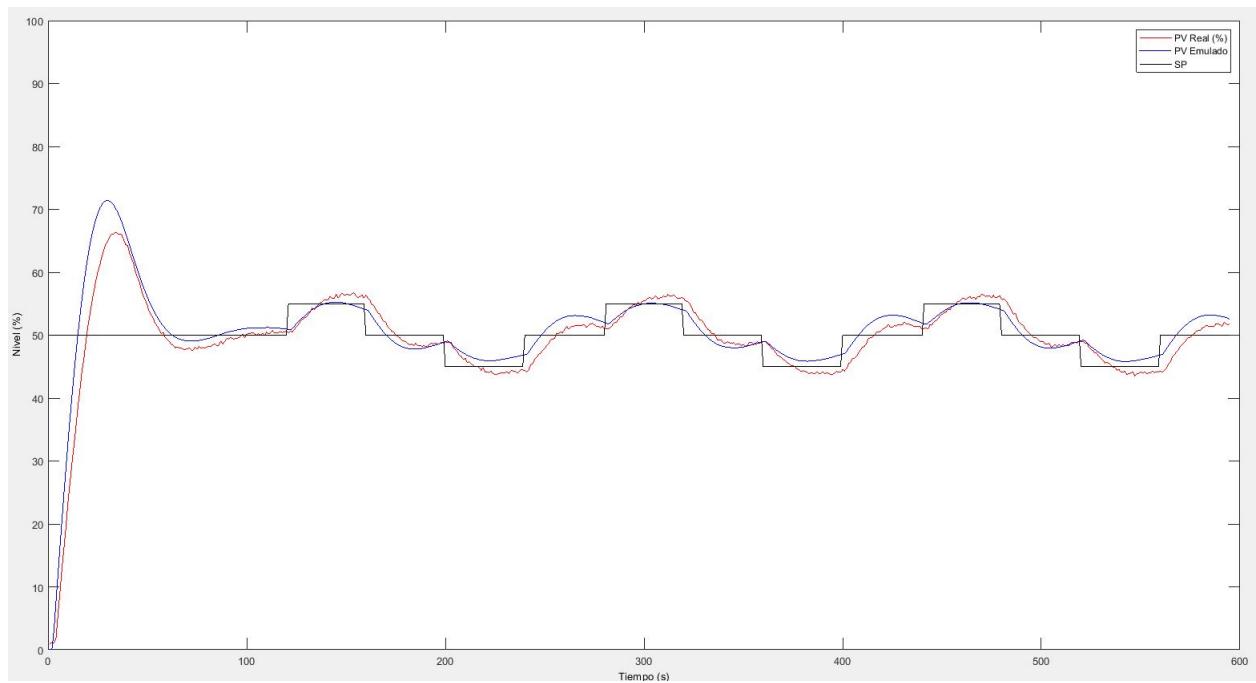


Figura 4.30 – Respuesta Sistema Emulado

NOTA: Los resultados obtenidos en el resto de puntos de operación se pueden encontrar en
https://github.com/Danielmendezb/TFM_Planta_Emulada/tree/main/2_Resultados/2_1_Comprobacion

5 DESPLIEGUE DEL MODELO

En el presente capítulo se desarrolla la implementación de las funciones de transferencia obtenidas en el proceso de identificación. La implementación se realiza en una Raspberry-Pi, a través de un fichero de Python. Este fichero es llamado cíclicamente para su ejecución por el motor de la emulación.

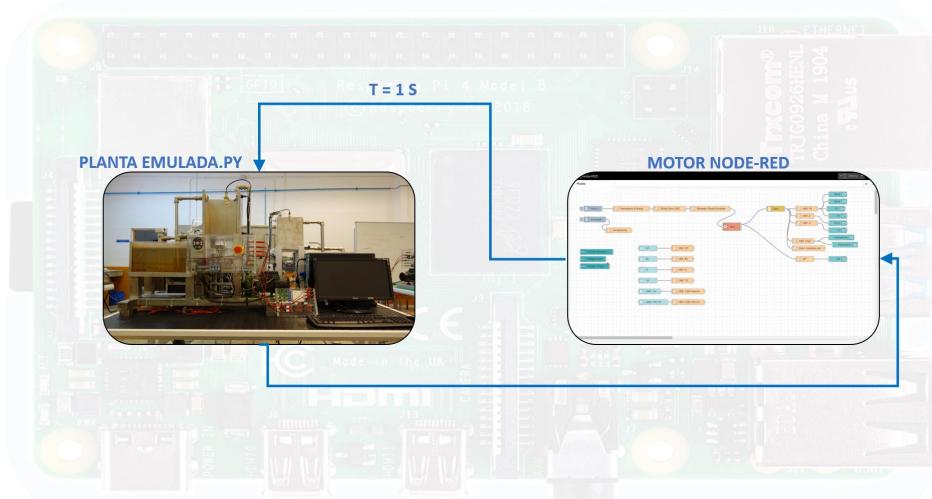


Figura 5.1 – Llamada cíclica a planta emulada

En el interior de este fichero se gestiona el intercambio de señales con el motor de la emulación, la implementación de las funciones de transferencia de la planta a emular y la implementación del regulador PID que actúa sobre la planta emulada.

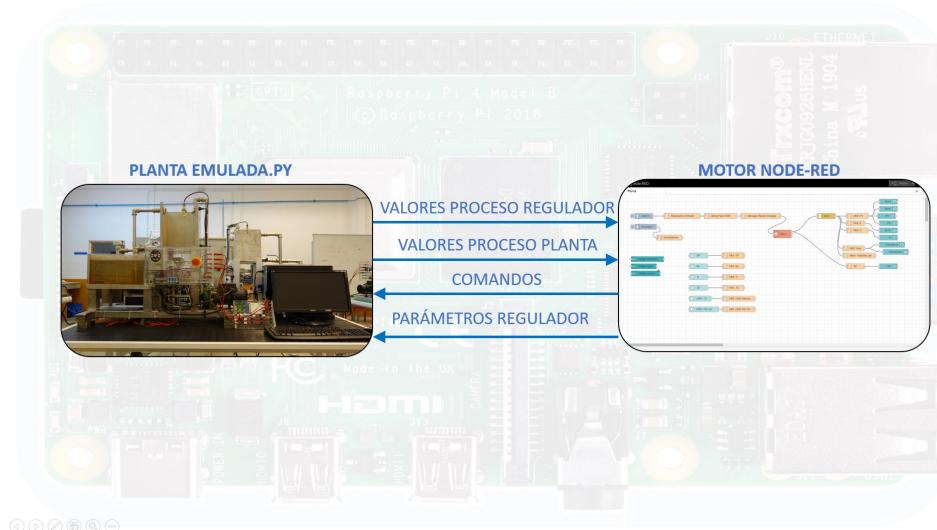


Figura 5.2 – Intercambio de señales

Las señales de entrada que recibe el fichero desde el motor de la emulación son:

- **Comandos:** Orden de marcha/paro del sistema emulado y orden de habilitar/deshabilitar acción del regulador. Ésta orden es emitida desde el interface gráfico.
- **Parámetros del regulador PID:** Parámetros de ajuste del regulador PID configurados por el usuario en el interface gráfico.

Las señales de salida que se envían desde el fichero al motor de la aplicación son:

- **Variables Proceso Planta:** Nivel y frecuencia de giro del variador calculados por el sistema emulado.
- **Variables Proceso Regulador:** Valores obtenidos a través del regulador PID como la señal de control, el error...

El fichero presenta la siguiente estructura:

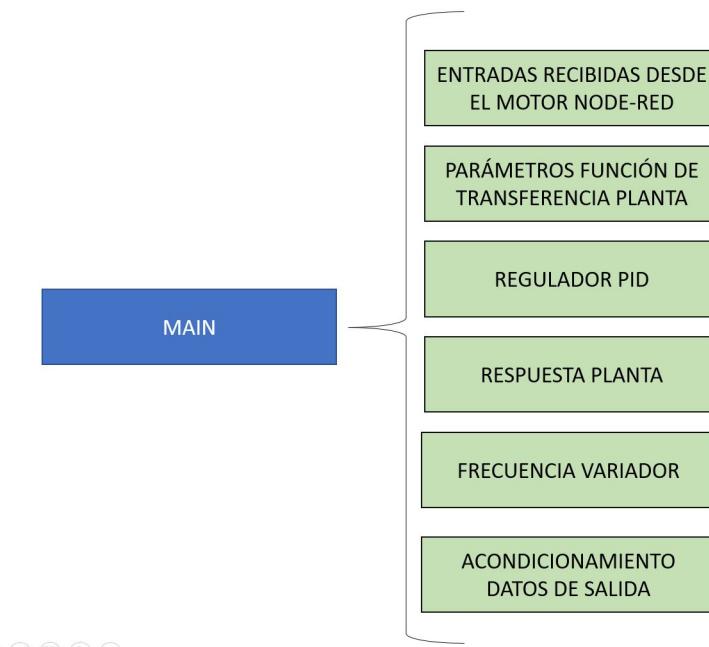


Figura 5.3 – Estructura fichero

5.1. Función Main

La función principal tiene el objetivo de realizar la llamada al resto de funciones del fichero. Se ejecuta cíclicamente por la llamada del motor de la emulación. Las funciones que son llamadas son:

- **Entradas recibidas desde el motor node-red**
- **Parámetros función de transferencia planta**
- **Regulador PID**
- **Respuesta planta**

- **Frecuencia Variador**
- **Acondicionamiento datos de salida**

Desde esta función se realiza el envío de datos al motor de la emulación con la instrucción *print*. Los datos se envían en formato JSON que tiene la siguiente estructura:

$$DatoEnvio = \{ 'NombreDat1' : ValorDat1, 'NombreDat2' : ValorDat2, 'NombreDatN' : ValorDatN \}$$

El código en python de esta función se encuentra en [13.1.1](#).

5.2. Función Entradas recibidas desde node-red

Esta función recoge los valores que son enviados desde el motor de emulación, los convierte en reales y los almacena en variables globales de python. Permite recoger hasta 30 variables.

$$Variable = float(sys.argv[n])$$

La instrucción *sys.argv[n]* pertenece a la librería *sys* y que es necesario importar. Esta instrucción permite recoger el valor de la variable que ocupa la posición *n* en la llamada al fichero desde el motor de la emulación.

import sys

El código en python de esta función se encuentra en [13.1.2](#).

5.3. Función Parámetros Función de Transferencia Planta

Esta función asigna los polos y los ceros de la función de transferencia en función de la consigna seleccionada por el usuario en el interface gráfico. Los valores que se asignan a los polos y los ceros se encuentran en la tabla [4.3](#).

El código en python de esta función se encuentra en [13.1.3](#).

5.4. Función PID

Esta función implementa el regulador PID y genera la señal de control y el error de posición. Las ecuaciones que se siguen para implementar el regulador PID son [4.1](#), [4.2](#).

El código en python de esta función se encuentra en [13.1.4](#).

5.5. Función Respuesta Planta

Esta función implementa la función de transferencia de la planta con los parámetros asignados en la función de Parámetros Planta. Esta función genera el nivel de la planta. Para realizar esta implementación se hace uso de las ecuaciones [4.3](#).

El código en python de esta función se encuentra en [13.1.5](#).

5.6. Función Display Variador

Esta función calcula la frecuencia de giro del variador en función de la señal de control que es generada por la función PID. El regulador PID genera una salida en un rango entre el 0 % y el 100 % y el variador tiene un rango entre 0 Hz y 50 Hz.

El código en python de esta función se encuentra en [13.1.6](#).

5.7. Función Paso a formato JSON

Esta función realiza la conversión de *string* a formato JSON.

varJSON = json.dumps(varString)

La instrucción *json.dumps* está dentro de la librería *json* y que es necesario importar.

import json

El código en python de esta función se encuentra en [13.1.7](#).

6 INTERFACE EN NODE-RED

Node-Red es una herramienta de programación gráfica que simplifica las tareas de programación con respecto a otros lenguajes que requieren desarrollo de código. Es un editor de flujo basado en el navegador donde se puede añadir o eliminar nodos y conectarlos entre sí. En este sentido, facilita la conexión entre dispositivos de hardware, APIs y servicios de línea. Además permite realizar una interface gráfica gracias a nodos que se muestran graficamente en el "dashboard".

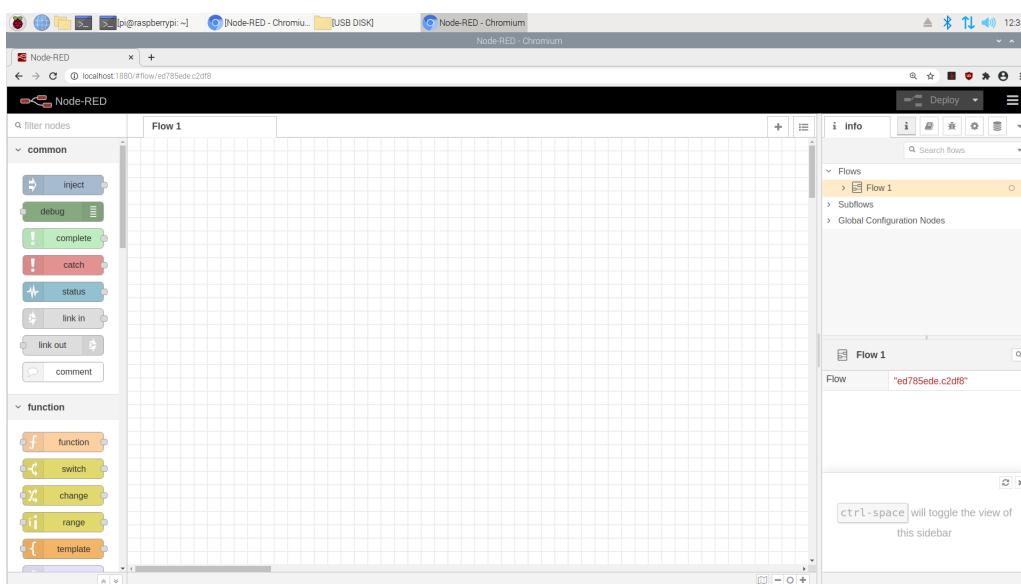


Figura 6.1 – Apariencia node-red

El flujo se transmite entre nodos compartiendo un mensaje. Atendiendo al ciclo de ejecución de los nodos se pueden distinguir cuatro grupos:

- **Nodos cíclicos:** Son nodos que se ejecutan cíclicamente con un periodo configurable. Se permite retrasar su llamada a través de un parámetro de tiempo configurable.
- **Nodos de arranque:** Son nodos que sólo se ejecutan al iniciar la aplicación. Estos nodos permiten retrasar su llamada a través de un parámetro de tiempo configurable.
- **Nodos función:** Son nodos llamadas desde los nodos cíclicos y los nodos de arranque. En ellos se programa la lógica de funcionamiento de la aplicación a través del mensaje recibido.
- **Nodos de programación:** Estos nodos permiten realizar programación en lenguaje html. No es necesario que sean llamados por un nodo cíclico o de arranque.

Dentro de los nodos función se pueden diferenciar nodos:

- **Sin Interface Gráfica:** Son nodos que no se representan graficamente en el dashboard. Se utilizan para realizar la lógica de la aplicación.
- **Con Interface Gráfica:** Son nodos que se representan graficamente en el dashboard. Se utilizan para representar variables de forma gráfica y para recibir órdenes y consignas desde el dashboard.

En el presente proyecto node-red tiene la función de ejercer de motor de la aplicación y la función de generar un interface gráfico que permite al usuario ver el estado del sistema e introducir consignas y comandos al sistema.

La estructura más común utilizada en node red se compone por un nodo cíclico, un nodo de arranque y los nodos función. El nodo de arranque tiene un tiempo de retardo inferior al tiempo de retardo configurado en el nodo cíclico para que se ejecute en primer lugar.

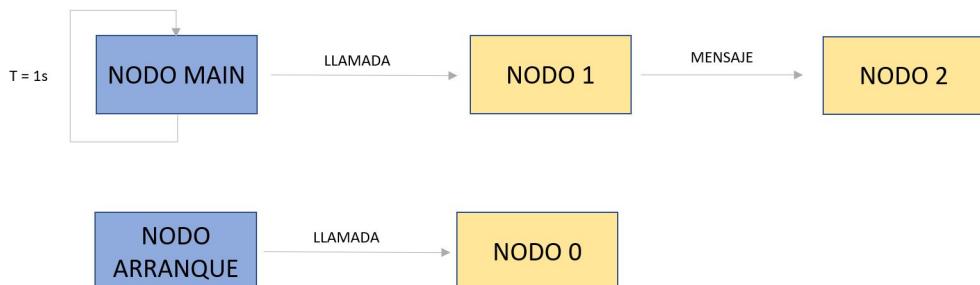


Figura 6.2 – Estructura básica node-red

La base de la arquitectura del presente proyecto es la estructura básica que se representa en la imagen anterior. En el siguiente esquema se representa la arquitectura implementada en Node-Red, en donde se representan los nodos y sus conexiones. Estos nodos se han representado con colores según su función:

- **Azul:** Nodos cíclicos
- **Amarillo:** Nodos función
- **Verde:** Nodos programación
- **Cian:** Nodos con interface gráfico que envian datos desde la visualización a Node-Red
- **Cian Oscuro:** Nodos con interface gráfico que reciben los datos desde el Node-Red.

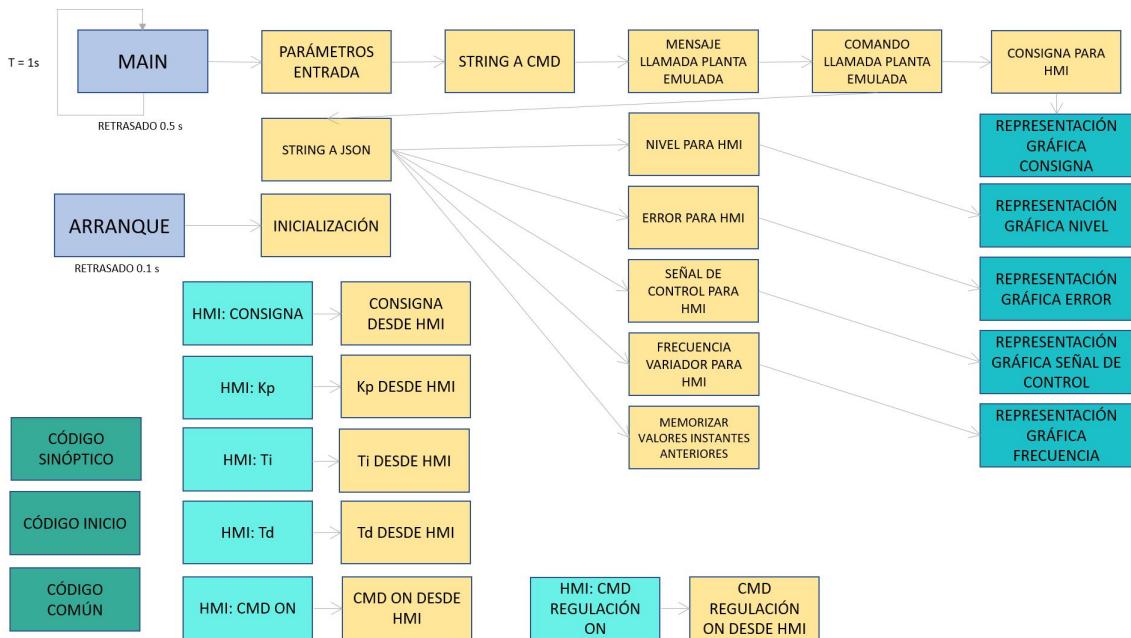


Figura 6.3 – Arquitectura planta de nivel emulada

En las siguientes subsecciones se explica la función de cada uno de los nodos que componen la arquitectura.

6.1. Motor de la aplicación

6.1.1. Nodo Cíclico Main

El nodo cíclico main se configura como nodo temporal (*timestamp*) en intervalos de un segundo. Se retrasa 0.5 segundos en el arranque para que se ejecute en primer lugar el nodo de arranque. Es el primer nodo en generar el mensaje de transmisión mediante la instrucción *msg*. El nombre que se le decide asignar a este mensaje es el de *payload*.

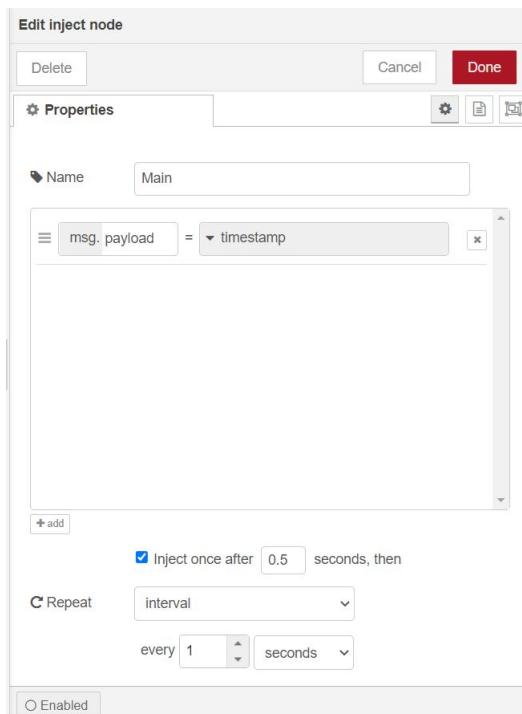


Figura 6.4 – Node-Red: Nodo Main

6.1.2. Nodo Arranque

El nodo de arranque sólo se ejecuta una vez al arrancar la aplicación. Se configura como nodo temporal (*timestamp*) sin repetición. Se retrasa 0.1 segundos en el arranque para garantizar que es el primer nodo en ser llamado. Este nodo llama a un nodo función en el que se inicializan las variables de proceso.

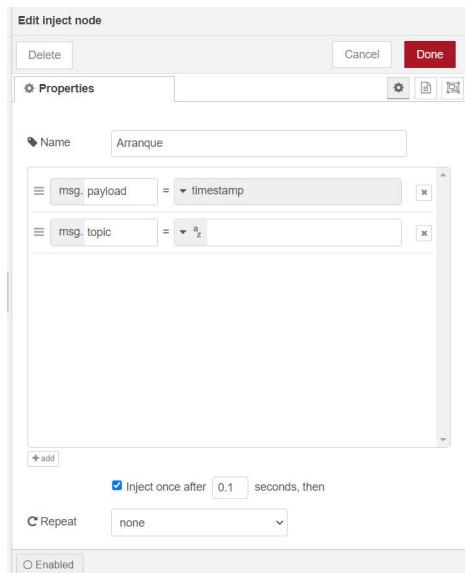


Figura 6.5 – Node-Red: Nodo Arranque

6.1.3. Nodo Función Inicialización

Este nodo es llamado por el nodo de arranque. En el interior de este nodo se inicializan las variables globales de proceso con el valor 0. Para ello se hace uso de la instrucción:

```
global.set('variable', valor)
```

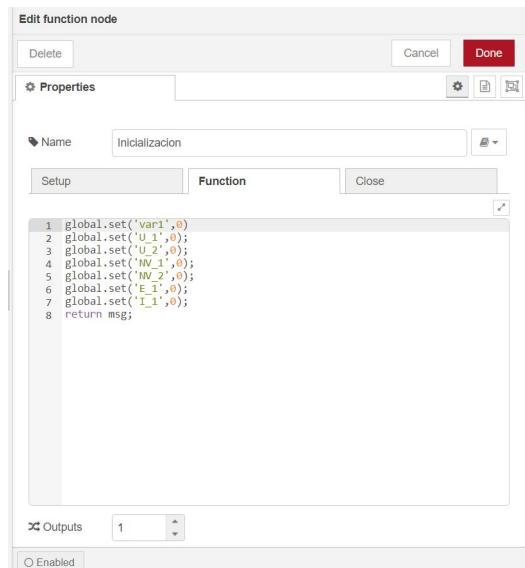


Figura 6.6 – Node-Red: Nodo Inicialización

6.1.4. Nodo Función Parámetros de Entrada

Este nodo función es llamado por el nodo cíclico main. En él, se recoge el valor de las variables globales de proceso haciendo uso de la instrucción:

```
global.get('variable')
```

Estos valores son asignados en una cadena de caracteres que presentan el siguiente formato:

```
"Var0" : global.get('variable0'), "Var1" : global.get('variable1'), "Varn" :
global.get('variablen')
```

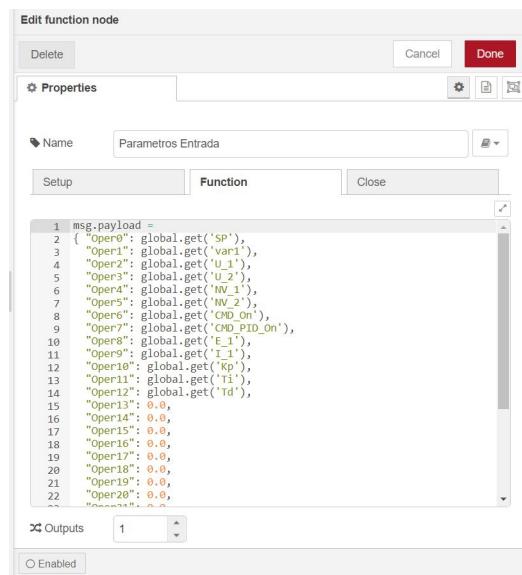


Figura 6.7 – Node-Red: Nodo Parámetros de entrada

6.1.5. Nodo Función String a CMD

Este nodo función es llamado por el nodo función parámetros de entrada y tiene como objetivo recoger los valores numéricos enviados desde el nodo anterior y concatenarlos con espacios para poder enviarlos como parámetros de entrada al fichero .py por la línea de comandos de Windows.

Para recoger el valor numérico se hace uso de la instrucción:

```
msg.payload.Var0
```

Esta concatenación se almacena en la variable temporal *In*, que es enviada a través del mensaje al nodo siguiente.

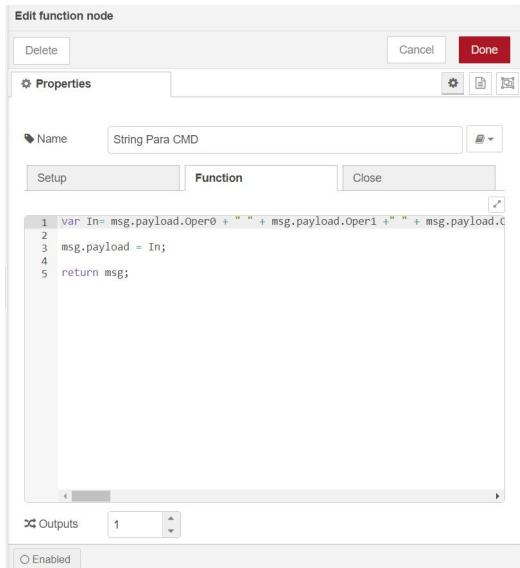


Figura 6.8 – Node-Red: Nodo String a CMD

6.1.6. Nodo Función Mensaje Llamada Planta Emulada

Este nodo función es llamado por el nodo función string a cmd y tiene como objetivo formar el mensaje que es enviado a la línea de comandos de la raspberry-pi. En este mensaje se llama al fuchero .py y se pasa por parámetros los datos de las variables globales del proceso.

La estructura que presenta este mensaje es:

”python/ubicacion/planta.py”variablesglobales

Esta concatenación se almacena en la variable temporal *cmd*, que es enviada a través del mensaje al nodo siguiente.

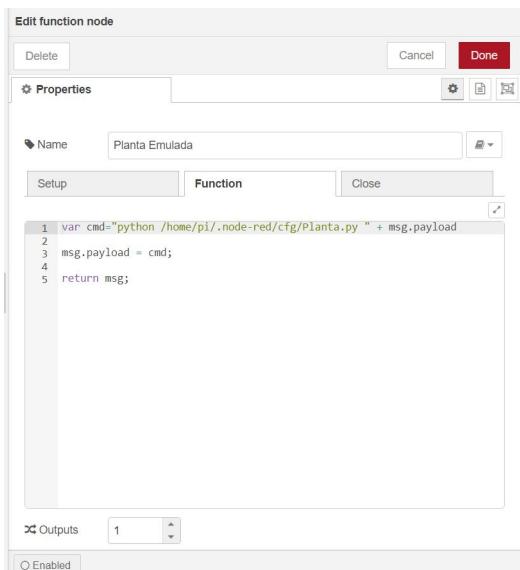


Figura 6.9 – Node-Red: Mensaje llamada a planta emulada

6.1.7. Nodo Función Comando Llamada Planta Emulada

Este nodo función es llamado desde el nodo función de mensaje llamada planta emulada y tiene el objetivo de lanzar en la línea de comandos de Windows el mensaje que recibe en la entrada.

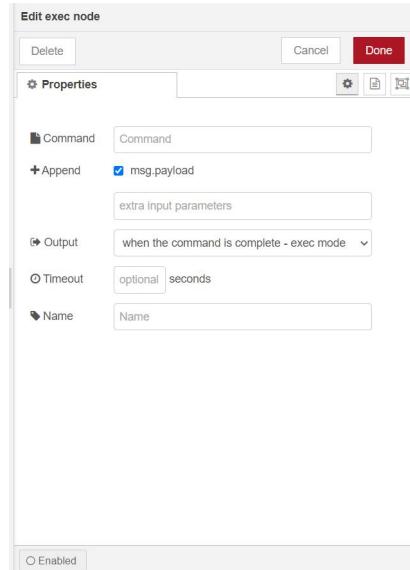


Figura 6.10 – Node-Red: Comando llamada a planta emulada

6.1.8. Nodo Función de String a JSON

Este nodo función es llamado desde el nodo función de comando llamada planta emulada y tiene el objetivo de pasar los datos de salida string que se reciben desde el fichero .py a formato JSON.

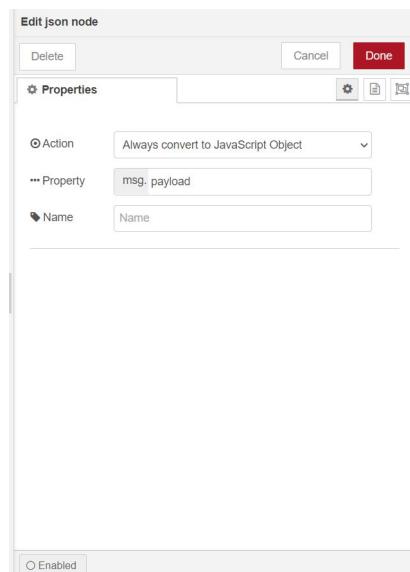


Figura 6.11 – Node-Red: Paso de formato string a json

6.1.9. Nodo función Memoria Estados Anteriores

Este nodo función es llamado desde el nodo función de string a JSON y tiene el objetivo de almacenar en variables globales de node-red los valores de proceso que son necesarios para realizar cálculos en cada llamada del fichero .py. Estas variables son:

- **U_1:** Señal de control en el instante anterior
- **U_2:** Señal de control en dos instantes anteriores
- **NV_1:** Nivel en el intante anterior
- **NV_2:** Nivel en dos instantes anteriores
- **E_1:** Error en el instante anterior
- **I_1:** Integral en el instante anterior

Para ello se hace uso de la instrucción:

```
global.set('variable', valor)
```

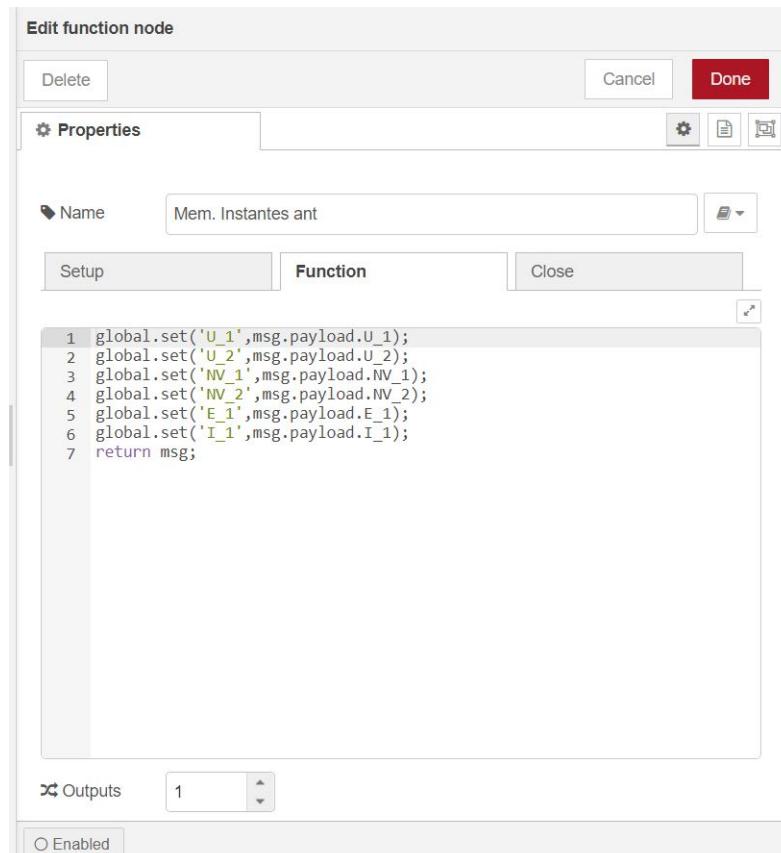


Figura 6.12 – Node-Red: Memoria estados anteriores

6.1.10. Datos para HMI

Estos nodos función son llamados desde el nodo función de string a JSON y tienen el objetivo de enviar el valor de cada variable del proceso a los nodos función que tienen interface gráfico en el dashboard.

Los nodos que recogen las variables son:

- Nivel para HMI
- Error para HMI
- Señal de control para HMI
- Frecuencia para HMI

La instrucción que se utiliza para recoger el valor de una variable específica del mensaje es:

msg.payload.dato

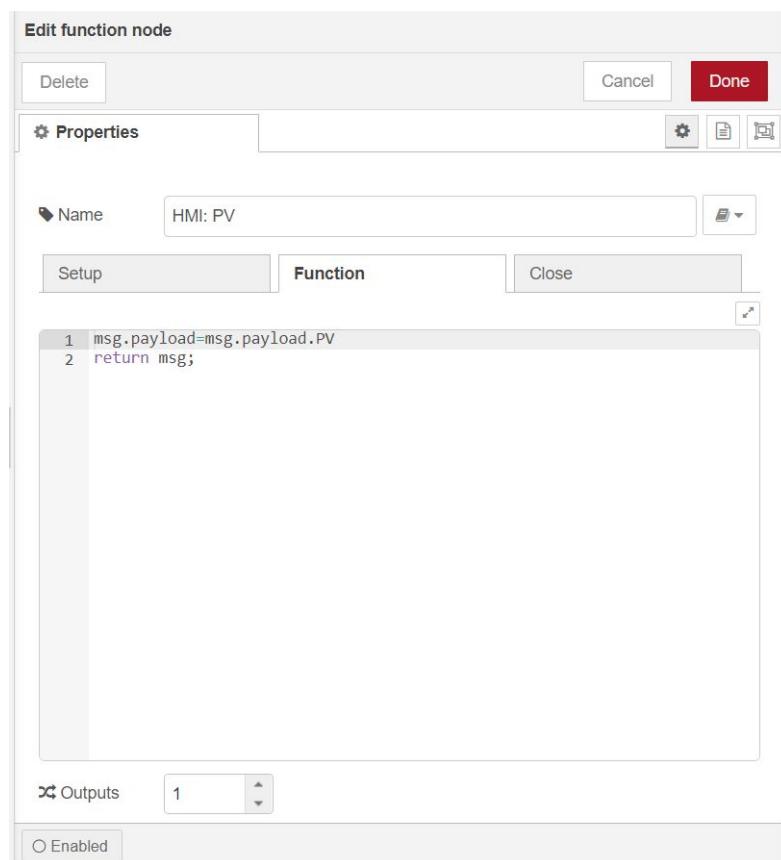


Figura 6.13 – Node-Red: Valores a HMI

En el caso del nodo función consigna para HMI es necesario recoger el valor de la variable global, ya que este parámetro es ajustado por el usuario desde el HMI. Para recoger este valor se hace uso de:

```
global.get('dato')
```

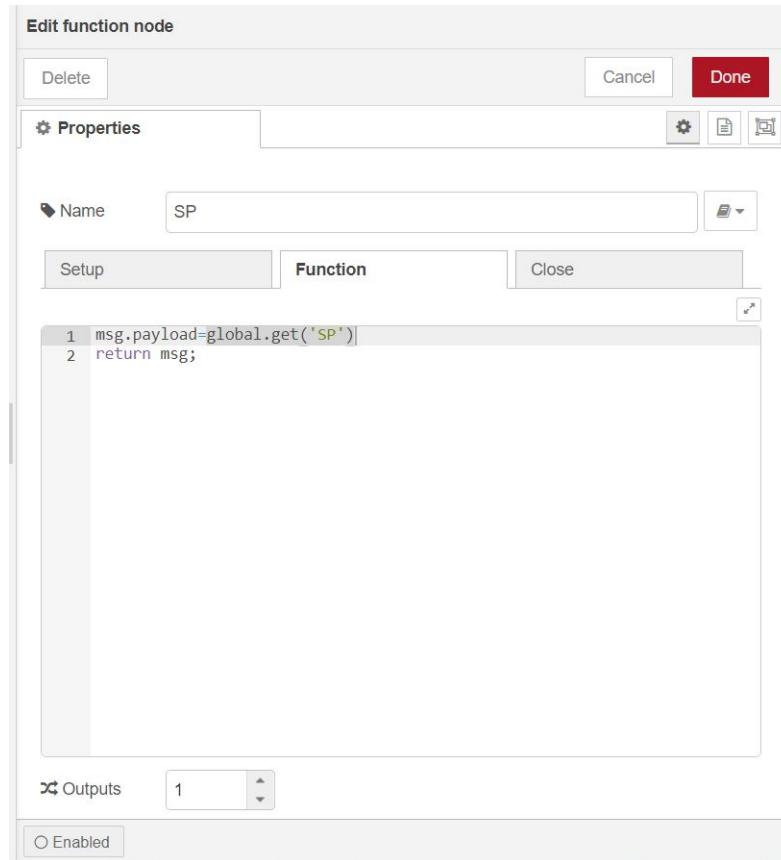


Figura 6.14 – Node-Red: Consigna a HMI

6.1.11. Datos desde HMI

Estos nodos función recogen los datos que introduce el usuario desde el HMI y los almacena en variables globales. Estos nodos son:

- Consigna desde el HMI
- Constante proporcional desde el HMI
- Constante integral desde el HMI
- Constante derivativa desde el HMI
- Comando de marcha/paro desde el HMI
- Comando de habilitar/deshabilitar regulación desde el HMI

La instrucción de la que se hace uso para recoger los datos es:

```
vardato = msg.payload; global.set('dato', dato); msg.payload = global.get('dato')
```

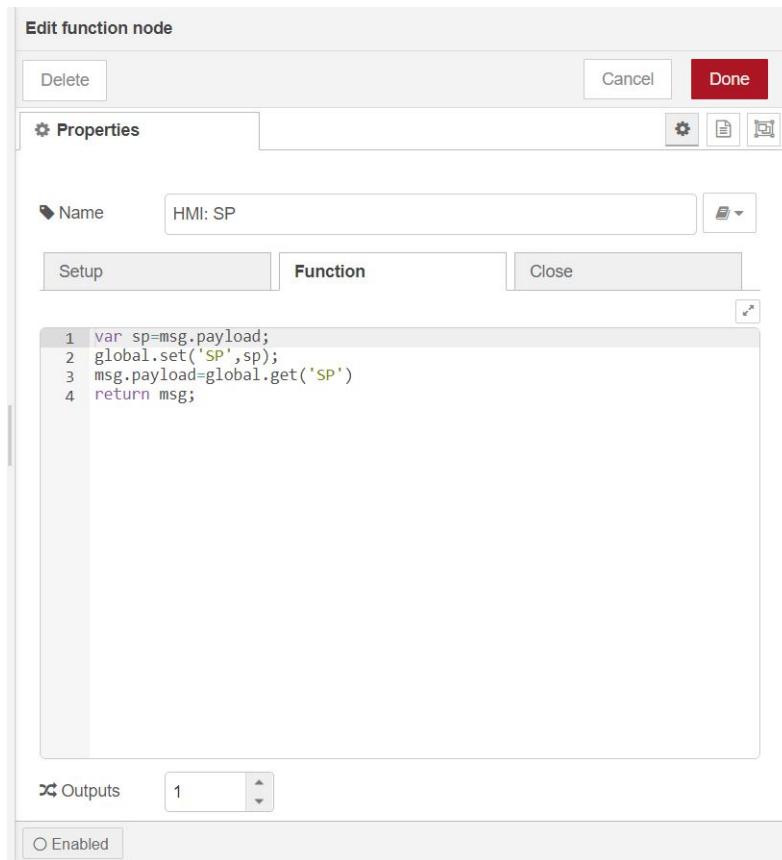


Figura 6.15 – Node-Red: Consignas/Comandos desde HMI

6.2. Interface Gráfico

Node-Red permite el diseño de un interface gráfico a través de nodos función basado en html. Se observa a través de un Web-Server en un navegador.

La aplicación gráfica del presente proyecto tiene el objetivo de representar el estado de la planta emulada, graficar las variables importantes del proceso (SP, señal de control, nivel, error...) y permitir la introducción de consignas del sistema (SP, Kp, Ti y Td). Para realizar esto se implementan cuatro pantallas.

- **Pantalla de inicio:** Pantalla inicial que se muestra al iniciar la aplicación
- **Pantalla Sinóptico:** Pantalla que muestra un sinóptico de la planta emulada y dónde se muestra el nivel y la frecuencia del variador.
- **Pantalla Ajustes:** Pantalla que permite introducir los parámetros del regulador, la consigna y dar orden de marcha al sistema. En ella también se muestra el nivel de la planta emulada, la frecuencia del variador y la representación en gráfica del nivel.
- **Pantalla Gráficos:** Pantalla destinada a las gráficas de la consigna, del nivel, del error y de la señal de control.

En node-red, cada pantalla se compone de grupos y cada grupo se compone de elementos. Las dimensiones de la pantalla dependen de las dimensiones de los grupos. Las pantallas de node-red no permite que existan huecos vacíos. Toda la pantalla se debe de llenar o con un elemento visible o con un elemento denominado *spacer* (elemento vacío). Los *spacer* se utilizan para posicionar los elementos visibles en la posición de la ventana que interesa. En la siguiente imagen se muestra un ejemplo de una composición de una pantalla en dónde los elementos azules son visibles y los blancos son *spacers*.



Figura 6.16 – Node-Red: Ejemplo Composición HMI

La apariencia del dashboard es la siguiente:

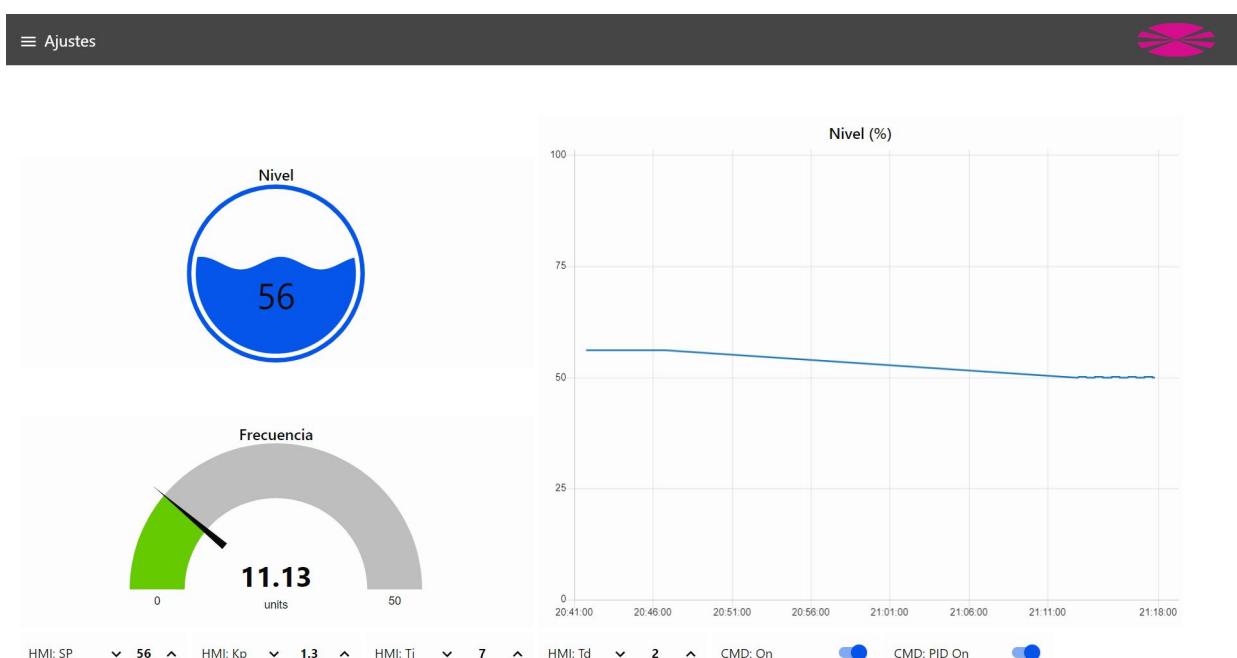


Figura 6.17 – Node-Red: Ejemplo Composición Dashboard HMI

6.2.1. Pantalla de inicio

La pantalla de inicio se muestra al abrir la aplicación. Esta pantalla contiene un grupo con un elemento en el que se ubica la imagen de inicio. Para llamar a la imagen, posicionar y escalarla correctamente se hace uso de un nodo función de tipo *template* denominado código inicio. La programación de este elemento se realiza en código HTML.

```
<style>
body {
    background-image: url("/img/inicio.png");
    background-repeat: no-repeat;
    background-position: left bottom;
    background-size: 102%;
}
</style>
```

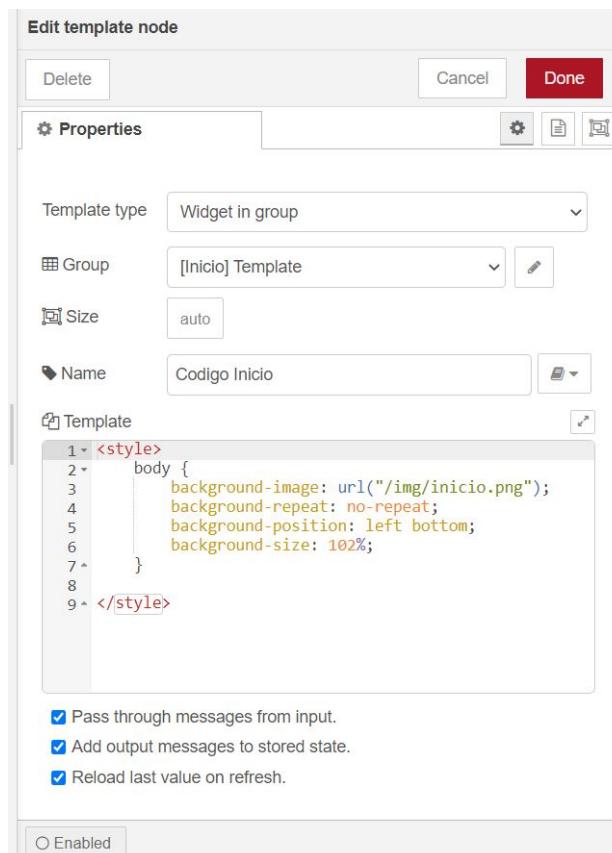


Figura 6.18 – Node-Red: Código inicio

La apariencia que tiene esta pantalla se muestra en la siguiente imagen:



Figura 6.19 – Node-Red: Inicio HMI

6.2.2. Pantalla Sinóptico

La pantalla de sinóptico muestra una representación de la planta real de nivel. En ella se muestra el nivel y la frecuencia del variador. Esta pantalla consta de varios grupos con elementos *spacer*, un grupo con la imagen de fondo, un grupo con la representación del nivel y un grupo con la representación de la frecuencia. Los elementos *spacer* son invisibles y se utilizan para posicionar correctamente el elemento de representación del nivel y de la frecuencia sobre la imagen.

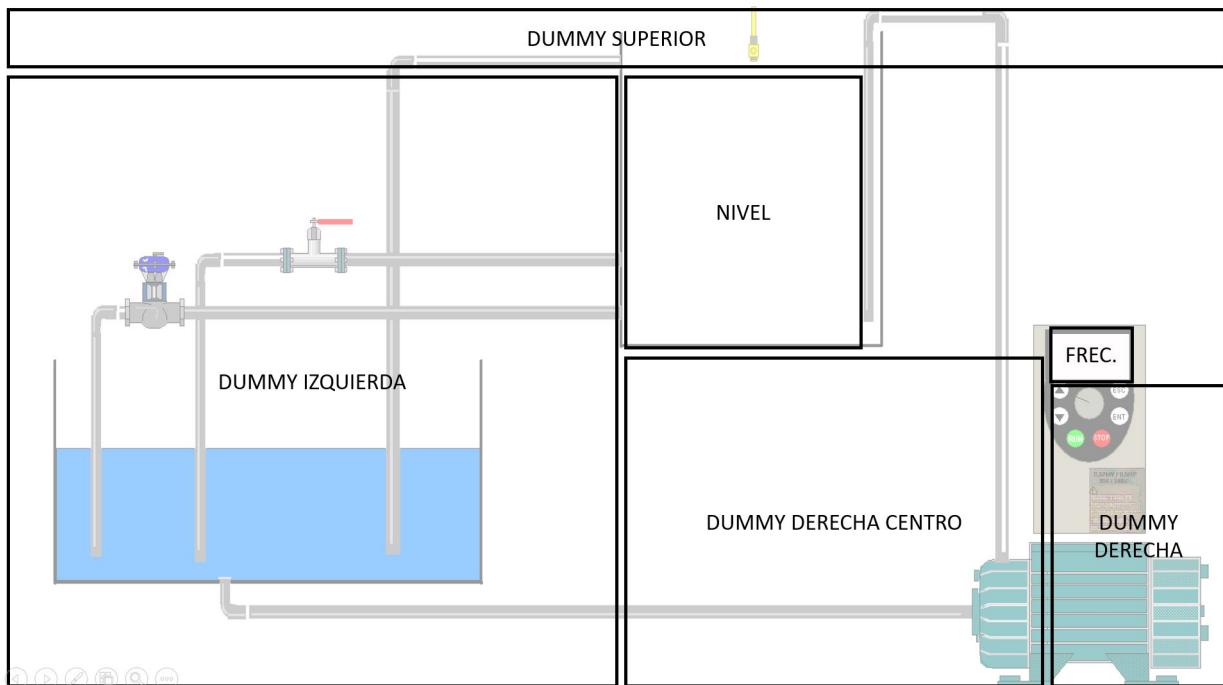


Figura 6.20 – Node-Red: Distribución Sinóptico

Para llamar a la imagen, posicionar y escalarla correctamente se hace uso de un nodo

función de tipo *template* denominado código sinóptico. La programación de este elemento se realiza en código HTML.

```
<style>
body {
    background-image: url("/img/Sinoptico_Planta.PNG");
    background-repeat: no-repeat;
    background-position: left bottom;
    background-size: 102%;
}
</style>
```

En el mismo elemento *template*, se programa la transparencia de los dummies y la eliminación de sus bordes con el siguiente código html.

```
<style>
#Sinoptico_Dummy_Sup{
    background-color:#11111100;
    border-style :none;
}
</style>
```

La configuración de este *template* se muestra en la siguiente imagen:

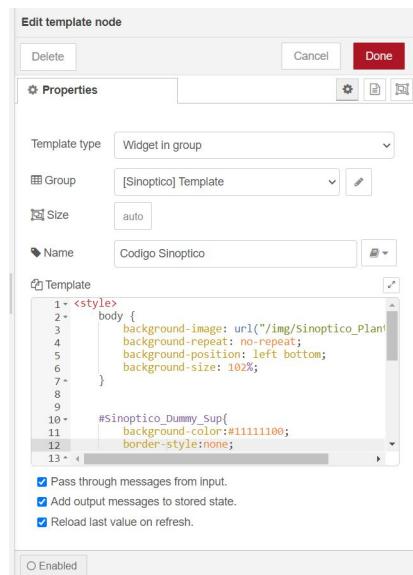


Figura 6.21 – Node-Red: Código sinóptico

Para representar el nivel se hace uso de un nodo función con representación en el dashboard de estilo *Level*. Este nodo tiene como entrada el nivel que es devuelto desde el nodo función nivel para HMI.

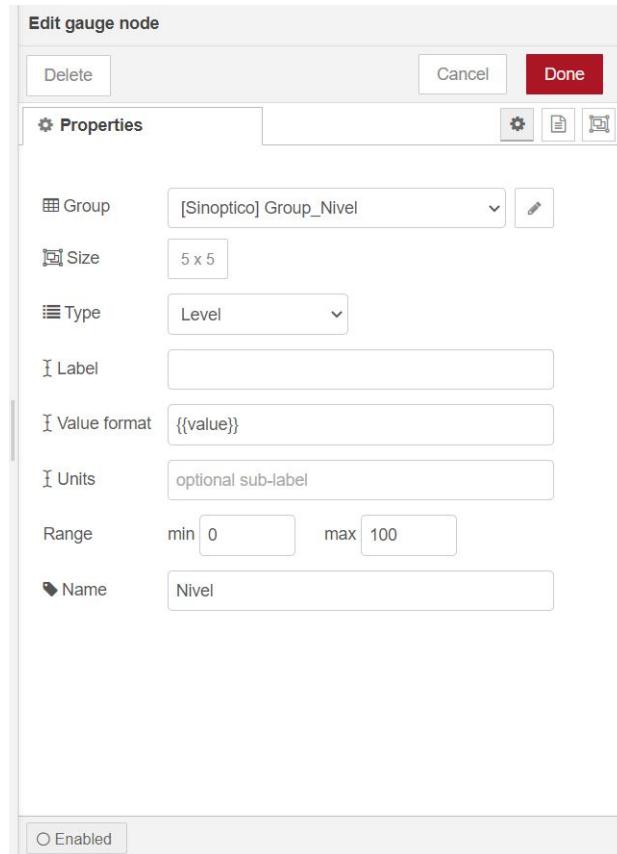


Figura 6.22 – Node-Red: Nivel sinóptico

Su representación en el dashboard es la siguiente:

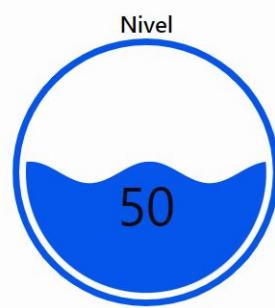


Figura 6.23 – Node-Red: Representación nivel

La frecuencia del variador se representa a través de un nodo función con representación en el dashboard de estilo texto. Este nodo tiene como entrada la frecuencia que es devuelta desde el nodo función frecuencia del variador para HMI.

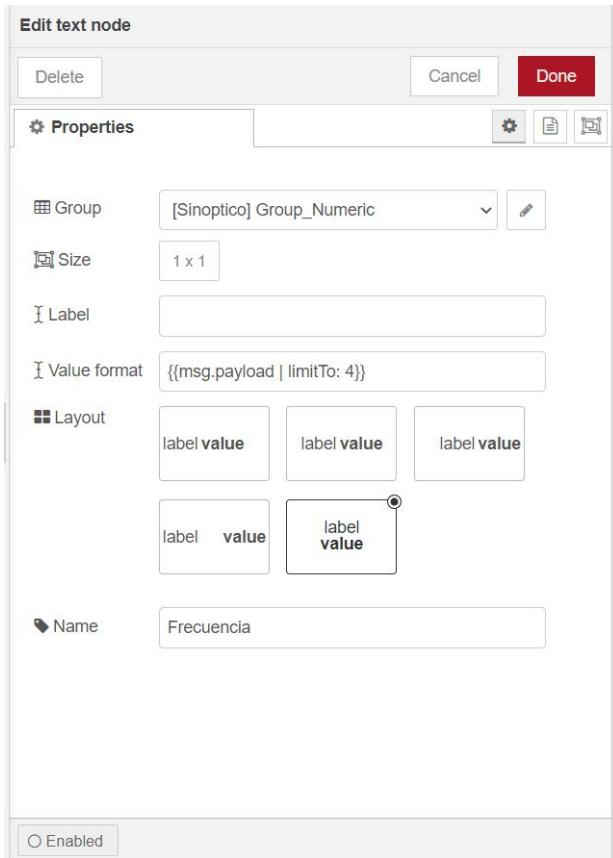


Figura 6.24 – Node-Red: Frecuencia sinóptico

La apariencia que tiene la pantalla de sinóptico se muestra en la imagen siguiente:

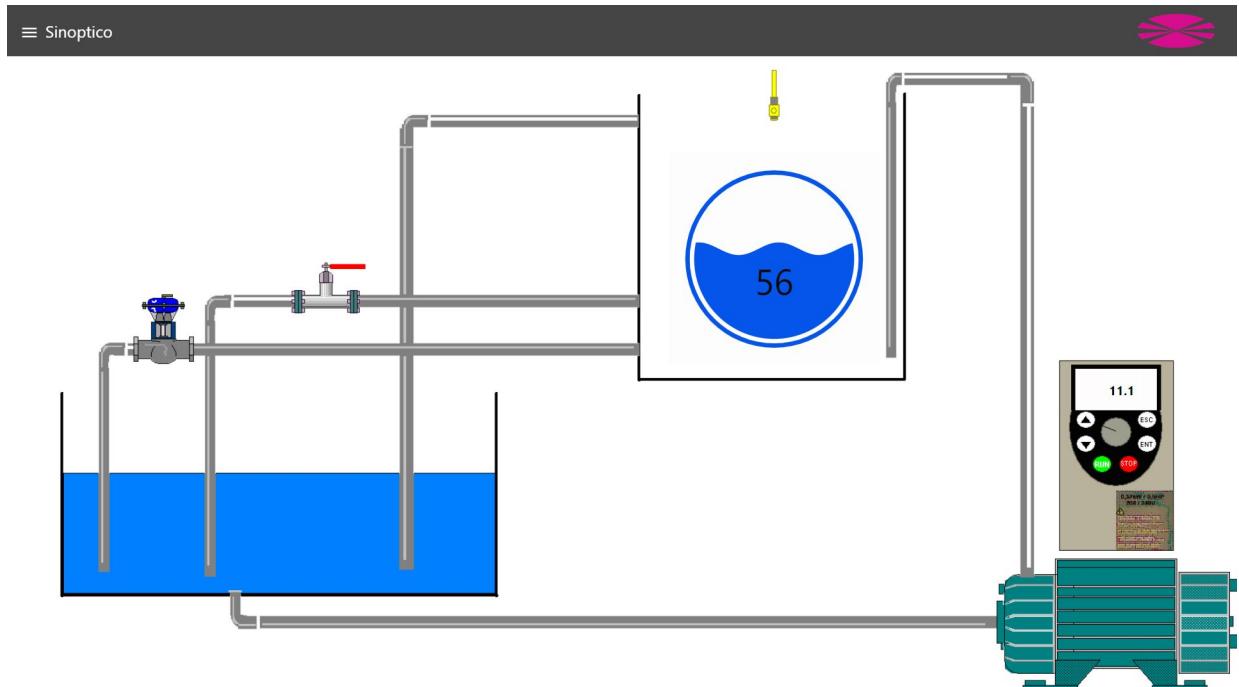


Figura 6.25 – Node-Red: Sinoptico HMI

6.2.3. Pantalla Ajustes

La pantalla de ajustes permite al usuario introducir la consigna de nivel, las constantes características del regulador PID y dar las órdenes de marcha/paro al sistema y habilitar/deshabilitar la acción de la regulación. También representa el nivel y la frecuencia. Existe una gráfica que permite ver la evolución del nivel.

La consigna y los parámetros característicos del regulador PID se introducen desde el dashboard a través de un nodo función con representación de estilo *introducción numérica*.

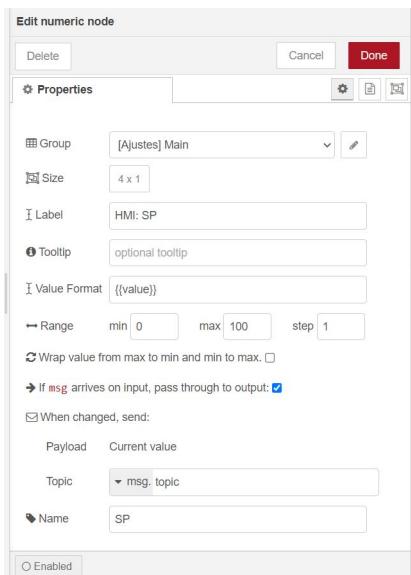


Figura 6.26 – Node-Red: Introducción numérica

La representación que tiene este nodo función en el dashboard es la siguiente:



Figura 6.27 – Node-Red: Representación campo introducción numérica

Los comandos de marcha/paro del sistema y de habilitar/deshabilitar regulación se realizan con un nodo función con representación en el dashboard del tipo *switch*.

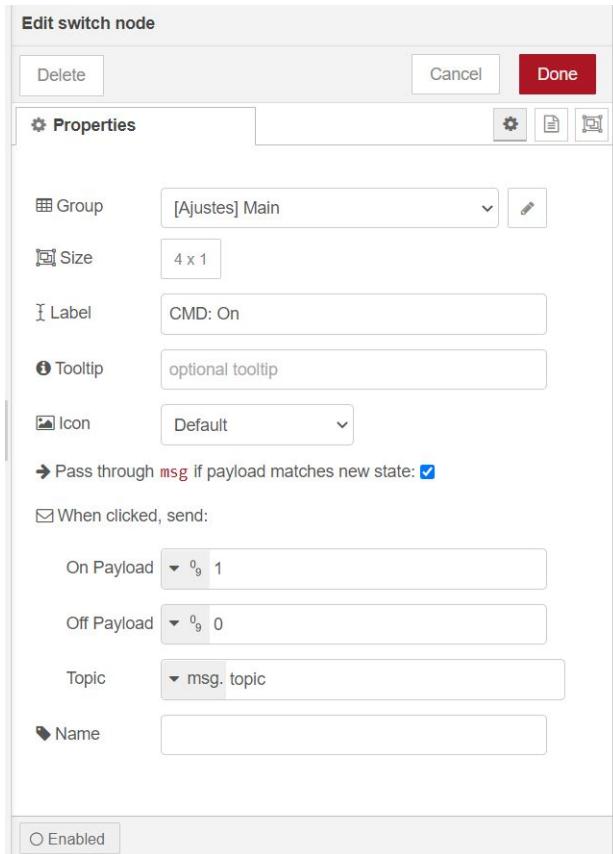


Figura 6.28 – Node-Red: Comando desde HMI

La representación gráfica de este nodo función es la siguiente:



Figura 6.29 – Node-Red: Representación comando

La representación del nivel es similar a la realizada en la pantalla sinóptico([6.22, 6.23](#)).

Para representar la frecuencia se hace uso de un nodo función con representación en el dashboard de estilo *Gauge*. Este nodo tiene como entrada el nivel que es devuelto desde el nodo función nivel para HMI.

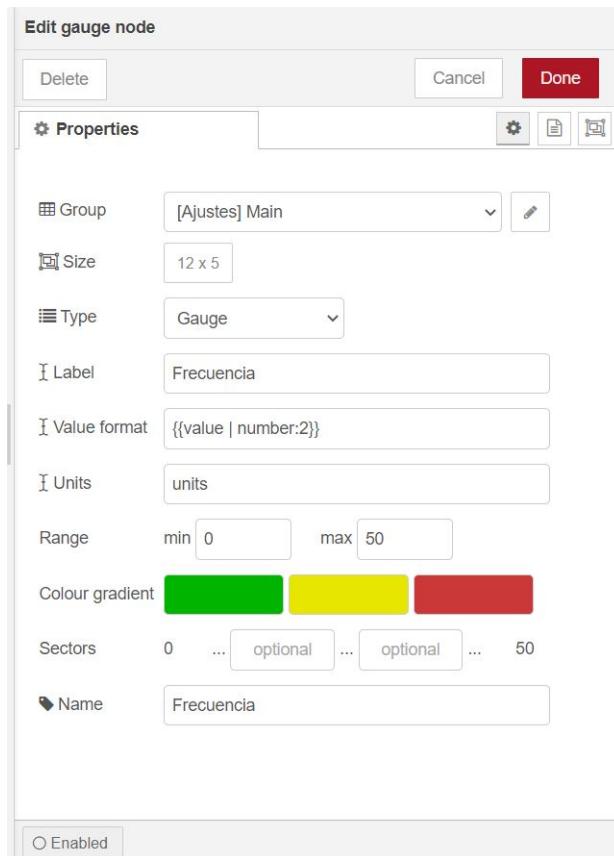


Figura 6.30 – Node-Red: Frecuencia Ajustes

Su representación en el dashboard es la siguiente:

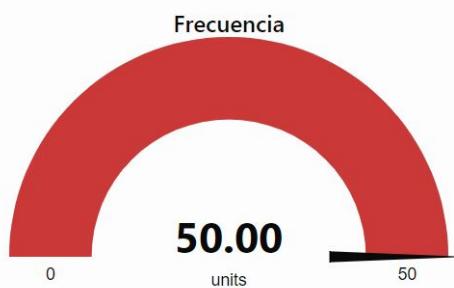


Figura 6.31 – Node-Red: Representación frecuencia (Gauge)

Para representar en una gráfica la evolución del nivel se hace uso de un nodo función con representación en el dashboard del tipo *Graph*.

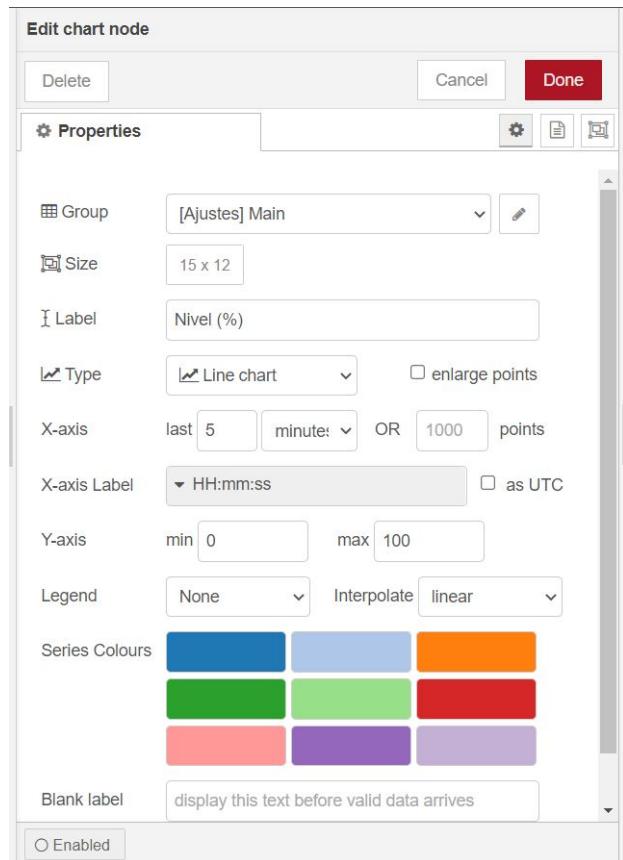


Figura 6.32 – Node-Red: Grafico Ajustes

Su representación en el dashboard es la siguiente:

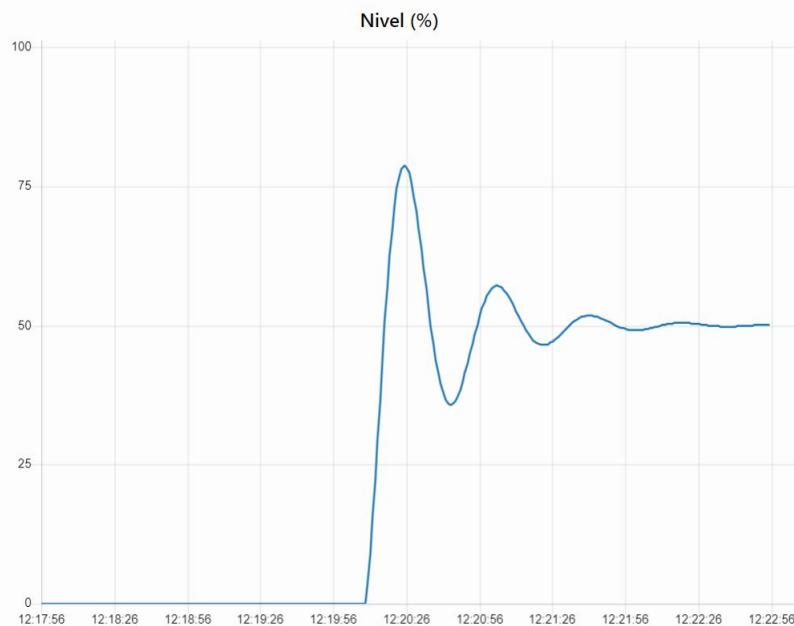


Figura 6.33 – Node-Red: Representación grafica nivel

La apariencia de la pantalla de ajustes se puede observar en la siguiente imagen:

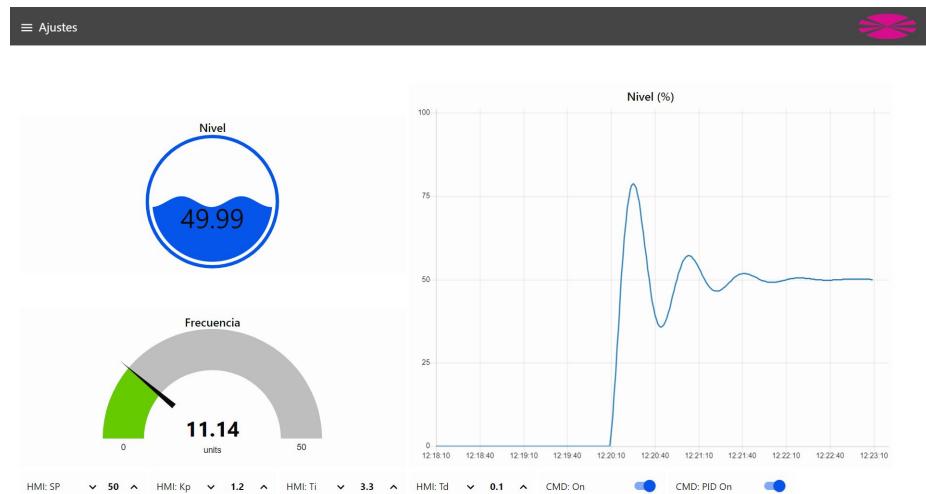


Figura 6.34 – Node-Red: Ajustes HMI

6.2.4. Pantalla Gráficos

Esta pantalla se compone de cuatro gráficos que permiten ver la evolución de las señales de nivel, error, señal de control y consigna. Para estas gráficas se hace uso de nodos función con representación gráfica en el dashboard del estilo Graph. La configuración y apariencia es la misma que la explicada en [6.32, 6.33](#).

La apariencia que muestra esta ventana se puede ver en la siguiente imagen:

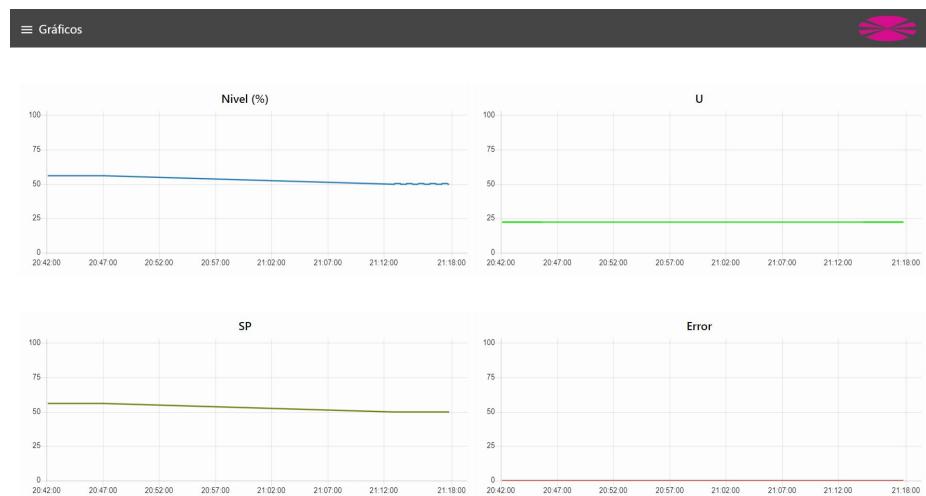


Figura 6.35 – Node-Red: Graficas HMI

6.3. Arquitectura en Node-Red

La apariencia de los nodos explicados en las secciones anteriores se puede observar en la siguiente imagen:

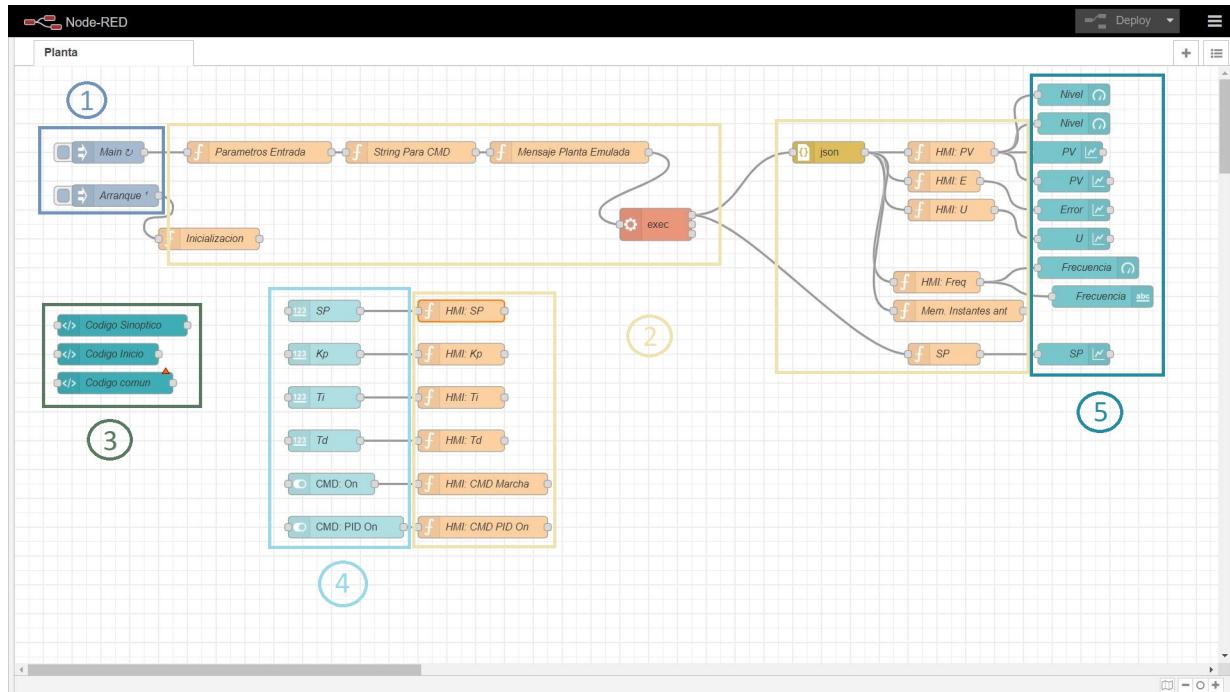


Figura 6.36 – Node-Red: Arquitectura

En la arquitectura implementada se pueden encontrar varias zonas que contienen nodos. Estas zonas son:

- **Zona 1:** Nodos cíclicos
- **Zona 2:** Nodos función
- **Zona 3:** Nodos plantilla de visualización
- **Zona 4:** Nodos con interface gráfico que envian datos desde la visualización a Node-Red
- **Zona 5:** Nodos con interface gráfico que reciben los datos desde el Node-Red.

TÍTULO: **EMULACIÓN DE PLANTA DE NIVEL BASADA EN SISTEMA
EMBEBIDO**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2022

AUTOR: EL/LA ALUMNO/A

Fdo.: DANIEL MÉNDEZ BUSTO

Índice del documento PRESUPUESTO

7 PRESUPUESTO MATERIALES	79
8 PRESUPUESTO MANO DE OBRA	79
9 PRESUPUESTO	79

7 PRESUPUESTO MATERIALES

IMAGEN	DESCRIPCIÓN	P.V.P	ud.	Total
	NinkBox Raspberry Pi 4 Modelo B, con HDMI, Fuente de Alimentación 5V/3A con Interruptor, Ventilador	195	1	195

Tabla 7.1 – Presupuesto materiales

8 PRESUPUESTO MANO DE OBRA

CONCEPTO	Nº HORAS	PRECIO/HORA	Total
Ingenieria	15	50	750
Desarrollo	300	50	15000

Tabla 8.1 – Presupuesto mano de obra

9 PRESUPUESTO

CONCEPTO	SUBTOTAL
Raspberry Pi	195
Ingenieria	750
Desarrollo	15000
Iva 21 %	3,348.45
TOTAL:	19,293.45

Tabla 9.1 – Presupuesto

TÍTULO: **EMULACIÓN DE PLANTA DE NIVEL BASADA EN SISTEMA
EMBEBIDO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: **JUNIO DE 2022**

AUTOR: **EL/LA ALUMNO/A**

Fdo.: **DANIEL MÉNDEZ BUSTO**

Índice del documento Pliego de Condiciones

10 CONDICIONES DE TRABAJO	85
11 Hardware y Software	86
11.1 Hardware	86
11.2 Software	86

10 CONDICIONES DE TRABAJO

Todas las pruebas del presente trabajo Fin de Máster han sido realizadas en el laboratorio de optimización y control de la Escuela Universitaria Politécnica de Serantes.

Se han realizado en un entorno limpio y seco, a una temperatura ambiente entre 19 °C y 22 °C.

El depósito inferior que surte el agua al depósito superior, siempre se ha encontrado al máximo de su capacidad para no dañar la integridad del sistema durante las pruebas. La planta de nivel se encuentra sobre una superficie nivelada y rugosa.

La válvula de vaciado manual del depósito se encuentra en una posición de 50 % de apertura.

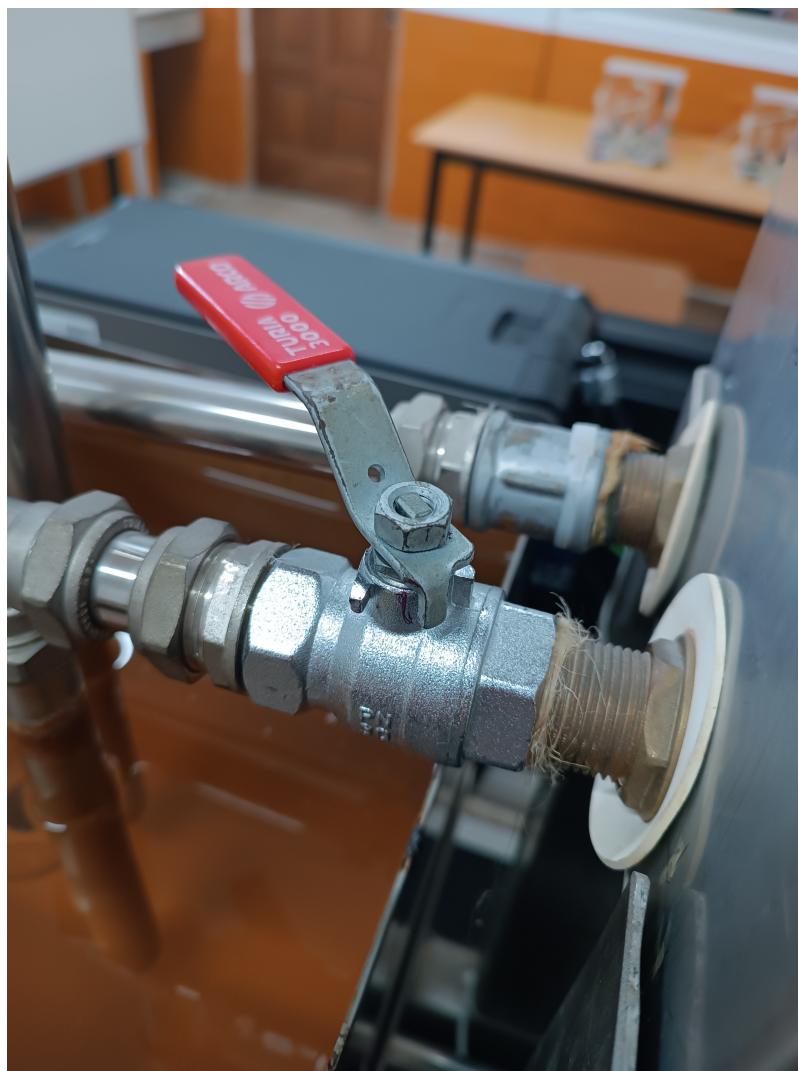


Figura 10.1 – Posición válvula manual de vaciado

Toda prueba realizada que no cumpla estas condiciones de trabajo no será valida.

11 Hardware y Software

11.1. Hardware

El hardware utilizado en el presente proyecto es:

- Raspberry Pi 4 Model B - 4 GB de RAM

11.2. Software

El software utilizado en el presente proyecto es:

- Sistema Operativo Raspberry: Raspberry-Pi OS 32-bit
- Matlab R2014b
- Node Red v16.15.1
- Python v3.9.0

TÍTULO: **EMULACIÓN DE PLANTA DE NIVEL BASADA EN SISTEMA
EMBEBIDO**

ANEXOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2022

AUTOR: EL/LA ALUMNO/A

Fdo.: DANIEL MÉNDEZ BUSTO

Índice del documento ANEXOS

12 Manual de funcionamiento	91
12.1 Instalaciones previas	91
12.1.1 Instalación del sistema operativo	91
12.1.2 Iniciando Raspberry-Pi	94
12.1.3 Instalación de Node-Red	95
12.1.3.1 Raspberry-Pi	95
12.1.3.2 PC	95
12.1.4 Configuración en Raspberry-Pi	96
12.2 Funcionamiento de la emulación	104
13 Códigos de Programación	109
13.1 Planta.py	109
13.1.1 Main	109
13.1.2 Entradas	109
13.1.3 Parámetros Planta	110
13.1.4 Regulador PID	111
13.1.5 Planta	111
13.1.6 Frecuencia Variador	112
13.1.7 Paso a JSON	112
14 Variables Scripts	113
14.1 Identificación	113
14.2 Comprobación	114

12 Manual de funcionamiento

12.1. Instalaciones previas

Para poder ejecutar la emulación en la raspberry-pi se tienen que realizar las instalaciones previas que se explican en las siguientes secciones.

12.1.1. Instalación del sistema operativo

En primer lugar se debe de instalar el sistema operativo en la Raspberry Pi. En este caso el sistema operativo que se va a instalar es el Raspberry Pi OS y se realiza desde un PC con Windows.

Se accede a la <https://www.raspberrypi.com/software/> y se selecciona *Download for Windows* para descargar el instalador del sistema operativo de la Raspberry-Pi.

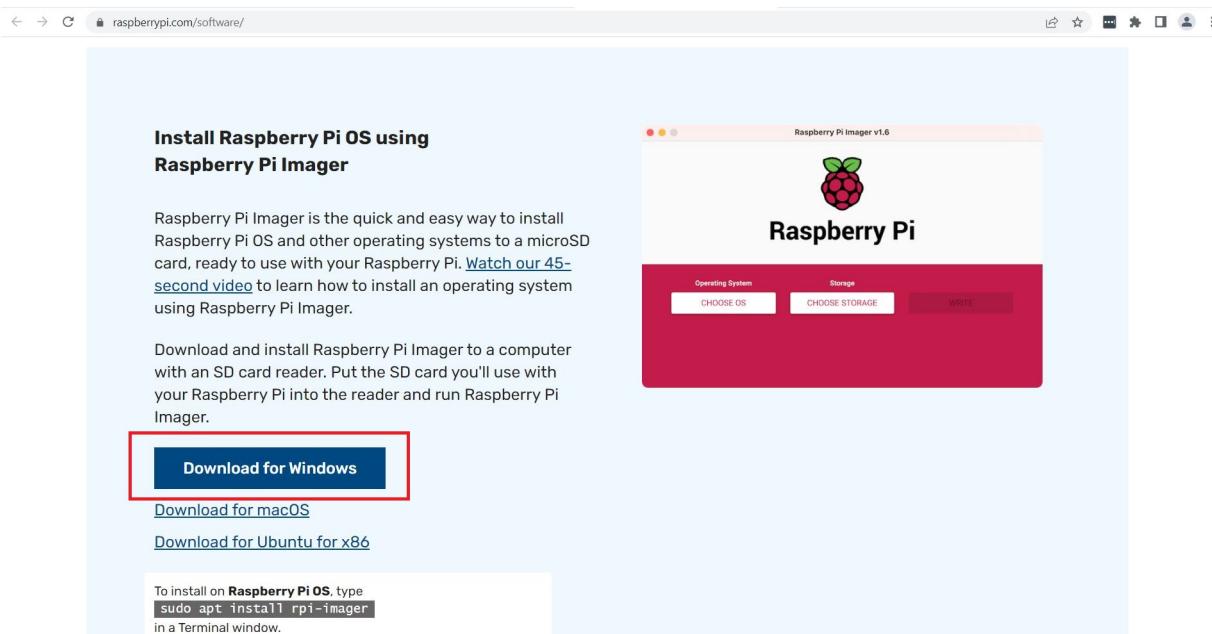


Figura 12.1 – Descarga de Sistema Operativo

Cuando finaliza la descarga se obtiene un ejecutable denominado *imager1,7,2.exe*. Se ejecuta y se ejecutan los pasos de la instalación.

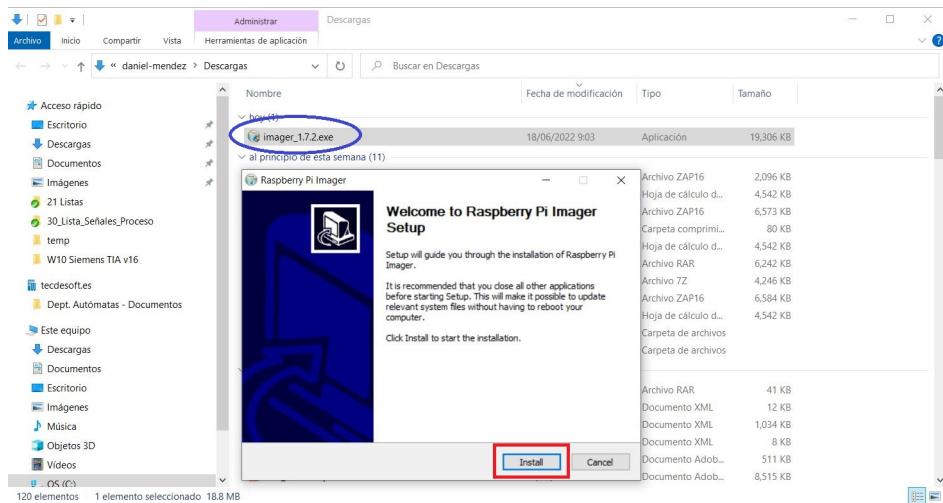


Figura 12.2 – Ejecutar instalador

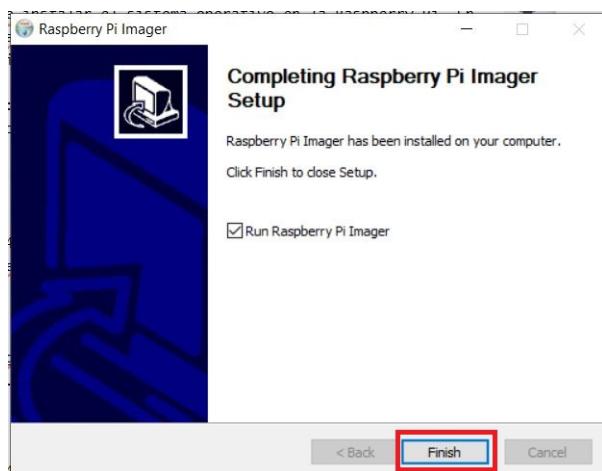


Figura 12.3 – Finalizar instalación instalador

Al finalizar la instalación del instalador se abre automáticamente el *Raspberry Pi Imager*.

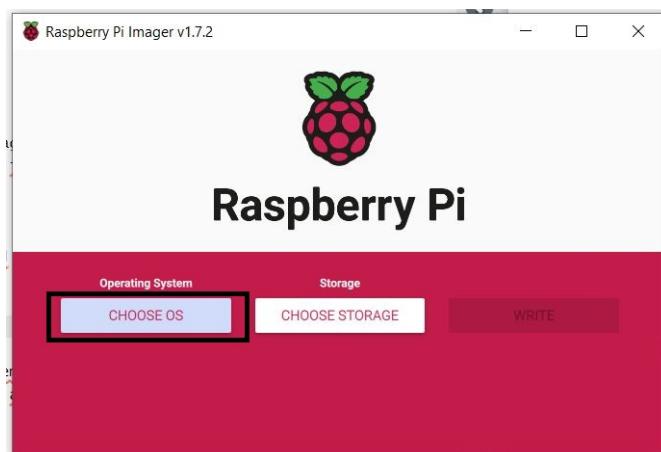


Figura 12.4 – Raspberry-Pi Imager

Se selecciona *Choose OS* y se muestra un desplegable con diferentes opciones de siste-

mas operativos. En este caso se selecciona el sistema operativo recomendado *Raspberry Pi OS (32 bit)*.

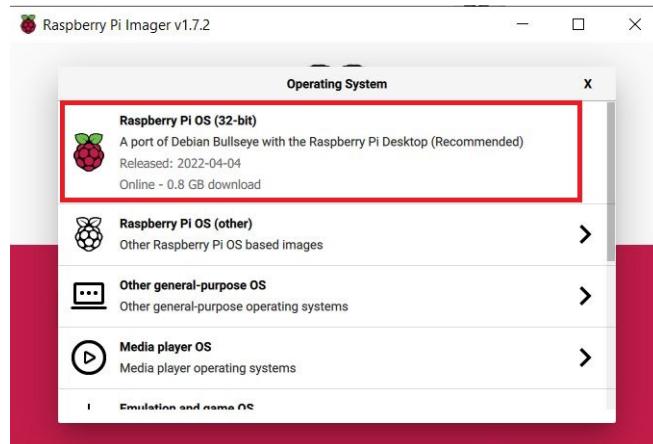


Figura 12.5 – Raspberry-Pi Imager - Selección Sistema Operativo

Al seleccionar el sistema operativo, se vuelve a mostrar el instalador *Raspberry-Pi Imager*.

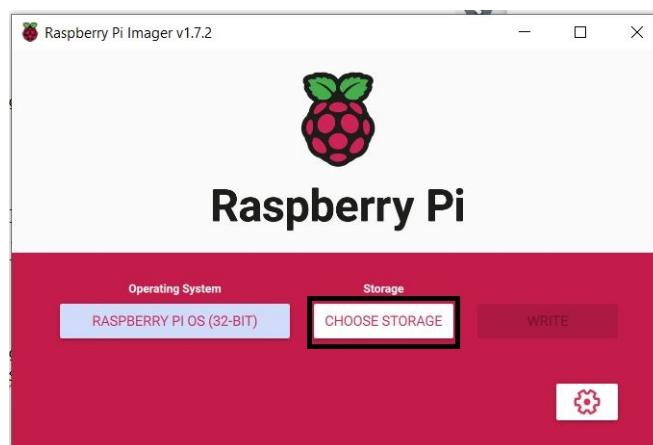


Figura 12.6 – Raspberry-Pi Imager 2

Se introduce la tarjeta de memoria en dónde se va a instalar el sistema operativo en el PC y se selecciona la opción *Choose Storage*.

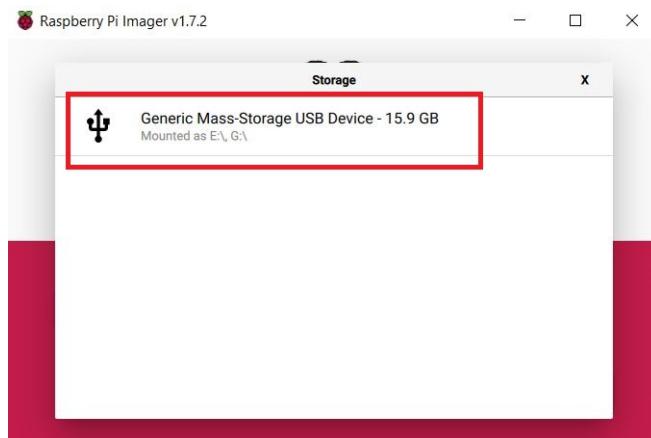


Figura 12.7 – Selección de medio de almacenamiento

Se muestran los medios de almacenamiento encontrados y se selecciona la tarjeta de memoria en dónde se quiere instalar el sistema operativo.



Figura 12.8 – Raspberry-Pi Imager 2 - Write

Al seleccionar el medio de almacenamiento, se vuelve a mostrar el *Raspberry-Pi Imager* indicando el sistema operativo y el medio de almacenamiento seleccionado. Se selecciona en *write* para comenzar con la instalación.

El software de instalación se puede encontrar en https://github.com/Danielmendezb/TFM_Planta_Emulada/tree/main/3_Instalacion/3_0_Ejecutables.

12.1.2. Iniciando Raspberry-Pi

Al iniciar el sistema operativo de Raspberry-Pi se configura el país, el idioma y la hora. A continuación se muestra el escritorio.

Es necesario conectar la Raspberry-Pi a internet mediante WIFI para poder realizar las instalaciones de software necesarias.

12.1.3. Instalación de Node-Red

12.1.3.1. Raspberry-Pi

Para instalar Node-Red en la Raspberry-Pi, se debe de ejecutar en la línea de comandos de la Raspberry-Pi el siguiente comando:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

Una vez finalizada la instalación se arranca node-red con el comando:

```
sudo node-red
```

12.1.3.2. PC

Para poder ver el interface gráfico de node-red alojado en la Raspberry desde un PC es necesario que el PC desde el que se accede tenga instalado Node-Red.

La descarga de Node-Red se realiza desde la <https://nodejs.org/en/>.

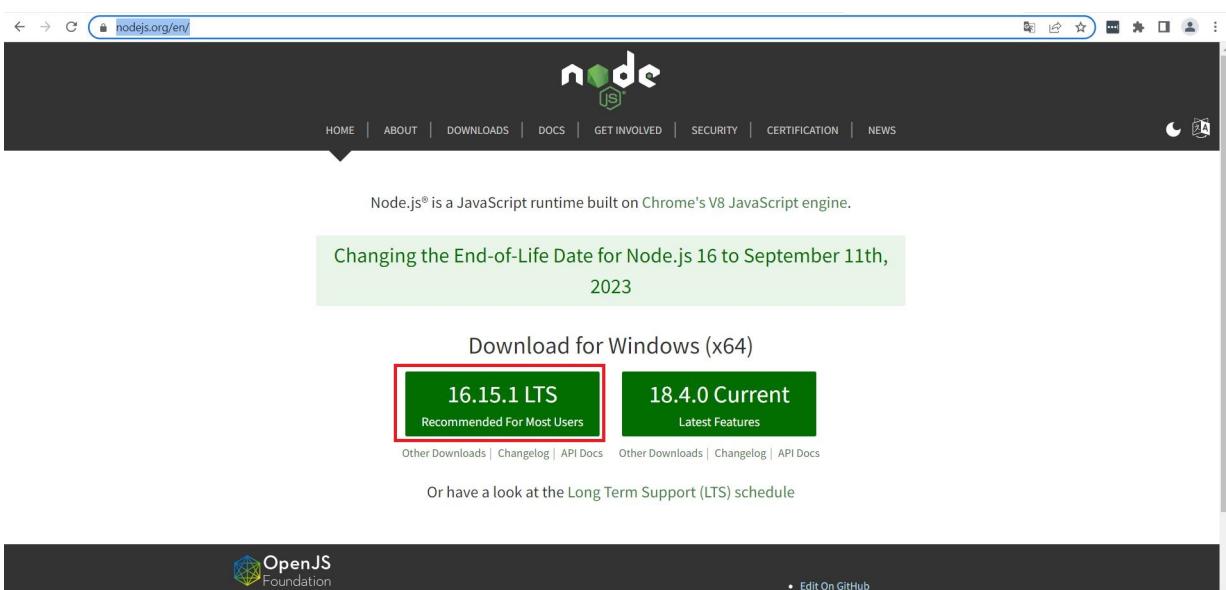


Figura 12.9 – Descarga intalador Node-Red PC

Se descarga un ejecutable denominado *node-v16.15.1-x64.msi*. Se ejecutan y se siguen los pasos que indica el instalador.

Una vez finalizada la instalación, se comprueba si se ha instalado correctamente. Para ello en la línea de comandos de Windows se ejecuta la siguiente instrucción:

```
node --version npm --version
```

Al ejecutar esta instrucción se devuelven las versiones instaladas.



Figura 12.10 – Comprobar versión de node-red

Para finalizar la instalación se debe de escribir la siguiente instrucción en la línea de comandos de Windows.

```
npm install -g --unsafe-perm node-red
```

El comando que se debe de ejecutar para lanzar node-red es:

```
node-red
```

El software de instalación se puede encontrar en https://github.com/Danielmendezb/TFM_Planta_Emulada/tree/main/3_Instalacion/3_0_Ejecutables.

12.1.4. Configuración en Raspberry-Pi

Para que la emulación se ejecute correctamente es necesario crear dos carpetas de configuración en la ruta `/home/pi/node-red`. Estas carpetas son:

- **cfg:** Contiene archivos de configuración de la aplicación (fichero de python de la emulación y *flow* de Node-Red).
- **img:** Contiene las imágenes de la aplicación.

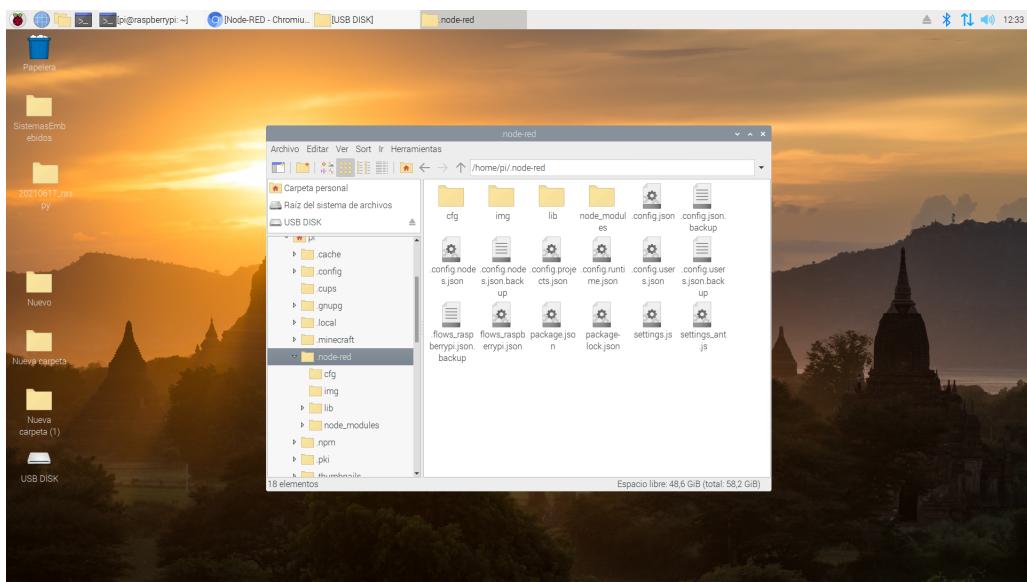


Figura 12.11 – Carpeta de configuración e imágenes

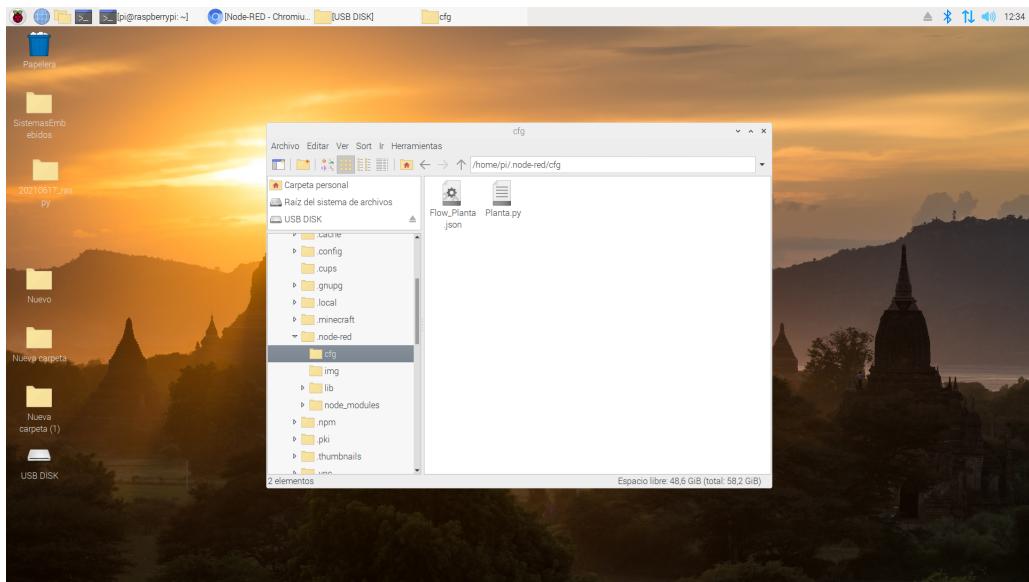


Figura 12.12 – Carpeta de configuración

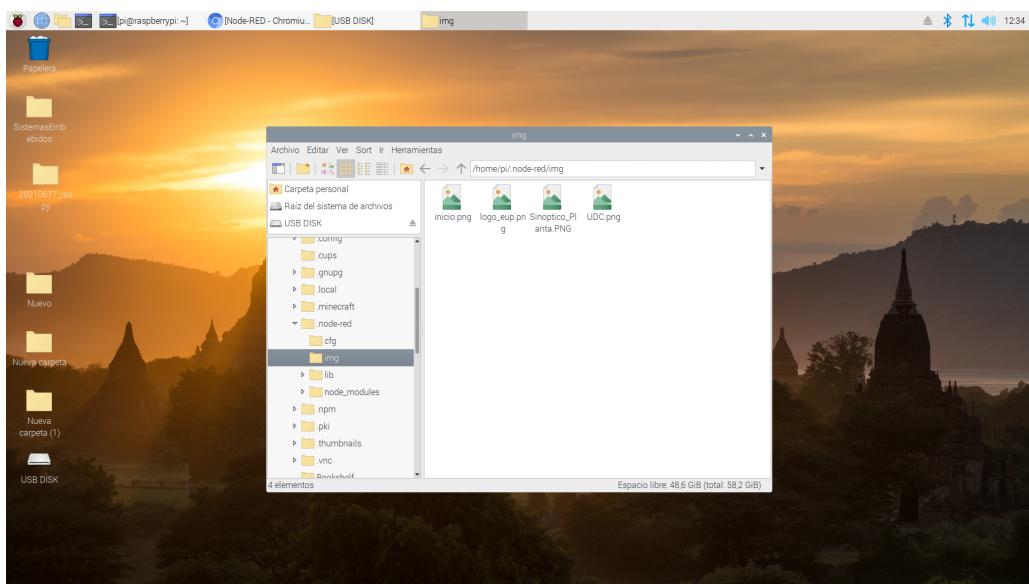


Figura 12.13 – Carpeta de Imágenes

El contenido de estas carpetas se puede encontrar en https://github.com/Danielmendezb/TFM_Planta_Emulada/tree/main/3_Instalacion.

En la ruta `/home/pi/node-red` existe un archivo de configuración de node-red denominado `settings.js`. Este fichero tiene que ser modificado para indicar la ruta de trabajo. Para ello se descomenta la línea que se indica en la imagen y se especifica la ruta `httpStatic: '/home/pi/.node-red/'`.

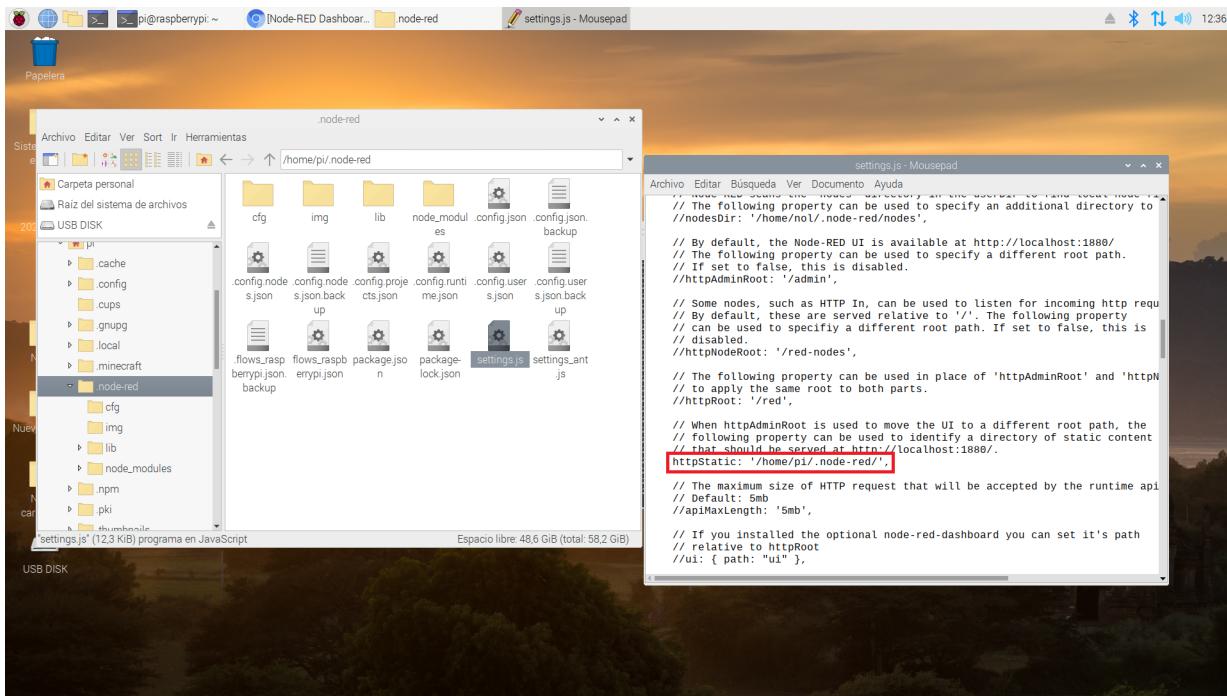


Figura 12.14 – Modificación de la configuración

Tras realizar estas configuraciones se arranca node-red y se accede al editor a través del navegador en la url *localhost:1880*.

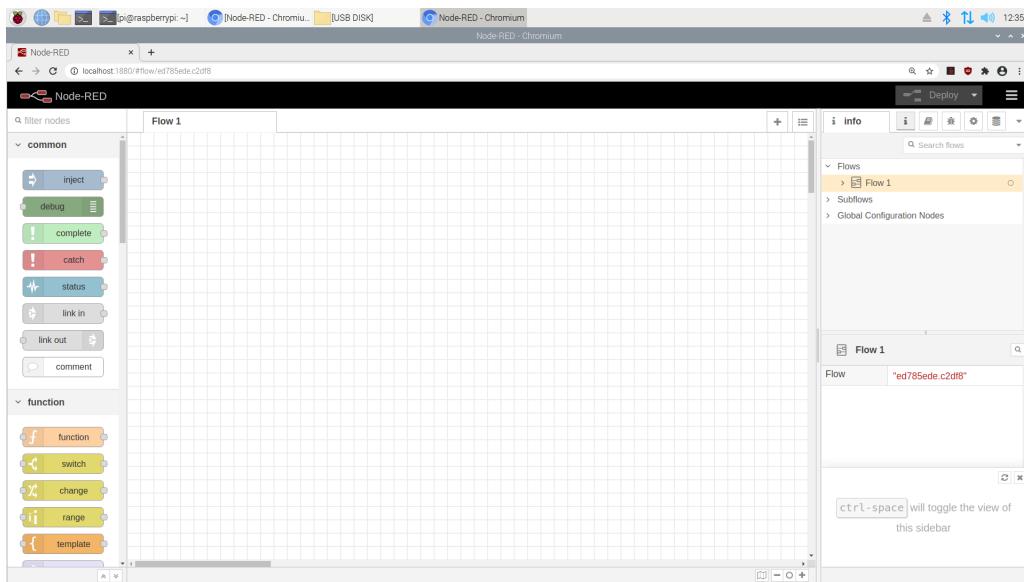


Figura 12.15 – Editor de node-red

Al abrirse la página, se muestra un *flow* vacío. Para importar el archivo que contiene la emulación, en el menú se selecciona importar.

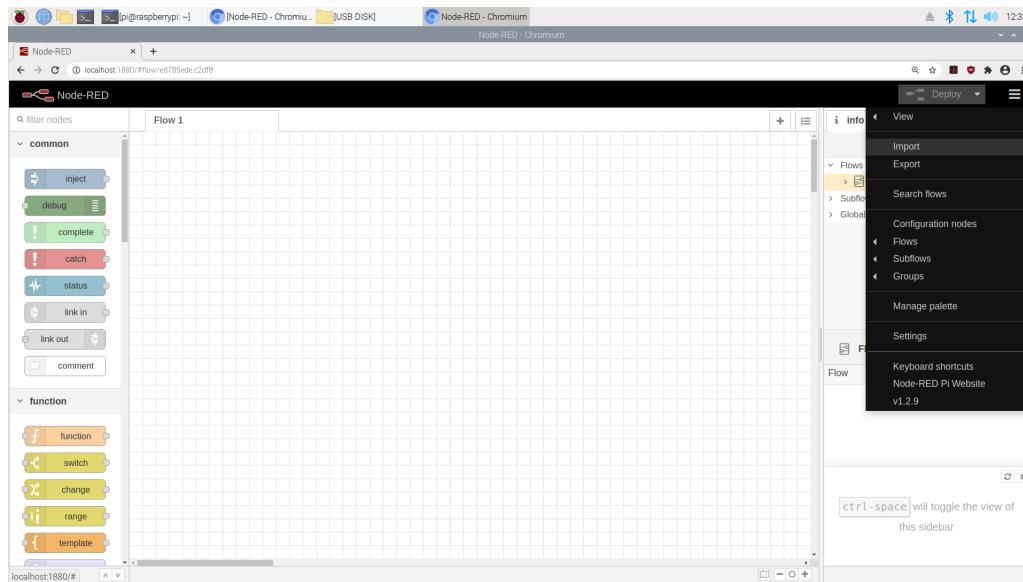


Figura 12.16 – Menú - Importar

En el menú de importar, se selecciona el archivo. Este archivo se encuentra dentro de la carpeta de configuración y se denomina *Flow_Planta.json*.

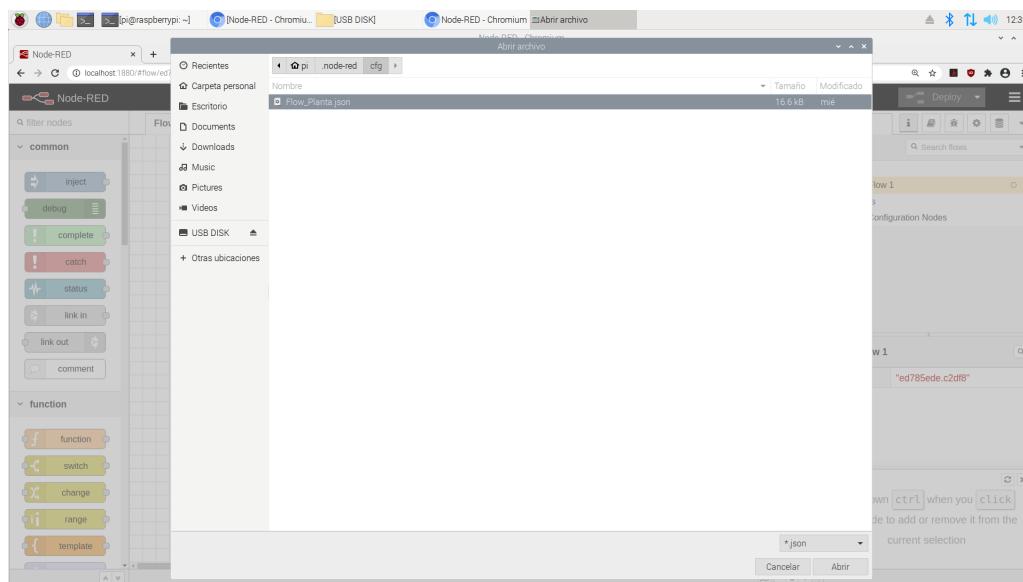


Figura 12.17 – Seleccionar archivo planta emulada

Una vez seleccionado, se pulsa en *abrir*. Esto nos abre la ventana de importación en dónde se selecciona *import*.

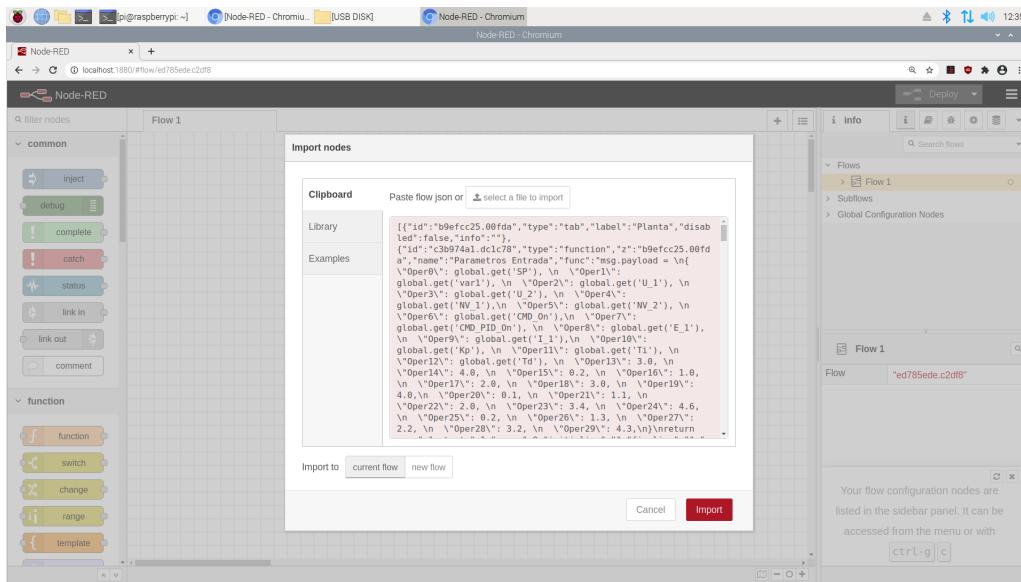


Figura 12.18 – Menú - Importar JSON

Al finalizar la importación se crea un nuevo *flow* en dónde se muestran los nodos de la planta emulada.

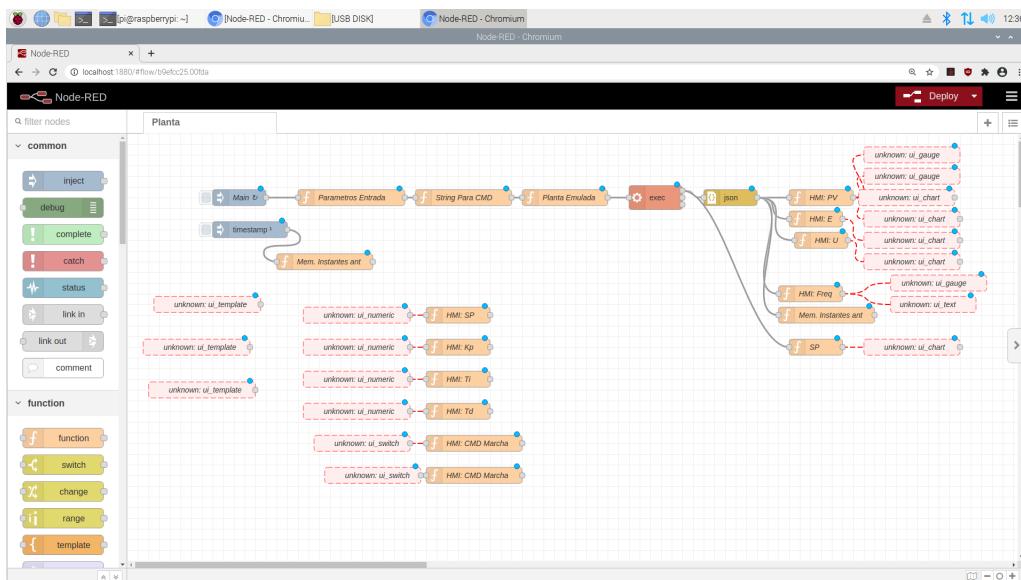


Figura 12.19 – Nodos

Cómo se puede apreciar en la imagen anterior, hay varios nodos que aparecen como no reconocidos. Ésto es debido a que es necesario instalar una *paleta* para que sean reconocidos. Para instalar la paleta, se debe de seleccionar en el menú *Manage Palette*.

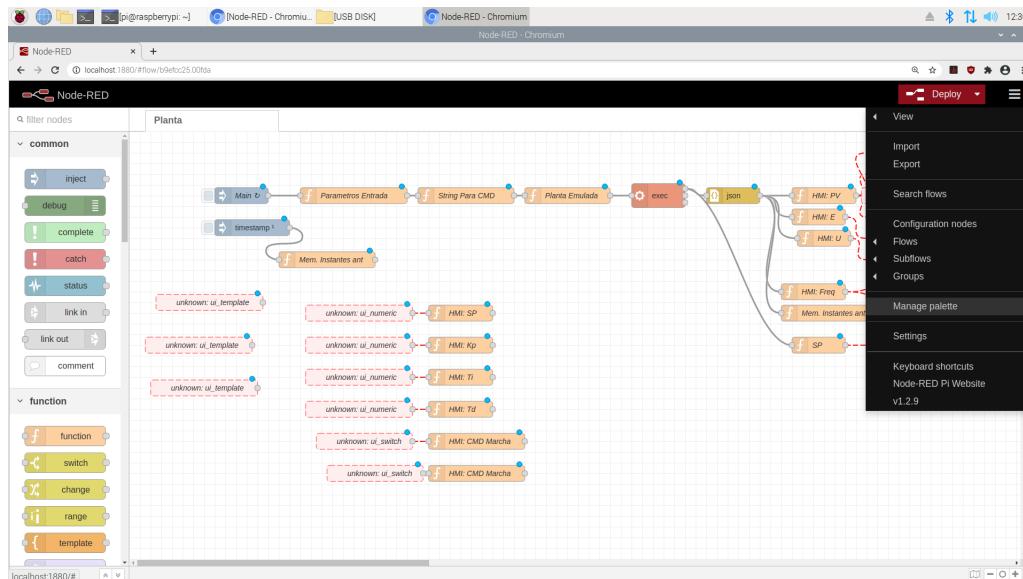


Figura 12.20 – Seleccionar instalación de paleta

En la ventana de *install*, hay que buscar la paleta *node-red-dashboard* y pulsar en *install*.

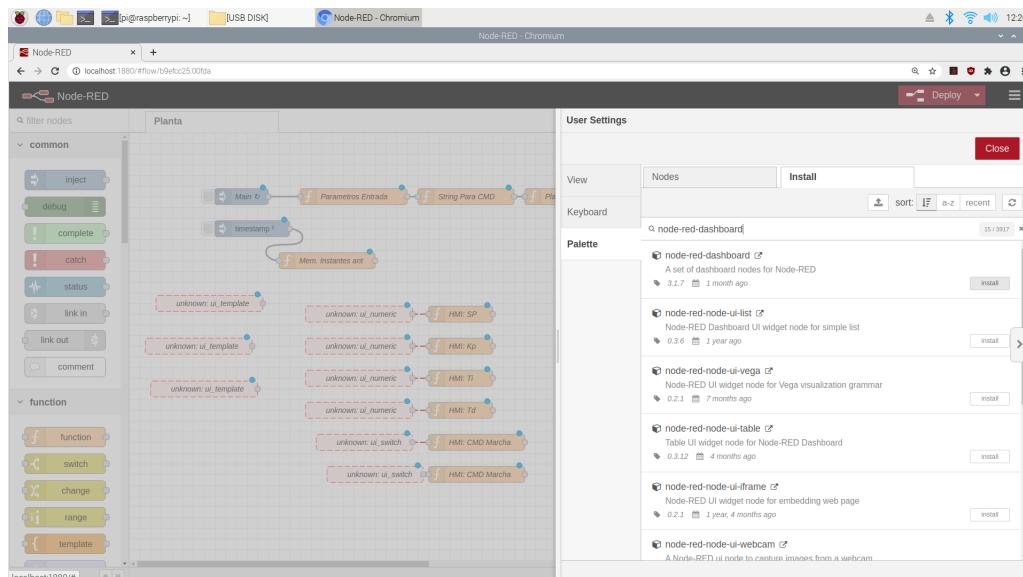


Figura 12.21 – Búsqueda de paquete

Se pide una segunda confirmación de instalar que debe de ser aceptada pulsando en *Install*.

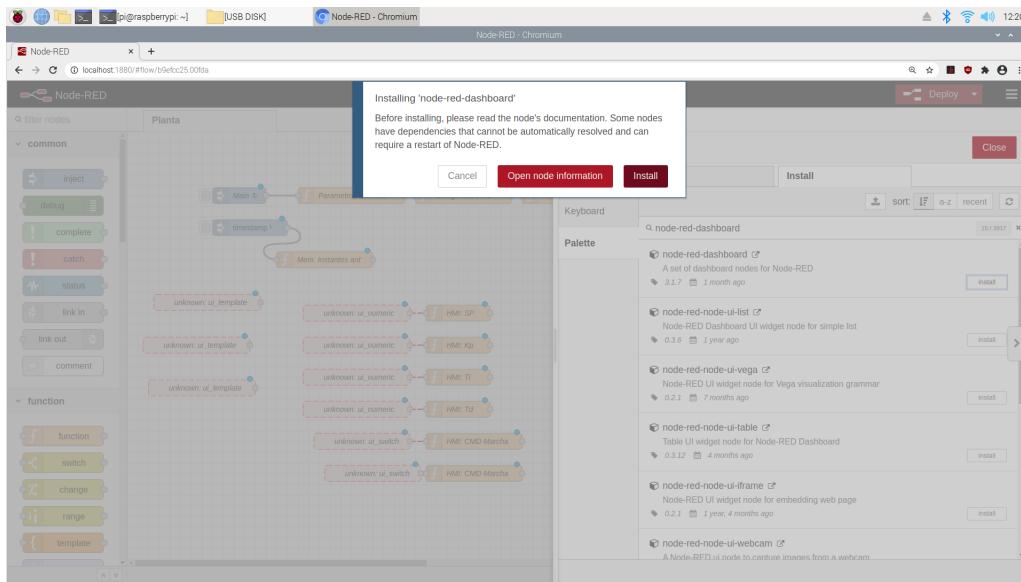


Figura 12.22 – Instalación de paleta

Tras finalizar la instalación de la paleta, todos los nodos son reconocidos. Pero algunos de ellos se muestran con un triángulo rojo.

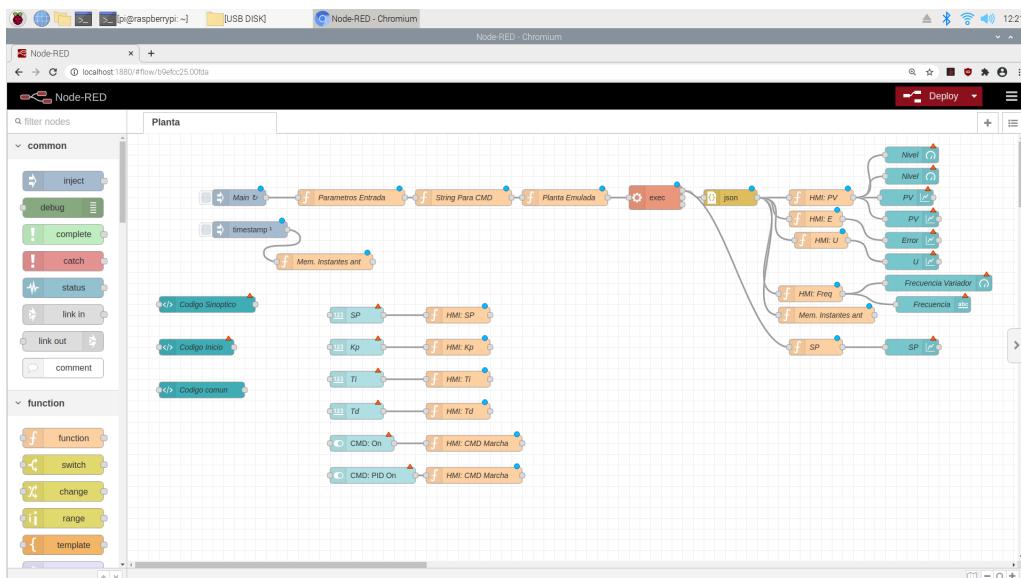


Figura 12.23 – Nodos reconocidos

Ésto es debido a que la paleta que contiene estos nodos acaba de ser instalada y por lo tanto node-red no asume la configuración automáticamente. Es necesario entrar en el interior de cada uno de los nodos y aceptar su configuración.

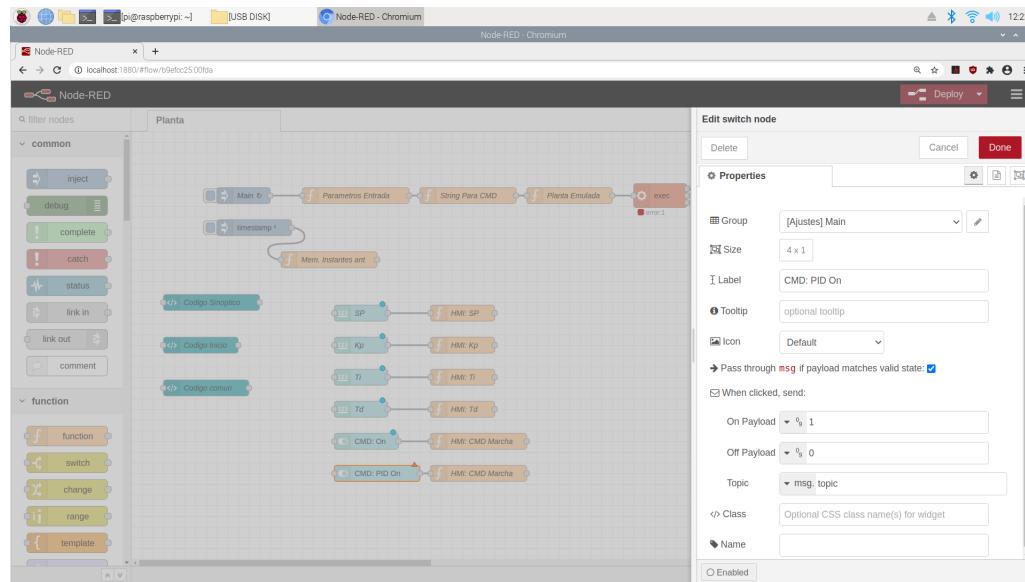


Figura 12.24 – Aceptar configuración nodos no configurados

Una vez realizados estos pasos, se deben de compilar todos los nodos en el botón de *Deploy*.

Para comprobar si la aplicación se está ejecutando correctamente, se debe de acceder al *dashboard* a través de la url *localhost:1880/ui* y ver que se muestra el interface de la planta emulada.

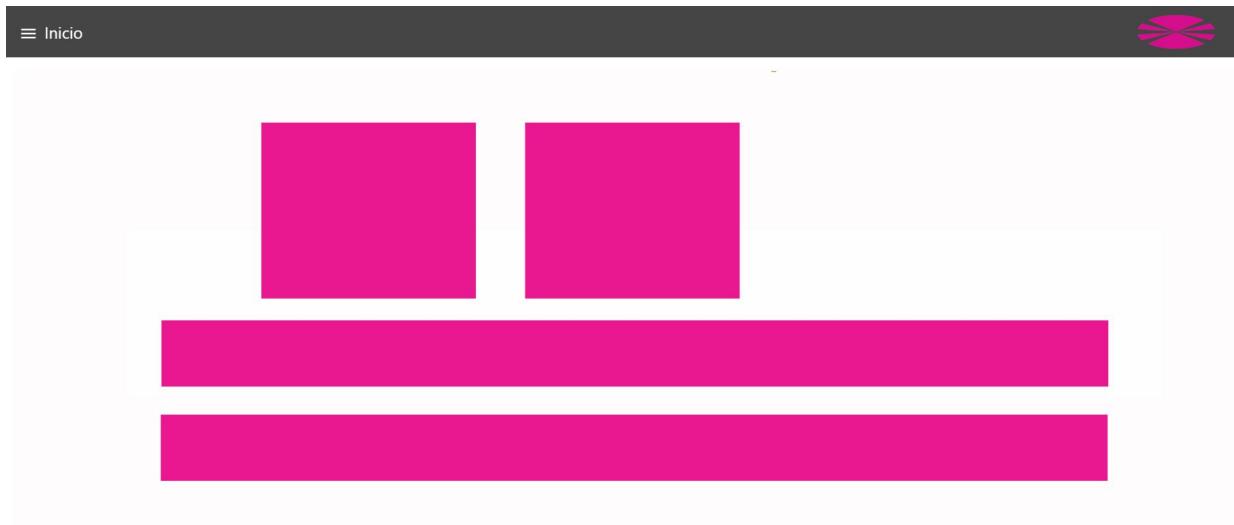


Figura 12.25 – Configuración correcta

12.2. Funcionamiento de la emulación

En el presente sección se explica los pasos que debe de seguir el usuario para poner en funcionamiento el emulador de la planta de nivel.

Al iniciar la aplicación se muestra la pantalla de inicio.



Figura 12.26 – Manual Funcionamiento: Pantalla de inicio

En la esquina superior izquierda se encuentra el menú de navegación en dónde se debe de seleccionar la página de configuración:



Figura 12.27 – Manual Funcionamiento: Menú de navegación

En este menú se deben de llenar en primer lugar los parámetros de consigna, contante proporcional, tiempo de integración y tiempo derivativo del regulador.

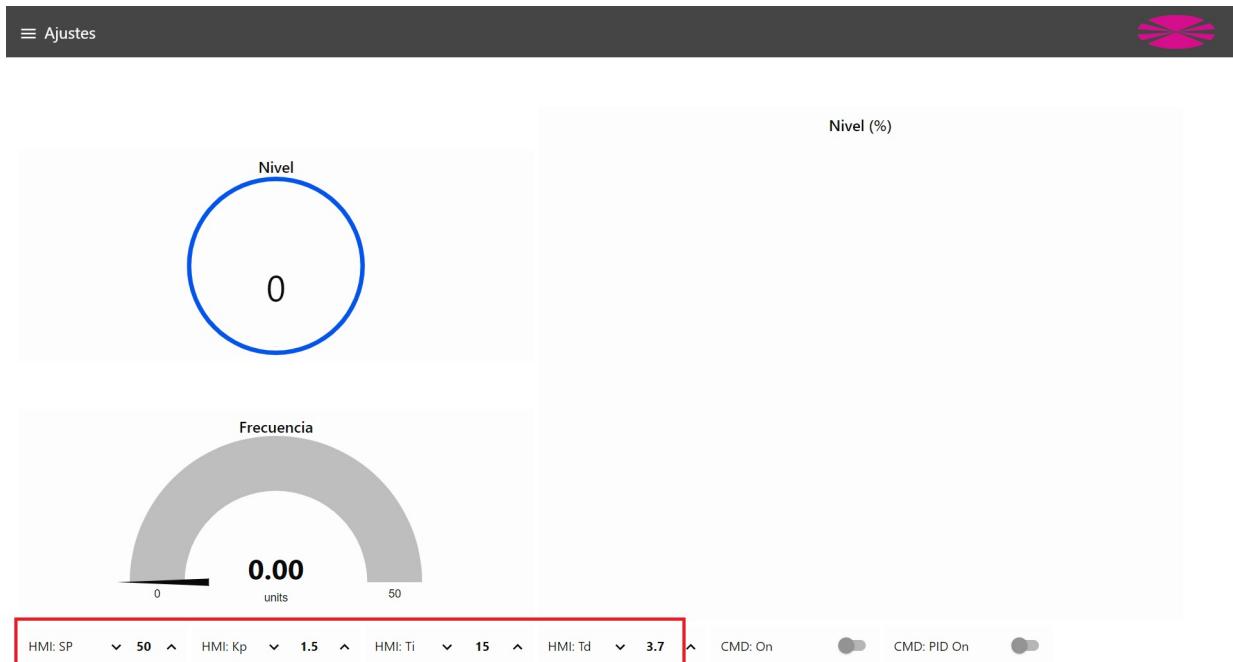


Figura 12.28 – Manual Funcionamiento: Ajuste de consigna y parámetros regulador

A continuación se debe de habilitar el regulador y después dar el comando de marcha.

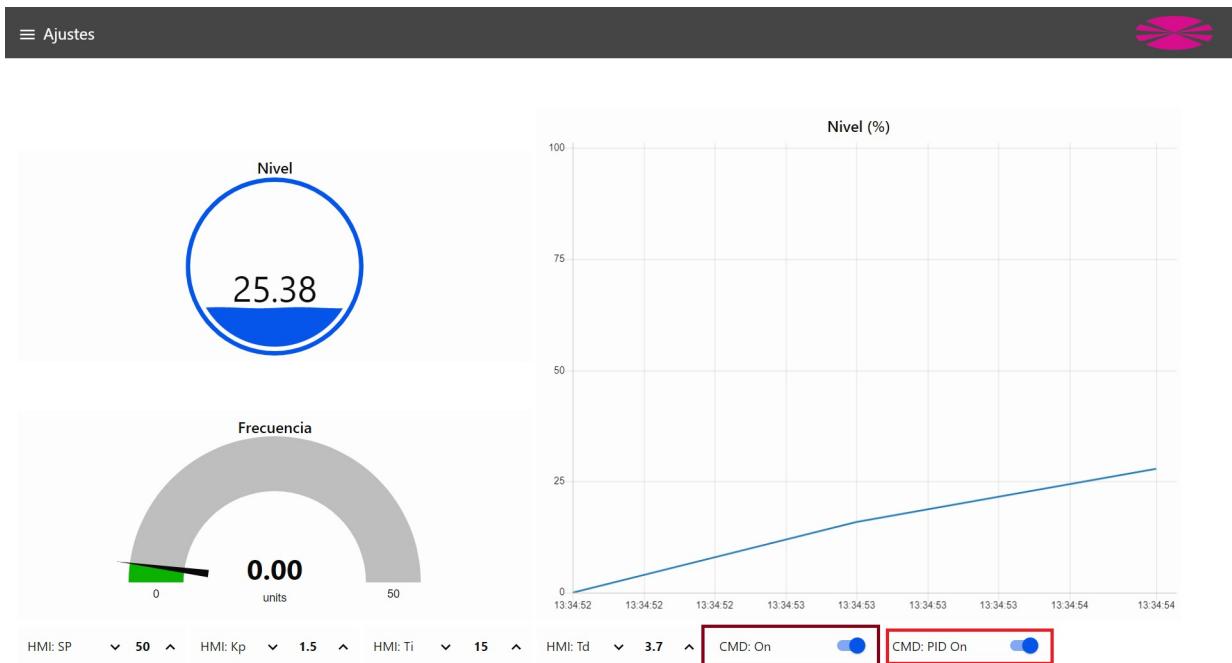


Figura 12.29 – Manual Funcionamiento: Órden de marcha

En la página del sinóptico se puede ver el layout de la planta emulada.

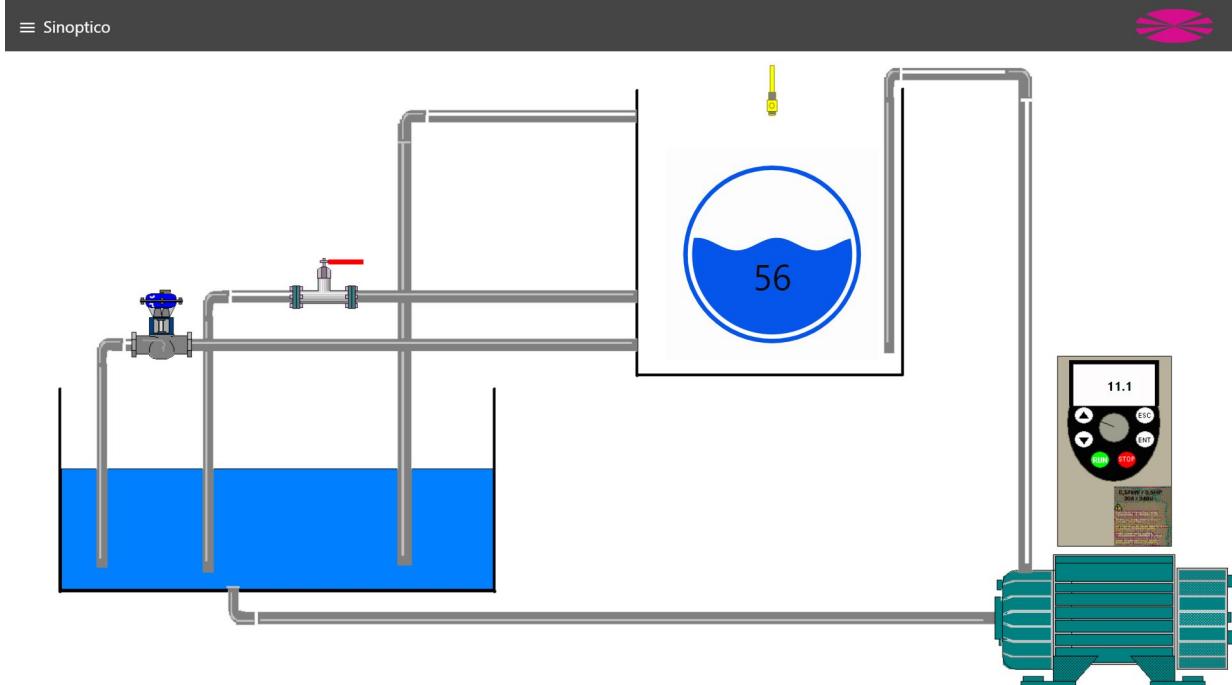


Figura 12.30 – Manual Funcionamiento: Sinóptico

En la página de gráficas se puede ver el comportamiento de la señal de control, el error, la consigna y el nivel.

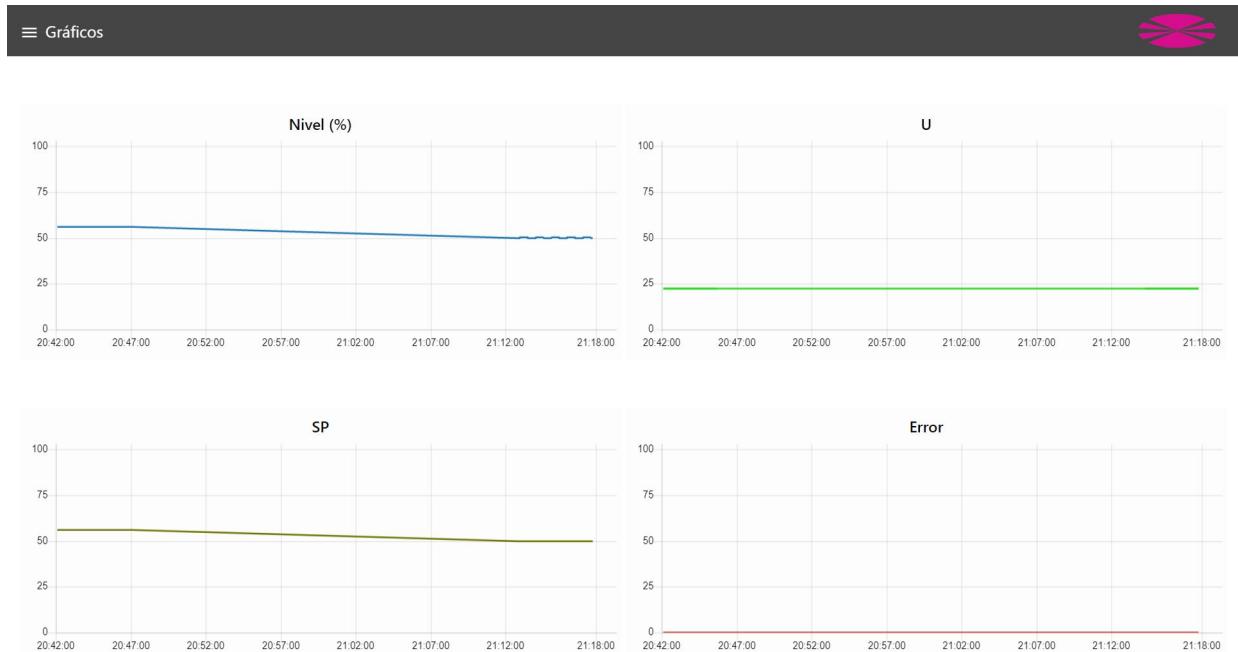


Figura 12.31 – Manual Funcionamiento: Gráficas

13 Códigos de Programación

13.1. Planta.py

13.1.1. Main

```
if __name__=="__main__":
    Entradas()
    Parametros_Planta()
    PID()
    Planta()
    freq_variador=Display_variador()
    out={ 'PV':NV, 'Freq':freq_variador , 'E':E, 'U_1':U_1 , 'U_2':U_2 , 'NV_1':NV_1 , 'NV_2':NV_2 , 'E_1':E_1 , 'I_1':I_1 }
    out_parseado=json.dumps(out)
    print(out_parseado)
```

13.1.2. Entradas

```
def Entradas():
    global SP,var1,U_1,U_2,NV_1,NV_2,CMD_start,CMD_PID_On,E_1,I_1,Kp,Ti,Td,var14,
           var15,var16,var17,var18,var19,var20,var21,var22,var23,var24,var25,var26,
           var27,var28,var29

    SP=float(sys.argv[1])
    var1=float(sys.argv[2])
    U_1=float(sys.argv[3])
    U_2=float(sys.argv[4])
    NV_1=float(sys.argv[5])
    NV_2=float(sys.argv[6])
    CMD_start=float(sys.argv[7])
    CMD_PID_On=float(sys.argv[8])
    E_1=float(sys.argv[9])
    I_1=float(sys.argv[10])
    Kp=float(sys.argv[11])
    Ti=float(sys.argv[12])
    Td=float(sys.argv[13])
    var14=float(sys.argv[14])
    var15=float(sys.argv[15])
    var16=float(sys.argv[16])
    var17=float(sys.argv[17])
    var18=float(sys.argv[18])
```

```
var19=float(sys.argv[19])
var20=float(sys.argv[20])
var21=float(sys.argv[21])
var22=float(sys.argv[22])
var23=float(sys.argv[23])
var24=float(sys.argv[24])
var25=float(sys.argv[25])
var26=float(sys.argv[26])
var27=float(sys.argv[27])
var28=float(sys.argv[28])
var29=float(sys.argv[29])
```

13.1.3. Parámetros Planta

```
def Parametros_Planta():
    global CA1,CA2,CA3,CB1,CB2,CB3
    if SP>=75:
        CA1=1
        CA2=-0.9815
        CA3=0.0038
        CB1=0
        CB2=0.452
        CB3=0.0215
    elif SP>=65:
        CA1=1
        CA2=-0.9772
        CA3=0.0018
        CB1=0
        CB2=0.0455
        CB3=0.0226
    elif SP>=55:
        CA1=1
        CA2=-0.9765
        CA3=0.0035
        CB1=0
        CB2=0.045
        CB3=0.0229
    elif SP>=45:
        CA1=1
        CA2=-0.9708
        CA3=0.0019
        CB1=0
        CB2=0.0445
        CB3=0.0248
    elif SP>=30:
        CA1=1
        CA2=-0.9601
        CA3=-0.0029
```

CB1=0 CB2=0.0444 CB3=0.0261

13.1.4. Regulador PID

```
def PID():
    global E, I_1 , E_1 , U, I

    if CMD_start==1 and CMD_PID_On==1:
        E=SP-NV_1
        I=I_1 +(E_1+E)/2
        U=Kp*E+Kp*(Td/1)*(E-E_1)+Kp*(1/Ti)*I

        E_1=E
        I_1=I
    else:
        E=0
        E_1=0
        I=0
        I_1=0
        U=0
```

13.1.5. Planta

```
def Planta():
    global NV, U_1 , U_2 , NV_1 , NV_2

    if CMD_start==1:
        U_B=U*CB1+CB2*U_1+CB3*U_2
        NV_A=CA2*NV_1+CA3*NV_2
        NV=(U_B-NV_A)/CA1

        if NV>100:
            NV=100
        elif NV<0:
            NV=0

        U_2=U_1
        U_1=U
        NV_2=NV_1
        NV_1=NV
    else:
        U_B=0
        NV_A=0
        NV=0
```

```
U_2=0  
U_1=0  
NV_2=0  
NV_1=0
```

13.1.6. Frecuencia Variador

```
def Display_variador ():  
    if U>100:  
        freq_variador=50  
    elif U<0:  
        freq_variador=0  
    else:  
        freq_variador=U/2  
    return(freq_variador)
```

13.1.7. Paso a JSON

```
def to_json(x):  
    y = json.dumps(x)  
  
    return(y)
```

14 Variables Scripts

14.1. Identificación

En el grupo de variables de proceso se encuentran:

- u: Señal de control
- integral: Valor de la parte integral del regulador PID
- integral_ant: Valor de la parte integral del regulador PID en el instante anterior
- derivada: Valor de la parte derivativa del regulador PID
- error: Valor de la diferencia entre la consigna y la variable del proceso
- error_ant: Valor de la diferencia entre la consigna y la variable del proceso en el instante anterior
- pv: Variable del proceso
- pv_ant: Variable del proceso en el instante anterior
- datos_PRBS: Matriz de almacenamiento
- datos_PRBS_aux: Matriz de almacenamiento auxiliar
- cont_save: Contador de guardado de datos

En el grupo de variables de configuración se encuentran:

- sp: Consigna
- T: Periodo de muestreo
- Kp: Constante proporcional regulador PID
- Ti: Constante integral regulador PID
- Td: Constante derivativa regulador PID
- t_estable: Tiempo para considerar tiempo estable
- t_save: Tiempo para iniciar el guardado de datos
- ruta: Ruta para el guardado de la matriz de datos

14.2. Comprobación

En el grupo de variables de proceso se encuentran:

- u: Señal de control
- sp_ar: Variable para graficar el SP
- sp_cte: Variable auxiliar del SP
- integral: Valor de la parte integral del regulador PID
- error: Valor de la diferencia entre la consigna y la variable del proceso
- derivada: Valor de parte derivativa del regulador PID
- lectura: Variable del proceso de la planta real
- NV: Variable del proceso de la planta emulada
- datos_PRBS_aux: Matriz de almacenamiento auxiliar
- cont: Contador sistema estabilizado
- U_B: Señal de control multiplicada por CB1
- NV_A: Variable de proceso emulada en el instante anterior multiplicado por CA2

En el grupo de variables de configuración se encuentran:

- sp: Consigna
- T: Periodo de muestreo
- Kp: Constante proporcional regulador PID
- Ti: Constante integral regulador PID
- Td: Constante derivativa regulador PID
- CA1: Parámetro grado dos del parámetro A de arx
- CA2: Parámetro grado uno del parámetro A de arx
- CA3: Parámetro grado cero del parámetro A de arx
- CB1: Parámetro grado dos del parámetro B de arx
- CB2: Parámetro grado uno del parámetro B de arx
- CB3: Parámetro grado cero del parámetro B de arx