UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC

AUTOR: JOSÉ RIBAS OLTRA

TUTOR: JUAN MANUEL HERRERO DURÁ

Curso Académico: 2017-18

AGRADECIMIENTOS

En este momento tan especial, quiero agradecer a todas aquellas personas que me han apoyado durante todos estos años. En particular, a mis padres que siempre han estado apoyándome en todo momento y que sin ellos, no hubiera sido posible realizar este hecho. Asimismo, agradecer a mi tutor toda la ayuda suministrada durante estos últimos meses, que ha sido imprescindible para el desarrollo satisfactorio del presente Trabajo Final de Máster (TFM). A todos ellos les doy las gracias.

RESUMEN

El presente TFM tiene por objetivo el desarrollo de una plataforma que sea capaz de realizar el control multivariable de procesos industriales mediante técnicas DMC. Esta plataforma está compuesta por un dispositivo empotrado de control y una serie de aplicaciones.

El dispositivo de control, se encuentra empotrado en una unidad que está formada por una Raspberry Pi y una tarjeta de adquisición de datos (TAD). El desarrollo de la TAD, que se encarga de obtener toda la información relevante del proceso a controlar, es también parte de este TFM.

La plataforma viene acompañada de dos aplicaciones que se encargan de implementar y configurar el sistema. La primera es la aplicación de control que se ejecuta en la Raspberry Pi y realiza el control mediante técnicas DMC. La siguiente aplicación es la interfaz gráfica remota que gestiona la configuración de la aplicación de control. Ambas aplicaciones se comunican mediante el protocolo de comunicación industrial Modbus TCP. El uso de dicho protocolo permite la configuración del sistema desde dispositivos/aplicaciones de terceros.

Para poder llevar a cabo la prueba de concepto de la plataforma se ha desarrollado un simulador Hardware-in-the-Loop (HIL) de un proceso industrial multivariable. En concreto el HIL simula una columna de destilación de aceite pesado de la compañía Shell Oil.

Las pruebas de concepto llevadas a cabo ponen de manifiesto el correcto funcionamiento de la plataforma desarrollada. Por tanto, estamos ante una solución eficiente y de bajo coste que permite el control DMC multivariable de procesos industriales.

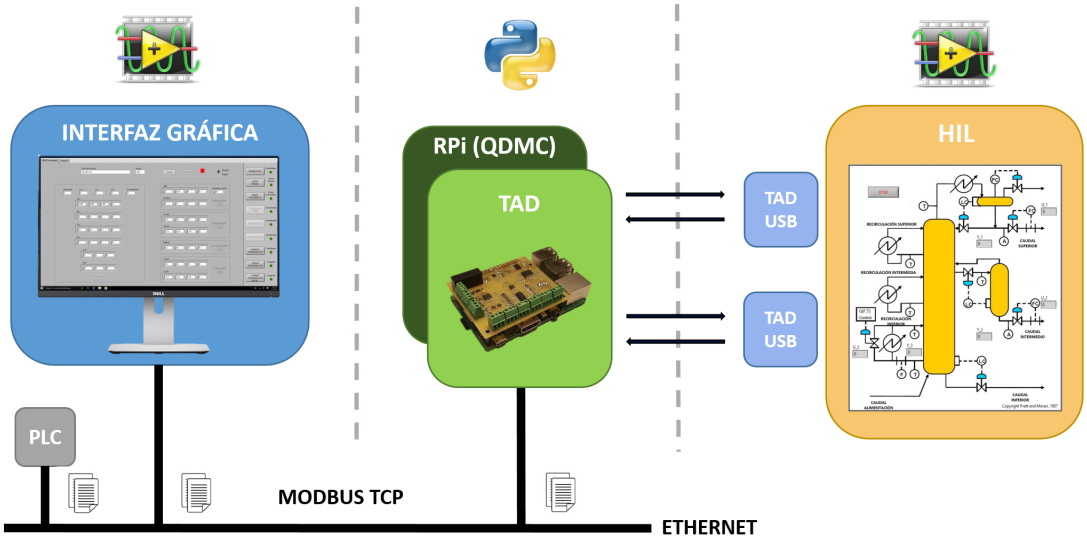


Figura 1: Estructura general del proyecto

Palabras clave: Raspberry Pi (RPi), Modbus TCP, TAD, HIL, Python, LabVIEW, aplicación, ethernet, I2C, DAC, ADC, digital expander, QDMC, DMC, MPC.

RESUM

El present TFM té per objectiu el desenvolupament d´una plataforma que siga capaç de realitzar el control multivariable de processos industrials mitjançant tècniques DMC. Aquesta plataforma està composta per un dispositiu encastat de control i una sèrie d´aplicacions.

El dispositiu de control, es troba encastat en una unitat que està formada per una Raspberry Pi i una targeta d´adquisició de dades (TAD). El desenvolupament de la TAD, que s´encarrega d´obtindre tota l´informació rellevant del procés a controlar, és també part d´aquest TFM.

La plataforma ve acompanyada de dues aplicacions que s'encarreguen d'implementar i configurar el sistema. La primera és l'aplicació de control que s'executa a la Raspberry Pi i realitza el control mitjançant tècniques DMC. La següent aplicació és la interfície gràfica remota que gestiona la configuració de l'aplicació de control. Les dues aplicacions es comuniquen mitjançant el protocol de comunicació industrial Modbus TCP. L'ús d'aquest protocol permet la configuració del sistema des de dispositius / aplicacions de tercers.

Per poder dur a terme la prova de concepte de la plataforma s'ha desenvolupat un simulador Hardware-in-the-Loop (HIL) d'un procés industrial multivariable. En concret el HIL simula una columna de destil·lació d'oli pesat de la companyia Shell Oil.

Les proves de concepte dutes a terme posen de manifest el correcte funcionament de la plataforma desenvolupada. Per tant, estem davant d'una solució eficient i de baix cost que permet el control DMC multivariable de processos industrials.

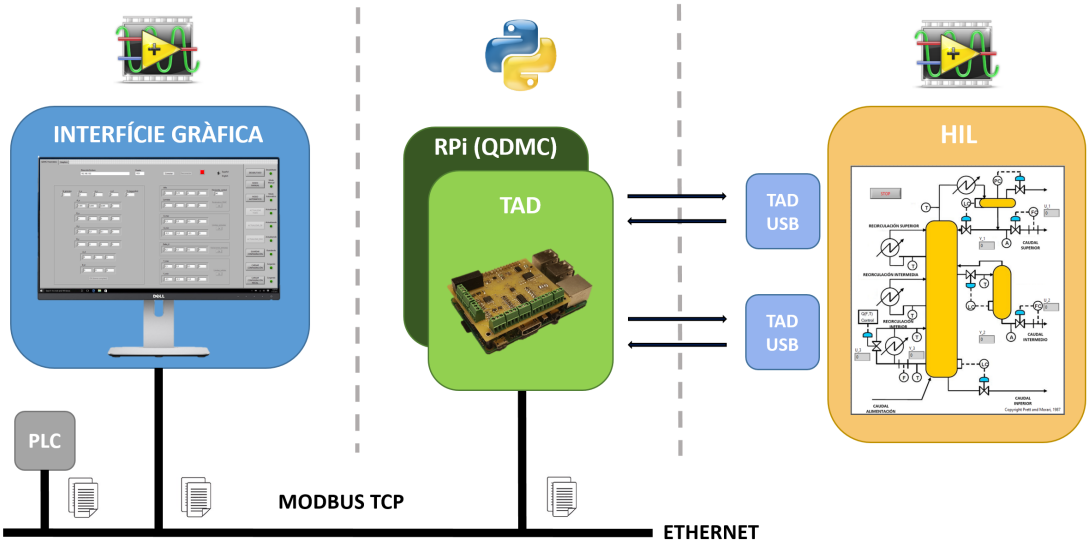


Figura 2: Estructura general del projecte

Paraules clau: Raspberry Pi (RPi), Modbus TCP, TAD, HIL, Python, LabVIEW, aplicació, ethernet, I2C, DAC, ADC, digital expander, QDMC, DMC, MPC.

ABSTRACT

The purpose of this TFM is to develop a platform that is capable of performing multivariable control of industrial processes using DMC techniques. This platform is composed of an embedded control device and some applications.

The control device is embedded in a control unit composed by a Raspberry Pi and a data acquisition card (DAQ). The development of the DAQ, which is responsible for obtaining all relevant information of the process to control, is also part of this TFM.

The platform has two applications which are in charge of implementing and configuring the system. The first is the control application that is executed in the Raspberry Pi and realizes the control through the DMC technics. The next application is the remote graphical interface that manages the configuration of the control application. Both applications communicate through the industrial communication protocol Modbus TCP. The use of this protocol allows to configure the system from other devices/applications.

In order to carry out the concept proof of the platform, a Hardware-in-the-Loop (HIL) simulator of a multivariable industrial process has been developed. Specifically, the HIL simulates a heavy oil distillation column of Shell Oil Company.

The tests carried out demonstrate the correct functioning of the developed platform. Therefore, we are faced with an efficient and low-cost solution that allows multivariable DMC control of industrial processes.

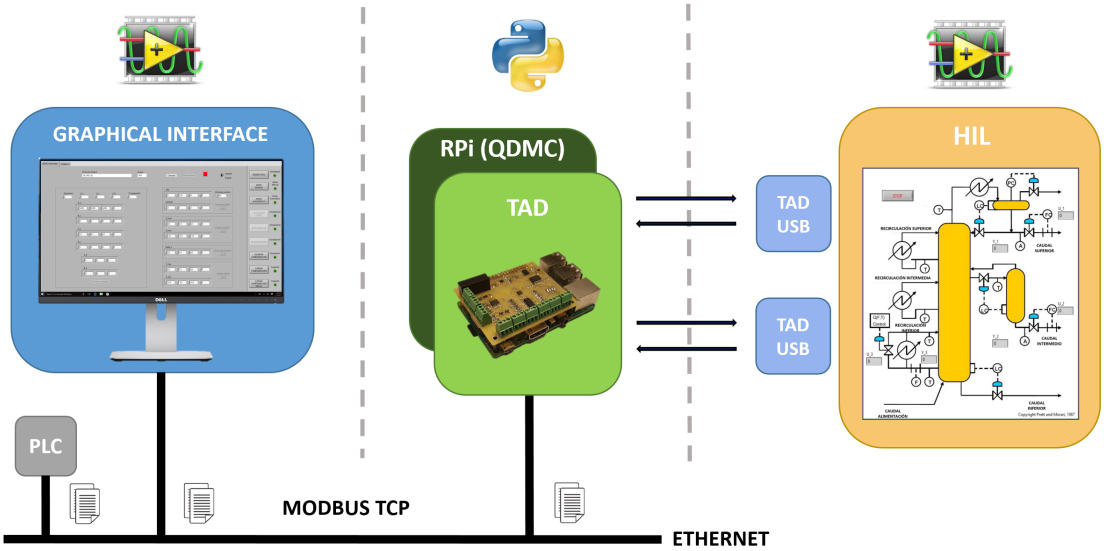


Figura 3: General structure of the project

Keywords: Raspberry Pi (RPi), Modbus TCP, TAD, HIL, Python, LabVIEW, applications, ethernet, I2C, DAC, ADC, digital expander, QDMC, DMC, MPC.

DOCUMENTOS CONTENIDOS EN EL TRABAJO FINAL DE MÁSTER:

- Memoria del proyecto
- Anexo 1: Manual de diseño de la tarjeta de adquisición de datos para la Raspberry Pi
- Anexo 2: Manuales de usuario de las aplicaciones diseñadas
- Anexo 3: Manuales de programación de la interfaz gráfica y del hardware-in-the-loop
- Anexo 4: Manuales de programación de las aplicaciones implementadas en la Raspberry Pi
- Presupuesto del proyecto

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC:

MEMORIA DEL PROYECTO

ÍNDICE:

ÍNDICE DE FIGURAS:

1. INTRODUCCIÓN

Este documento conforma la memoria del Trabajo de Final de Máster titulado "Desarrollo de un dispositivo basado en Raspberry PI para el control multivariable de procesos mediante técnicas DMC" de la titulación Máster Universitario en Ingeniería Industrial, cursado en la Escuela Técnica Superior de Ingenieros Industriales, de la Universidad Politécnica de Valencia desde el año 2016 hasta el año 2018.

A lo largo de este documento, se pretende mostrar todo el trabajo realizado sobre el desarrollo de un dispositivo basado en Raspberry Pi para el control multivariable de procesos mediante técnicas DMC (Dynamic Matrix control). Para ello, se tratarán los diferentes elementos que se han visto involucrados en el desarrollo y posterior puesta en marcha del dispositivo en cuestión.

El presente proyecto se ha desarrollado entre los meses de febrero hasta finales de agosto, donde se han empleado una cantidad sutilmente superior a las 300 horas, para realizar todas las partes que lo conforman. Estas horas se han empleado tanto en desarrollar el dispositivo de control como a recoger y documentar toda la metodología seguida a lo largo del mismo y el posterior análisis de los resultados obtenidos.

Finalmente, destacar que para conseguir desarrollar correctamente todo el proyecto, ha sido necesario establecer una serie de objetivos y especificaciones técnicas desde el primer momento que se comenzó a realizar. No obstante, estos han ido variando durante el transcurso del mismo por diferentes motivos que se expondrán en la memoria, eso sí, sin variar el objetivo principal.

1.1. Objetivo y alcance del proyecto

El objetivo principal del proyecto en sí, es el de realizar el desarrollo de un dispositivo basado en Raspberry Pi para el control multivariable de procesos mediante técnicas DMC. No obstante, para realizar este hecho es necesario definir una serie de objetivos de un nivel inferior, para ir poco a poco consiguiendo el principal objetivo del proyecto. A continuación, se van a enumerar estos objetivos:

- Desarrollo de un controlador de bajo coste.
- Desarrollo integral de una tarjeta de adquisición de datos que se conecte mediante el puerto de expansión de la Raspberry PI. Esta debe disponer de 4 entradas y 4 salidas de carácter analógico y además, de entradas/salidas de tipo digital. Asimismo, debe poseer unas dimensiones semejantes a la RPi.
- Programación de un algoritmo de control de tipo DMC. Este se debe desarrollar, por especificación del cliente, en el lenguaje de programación de Python.
- Capacidad del sistema de realizar la monitorización del proceso y la configuración del algoritmo tipo DMC de forma remota mediante el protocolo de comunicación industrial Modbus TCP, por especificación del cliente.
- Programación de la aplicación de monitorización, configuración y gestión de forma remota en LabVIEW. También por especificación del cliente.
- Programación en Python del automatismo que se encarga de gestionar los modos de funcionamiento del controlador y las variables procedentes del protocolo Modbus.
- Programación del cliente y del servidor del protocolo Modbus TCP.

- Capacidad de almacenamiento del controlador de la última configuración deseada por el usuario.
- Prueba de concepto de todo el hardware/software diseñado, mediante el desarrollo de un sistema tipo Hardware-in-the-loop (HIL) que simule el proceso a control.

Por tanto, el alcance del proyecto conlleva el desarrollo tres aplicaciones para realizar tanto el dispositivo de control como su posterior puesta a prueba. Para ello, la primera aplicación es la que se ejecuta en la Raspberry Pi, que se encarga de recoger y mandar toda la información de la TAD, calcular la acción de control óptima en cada momento, gestionar el controlador y actualizar las variables del protocolo Modbus. La segunda, consiste en una interfaz gráfica cuya función es realizar la monitorización y configuración de forma remota del algoritmo de control. Finalmente, la última aplicación implementará el HIL que se encarga de la simulación vía software del proceso a controlar. Las dos primeras aplicaciones necesitan una conexión entre ellas de tipo Ethernet, donde es necesario establecer la dirección IP del servidor y un puerto de conexión. Por otro lado, las dos últimas necesitan una conexión cableada entre las diversas TAD que componen el sistema. El alcance llega hasta la prueba del sistema, el cual debe realizar correctamente el control de un determinado proceso, y la posterior redacción de la presente memoria.

1.2. Motivaciones

La principal motivación que me ha llevado a escoger este proyecto es que, a lo largo del mismo, se han usado conocimientos de las áreas que más me apasionan, la electrónica y el control de procesos. Por este motivo, decidí cursar durante este año la especialización de control, automatización y robótica, ya que considero es el complemento perfecto para finalizar mis estudios.

Otro de los principales motivos, es que para este proyecto era necesario manejar la Raspberry Pi, la cual nunca antes había usado ni programado y que actualmente, se usa en infinidad de aplicaciones. Asimismo, el hecho de diseñar desde cero una TAD, tampoco lo había realizado nunca y me resultaba muy atractivo en el momento de la selección de este proyecto. Por otro lado, la comunicación de tipo Modbus, se había abordado de forma teórica durante la especialización pero no se había llevado a la práctica. Finalmente, el hecho de aprender a manejar nuevos programas como DesignSpark o lenguajes de programación como Python, me hicieron motivarme aún más para realizar este proyecto.

Finalmente, otra motivación ha sido el hecho de desarrollar un dispositivo de bajo coste que permite implementar un control mediante técnicas DMC para procesos multivariables y que además, puede emplearse en las prácticas de aula de la especialidad del DISA (Dpto. de Ingeniería de Sistemas y Automática), donde se ha desarrollado este proyecto.

En definitiva, este proyecto me ha dado la oportunidad de afianzar y adquirir muchos conocimientos que no se habían visto con gran profundidad a lo largo de mi titulación, pero que a la vez son muy útiles y valorados en el mundo industrial, lo cual es una gran motivación para mi inminente futuro.

1.3. Estructura del documento

Este documento está formado por tres partes principales, la memoria, los anexos y el presupuesto del proyecto. La memoria está formada por serie de capítulos, los cuales relatan todo el trabajo realizado a lo largo del mismo. En primer lugar, esta narra la parte teórica en la que se ha basado la realización de todo el sistema de control, desde el algoritmo de control hasta el hardware comercial empleado. Seguidamente, se prosigue con el desarrollo de la TAD diseñada y las diversas aplicaciones que hacen posible el correcto funcionamiento de todas las funciones del dispositivo de control. Finalmente, se aborda la prueba de todo el sistema mediante el control de un determinado proceso industrial.

La segunda parte está compuesta por cuatro anexos que explican el diseño, programación y metodología de uso del sistema control. El primero de ellos, corresponde con el manual de diseño de la TAD, donde se explica la metodología seguida para su diseño. Seguidamente, el manual de programación de la aplicación de control en la Raspberry Pi y de la aplicación que actúa como interfaz gráfica, donde se explica cómo se han programado ambas aplicaciones. Finalmente, el último anexo consta del manual de usuario de la aplicación de control completa.

Por último, el documento del presupuesto que se encarga de recoger todo el desembolso económico de la realización del proyecto y la posterior amortización del mismo.

2. ESTRUCTURA DEL PROYECTO

El presente proyecto está formado por diversas partes (ver Figura 1), las cuales se explicarán a continuación. En primer lugar, se va describir la parte que actúa como interfaz gráfica cuya función es realizar la configuración remota del algoritmo de control y de monitorizar las variables de interés. Esta aplicación se ha programado mediante LabVIEW y se ejecuta en un ordenador común para su correcto funcionamiento.

Por otro lado, se tiene la parte encargada de realizar toda la gestión y control del proceso que se desea controlar, cabe destacar que en este caso se ha recurrido a las técnicas DMC (Dynamic Matrix Control) como algoritmo de control. Este software se ejecuta en la Raspberry Pi, que actúa como controlador y además, cuenta con una tarjeta de adquisición de datos. Esta se encarga de obtener las medidas de las salidas del proceso, así como de proporcionar las acciones de control pertinentes, para realizar el control del mismo. Asimismo, en este dispositivo también se ejecuta el software correspondiente a la parte que actúa como servidor del protocolo de comunicación Modbus TCP.

El último elemento que conforma el proyecto, consiste en el proceso que se desea controlar. Lo ideal hubiese sido realizar el control sobre una planta real pero no se dispone de los medios necesarios para realizar este hecho. Por este motivo, se ha recurrido a realizar un Hardware-in-the-Loop (HIL) que se encarga de realizar la simulación de un proceso a partir de las funciones de transferencia pertinentes del mismo. El HIL se ha implementado en LabVIEW y para su correcto funcionamiento es necesario el uso de tarjetas de adquisición de datos, tanto para recibir como mandar la información correspondiente y necesaria para el correcto funcionamiento de la aplicación.

La relación entre las diversas partes es la siguiente. La aplicación que actúa como interfaz gráfica se comunica con la Raspberry Pi (RPi) para realizar la correspondiente configuración del programa encargado de realizar el control del proceso, mediante el protocolo de comunicación industrial Modbus TCP, para el cual únicamente es necesario una conexión internet de tipo Ethernet/WIFI para ambos dispositivos.

Una vez realizada la configuración, la Raspberry Pi comenzará a realizar el control del proceso mediante la tarjeta de adquisición de datos (TAD) que está directamente cableada a las tarjetas de adquisición de datos que forman parte del HIL para realizar la simulación del proceso.

Finalmente, la RPi recoge toda la información de las variables de interés del proceso y la envía mediante el protocolo Modbus TCP a la interfaz gráfica, que realiza la monitorización de dichas variables.

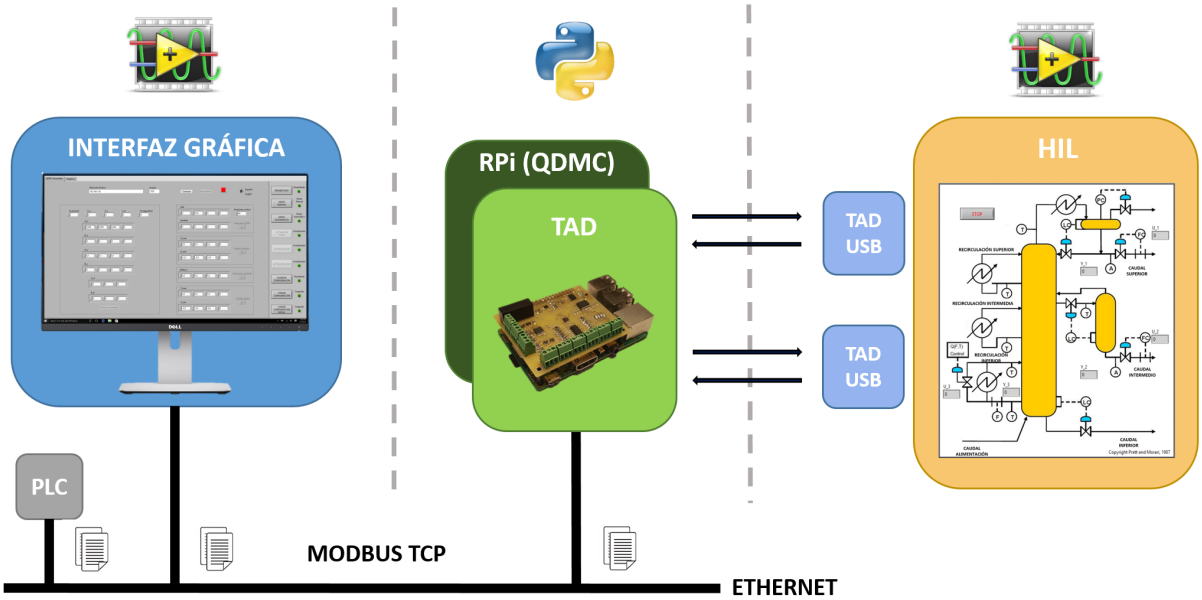


Figura 1: Estructura general del proyecto

3. ANTECEDENTES

3.1. Control Predictivo: QDMC (Quadratic Dynamic Matrix Control)

3.1.1. Introducción al control predictivo

Para esta aplicación se ha decido optar como algoritmo de control las técnicas Quadratic Dynamic Matrix Control (QDMC)[9], las cuales son un tipo concreto de control predictivo de procesos (MPC) que se encarga de utilizar técnicas más potentes que los tradicionales controladores PID [19] especialmente en sistemas mutivariables.

El control predictivo de procesos consiste en un conjunto de técnicas cuyo principal fin es llevar las salidas del proceso a una determinada referencia en el menor tiempo posible. Para ello, el MPC hace uso de los errores entre la referencia y las variables a controlar de los instantes anteriores, y de los valores de estas variables en los futuros instantes (de ahí su carácter de predictivo) mediante la respuesta del sistema ante una determinada entrada a calcular de forma óptima. Para realizar este hecho, el MPC emplea el parámetro denominado horizonte de predicción (p), el cual corresponde con el número de instantes futuros que se desea tener en cuenta para realizar el cálculo de las futuras acciones de control (c). No obstante, también es necesario el uso de una función de coste que proporcione un determinado índice cuyo principal objetivo es minimizarlo para obtener la solución óptima dentro de unas determinadas restricciones que puede incorporar el optimizador. Este optimizador puede poseer diferentes características dependiendo de qué tipo de control predictivo se esté empleando. Una vez resuelto este paso, el optimizador devuelve las acciones de control futuras que se aplicarán al proceso a controlar.

3.1.2. Antecedentes históricos [25]

Los primeros algoritmos de tipo MPC surgieron en la década de los 70, con el fin de realizar el control de las refinerías de Shell Oíl [9]. El primer algoritmo de estas características fue el Model Predictive Heuristic Control (MPHC) presentado por Richalet en 1978. Seguidamente, se presentó el algoritmo denominado Dynamic Matrix Control (DMC), concretamente en 1979 de la mano de Cutler y Ramaske, como una versión mejorada del anterior. La principal diferencia entre ambos es que el primero de ellos necesita la respuesta del modelo de la planta ante una entrada de tipo impulso, y el DMC emplea para sus cálculos la respuesta del modelo ante una entrada de tipo escalón. Finalmente, en la década de los 80 se presentó una nueva versión del DMC, denominada Quadratic Dynamic Matrix Control (QDMC), la cual posee las mismas características que su antecesor, pero añadiendo restricciones al modelo, referentes tanto los valores mínimos y máximos de las acciones de control como a las salidas del proceso. Asimismo, también se añade la restricción de variación máxima de las variables manipuladas. Para la aplicación de control que se va a desarrollar, se va a emplear este algoritmo de control.

Posteriormente, se han desarrollado otros algoritmos de control como el Adersa Identification Command Multivariable o Adersa Predictive Functional Control, los cuales se caracterizan por usar espacio de estados en los modelos de los procesos, pero que no son de gran interés para este caso.

Finalmente destacar que actualmente estas técnicas se usan en una gran variedad de procesos, gracias al aumento de la capacidad computacional de los ordenadores actuales, ofreciendo la posibilidad de realizar el control de procesos más rápidos en comparación con aquellos donde se empleaban inicialmente estas técnicas de control.

3.1.3. Metodología MPC

A continuación, se van a explicar la metodología que emplea el MPC [5] [6] [16] para realizar el control de un determinado proceso. En la Figura 2 se observa el diagrama de bloques genérico que describe el comportamiento del control predictivo.

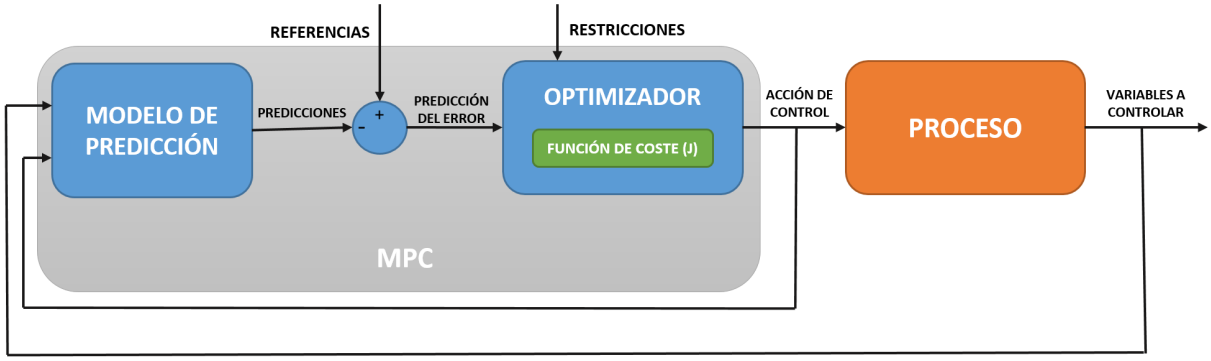


Figura 2: Diagrama de bloques del MPC

El MPC parte del modelo de predicción donde obtiene la predicción del comportamiento del sistema para un número de instantes determinado por el horizonte de predicción. Estos instantes son múltiplos del periodo de muestreo empleado en el bucle de control. Las predicciones obtenidas tienen en cuenta los valores de las acciones de control y de las variables a controlar de los instantes anteriores como de los instantes futuros mediante el modelo de predicción. En la Figura 3 se observa dicho modelo, que está compuesto por la respuesta del modelo del proceso y de las perturbaciones medibles ante unas determinadas acciones de control, y de aquellas perturbaciones no mediables o los posibles errores de modelado cometidos. Cabe destacar que el modelo del proceso como el de las perturbaciones medibles difieren según el algoritmo de control predictivo empleado.

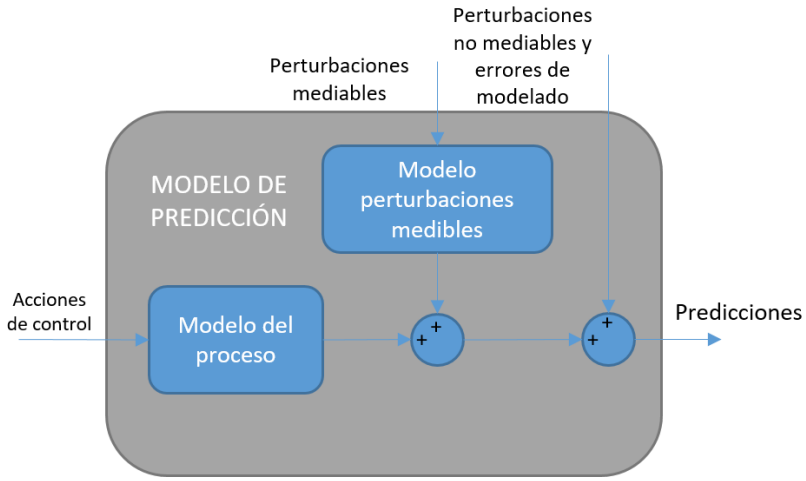


Figura 3: Modelo de predicción

Por otro lado, se define un perfil de referencias, el cual puede describir las futuras trayectorias del proceso o únicamente un valor, pero este debe poseer tantos elementos como el horizonte de predicción (p) indique. Seguidamente, se obtiene la predicción del error para los p instantes, restando el perfil de referencias a las predicciones del modelo. Una vez obtenido, se dispone de la función de coste [16] que penaliza en gran medida los errores de predicción, obligando a que las variables a controlar estén cercanas al valor de la referencia correspondiente.

$$\boldsymbol{J(p,c) = \sum_{j=1}^{p} \alpha(j)\,[\hat{y}(k+j \mid k) - R(k+j)]^2 + \sum_{j=1}^{c} \lambda(j)\,[\Delta u(k+j-1)]^2} \quad (1)$$

En la ecuación (1), se muestra una de las funciones de coste más empleada en este tipo de algoritmos, donde el primer término de la misma corresponde con la diferencia entre las predicciones del modelo ($\hat{y}$) y las referencias futuras (w) para p instantes, y el segundo, con las variaciones de las acciones de control (Δu). Por otro lado, el parámetro α pondera el primer término de la ecuación de forma que si se aumenta su valor penaliza aún más las predicciones del error. Asimismo, el parámetro λ penaliza el segundo término, provocando que cuando aumente su valor, las variaciones de las acciones de control sean más suaves. El principal objetivo es minimizar la diferencia entre la referencia y los valores de la predicción en el horizonte de predicción y las variaciones de la acción de control a lo largo del horizonte de control establecidos.

Una vez obtenido el índice de la ecuación de coste, se pretenden calcular las acciones de control para c instantes de tiempo. Para ello existen diferentes tipos de optimizadores desde algoritmos de mínimos cuadrados hasta rutinas de programación cuadrática. Cabe destacar, que aunque el algoritmo calcule un número determinado de acciones de control, correspondientes al valor del horizonte de control (c), únicamente se aplica la primera de ellas por cada una de las variables manipuladas del proceso.

Recapitulando, los principales pasos que siguen la mayoría de los algoritmos de tipo MPC en cada iteración son los siguientes:

- En primer lugar, el controlador calcula la salida del sistema hasta un determinado instante, correspondiente al valor del horizonte de predicción. Este hecho es denominado como la respuesta libre del sistema.
- A continuación, se calcula las acciones de control pertinentes a cada una de las variables a manipular. Para cada variable manipulada se calcula un determinado número de estas, el cual corresponde con el parámetro llamado horizonte de control (c). Todo ello mediante un determinado algoritmo de optimización.
- Finalmente, se aplican al proceso las acciones de control calculadas, pero únicamente la primera de ellas para cada variable.

3.1.4. Dynamic Matrix Control (DMC)

En este apartado se va tratar en que consiste el Dymanic Matrix Control (DMC) [8] [17], que consiste en un tipo de MPC que se caracteriza por emplear como modelo del proceso, el comportamiento del proceso ante un determinado tipo de entrada, que puede ser de tipo impulso o de tipo escalón. No obstante, en este caso, se va explicar cómo obtener la respuesta de tipo escalón.

Para ello, se parte del proceso en el punto de equilibrio correspondiente y seguidamente se introduce un escalón unitario a la variable manipulada correspondiente. Después, se almacenan en un vector los N instantes del ensayo hasta que cada una de las salidas del proceso se hayan estabilizado. Finalmente, se resta el valor del punto de equilibrio de cada una de las variables a cada uno de los vectores, dejando los valores de la respuesta en forma incremental.

Por otro lado, las predicciones calculadas por el algoritmo DMC están compuestas por la suma de la respuesta libre y forzada del proceso, la respuesta de las perturbaciones medibles y el efecto de las perturbaciones no medibles. La respuesta libre (Figura 4) consiste en el comportamiento que posee el sistema ante unas determinadas variaciones en las entradas en instantes pasados, es decir, corresponde con la respuesta del sistema asumiendo que las variaciones en las entradas en los instantes presentes y futuros son nulos.

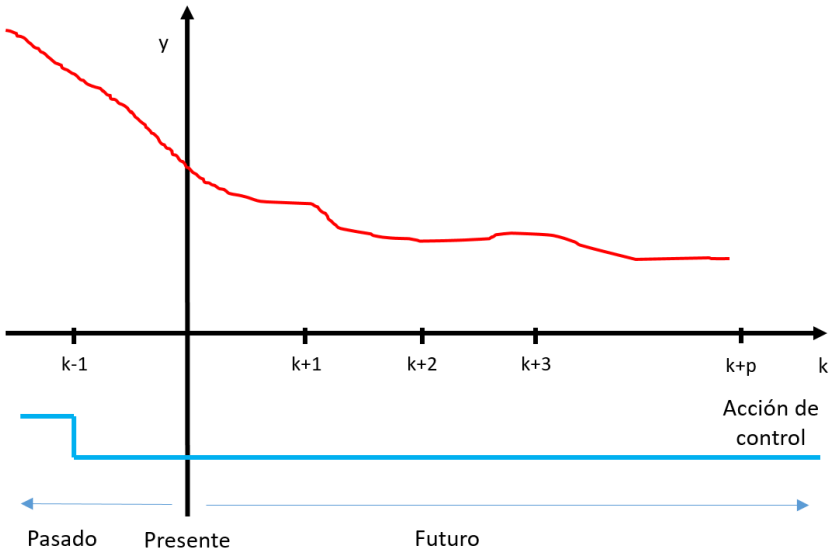


Figura 4: Respuesta libre del DMC

La respuesta forzada (Figura 5) corresponde al comportamiento del proceso ante variaciones en las entradas en los instantes presentes y futuros, estos son calculados a partir de las Δu a lo largo del horizonte de control (c).

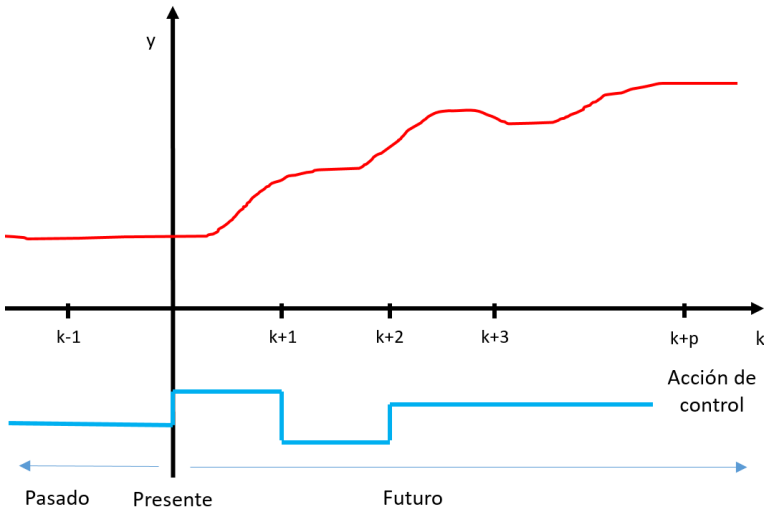


Figura 5: Respuesta forzada del DMC

Una vez obtenida las predicciones se procede al cálculo de las acciones de control para una determinada iteración del algoritmo. No obstante, cabe destacar que en el algoritmo DMC no se pueden establecer ningún tipo de restricciones en las entradas o salidas del proceso.

$$\Delta u = (G^T \alpha G + \lambda)^{-1} G^T \alpha (sp - y_{libre}) = D(sp - y_{libre}) \quad (2)$$

En la ecuación (2) se observa la ley de control para obtener los Δu en una determinada iteración. La matriz G representa la respuesta dinámica del modelo para cada una de las entradas del sistema y se construye a partir de los primeros coeficientes de la respuesta escalón del mismo hasta alcanzar el número de filas del horizonte de control (p), en el caso de un proceso con una única salida. Asimismo la matriz G posee una forma matricial triangular inferior. Cabe destacar que la ecuación anterior se obtiene de derivar la función de coste respecto Δu, agrupando todos los términos dependientes en la variable D. Asimismo, el término $(sp - y_{libre})$ representa el error entre la referencia y la respuesta libre del sistema.

3.1.4.1. Parámetros configurables del algoritmo DMC

Por otro lado, todos los algoritmos anteriormente mencionados, concretamente el DMC, poseen determinados parámetros para ajustar el comportamiento del algoritmo ante un determinado proceso. Los parámetros de mayor interés para este trabajo son los siguientes:

- **Horizonte de control (c)**: consiste en el número de instantes futuros que se desea calcular de las acciones de control del controlador en cada iteración.
- **Horizonte de predicción (p)**: consiste en el número de instantes que se tiene en cuenta para la generación de la predicción de las salidas del sistema y posterior cálculo de las acciones de control.
- **Lambda (λ)**: parámetro que pondera las variables manipuladas del sistema el índice a optimizar. A cada variable manipulada le corresponde un valor de lambda. Cuanto mayor es su valor, más suave es el comportamiento de las acciones de control.
- **Alfa (α)**: parámetro que determina la rapidez con la que el controlador debe intentar seguir a la referencia. Cuanto mayor es su valor mayor rapidez en alcanzarla.

3.1.5. Quadratic Dynamic Matrix Control (QDMC)

Para establecer restricciones al algoritmo de control DMC se emplea las técnicas Quadratic Dynamic Matrix Control (QDMC) [9] [18], que consiste en emplear todos los algoritmos del DMC pero cambiando el optimizador por un algoritmo QP. Las restricciones que se pueden implementar son los límites superior e inferior de las variables manipuladas y de las salidas del proceso, y la velocidad de variación de las acciones de control.

$$min \quad J_{QP} = \frac{1}{2}x^T Hx + T^T x = \frac{1}{2}x^T Hx + (-G\alpha e)x$$
$$sujeto\ a \quad Ax \leq B \tag{3}$$

La ecuación (3) muestra la ecuación a minimizar donde la variable H corresponde con el hessiano (H) de la ley de control. Por otro lado, el parámetro T del segundo término se debe actualizar en cada una de las iteraciones del controlador ya que para calcularla es necesario el error (e) entre la referencia y la respuesta libre (y_{libre}) en cada una de ellas, mientras que H es constante. El vector x es la solución óptima que minimiza la ecuación anterior, que en este caso son las variaciones de las acciones de control para un horizonte de control determinado.

La segunda ecuación representa las restricciones del QP donde la matriz A y B están formadas por un subconjunto de matrices correspondientes a las restricciones de cada una de las variables que forman el proceso.

A la hora de establecer las restricciones, se pueden clasificar de en dos tipos, las restricciones duras y las blandas. Las restricciones duras corresponden a aquellas que el algoritmo QP siempre debe cumplir, pero este hecho puede causar que en algunas iteraciones del algoritmo optimizador, no se encuentre una solución que sea capaz de satisfacer todas ellas. Las restricciones blandas son aquellas que le otorgan cierta holgura a las restricciones, pueden ser sobrepasadas en cierta manera en alguna de las iteraciones del QP. Para ello, se añade el parámetro ε al final de la matriz B, el cual determina cual es la holgura máxima que estas restricciones pueden tener. Para la aplicación de control que se ha desarrollado se han empleado restricciones blandas pero con una holgura mínima para que el algoritmo QP fuese capaz de obtener las acciones de control óptimas en cada una de las iteraciones. Este hecho se abordará con mayor profundidad en el capítulo de la prueba de concepto del sistema de control.

3.1.5.1. Parámetros configurables del algoritmo QDMC

Los parámetros configurables del QDMC son los mismos que se han mencionado en el algoritmo DMC pero añadiendo el valor de cada una de las restricción del proceso a controlar. Estos parámetros son los siguientes:

- **Umax**: valor máximo que debe poseer cada una las variables manipuladas.
- **Umin**: valor mínimo que debe poseer cada una las variables manipuladas.

- **Δu**: valor máximo que se debe variar el valor de cada una de las variables manipuladas por iteración.
- **Ymax**: valor máximo que debe poseer cada una las variables a controlar.
- **Ymin**: valor mínimo que debe poseer cada una las variables a controlar.

3.2. Protocolo de comunicación: Modbus TCP

3.2.1. Introducción

Un protocolo de comunicación es un conjunto de normas que habilita la comunicación entre varios dispositivos, los cuales pueden tener características muy diferentes entre sí. No obstante, si emplean el mismo protocolo, podrán establecer una comunicación entre ellos a través de un medio.

Para este caso en concreto, se va a emplear el protocolo de comunicación Modbus [10] que permite a sus usuarios enviar información de una forma ordenada, concretamente en unos espacios de memoria que ya vienen definidos por defecto en dicho protocolo. Se caracteriza por ser un protocolo de uso industrial, gratuito y fácil de usar, concretamente está en gran medida implantado en todo ambiente industrial donde existe la presencia de autómatas programables (PLC). Cabe destacar que el Modbus se basa en la estructura de comunicación cliente/servidor y pertenece a la capa número 7 de la capa del modelo de interconexión de sistemas abiertos (OSI) [23].

En la actualidad existen diferentes versiones del protocolo Modbus, donde las más conocidas son Modbus RTU [10] [11], el cual emplea la comunicación es de tipo serie, y Modbus TCP [11], el cual emplea para establecer comunicación las redes de tipo TCP/IP [23]. Para esta aplicación, se ha empleado el protocolo Modbus TCP ya que ofrece la posibilidad de establecer conexión entre los diversos dispositivos vía Ethernet/WIFI, a través de una dirección IP y un puerto de conexión, establecidos previamente. En este caso, el protocolo Modbus TCP no necesita ningún comando de comprobación de redundancia cíclica de datos (CRC), ya que el propio protocolo TCP/IP ya incorpora uno de ellos en su propia trama. La principal finalidad del comando CRC, es que cuando el cliente recibe la trama desde el servidor, sea capaz de comprobar que la información recibida es correcta como afirma National Instruments en su página web [1].

En la Figura 6, se observa una red que incorpora ambas versiones de Modbus anteriormente mencionadas, formadas por sus correspondientes clientes y servidores.

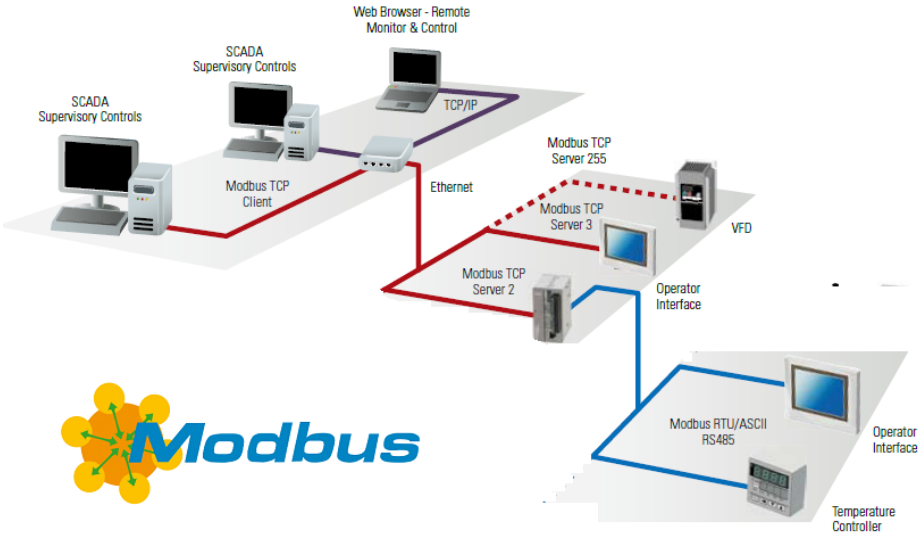


Figura 6: Esquema de conexión de una red que emplea el protocolo Modbus. Obtenida de la web de Modbus

3.2.2. Descripción protocolo Modbus TCP

El protocolo Modbus TCP [1] [11], como se ha comentado anteriormente, emplea las redes de tipo ethernet para establecer comunicación entre los diversos dispositivos presentes en dicha red. Para ello, emplea el protocolo TCP donde en la parte de la trama que contiene la información referente al mensaje, se encapsula la trama cifrada mediante el protocolo de comunicación Modbus. La trama del protocolo Modbus TCP está dividida en dos partes, la unidad de aplicación (ADU), referente a la trama del protocolo TCP (Figura 7), y la unidad de datos de protocolo (PDU), que corresponde al protocolo Modbus (Figura 8).

Transacción	Protocolo	Longitud	Unidad ID	Modbus PDU

Figura 7: Unidad de aplicación (ADU) de TCP/IP

La ADU se compone de varias partes. La primera de ellas, hace referencia al número de transacción, útil en aquellos sistemas que pueden mandar varias solicitudes simultáneamente. Seguidamente, el identificador del protocolo que normalmente posee un valor igual a cero. A continuación, se encuentra el campo referente a la longitud del mensaje y el ID de la unidad, que normalmente no se emplea en los dispositivos TCP/IP.

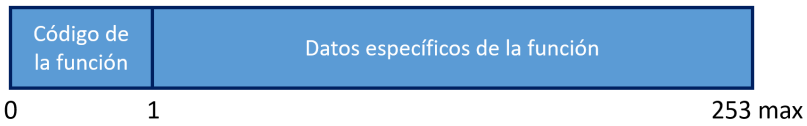


Figura 8: Unidad de datos de protocolo (PDU) de Modbus

La PDU de Modbus, se compone de dos únicos campos, el código de función y los datos correspondientes a dicha función. Los códigos de función consisten en las acciones que desea realizar el servidor o cliente para leer o modificar cada uno de los diferentes bloques de memoria. Las funciones más comunes corresponden con la escritura y lectura de los diversos tipos de datos que puede soportar el protocolo Modbus, las cuales se pueden observar en la siguiente tabla:

Tabla 1: Bloques de memoria del protocolo Modbus

Bloque de memoria	Tipo de dato	Tamaño (bits)	Acceso maestro	Acceso esclavo
Bobinas	Booleano	1	Lectura/Escritura	Lectura/Escritura
Entradas discretas	Booleano	1	Lectura	Lectura/Escritura
Registros de retención	Palabra sin signo (U16)	16	Lectura/Escritura	Lectura/Escritura
Registros de entrada	Palabra sin signo (U16)	16	Lectura	Lectura/Escritura

Para cada bloque de memoria se posee un tamaño de memoria de 65536, donde las direcciones de memoria van de 0 hasta 65535. Cada uno de los bloques de memoria posee un prefijo diferente para identificar qué tipo de datos se está empleando en cada momento. No obstante, la PDU de Modbus posee 253 bytes de memoria por cada mensaje que se pretende mandar entre cliente y servidor. Esto es debido a que este protocolo está basado en el protocolo TCP que posee un determinado número de bytes por cada trama enviada.

Una de las principales limitaciones que posee este tipo de protocolo es que únicamente se pueden enviar datos de tipo booleano o datos que posean un tamaño de 16 bits, como por ejemplo datos de tipo entero. En este caso en particular, este hecho es un problema puesto que uno de los principales objetivos de este proyecto es ser capaces de enviar datos de tipo flotante. Para ello, se ha tenido que dividir el número flotante en dos enteros de 16 bits, ocupando dos espacios de memoria por cada variable de este tipo. Posteriormente, el cliente se encargará de coger ambos datos de tipo

entero y convertirlos de nuevo a un dato de tipo flotante. Este procedimiento se llevará a cabo para cada variable de tipo float32 que se desde transmitir entre el servidor y el cliente. Asimismo, este hecho se verá con más detalle en el anexo correspondiente a la programación de la aplicación que actúa como interfaz gráfica.

Finalmente, destacar que en esta aplicación se han empleado los bloques de memoria correspondientes a las bobinas (coils), de las direcciones 0 a 15, y a los registros de retención (holding registers), de las direcciones 0 a 148. Asimismo, se especificará con más detalle en el correspondiente anexo de programación.

3.3. Raspberry PI

3.3.1. Introducción

Raspberry PI, como afirma la propia web de la compañía, consiste en una placa de tamaño reducido, concretamente de 85.60mm de largo por 56 mm de ancho, que se comenzó a desarrollar en Reino Unido por la Fundación Raspberry Pi a partir de 2009 [4]. A partir de esta fecha se han ido presentando diversas versiones, que se pueden distinguir por su rendimiento y tamaño. El principal fin de este producto, es crear un dispositivo de coste reducido, ofreciendo la posibilidad de integrar en mayor medida los lenguajes de programación dentro de las aulas. Esta placa ofrece todas las características y componentes de un ordenador convencional, como son memoria RAM, una GPU, una CPU, etc. Pero como es de esperar de forma limitada, es decir, posee las características de un ordenador, pero no posee el potencial necesario para ciertas aplicaciones. Este dispositivo es capaz de soportar diferentes sistemas operativos, pero normalmente el más empleado, es el que suministra la propia compañía, el llamado Raspbian, que proviene directamente del proyecto Debian [20]. En la actualidad, este tipo de dispositivos han ganado una gran popularidad, ya que se emplean en múltiples disciplinas desde la automatización de viviendas hasta proyectos de carácter DIY (Do it Yourself) debidos a su bajo costo y facilidad de uso.

3.3.2. Raspberry Pi 3 Model B

Para esta aplicación se ha optado por emplear el dispositivo Raspberry Pi 3 Model B [4], que corresponde con uno de los últimos modelos lanzados por la compañía, el cual posee algunas características mejoradas con respecto a sus versiones anteriores. Según la compañia, el principal aspecto de mejora, es su procesador que aumenta su velocidad hasta 1.2Ghz, lo cual es en gran medida beneficioso para el desarrollo y posterior ejecución del algoritmo de control que se introducirá en posteriores capítulos de esta memoria. Otra de las características de este dispositivo, es que posee un gran número de entradas y salidas digitales en el rango de tensión de 0 a 3,3 V, lo cual aumenta considerablemente sus posibilidades de uso.

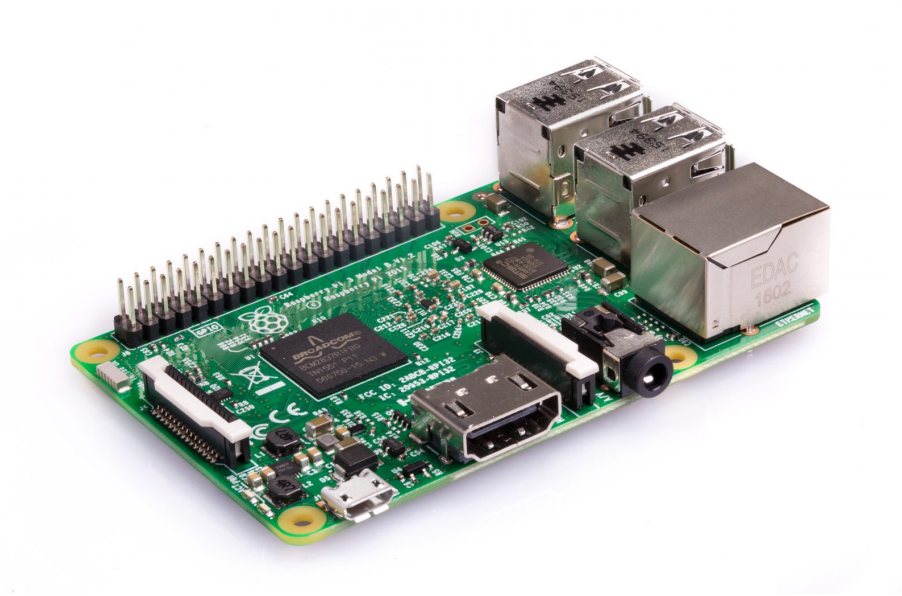


Figura 9: Raspberry Pi 3 Model B. Obtenida de la página web de Raspberry Pi

Uno de los elementos más útiles que posee este dispositivo es su puerto de expansión o también denominado GPIO de 40 pines, mediante el cual se tiene acceso a las entradas/salidas digitales que se han mencionado anteriormente, a las fuente de tensión de 3.3 V y 5 V, a los puertos correspondientes a SCA y SCL del protocolo de comunicación I^2C, a los puertos de comunicación serie y a los puertos de comunicación SPI. Estos últimos pines posibilitan la incorporación de más chips que empleen estos protocolos de comunicación, ampliando las funciones del dispositivo en cuestión. Este hecho es de gran utilidad, ya que el puerto GPIO será empleado en este caso, para añadir una tarjeta de adquisición de datos para ampliar la funcionalidad del dispositivo y adaptarla a las necesidades de esta aplicación.

3.3.3. Protocolo de comunicación I^2C

Es uno de los protocolos o buses de comunicación más empleado actualmente para la comunicación entre microprocesadores y sensores. Este bus [24] se caracteriza por poseer una velocidad media de transmisión de datos (como máximo 1Mhz) y ser capaz de establecer comunicación con diversos dispositivos siguiendo la configuración Maestro/Esclavo, donde cada uno de los elementos posee una dirección única dentro del mismo bus. Asimismo, este protocolo se caracteriza por poseer únicamente dos líneas (sin tener en cuenta la de alimentación del bus) para establecer la comunicación, el SCL que se encarga de transmitir la señal reloj para establecer la sincronización entre varios componentes, y el SDA que es el canal por donde se transmiten los datos de interés. Cabe destacar que el encargado de generar la señal reloj es el dispositivo que actúa como maestro. En la Figura 10, se muestra un ejemplo de un esquema de conexión entre diferentes componentes mediante este tipo de bus.

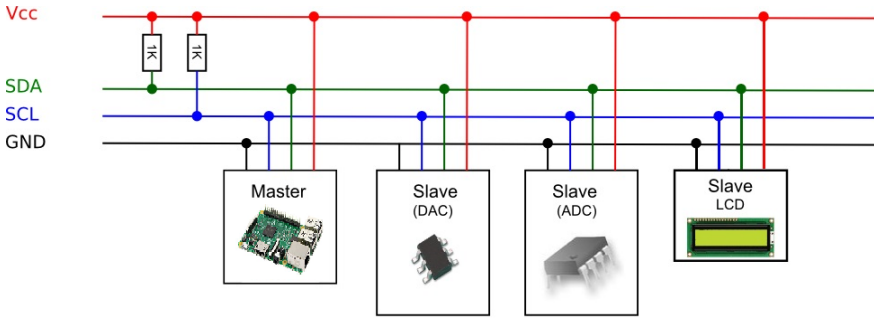


Figura 10: Esquema de conexión del bus I^2C. Obtenida de la página web Prometec

Además, cada uno de los componentes que conforman la red, posee la capacidad de enviar o recibir datos del correspondiente maestro, que en este caso este papel lo desempeña la Raspberry Pi. La conexión específica de cada uno de los componentes descritos en el apartado anterior, se verá en el anexo del manual de diseño de la TAD.

La trama que este protocolo emplea está formada por diversas partes que se explican a continuación.

Figura 11: Trama de inicio de la comunicación en el bus

En la Figura 11, se puede observar la trama de inicio que el maestro envía al correspondiente esclavo con el que desea establecer una comunicación. Está formada por:

- **Bit de inicio**: Bit que representa el inicio de la comunicación entre maestro y esclavo.
- **Dirección**: Está formada por una serie de bits, que dependiendo del hardware empleado puede poseer una longitud entre 7 y 10 bits. En esta parte de la trama el maestro envía la dirección del dispositivo que actúa como esclavo con el que quiere establecer una comunicación.
- **Lectura/Escritura**: En este bit se manda la acción que se desea realizar, si el bit es igual a 1 entonces corresponde con el modo lectura y si es 0, corresponde con el modo escritura.
- **ACK**: Este bit representa que la trama anteriormente descrita se ha recibido satisfactoriamente.

Una vez establecida ya la dirección del esclavo, la forma de la trama para el resto de la comunicación es la siguiente:

Figura 12: Trama de transmisión de datos entre dispositivos I^2C

- **Datos:** conjunto de bits que contiene la información que se desea transmitir, normalmente posee un tamaño de 8 bits.
- **ACK**: Este bit representa que la trama anteriormente descrita se ha recibido satisfactoriamente.
- **Parada**: una vez ya realiza la comunicación entre maestro y esclavo, se estable el bit de parada con el propósito de liberar el bus. De esta forma, el maestro a partir de este momento puede establecer comunicación con otros dispositivos del bus.

3.3.4. Python

Para realizar la programación tanto de la TAD como del algoritmo de control en la Raspberry Pi, se ha empleado el lenguaje de programación interpretado llamado Python [12]. Este se caracteriza por soportar programación orientada a objetos y programación imperativa. Asimismo, la principal virtud de Python es que es totalmente gratuito y además actualmente empleado en diversas disciplinas como en el desarrollo de páginas web, en bases de datos o en todo tipo de aplicaciones científicas. Por otro lado, posee la funcionalidad de poseer en gran variedad de librerías que se han empleado para las diversas partes del proyecto, como las librerías referentes a los componentes de la TAD mediante el SMBus (System Management Bus), o librerías matemáticas para realizar los cálculos pertinentes de la parte del controlador. Finalmente destacar que Python se trata de un lenguaje interpretado, ofreciendo la posibilidad de ejecutar y depurar cualquier programa realizado con este lenguaje.

4. HARDWARE DISEÑADO: TARJETA DE ADQUISICIÓN DE DATOS

Dado que la Raspberry PI no dispone de entradas/salidas analógicas, ha sido necesario el desarrollo integral de una tarjeta de adquisición de datos (TAD) para poder cubrir las necesidades del proyecto. Las principales características de esta TAD, es que debe ser de dimensiones semejantes a las de la Raspberry Pi y debe ser compatible con el puerto GPIO, ya que la comunicación entre la Raspberry Pi y la TAD se va a realizar a través de este, concretamente a través de los pines correspondientes al SCA y SCL del protocolo de comunicación I^2C. Asimismo, esta debe poseer una ciertas cantidad de entradas y salidas analógicas para poder realizar las acciones de control pertinentes y leer las variables a controlar del proceso deseado. Finalmente, también se deben añadir entradas y salidas de tipo digital a la placa en cuestión. En la siguiente figura, se puede observar un esquema genérico de conexión de sensores y actuadores con la TAD.

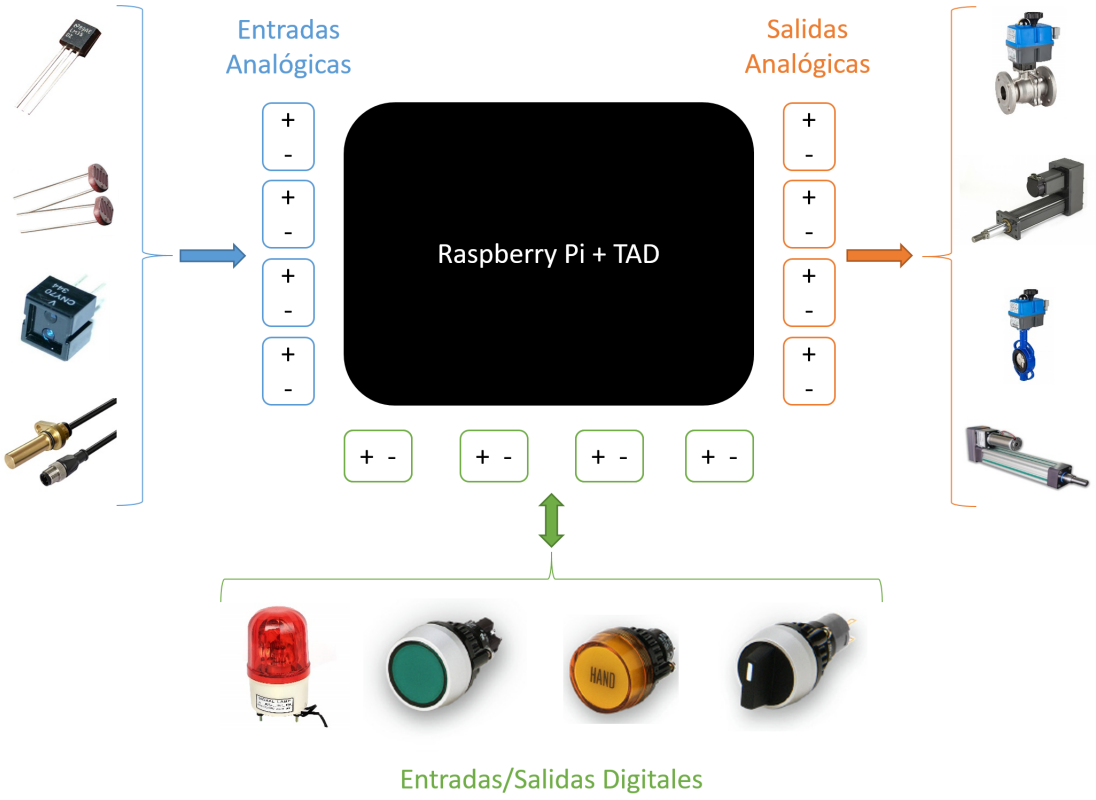


Figura 13: Esquema de conexión de sensores y actuadores con la TAD

Las características principales de la TAD son las siguientes:

- Cuatro entradas analógicas que poseen un rango en tensión [-10; 10] V.
- Cuatro salidas analógicas que poseen un rango en tensión [-10; 10] V.
- Ocho entradas/salidas digitales que poseen un rango de tensión [0; 5] V. Estos pines son configurables mediante software para establecerlos como entradas o salidas.

Para conseguir estas características técnicas se ha empleado una serie de componentes electrónicos, compuestos por convertidores analógicos digitales (DAC y ADC), amplificadores operacionales, diodos... que se abordarán con posterioridad en el anexo de diseño. No obstante, uno de los principales objetivos es que la TAD fuese lo más compacta posible, lo cual ha supuesto un gran esfuerzo a la hora de realizar el diseño de la misma. Asimismo, la ubicación de cada uno de los conectores, se han establecido de una forma concreta para poder aprovechar el uso de carcasas de protección ya existentes en el mercado actual como por ejemplo la Piface 2 case [13]. Si no se hubiese realizado de esta forma, el coste del presupuesto se habría encarecido en gran medida, ya que realizar el diseño de una carcasa desde cero resulta mucho más costoso en términos económicos.

En la Figura 14, se pueden observar las diversas partes de la TAD desarrollada, que son las siguientes:

- Zona roja: contiene los componentes electrónicos que conforman las entradas analógicas de la TAD.
- Zona negra: contiene los componentes que conforman las salidas analógicas de la TAD.
- Zona amarilla: contiene los componentes que conforman las entradas y salidas digitales.
- Zona azul: contiene los componentes necesarios para la adaptación de niveles de tensión del protocolo de comunicación I^2C entre la Raspberry Pi y los componentes correspondientes.

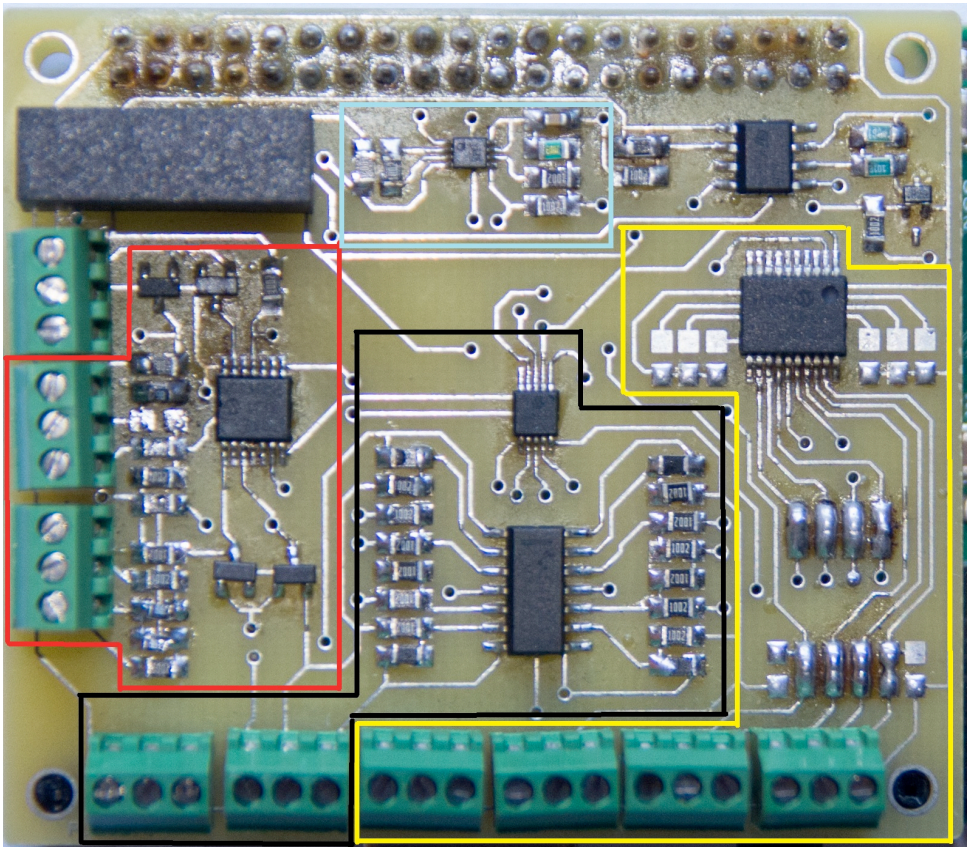


Figura 14: Vista de planta de la TAD diseñada ya ensamblada

El papel que desempeña el conjunto Raspberry Pi + TAD una vez iniciada la aplicación es el siguiente. En primer lugar, la TAD recibe los valores de las variables a controlar procedente normalmente de los sensores del proceso industrial en cuestión. Seguidamente, esta tensión irá al convertidor analógico-digital, que se encarga de convertir la información expresada en tensión a bits. Una vez realizado este hecho, esta información será enviada a la Raspberry Pi a través del protocolo de comunicación I^2C y a continuación, esta calculará las acciones de control correspondientes, mediante el algoritmo de control QDMC. El siguiente paso es coger estas acciones y enviarlas mediante el protocolo I^2C al convertidor digital-analógico (DAC) que convierte la información de bits a tensión y mediante el cableado correspondiente, comunicar estas tensiones a los actuadores del proceso. En la Figura 15 se observa el flujo de información que circula por la TAD.

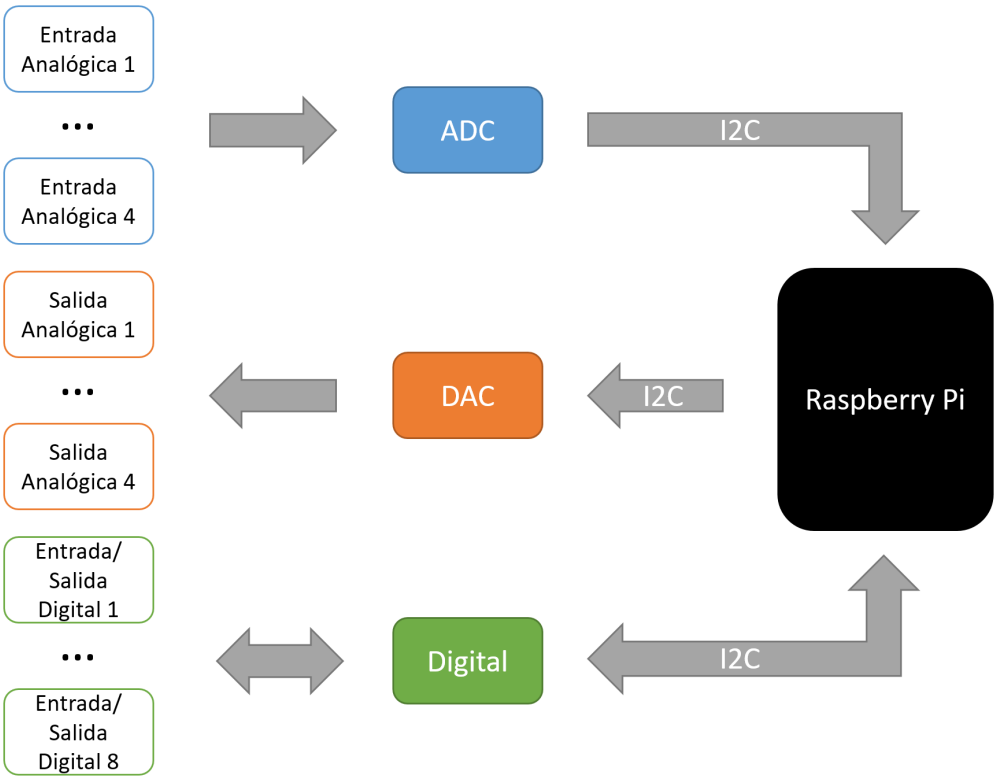


Figura 15: Flujo de información en la tarjeta de adquisición de datos

Todos los aspectos más referentes al propio diseño y como se ha llegado a las especificaciones establecidas, se pueden ver con un mayor nivel de detalle en el anexo correspondiente al manual de diseño de la TAD.

5. APLICACIÓN DE CONTROL QDMC EMPOTRADA.

La aplicación de control posee las funciones de recibir y mandar información mediante Modbus, realizar la gestión de los diversos modos de funcionamiento del controlador, gestionar el propio controlador QDMC y el manejo de las entradas y salidas de la TAD. Para ello, la aplicación está compuesta por diversas partes fundamentales. La primera corresponde con la inicialización tanto del cliente Modbus, estableciendo la IP del servidor y el puerto conexión. Asimismo, inicializa las variables necesarias para hacer funcionar correctamente la aplicación, donde esta cargará un fichero .txt con la configuración del proceso que viene por defecto. No obstante, si la aplicación detecta un fichero extra con la configuración de un determinado proceso, esta cargará este fichero ignorando el correspondiente a la configuración por defecto. Y finalmente, la inicialización de las direcciones de los chips de entradas y salidas digitales, ADC y DAC de la TAD.

La siguiente parte es la referente a la comunicación Modbus, donde la aplicación está leyendo y escribiendo en los correspondientes bloques de memoria del protocolo, tanto para actualizar algunas de las variables de configuración del controlador, como para enviar cada intervalo de tiempo, la información referente a las referencias, variables manipuladas y variables a controlar procedente del controlador, para su posterior monitorización en la interfaz gráfica remota. Es importante recalcar, que cuando una variable se actualiza en el bloque de memoria del Modbus y se lee por la aplicación empotrada, esta es almacenada en una determinada variable auxiliar local, que dependiendo del modo de funcionamiento en que se encuentre, actualizará directamente la variable local correspondiente que afecta al controlador o se esperará a estar en el modo correcto para actualizar dicho parámetro.

A continuación, le precede la parte referente al controlador, en el cual se diferencian dos partes, una parte que actúa de forma offline y otra que actúa online. La parte offline es la que únicamente es necesario ejecutarla una sola vez, sino se cambia ninguna de las variables de configuración referentes al algoritmo DMC. Si se produce algún cambio en ellas, se debe ejecutar para recalcular todas las variables del algoritmo. La parte online es la que se encarga de calcular las acciones de control para llevar las variables a controlar a las referencias establecidas. Es importante destacar que el controlador se ejecuta cada periodo y posee tres posibles modos de funcionamiento, que son gestionados por la siguiente parte de la aplicación. Dependiendo del modo de funcionamiento seleccionado, el controlador necesitará unas determinadas variables. No obstante, las funciones de lectura de las referencias, procedente de la aplicación remota (Modbus), de lectura de las variables a controlar y de escritura de las variables manipuladas, procedentes de la TAD, son comunes a todos los modos de funcionamiento.

Los modos de funcionamiento en los que se pueden encontrar el controlador se muestran a continuación:

- **Modo DESHABILITADO:** corresponde cuando el controlador se encuentra apagado. En este modo proporciona un valor nulo a todas las acciones de control.
- **Modo MANUAL:** el controlador QDMC se encuentra apagado, pero el usuario es capaz de suministrar los valores de las acciones de control a las variables manipuladas mediante la interfaz gráfica remota a través del protocolo Modbus.

- **Modo AUTOMÁTICO:** donde el control se gestiona de forma totalmente automática, siendo gestionado por el controlador QDMC. Para su correcto funcionamiento son necesarias las variables que se han calculado previamente de forma offline.

Finalmente, la última parte de la aplicación empotrada es el automatismo, que gestiona el modo de funcionamiento del controlador y cuando se deben actualizar las variables del mismo, ya que dependiendo del modo en que se encuentra el controlador, se pueden o no actualizar determinadas variables locales. Esta parte de la aplicación también se encarga de gestionar los ficheros de configuración que se encuentran en las Raspberry Pi, por si no se desea emplear los parámetros de configuración procedentes de la aplicación remota.

Cabe destacar que cada una de las partes anteriormente descritas, funcionan con un determinado periodo de tiempo, el cual, no tiene por qué ser el mismo para cada uno de los bucles de la aplicación. De hecho en este caso, cada uno de los bucles funciona a periodos diferentes, siendo el más rápido el que gestiona la lectura y la escritura de los bloques de memoria de Modbus.

En el modo DESHABILITADO, se pueden actualizar todos los parámetros de configuración desde el valor del identificador del proceso, el número de entradas y salidas del sistema, periodo de muestreo,… hasta todas las variables tanto de algoritmo DMC como del QP del controlador. Asimismo, desde este modo también se pueden cargar los ficheros referentes a las configuraciones del controlador, que se encuentran en la unidad de almacenamiento de la Raspberry Pi. Además, se tiene la posibilidad de cargar el fichero que corresponde a la configuración por defecto o a la última configuración guardada por el usuario. Si se realiza cualquier cambio en este modo, se ejecuta la parte offline del controlador para actualizar las variables del DMC.

En el modo manual, se supone que ya se han configurado los parámetros referentes a las características del proceso (identificador del proceso, número de entradas y salidas,…) y por tanto, únicamente se pueden modificar las variables del DMC (alfas, lambdas y horizonte de control) y las variables referentes al optimizador QP (variaciones, máximos y mínimos de las variables del proceso).

En el modo AUTOMÁTICO, únicamente se pueden cambiar en línea los parámetros del optimizador QP, ya que en este modo el controlador se encuentra en funcionamiento.

Por último, esta parte de la aplicación también se encarga de almacenar en un fichero .txt la configuración que posee el controlador actualmente, mediante una orden del usuario a través de la interfaz gráfica remota. Esta acción se puede realizar en cualquier momento mientras se está ejecutando la aplicación empotrada.

Otra de las cuestiones que se han tenido en cuenta a la hora del diseño de la aplicación empotrada, es el tema relacionado con la seguridad a la hora de la gestión del controlador en caso de fallo del algoritmo. La primera medida de seguridad que se ha implantado es el denominado perro guardián o watchdog, que consiste en un algoritmo que se ejecuta cada cierto tiempo, escribiendo en cada iteración un determinado valor en una variable interna del sistema. Si por cualquier motivo no se escribe sobre esa variable durante un determinado periodo de tiempo, la Raspberry Pi se reiniciará de forma automática y una vez iniciada de nuevo, ejecutará el programa correspondiente a la aplicación empotrada. Este algoritmo se ha implementado en la parte donde se ejecuta de forma cíclica el algoritmo QDMC. Otra medida de seguridad que se ha implementado es a la hora de ejecutar el

algoritmo de optimización QP, que puede devolver un vector vacío o un determinado error debido a que no puede cumplir con las restricciones establecidas, lo que puede hacer colapsar la aplicación. Para ello se ha implementado un algoritmo que en el momento que detecte alguno de estos errores, envié a la TAD las acciones de control referentes al punto de equilibrio del proceso.

Si se desea un mayor nivel de detalle sobre cada una de las partes de la aplicación empotrada o de su programación, se puede consultar el anexo del manual de programación de la aplicación empotrada en la Raspberry Pi.

6. INTERFAZ GRÁFICA DE CONFIGURACIÓN REMOTA

La interfaz gráfica proporciona la posibilidad de realizar la configuración del controlador de forma remota, únicamente siendo necesario una conexión de tipo Ethernet. No obstante, esta aplicación también proporciona la posibilidad de gestionar cada uno de los modos de funcionamiento del controlador y de monitorizar todas las variables de interés del proceso.

En esta ocasión se ha recurrido al lenguaje de programación de LabVIEW de la compañía National Instruments [22], el cual ofrece una programación gráfica multi-hilo mediante la ventana denominada diagrama de bloques y además está formado por una ventana denominada panel frontal, lo cual resulta ideal para este tipo de aplicaciones donde el usuario puede interactuar con el programa mientras este se encuentra en ejecución.

Para realizar cada una de las funciones que se han mencionado anteriormente, el panel frontal de la aplicación se ha dividido en tres partes, dos de ellas son pestañas que se puede cambiar de una a otra y la última corresponde a un panel que siempre se encuentra a la vista del usuario.

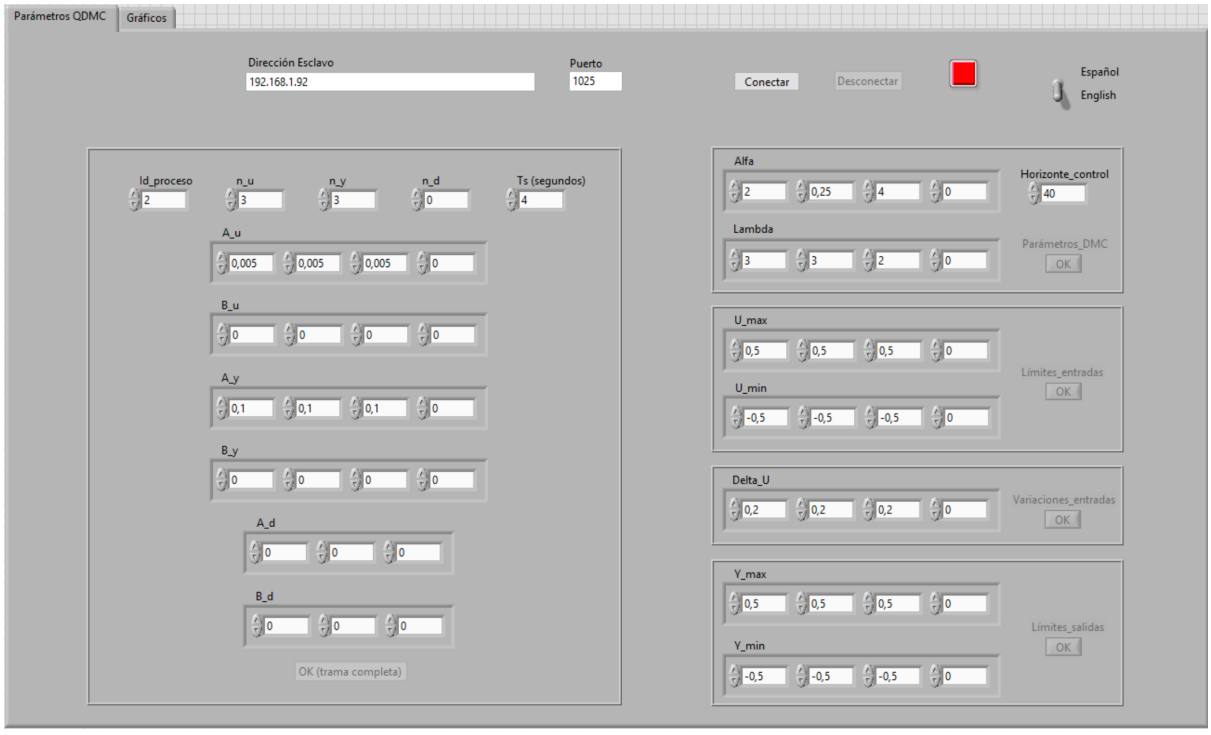


Figura 16: Pestaña 1 (Parámetros QDMC) de la interfaz gráfica.

La primera pestaña de la aplicación muestra todos los parámetros configurables del controlador como se puede observar en la Figura 16. En primer lugar, se encuentra la parte de configuración de la dirección IP del servidor y el puerto de conexión del protocolo Modbus. Seguidamente, en la parte izquierda de la misma, se encuentra los parámetros de configuración del controlador desde el identificador del proceso hasta las rectas de conversión de voltios a la magnitud correspondiente del proceso a controlar, correspondientes a las entradas y salidas de la TAD.

Asimismo, en la parte inferior se observa un pulsador booleano cuya finalidad es actualizar todos los bloques de memoria del protocolo Modbus, ya que se supone que en el momento que se varía algún parámetro de este bloque es necesario reconfigurar todos los parámetros de la aplicación empotrada.

En la parte derecha de la pestaña está situado, en la parte superior, el interruptor cuya función es la de cambiar el idioma de la aplicación de español a inglés y viceversa, y los parámetros de configuración del algoritmo DMC desde los parámetros alfa y lambda de cada una de las variables hasta el horizonte de control. Descendiendo por esta parte, se encuentra los parámetros que hacen referencia a las restricciones del algoritmo QP, estableciendo de arriba hacia abajo los límites, en magnitud y en variación, de las acciones de control por iteración y los límites de las variables a controlar, salidas del proceso. Además, al lado de cada uno de estos conjuntos de parámetros, hay una serie de pulsadores booleanos que actualizan únicamente las variables que se encuentran encapsuladas en el mismo rectángulo.

En la Figura 16 se observa que todos los pulsadores se encuentran deshabilitados, esto es debido a que únicamente se puede interactuar con ellos en el momento que se produce un cambio en las magnitudes correspondientes.

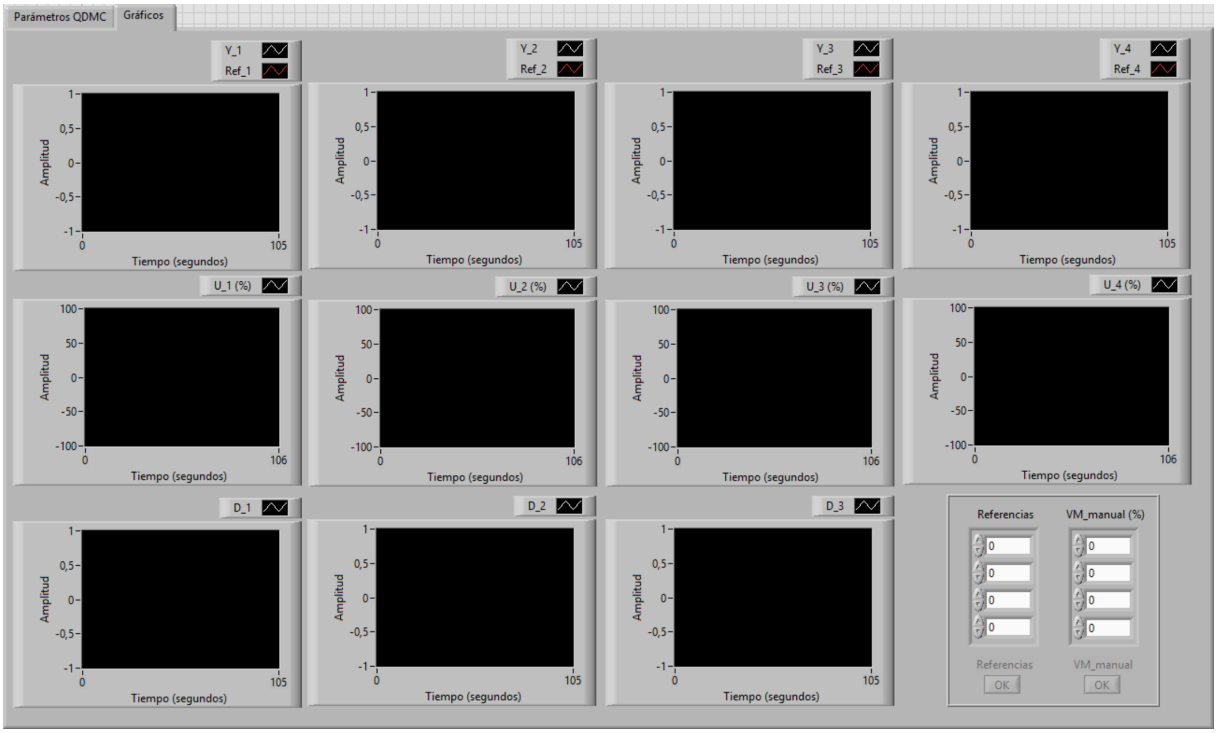


Figura 17: Pestaña 2 (Gráficos) de la interfaz gráfica

En la segunda pestaña de la interfaz gráfica (Figura 17) se encuentran 11 graficas que muestran la evolución de cada una de las variables del sistema siendo las cuatro de la primera fila donde se representa la evolución de las salidas del sistema y sus correspondientes referencias. En la segunda fila, se muestran las cuatro gráficas del comportamiento de las acciones de control y en la última, las

posibles perturbaciones medibles del proceso. Cabe recordar que la TAD únicamente posee cuatro entradas y cuatro salidas analógicas, por tanto, de estas once graficas únicamente pueden funcionar simultáneamente ocho de ellas, siendo posibles las cuatro referentes a las variables manipuladas y después puede variar según en número de variables a controlar y de perturbaciones medibles, cuya suma debe ser inferior o igual a cuatro.

En el margen inferior derecho, se observan las casillas referentes a los valores de las referencias y de las variables manipuladas cuando se encuentran en modo manual. Asimismo, se dispone de sendos controles booleanos para escribir las magnitudes en sus bloques de memoria del protocolo de comunicación.

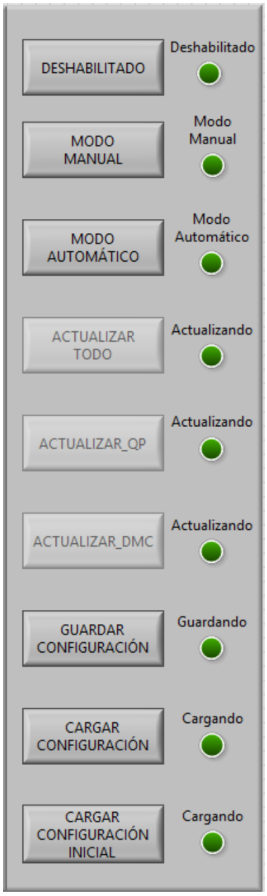


Figura 18: Pestaña fija de la interfaz gráfica

Finalmente, la pestaña fija de interfaz que se muestra en la Figura 18, donde se encuentra los diferentes pulsadores que habilitan las posibles funciones de la aplicación empotrada. Además, se dispone de una serie de indicadores que muestran los diferentes modos en el que se encuentra el controlador o que acciones se están ejecutando en cada momento. Los tres primeros booleanos corresponden con los posibles modos de funcionamiento del controlador. Seguidamente, se muestran tres pulsadores que se habilitan únicamente cuando han detectado alguna variación en alguno de los parámetros de configuración, cuya función es la de mandar la orden correspondiente al controlador para que actualice ciertos parámetros en su memoria local, si el modo de funcionamiento en el que se

encuentra la aplicación empotrada se lo permite. Finalmente, los pulsadores que leen y salvan las configuraciones del controlador de los ficheros de texto que se encuentran en la Raspberry Pi. Por otro lado, cada uno de los pulsadores posee un indicador, que en el caso de los tres primeros muestra el modo de funcionamiento del controlador, y en el resto de casos cuando se activan, indican que se ha mandado la orden de actualizar algún tipo de variable y se desactivan en el momento que se han hecho efectivos los cambios establecidos por el usuario en la aplicación empotrada.

Es destacable que la finalidad de esta interfaz gráfica no es la de realizar el papel de una aplicación de tipo SCADA, ya que el verdadero fin de esta es servir de complemento para el posible SCADA ya existente en el proceso industrial. Por este motivo, únicamente se muestran las variables de mayor interés de una forma genérica, para que esta representación sea compatible con la mayoría de procesos industriales a controlar.

Finalmente, remarcar que no es necesario que la interfaz gráfica remota este en continuo funcionamiento cuando la aplicación de control empotrada se encuentre en ejecución en la RPi, ya que la aplicación de control está programada para que pueda actuar de forma totalmente independiente. Es decir, la aplicación empotrada puede estar en funcionamiento continuo y en un determinado momento, iniciar la interfaz gráfica desde un ordenador de forma remota y realizar la configuración de los parámetros pertinentes. Una vez realizados, se puede cerrar la interfaz gráfica y la aplicación empotrada seguirá funcionando con la configuración establecida.

7. PRUEBA DE CONCEPTO DEL SISTEMA DISEÑADO

7.1. Proceso industrial a controlar

Para realizar la prueba de concepto del sistema de control diseñado, es necesario elegir un proceso industrial el cual debe poseer las siguientes características. En primer lugar, el proceso no debe poseer una dinámica muy rápida, es decir debe poseer un tiempo de establecimiento elevado, ya que el periodo del bucle de control que se ejecuta en la Raspberry Pi, no debe ser menor de los dos segundos, puesto que según en la situación que se encuentre el algoritmo QP, cada iteración puede tardar más de esa cantidad de tiempo, y por tanto, no realizar el control del proceso correctamente. La segunda característica, es que el proceso escogido no posea en ninguna de sus salidas una respuesta de tipo rampa ante una determinada entrada del sistema, ya que este hecho correspondería con un integrador en la función de transferencia correspondiente. Si esto ocurriese, el controlador QDMC no sería capaz de realizar el control de la misma de forma directa. No obstante, una posible solución hubiese sido colocar en controlador de tipo PID en la salida y entrada del sistema correspondiente, y el algoritmo QDMC hubiese realizado el control de forma indirecta a través de este PID.

Tras la búsqueda de varios procesos, finalmente se ha escogido una columna de destilación de aceite pesado de la compañía Shell Oil. El modelo de la misma fue obtenido por Prett and Morari en 1987, el cual originalmente consta de 7 salidas y 7 variables manipuladas. No obstante, existe el modelo simplificado a un proceso de 3 salidas y 3 entradas [27]. A continuación, se muestra la matriz de transferencia del proceso en cuestión.

$$\begin{bmatrix} Y_{1(s)} \\ Y_{2(s)} \\ Y_{3(s)} \end{bmatrix} = \begin{bmatrix} \frac{4.05e^{-27s}}{1+50s} & \frac{1.77e^{-28s}}{1+60s} & \frac{5.88e^{-27s}}{1+50s} \\ \frac{5.39e^{-18s}}{1+50s} & \frac{5.72e^{-14s}}{1+50s} & \frac{6.09e^{-15s}}{1+50s} \\ \frac{4.38e^{-20s}}{1+33s} & \frac{4.42e^{-22s}}{1+44s} & \frac{7.02}{1+19s} \end{bmatrix} \begin{bmatrix} U_{1(s)} \\ U_{2(s)} \\ U_{3(s)} \end{bmatrix} \quad (3)$$

Las salidas $Y_{1(s)}, Y_{2(s)}$ y $Y_{3(s)}$ corresponden respectivamente con la composición del flujo de la parte superior de la columna, con la composición del flujo de la parte intermedia de la columna y con la temperatura del caudal de recirculación de la parte inferior de la misma.

Las entradas $U_{1(s)}, U_{2(s)}$ y $U_{3(s)}$ corresponden respectivamente con la apertura de las válvulas de caudal de la parte superior, parte intermedia y recirculación inferior de la columna. En la Figura 19 se puede observar con mayor nivel de detalle las entradas, marcadas de color verde, y salidas del proceso, marcadas de color rojo.

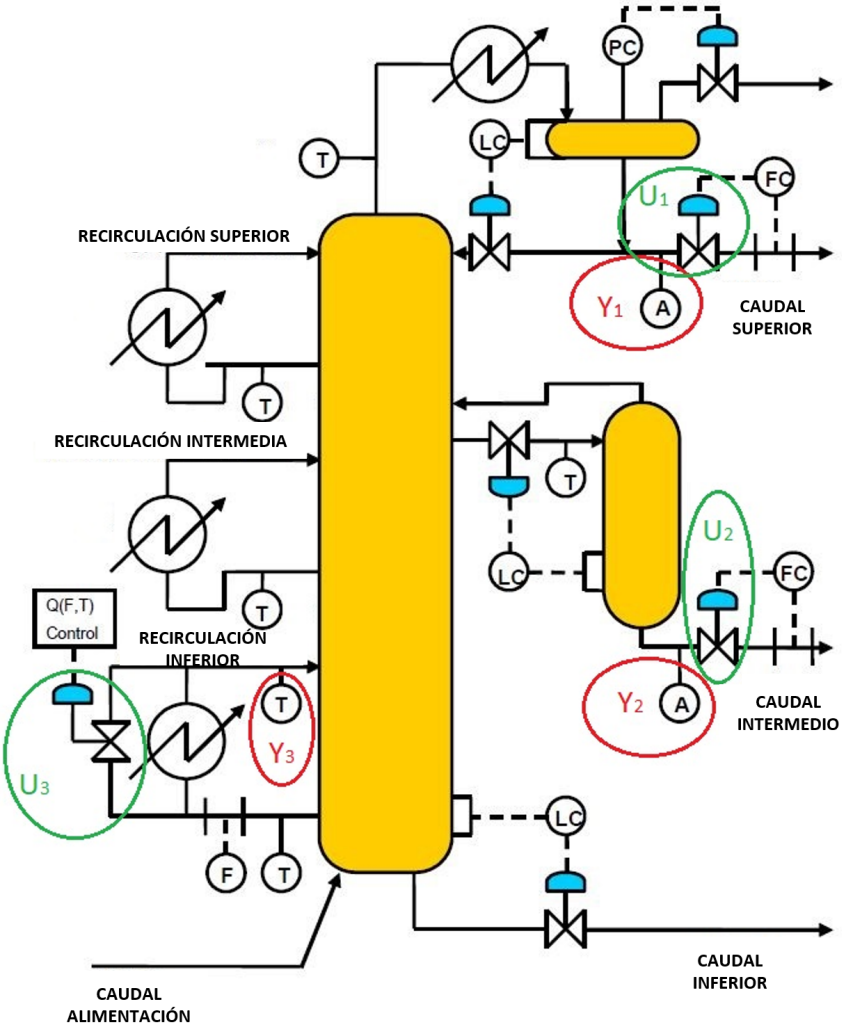


Figura 19: Esquema de la columna de destilación Shell Oil. Modificado a partir de [26]

Otra simplificación que se ha realizado al proceso, es que originalmente las unidades de tiempo del mismo están expresadas en minutos. Entonces, para poder realizar las simulaciones a mayor velocidad, se ha optado por cambiar la unidad de tiempo de minutos a segundos, haciendo que el modelo se estabilice en una menor cantidad de tiempo.

Como se observa en la matriz del modelo, el retardo mínimo de cada una de las variables de salida son 27, 14 y 0 segundos respectivamente. Asimismo, la variable con una dinámica más rápida es la tercera salida ya que en una de sus funciones de transferencia, posee el polo más rápido de todo el proceso, concretamente es la función que relaciona la tercera salida con la tercera entrada.

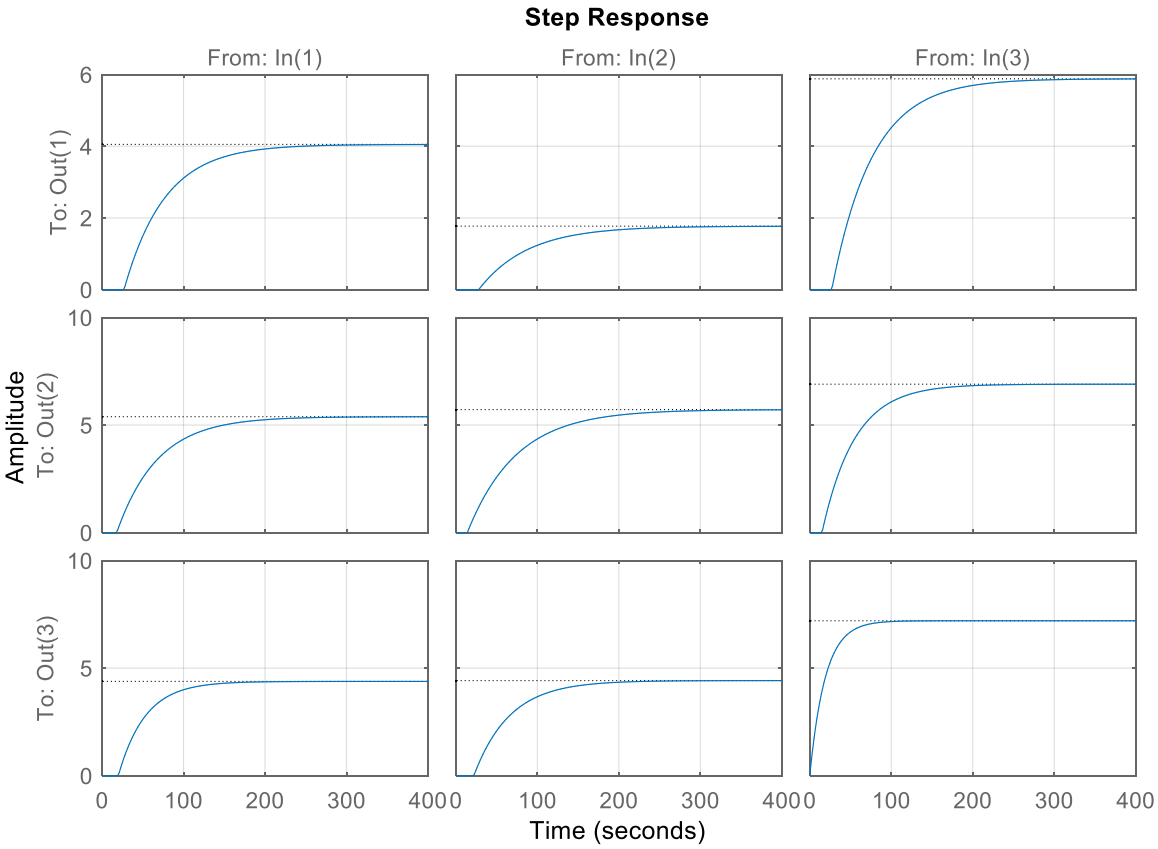


Figura 20: Respuesta del modelo ante una entrada de tipo escalón unitario

En la Figura 20, se encuentra la respuesta del modelo ante una entrada de tipo escalón unitario, la cual se ha obtenido mediante el programa Matlab [15]. La tercera salida posee el menor tiempo de establecimiento de todo el proceso, estabilizándose aproximadamente sobre los 200 segundos. Asimismo, la primera y segunda salida posee un tiempo de establecimiento muy similar ya que el polo más lento que afecta a ambas salidas posee en mismo valor, aproximadamente tarda unos 300 segundos.

En lo que concierne a las restricciones del modelo de la columna de destilación son las siguientes:

- Límites de las acciones de control:
  - $-0.5 \leq u_1 \leq 0.5$; $-0.5 \leq u_2 \leq 0.5$; $-0.5 \leq u_3 \leq 0.5$
  - $-0.05 \leq |\Delta u_1| \leq 0.05$; $-0.05 \leq |\Delta u_2| \leq 0.05$; $-0.05 \leq |\Delta u_3| \leq 0.05$ (Variación por segundo)
- Límites de las salidas del proceso:
  - $-0.5 \leq y_1 \leq 0.5$; $-0.5 \leq y_2 \leq 0.5$; $-0.5 \leq y_3 \leq 0.5$

7.2. Simulación del control QDMC mediante Matlab

Antes de realizar el ensayo sobre la aplicación de control desarrollado, se ha realizado una simulación mediante Matlab para comprobar si el algoritmo QDMC implementado es capaz de realizar el control dentro de las restricciones que impone el proceso. Cabe destacar que el algoritmo QDMC implementado en Matlab y en Python es el mismo, únicamente empleando el código de programación correspondiente para cada uno de los lenguajes de programación.

Para hacer la simulación en Matlab es necesario discretizar el modelo, para ello es necesario escoger un periodo de muestreo (Ts) adecuado. En este caso se ha elegido un Ts igual a 4 segundos obteniendo la matriz del modelo que se puede observar en la ecuación 5. Se ha elegido este periodo en base a que el polo más rápido del proceso es -1/19, siendo el tiempo de establecimiento de la señal al 98%:

$$te = \frac{4}{1/19} = 76\ segundos \tag{4}$$

Y empleando el criterio que el periodo de muestreo debe ser una décima parte del tiempo de establecimiento, se obtiene un Ts máximo de 7,6 segundos. No obstante, para estar del lado de la seguridad se ha usado aproximadamente la mitad del periodo de muestreo máximo, es decir 4 segundos. La matriz obtenida con un periodo de 4 segundos se puede observar en la siguiente ecuación.

$$\begin{bmatrix} Y_{1(z)} \\ Y_{2(z)} \\ Y_{3(z)} \end{bmatrix} = \begin{bmatrix} \frac{0.0802z + 0.2312}{z^7(z - 0.9231)} & \frac{0.1142}{z^7(z - 0.9355)} & \frac{0.1164z + 0.3356}{z^7(z - 0.9231)} \\ \frac{0.2113z + 0.2031}{z^5(z - 0.9231)} & \frac{0.1875z + 0.1814}{z^4(z - 0.9355)} & \frac{0.1704z + 0.4863}{z^4(z - 0.9048)} \\ \frac{0.5}{z^5(z - 0.8858)} & \frac{0.1964z + 0.1877}{z^6(z - 0.9131)} & \frac{1.367}{z - 0.8102} \end{bmatrix} \begin{bmatrix} U_{1(z)} \\ U_{2(z)} \\ U_{3(z)} \end{bmatrix} \tag{5}$$

Asimismo, para comprobar que el modelo discreto describe correctamente el comportamiento dinámico del proceso continuo, se ha realizado una comparación de ambos modelos ante una entrada de tipo escalón unitario. Los resultados se pueden apreciar en la Figura 21, donde el modelo discreto describe perfectamente el comportamiento del modelo.

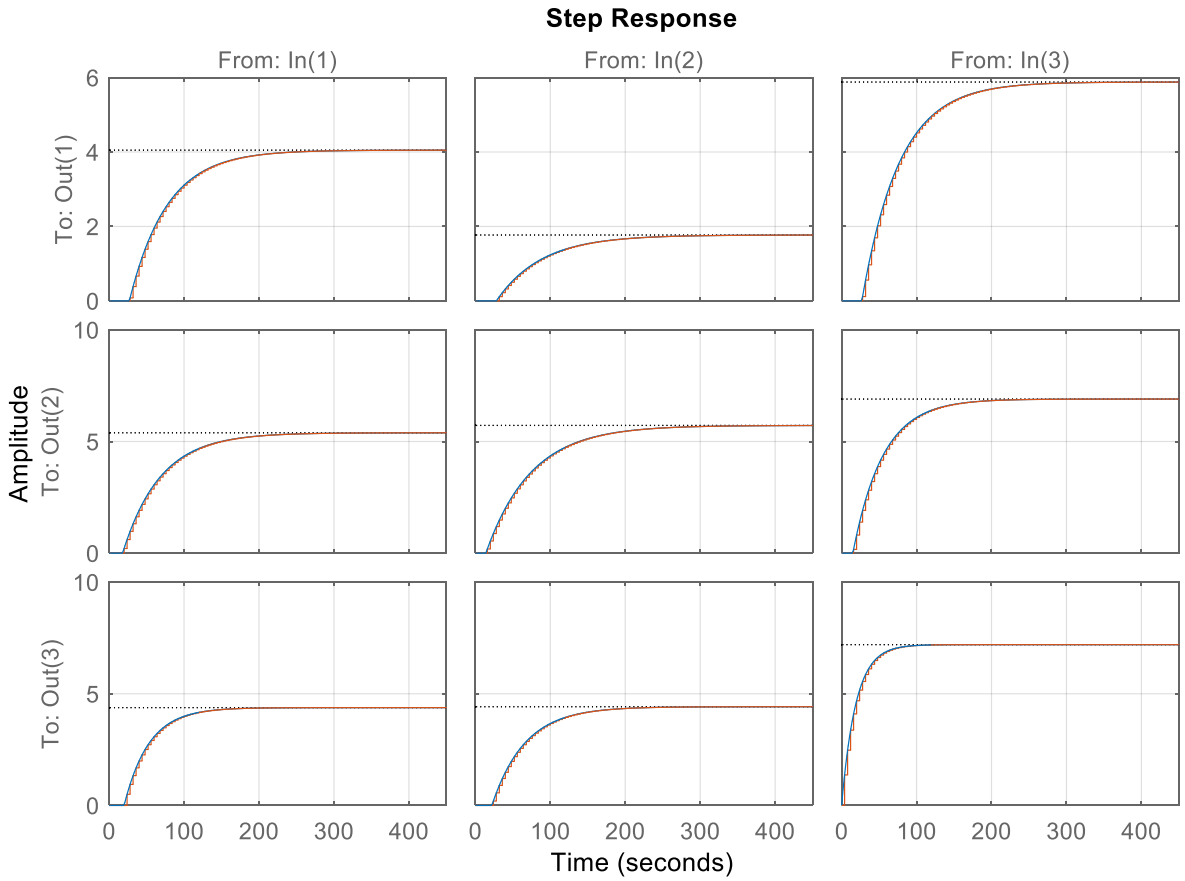


Figura 21: Comparación de la respuesta ante un escalón unitario del proceso continuo (azul) y discreto (rojo) a un Ts = 4 segundos

Una vez realizada la discretización se procede a realizar el control de la planta mediante una simulación en Matlab, con el siguiente perfil de referencias:

- En el instante inicial todas las referencias de la planta poseen un valor igual a cero.
- En el segundo cuarenta de la simulación, la referencia cambia a 0.5, 0.3 y 0.1 respectivamente para cada una de las salidas del proceso.
- En el instante de seiscientos segundos, se cambia la referencia de la primera variable de 0.5 a 0.4.

Los valores de los parámetros para la simulación del algoritmo QDMC son los siguientes:

- **Horizonte de control (c)**: 40
- **Horizonte de predicción (p)**: se ha establecido como la suma del horizonte de control más los instantes que tarda en estabilizarse el sistema, que son aproximadamente de 400 segundos y como el periodo de muestreo es de 4 segundos, entonces el número de instantes es igual a 100. Por lo tanto, el horizonte de predicción es igual a 140.
- **Lambda (λ)**: [3; 3; 2]
- **Alfa (α)**: [2; 0.25; 4]
- **Umax:** [0.5; 0.5; 0.5]

- **Umin**: [-0.5; -0.5; -0.5]
- **Δu**:[0.2; 0.2; 0.2] (en realidad es 0.05/segundo, pero como cada iteración es cada 4 segundos, entonces es 0.2/iteración)
- **Ymax**: [0.5; 0.5; 0.5]
- **Ymin**: [-0.5; -0.5; -0.5]

Una vez establecidos los parámetros, ejecutamos la simulación y el algoritmo QDMC, obteniendo los siguientes resultados que se muestran en la siguiente figura:

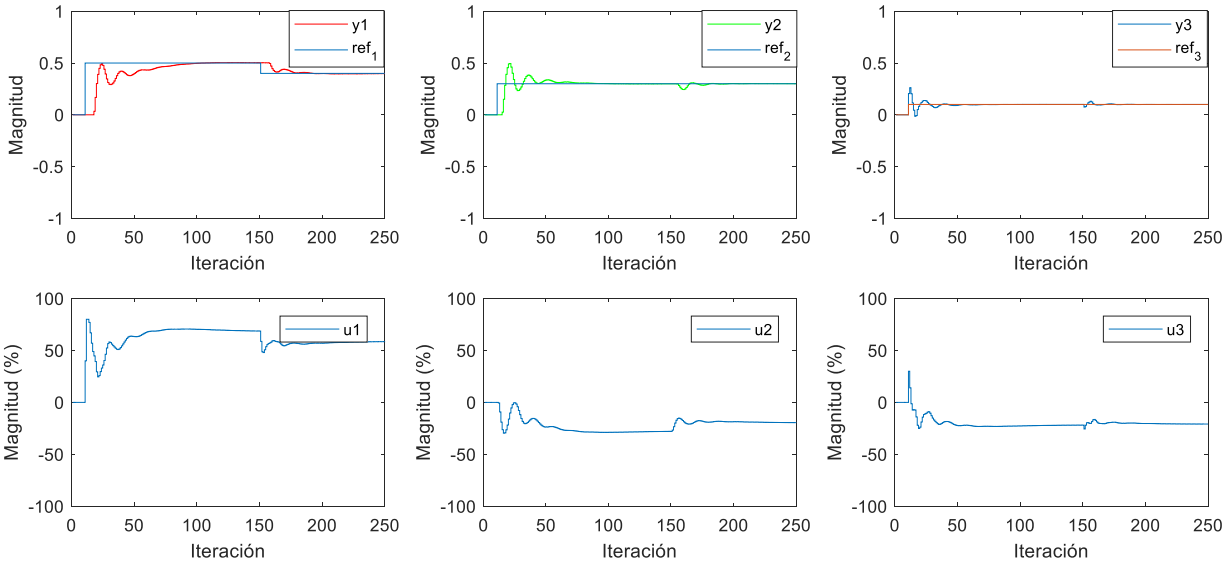


Figura 22: Resultados obtenidos de la simulación del control del proceso mediante Matlab

Como se observa, el algoritmo QDMC es capaz de seguir el perfil de referencias establecidos satisfactoriamente. No obstante, durante el transitorio de la variable de salida 1, en el primer cambio de la referencia, se produce cierta oscilación en las variables 2 y 3 hasta que la variable 1 alcanza la referencia establecida.

7.3. Ensayo del hardware y aplicación de control mediante simulación hardware-in-the-loop

Para realizar el ensayo del hardware y de las aplicaciones diseñadas, es necesario realizar un HIL (hardware-in-the-loop) [14] para realizar la simulación de la planta, ya que no se dispone de la planta real. Esta aplicación se ha realizado en LabVIEW mediante dos tarjetas de adquisición, modelo USB-6001 de datos de National Instruments [2] para recoger y mandar la información de las entradas y salidas del proceso. Según la compañía, estas tarjetas se caracterizan por tener 8 entradas analógicas, 2 salidas analógicas y 13 entradas/salidas digitales. No obstante, en este caso, la tarjeta se encuentra instalada en una carcasa, donde se tiene acceso a 6 entradas analógicas, 2 salidas analógicas y 8 entradas/salidas digitales. Cabe destacar que el rango de tensiones de las entradas y salidas digitales es de [-10; 10] voltios. Al poseer todas las tarjetas de adquisición de datos el mismo rango de tensiones,

no hay ningún riesgo de dañar durante el ensayo ninguna de ellas. En la Figura 23, se observa la una de las TAD con su correspondiente carcasa.

Figura 23: A la izquierda TAD USB-6001 de NI fuente National Instruments. A la derecha bornera donde se instala la TAD y que facilita el conexionado de las entradas y salidas de la misma.

En la Figura 24, se muestra el esquema básico del cableado correspondiente al montaje para realizar la simulación, donde las salidas analógicas de la Raspberry Pi + TAD se conectan a las entradas analógicas de la TADs Usb 6001 y viceversa.

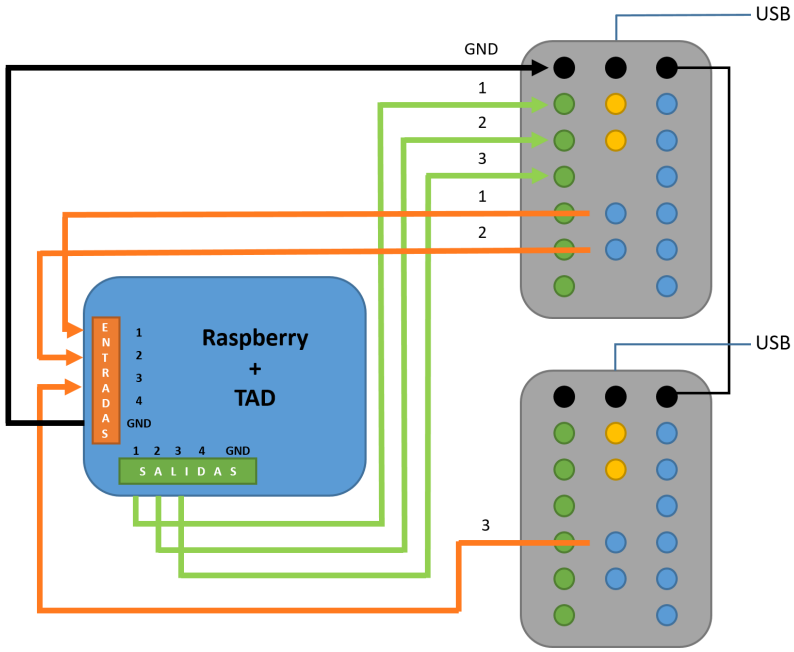


Figura 24: Diagrama básico del cableado entre las tarjetas de adquisición de datos

En la Figura 25 se muestra el montaje definitivo para realizar el ensayo de la aplicación de control diseñada, donde en la pantalla de la izquierda se observa el panel frontal del HIL y la consola de comandos de la RPi donde se muestran en cada iteración las referencias, las acciones de control y las salidas del proceso. Asimismo, también muestra el tiempo sobrante de cada iteración para saber si el algoritmo QP se realiza dentro del periodo establecido para el bucle del controlador. Por otro lado, en la pantalla de la derecha se muestra la interfaz gráfica diseñada, mostrando la evolución de cada una de las variables de interés del proceso.

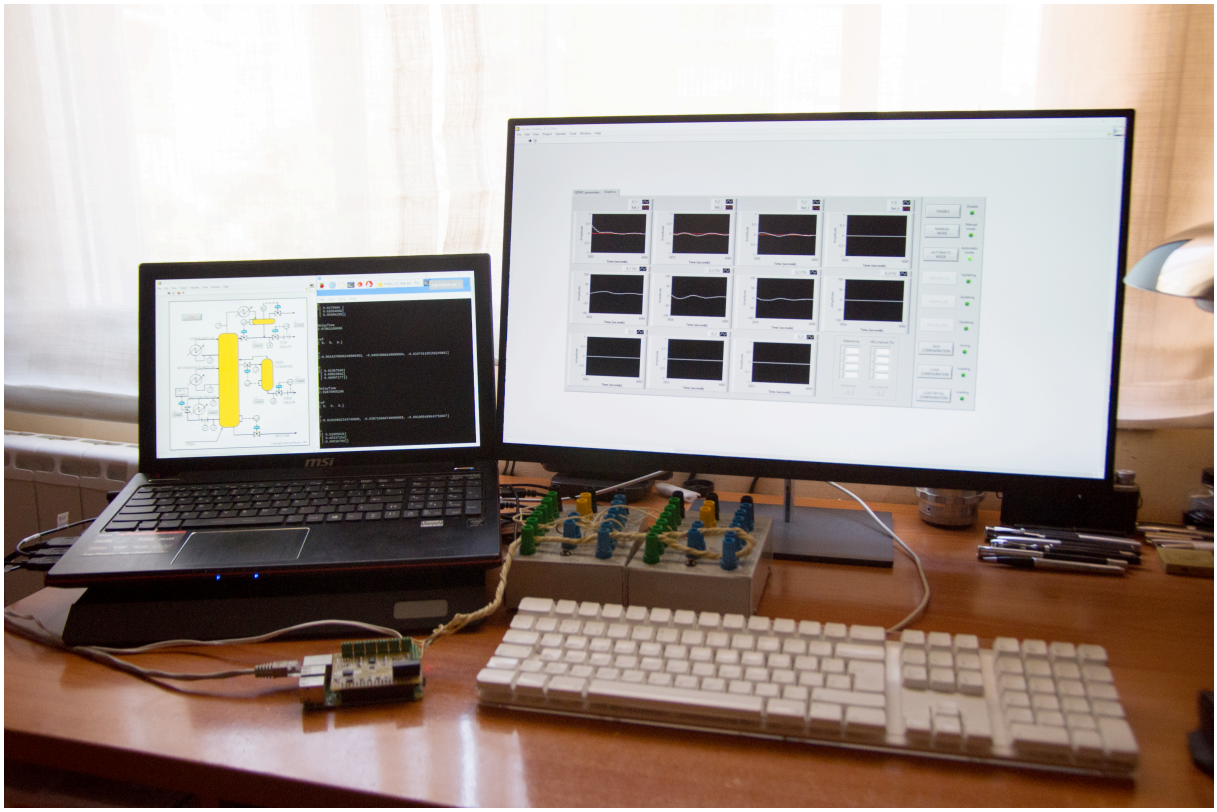


Figura 25: Montaje de la aplicación de control con su correspondiente hardware

En la Figura 26, se observa el panel frontal de la aplicación HIL, donde se muestra una ilustración del proceso y una serie de indicadores que muestran los valores de las entradas y salidas del sistema en todo momento.

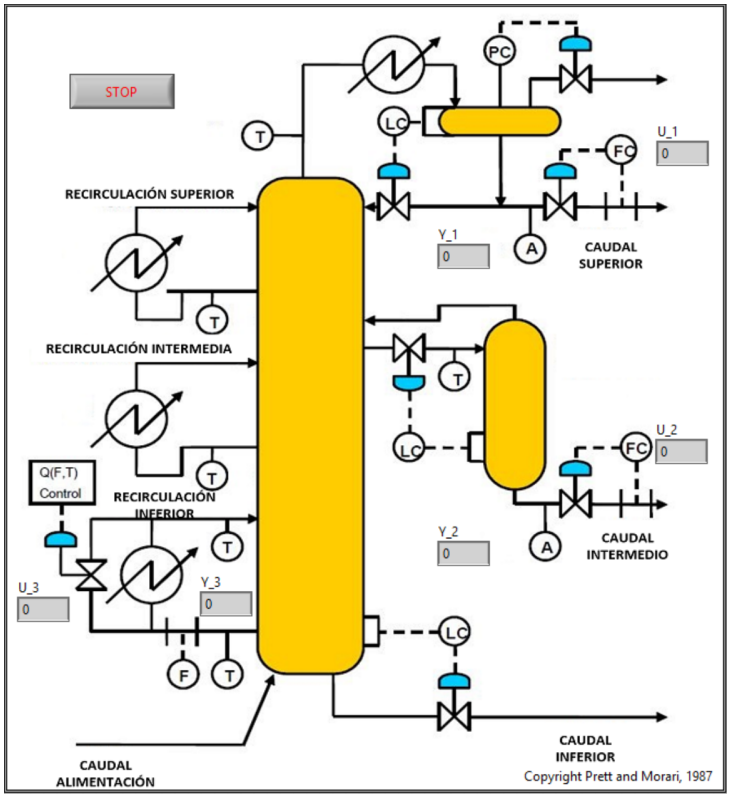


Figura 26: Panel frontal de la aplicación hardware-in-the-loop

Seguidamente se procede con el ensayo correspondiente, el cual se han establecido los mismos parámetros y mismo perfil de referencias, que en la simulación realizada con Matlab para comparar los resultados obtenidos. En la Figura 27, se muestra el panel frontal de la interfaz gráfica con la evolución de cada una de las variables del proceso durante el ensayo.

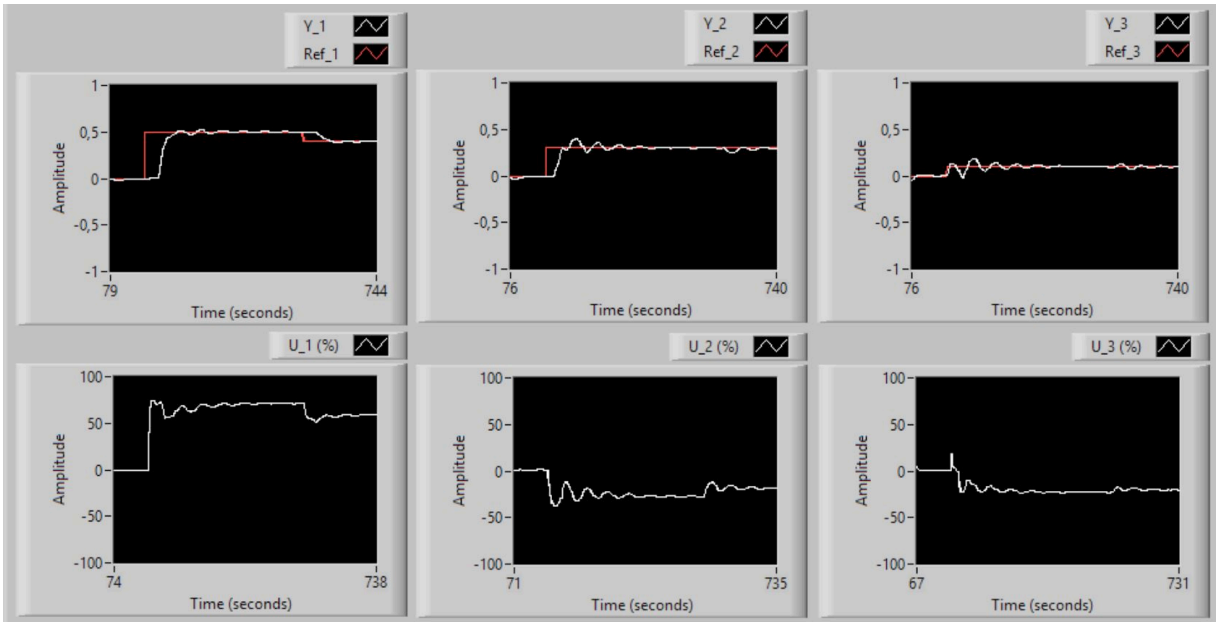


Figura 27: Ensayo completo de la aplicación de control diseñada

La aplicación de control, junto con el correspondiente hardware, es capaz de realizar en gran medida el seguimiento de la referencia a lo largo de todo el ensayo. No obstante, en comparación con los resultados obtenidos mediante Matlab, difieren en algunos tramos de la simulación. El ensayo con la aplicación diseñada, las variables 2 y 3 poseen una cierta oscilación alrededor de la referencia debido a que durante ese periodo de tiempo, la variable 1 aún no ha alcanzado su correspondiente referencia. No obstante, en el momento que todas han alcanzado la referencia, aun se observa cierta oscilación en cada una de las salidas del proceso. Esto es debido a que este ensayo es real y no es una simulación realizada mediante software como la realizada mediante Matlab, y el ensayo se ve expuesto a ciertos fenómenos como retardos a la hora de aplicar las acciones de control por el tiempo de la optimización o el error de cuantificación que se introduce en la medida debido a los componentes que forman las TADs empleadas.

Otros de los fenómenos observados durante el ensayo, es que en el momento que el proceso se acerca a alguna de las restricciones de las variables de salida, como es el caso de la variable 1, el algoritmo de optimización QP comienza a necesitar mayor capacidad de computación. Este hecho provoca que la Raspberry Pi comience a calentarse debido a la demanda de recursos por parte del algoritmo y a su vez, llegada a una cierta temperatura, la RPi comienza a reducir el rendimiento de la misma para evitar que alguno de los componentes se sobrecalienten y produzca algún fallo en el hardware, sobretodo en el procesador de la misma. Todo esto provoca que en algunas de las iteraciones con estas características, se sobrepase el periodo establecido para la duración de cada una de las iteraciones del bucle de control en unos milisegundos. Este hecho a tener en cuenta, puede provocar que no se calculen o apliquen las acciones de control óptimas en alguna de las iteraciones.

En comparación con los resultados obtenidos mediante Matlab, son bastante similares excepto en el primer cambio de la referencia de la variable 1, donde en el ensayo realizado mediante el sistema diseñado, esta alcanza la referencia de una forma suave y progresiva. Sin embargo, en la simulación mediante Matlab, dicha variable se comporta de una forma más agresiva, produciéndose algunas oscilaciones mientras se produce el transitorio hasta alcanzar la referencia.

Por otro lado, en el momento correspondiente al cambio de la referencia 1 de 0.5 a 0.4, no se producen tantas oscilaciones y posee un comportamiento muy similar al obtenido en la simulación realizada mediante Matlab, como en el resto de la simulación.

8. CONCLUSIONES

En lo referente a las conclusiones del proyecto, se han ido mencionando a lo largo de la memoria. No obstante, a continuación se va a hacer una recapitulación de las más importantes.

La mayor satisfacción ha sido que se ha cumplido el objetivo de mayor importancia de este proyecto, que consiste en que todas las partes que conforman el dispositivo de control hayan funcionado perfectamente a lo largo de todas los ensayos realizados y además, que todas estas partes hayan cumplido las especificaciones que se han establecido a lo largo del desarrollo del mismo. Asimismo, se han establecido las condiciones bajo las cuales, el dispositivo de control es capaz de funcionar correctamente.

Por otra parte, destacar el hecho de que este sistema de control únicamente es capaz de realizar el control de procesos relativamente "lentos", donde el tiempo de establecimiento del polo más rápido del sistema debe ser aproximadamente 10 veces superior al periodo mínimo de muestreo del bucle del controlador, que según los parámetros establecidos en los horizontes de control y predicción, se puede llegar a establecer en aproximadamente 2 segundos, lo cual es bastante razonable para un dispositivo de bajo coste como el que se ha empleado a lo largo de este proyecto.

Asimismo, todas las aplicaciones aquí diseñadas son susceptibles de ser mejoradas, pero que por falta de tiempo o recursos no se han podido realizar. A continuación, se van a relatar algunas de las posibles mejores a realizar en el dispositivo bajo mi punto de vista:

- Posibilidad de colocar varias tarjetas de adquisición de datos en la Raspberry Pi a través de su puerto de expansión, para ser capaces de realizar el control de procesos con un mayor número de entradas o salidas.
- Establecer algún sistema de refrigeración a los diversos procesadores de la RPi, ya que como se ha mencionado anteriormente, el sobrecalentamiento de los componentes hace decrecer el rendimiento del dispositivo.
- Instalar en la TAD un microcontrolador que este sea capaz de controlar mediante PIDs los lazos de control de bajo nivel y que el algoritmo de control QDMC realiza el control sobre estos PIDs estableciendo así un control jerárquico.
- Emplear como unidad de procesamiento la última versión de la RPi, la cual proporcionará un mayor rendimiento al sistema de control.
- Optimización del código del controlador.
- Aumentar el número de funcionalidades de la interfaz gráfica.
- Realizar pruebas con otros algoritmos de control que no se han empleado a lo largo de este proyecto.

Para finalizar con esta memoria, mencionar que uno de los mayores retos ha sido hacer frente a situaciones a resolver que nunca antes había realizado, lo cual ha supuesto un gran esfuerzo que finalmente ha dado sus correspondientes frutos.

9. BIBLIOGRAFÍA

[1] National Instruments (s.f.) *Información detallada sobre el Protocolo Modbus.* Obtenido el 16 de Junio de 2018 de http://www.ni.com/white-paper/52134/es/

[2] National Instruments (s.f.) *Especificaciones técnicas del dispositivo USB-6001.* Obtenido el 15 de Julio de 2018 de http://www.ni.com/es-es/support/model.usb-6001.html

[3] López-Guede, JM; Fernández-Gauna, B y Graña, M (2013) *On the influence of the prediction horizon in dynamic matrix control.* Obtenido el 30 de Junio de 2018 de http://article.sapub.org/pdf/10.5923.j.control.20130301.03.pdf

[4] Raspberry Pi (s.f.) *What is a Raspberry Pi?* Obtenido el 10 de Julio de 2018 de https://www.raspberrypi.org/help/faqs/

[5] C. Bordóns (2000). *Control Predictivo: metodología, tecnología y nuevas perspectivas*. Universidad de Sevilla

[6] C.E. García., D.M. Prett., M. Morari. (1989) *Model predictive Control: Theory and Practice – a Survey*. Publicación, International Federation of Automatic Control, Gran Bretaña.

[7] J.M. Maciejowski (2002). *Predictive Control with Constraints*. Prentice Hall.

[8] D.M. Prett., B.L. Ramaker., C.L. Cutler (1979). *Dynamic Matrix Control Method*. United States Patend.

[9]E. García, C.; M. Morshedi, A. (1985) *Chemical Enginering Communications: Quadratic Programming Solution of Dynamic Matrix Control (QDMC).* Shell Development Company, Texas.

[10] MODICON (1996) *Modicon Modbus Protocol Reference Guide*.

[11] Kuang, J.; Wang, G. y Bian, J. (2012) *A Modbus Protocol Stack Compatible with RTU/TCP Frames and Embedded Application*. In: Zhu M. (eds) Business, Economics, Financial Sciences, and Management. Advances in Intelligent and Soft Computing, vol 143. Springer, Berlin, Heidelber

[12] Python Software Foundation (s.f.) *About Python.* Obtenido el 15 de Julio de 2018 de https://www.python.org/about/

[13] SB Component (s.f.) *Piface Digital 2 Case.* Obtenido el 15 de Julio de 2018 de https://shop.sb-components.co.uk/products/pi-face-digital-2-clear-case

[14] National Instruments (s.f.) *Hardware-in-the-loop.* Obtenido el 12 de Julio de 2018 de http://www.ni.com/es-es/innovations/automotive/hardware-in-the-loop.html

[15] Matlab r2017b (s.f.) *Context Help.*

[16] Sanchis, J. (2018) *Basics of Industrial Model Predictive Control.* Control Industrial Avanzado, Máster Ingeniero Industrial. ETSII.

[17] Sanchis, J. (2018) *Dynamic Matrix Control (DMC).* Control Industrial Avanzado, Máster Ingeniero Industrial. ETSII.

[18] Sanchis, J. (2018) *Quadratic Dynamic Matrix Control (QDMC).* Control Industrial Avanzado, Máster Ingeniero Industrial. ETSII.

[19] Astrom, K. J.; Hagglund, T. (1995) *PID controllers: Theory, Design and Tuning.* Instruments Society of America

[20] Debian (s.f.) *Acerca de Debian.* Obtenido el 5 de Julio de https://www.debian.org/intro/about

[21] Spurgeon, C. (2009) *Ethernet: The Definitive Guide*. O´Reilly Media

[22] National Instruments (s.f.) *¿Qué es LabVIEW?* Obtenido el 6 de Julio de http://www.ni.com/es-es/shop/labview/labview-details.html

[23] Wilder, F. (1998) *Guide to the TCP/IP Protocol Suite.* Artech House Telecommunications Library

[24] Moreno Fernández, A. (2004) *El bus I2C.* Obtenido el 28 de Julio de 2018 de http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

[25] Lee, J.H. Int. J. (2011) Control Autom. Syst. Obtenido el 15 de Julio de 2018 de https://doi.org/10.1007/s12555-011-0300-6

[26] Kortela, J. (2016) *CHEM-E7145. Advanced Process Control Methods.* Obtenido el 17 de Julio de 2018

[27] F. Camacho, E. y Bordons Alba, C. (2013) *Springer Science & Business Media.* Obtenido el 10 de Julio de 2018

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC:

ANEXO 1: MANUAL DE DISEÑO DE LA TARJETA DE ADQUISICIÓN DE DATOS PARA LA RASPBERRY PI

ÍNDICE

ÍNDICE DE FIGURAS:

1. INTRODUCCIÓN

En este documento, se va a explicar todo lo relacionado con la tarjeta de adquisición de datos (TAD) diseñada para esta aplicación, desde las especificaciones y objetivos de diseño hasta la implementación de un prototipo completamente funcional. Asimismo, se expondrán todos los componentes empleados para el desarrollo de la misma y el motivo de su uso.

2. ESPECIFICACIONES DE DISEÑO

Las especificaciones de diseño para la TAD son las siguientes:

- Las dimensiones de la TAD deben ser de 65 mm de largo, por 56 mm de ancho. Estas dimensiones son relativamente más pequeñas que las de la Raspberry Pi, para que la TAD no colisione con las conexiones USB y Ethernet, que esta dispone.
- Cuatro entradas de tipo analógico con un rango de tensión de ±10 V.
- Cuatro salidas de tipo analógico con un rango de tensión de ±10 V.
- Ocho entradas/salidas de tipo digital configurables vía software, con un rango en tensión de [0; 5] V
- Establecer comunicación con la RPi mediante el puerto de expansión de la misma.
- Protocolo I^2C para establecer comunicación con los diferentes chips empleados.
- Sistema de protección de las entradas analógicas, en caso de introducir una tensión fuera del rango establecido.
- Fuente de alimentación propia de ±12 V para alimentar algunos de los componentes de la TAD.
- Disponer de un terminal para alimentar a 5 V la TAD y la RPI.

3. COMPONENTES HARDWARE EMPLEADOS

En este apartado se van a describir el hardware empleado, sobretodo los chips Digital-to-Analog Converter (DAC), Analog-to-Digital Converter (ADC) y Digital I/O expander.

La Figura 1 muestra un diagrama de bloques de las diferentes partes que forman la TAD para realizar las funciones descritas anteriormente. A continuación se explicarán cada uno de estos bloques.

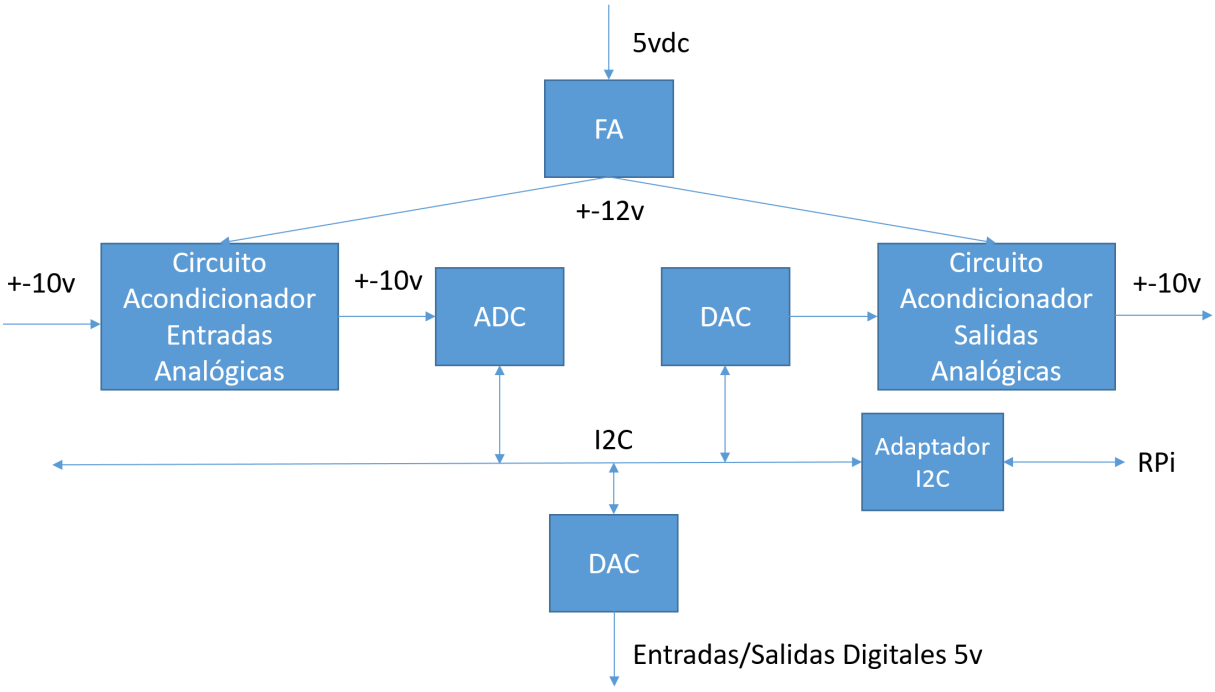


Figura 1. Diagrama de bloques funcional de la TAD diseñada.

3.1. Bloque Entradas Analógicas

Este bloque lo forma una circuitería analógica y el conversor ADC.

En primer lugar, se va a describir el ADC empleado para esta aplicación, el MCP3424 [1] del fabricante Microchip, que se caracteriza por tratarse de un dispositivo con bajo ruido blanco y una alta precisión en la conversión, ya que posee un sistema de conversión de hasta 18 bits. Este convertidor se encuentra disponible en diferentes encapsulados, pero en esta ocasión se va emplear el encapsulado SOP, debido a que es el de menor tamaño disponible. Por otro lado, según el datasheet del fabricante, este ADC posee 4 entradas de tipo analógico con un rango de tensión diferencial entre los pines de cada canal de ± Vref (tensión de referencia). Para este caso se va a emplear la Vref que posee el propio chip (2.048 V). No obstante, en este caso únicamente se va a poder aprovechar el rango de 0 a 2.048 V, puesto que uno de los terminales de cada entrada va a estar conectado a masa, provocando que únicamente se puede introducir voltajes no diferenciales.

Asimismo, el datasheet indica que posee varios modos de funcionamiento configurables vía software para la adquisición de datos, donde se configuran los bits y la velocidad de adquisición. Conforme se disminuye la resolución del ADC, mayor es la cantidad de muestras por segundo es capaz de adquirir este dispositivo. Posee cuatro modos diferentes, donde en esta aplicación se va a emplear

el modo de 12 bits que proporciona una tasa de muestreo de 240 SPS. Esta tasa es lo suficiente rápida para el proceso a controlar, ya que el periodo de muestreo del mismo es de 4 segundos.

El bloque de circuitería analógica se encarga de adaptar el rango de tensión de entrada de ±10 V al rango compatible con el chip ADC. Este bloque se explicará con un mayor detalle en los posteriores apartados.

3.2. Bloque salidas analógicas

Este bloque está formado por el conversor DAC y una circuitería analógica.

En lo que concierna al DAC, se ha empleado el MCP 4728 [2] de la compañía Microchip, que se caracteriza por ser un dispositivo de bajo consumo y con amplificador a la salida de alta precisión. Según el datasheet, este posee 4 salidas de tipo analógico, cada una de ellas con una resolución de 12 bits y un rango de tensión a la salida de 0 a 4.096 V. Asimismo, es compatible con el protocolo I^2C y está disponible con el encapsulado SOP. Finalmente, remarcar que posee dos modos de funcionamiento, uno que envía la información de cada canal en el momento que este recibe un nuevo valor, o puede esperar a que todos los canales del dispositivo posean el nuevo dato y volcarlos simultáneamente a las salidas del DAC. La configuración de estos modos se realiza mediante el terminal LDAC de carácter digital.

El bloque de circuitería posee la función de convertir el rango de tensiones del DAC al rango de tensiones de salida de ±10 V.

3.3. Bloque entradas/salidas digitales

Por otro lado, el digital I/O expander se ha instalado por si fuese necesario algún tipo de entrada o salida de tipo digital y así no fuese necesario emplear las que dispone la RPi. El chip empleado es el MCP23008 [3] del fabricante Microchip. Según el datasheet del componente, está compuesto por 8 entradas o salidas digitales configurable vía software con un rango de tensión de 0 a 5 V. El encapsulado empleado para esta aplicación es el SOP, como en el resto de convertidores.

3.4. Otros bloques

Otros de los bloques necesarios es el que corresponde con la fuente de alimentación (FA) que proporciona la tensión necesaria (±12 V) para alimentar parte de la circuitería analógica, a partir de la tensión de la RPi de 5 V.

Por último, cabe destacar que todos los componentes empleados usan el protocolo I^2C pero a una tensión de 5 V. Para que sea compatible con la RPi es necesario adaptar los niveles de tensión, puesto que esta emplea este protocolo a una tensión de 3.3 V. Para ello se emplea el bloque adaptador de tensión I^2C.

4. PROGRAMAS DE SIMULACIÓN Y DISEÑO EMPLEADOS

Los programas que se han empleado para la realización de la tarjeta de adquisición de datos son los siguientes. El primero de ellos es Multisim [9] de la compañía National Instruments, que se caracteriza por ser un programa de diseño de circuitos electrónicos que ofrece la posibilidad de realizar simulaciones en línea de estos. Mediante la herramienta scope, se pueden visualizar por pantalla los diversos valores referentes a voltajes e intensidades que teóricamente circulan por las diferentes partes del circuito diseñado. En la Figura 2, se observa el entorno gráfico de este programa donde los bocadillos de color amarillo corresponden con los scopes del programa. En este caso, Multisim ha sido útil para diseñar toda la circuitería de la TAD, pero en especial para toda aquella que se refiere a las entradas y salidas analógicas.

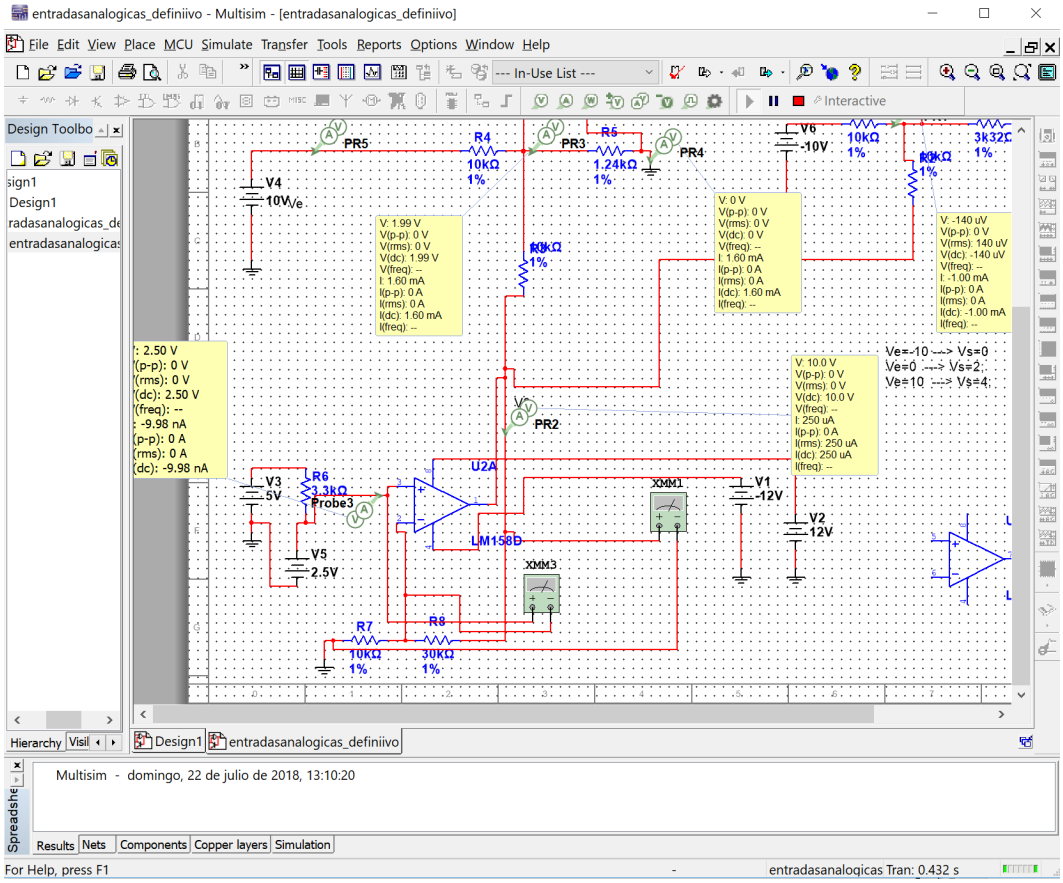


Figura 2: Entorno gráfico del programa Multisim

Por otro lado se ha empleado el programa gratuito DesignSpark PCB [8], propiedad de la compañía suministradora de componentes electrónicos RS Components. Este posee las funcionalidades de diseño de esquemas y de circuitos impresos (PCB). No obstante, no ofrece la herramienta de simulación de la circuitería diseñada, por este motivo ha sido necesario emplear el programa anteriormente mencionado. DesignSpark consta de dos entornos principales, el primero hace referencia al diseño de circuitos electrónicos y el segundo, al diseño de PCBs.

Finalmente, la metodología de trabajo ha sido la siguiente. En primer lugar se realiza el diseño de la circuitería de las diferentes partes de la TAD en Multisim y se realiza la simulación de cada uno

de los circuitos, comprobando que funcionan de forma correcta. Seguidamente, se pasan de forma manual los esquemas realizados en Multisim a DesignSpark. Una vez ya realizada todo el diseño de la circuitería de la TAD, se comienza a realizar el diseño de la PCB en este mismo programa.

No se ha usado el software Utliboard de NI compatible con MultiSim por no disponer de licencia gratuita.

5. CIRCUITERÍA DE LA TARJETA DE ADQUISICIÓN DE DATOS

A continuación se van a explicar cada una de las partes de la que consta la circuitería de la TAD. La primera parte que se va a describir, es la correspondiente a las entradas analógicas. Es necesario diseñar un circuito que sea capaz de reducir el rango de tensión de ±10 V al de 0 a 2.048V, que es el rango admitido por el ADC.

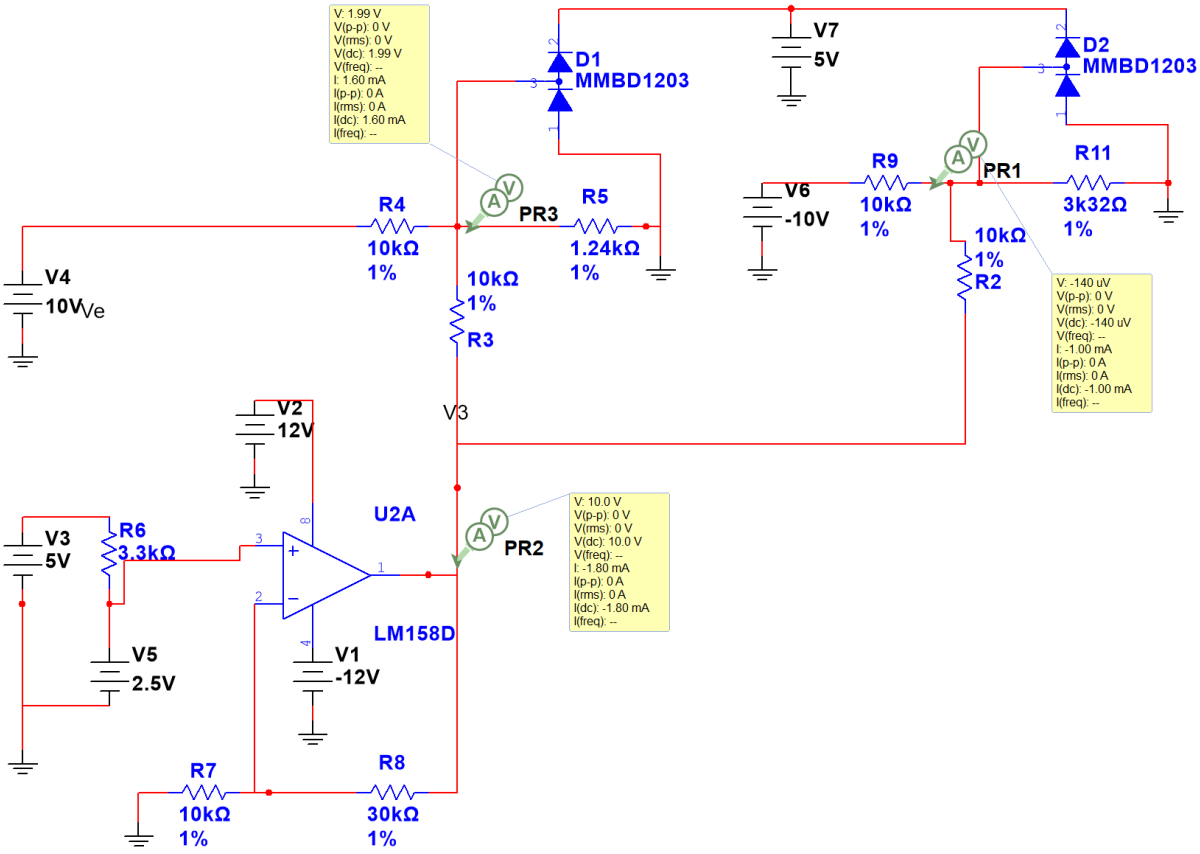


Figura 3: Simulación del circuito de dos de las entradas analógicas de la TAD mediante Multisim

El circuito encargado de hacer esta conversión está compuesto de tres partes fundamentales. La primera, consta de un amplificador operacional (LM158 de la compañía Texas Instruments) [6] que proporciona la tensión de referencia necesaria para la siguiente parte del circuito, que debe poseer un valor de 10 V. Para obtener este valor es necesario una fuente de alimentación de 2.5 V a la entrada del operacional, que es proporcionada por la tensión del diodo zener (ADR5041) [10] en polarización inversa.

La segunda parte del circuito es un divisor resistivo que mediante el voltaje de referencia y las resistencias correctas, es capaz de reducir el valor de ±10 V (Vin) al rango de 0 a 2 V (Vadc). La recta de conversión será la siguiente:

$$Vadc = 0.1Vin + 1 \quad (1)$$

Finalmente, la última parte del circuito es la que corresponde con los diodos de protección (MMBD1204), los cuales conducen corriente en el momento que se supera la tensión máxima (5 V) que soporta cada uno de los canales del ADC, evitando dañar el convertidor.

En la Figura 3, se muestra la simulación de 2 entradas analógicas, donde se observa el correcto funcionamiento del circuito en sí. Asimismo, en la siguiente figura se muestra el esquema definitivo de todas las entradas analógicas, donde el terminal V_out_ai corresponde con el voltaje de referencia mencionado anteriormente y los terminales CH, a las entradas de cada uno de los canales del ADC. Por otro lado, los terminales J6 y J7 es donde se conectarían los terminales de los sensores del proceso, los cuales cada dos entradas poseen un terminal de masa común para economizar el número de terminales en la PCB.

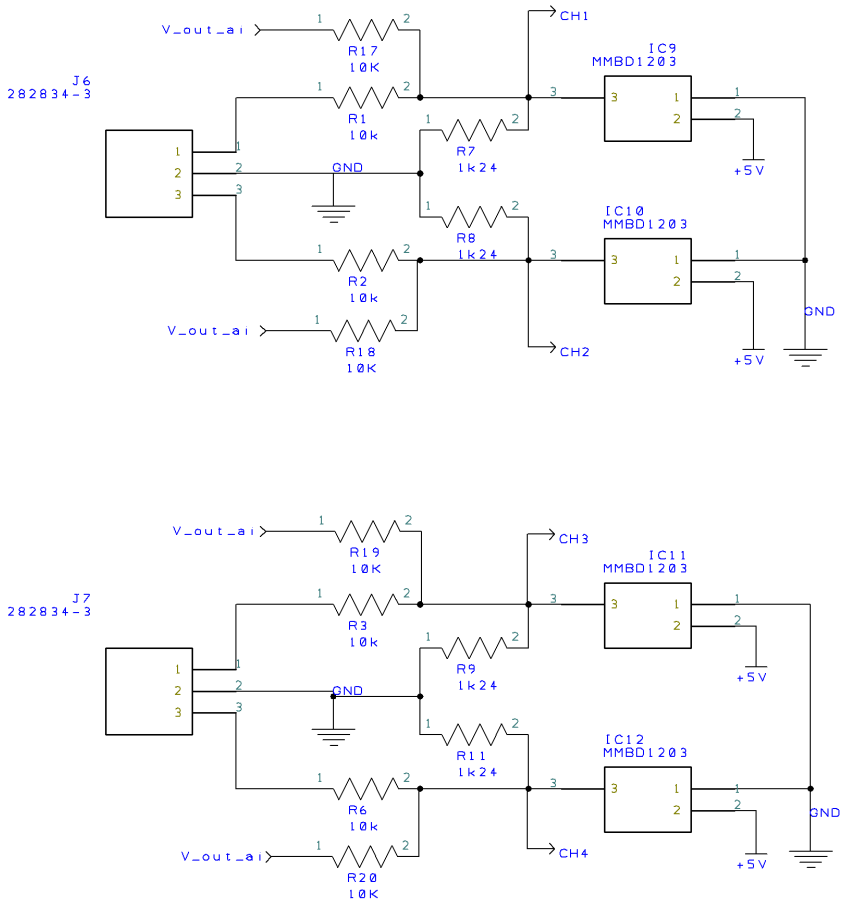


Figura 4: Esquema de las cuatro entradas analógicas en DesignSpark

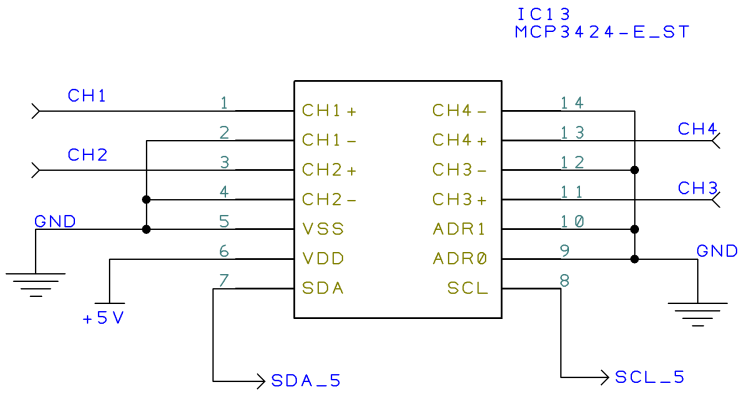


Figura 5: Diagrama de las conexiones del MCP3424

En la Figura 5 se observa el diagrama del MCP3424, donde el terminal positivo se ha conectado al CH del canal correspondiente y el terminal negativo a masa. Por otro lado, los terminales que se refieren a los últimos bits de la dirección del dispositivo en el bus, se han establecido a masa atribuyendo dos ceros a estos bits, ya que únicamente se va establecer un dispositivo de este tipo en la TAD. Los terminales Vdd y Vss se han conectado a los 5 V de la fuente de alimentación y a masa respectivamente. Finalmente, los terminales SDA y SCL se han conectado a la parte del bus que posee una tensión de 5 V.

En lo que concierne a la circuitería de las salidas analógicas, esta debe de convertir el rango de voltaje [0; 4] (Vdac) hasta el valor de ± 10 V (Vout). La recta de conversión es la siguiente:

$$Vout = 5Vdac - 10 \quad (2)$$

Para ello se van a emplear una serie de amplificadores operacionales, concretamente cinco, donde uno (LM158) [6] de ellos posee la función de generar el voltaje de referencia (-2 V) y el resto (LM124) [5] de amplificar la señal de cada uno de los canales del DAC.

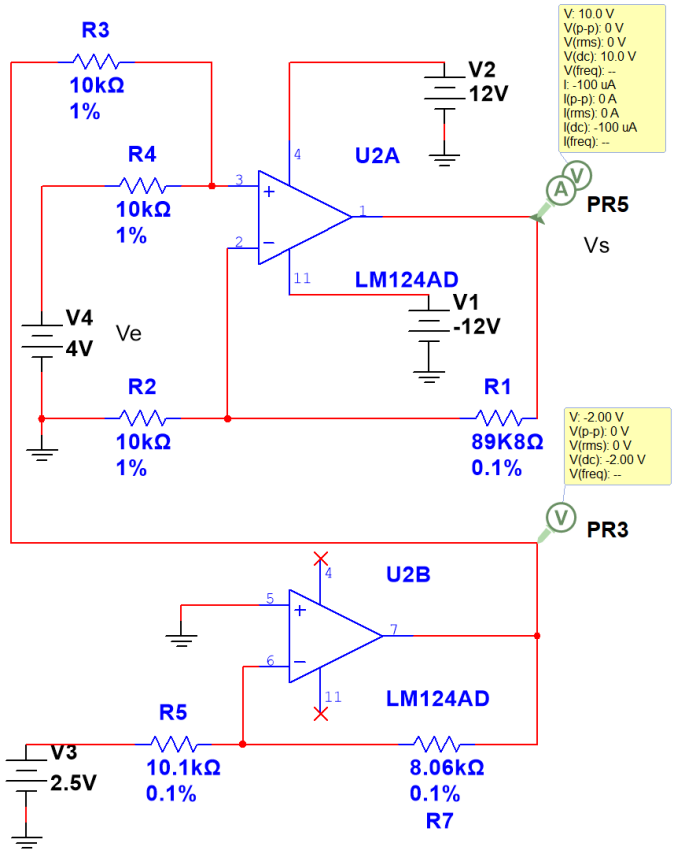


Figura 6: Simulación del circuito de una salida analógica de la TAD

En la Figura 6, se observa en la parte inferior, la circuitería referente a la gestión de la tensión de referencia y en la parte superior la parte que corresponde con la amplificación de la señal de cada uno de los canales.

En la Figura 7, se puede apreciar los diagramas del MCP4728 y del LM124. En primer lugar se va describir la conexión de los terminales del primero de ellos, donde los terminales de alimentación (Vdd y Vss) se han conectado a la alimentación de la TAD de 5 V y masa respectivamente. Seguidamente, los terminales Vout, se han conectado a cada salida de la TAD que previamente pasan por el amplificador operacional para realizar la conversión al voltaje deseado. Los terminales SDA y SCL se han conectado a la parte del bus de 5V y el terminal LDAC a la salida digital 5 de la RPi. En lo referente al LM124, los terminales IN+ y IN- de cada uno de los canales, se conectan a una serie de resistencias, que se observan en la Figura 8, las cuales configuran la ganancia del amplificador operacional. Por otro lado, los terminales V+ y V- se conectan con la alimentación de ±12V. Finalmente, los terminales Vout se conectan con los terminales J9 y J10 (Figura 8), que siguen la misma configuración que los anteriores, por cada dos salidas se comparte un terminal de masa.

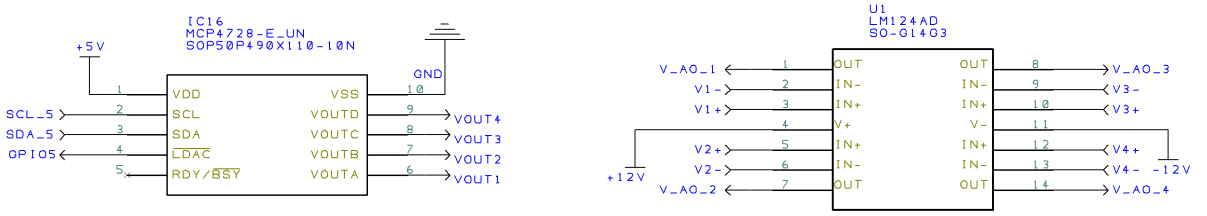


Figura 7: Diagrama del MCP4728 (izquierda) y del LM124 (derecha)

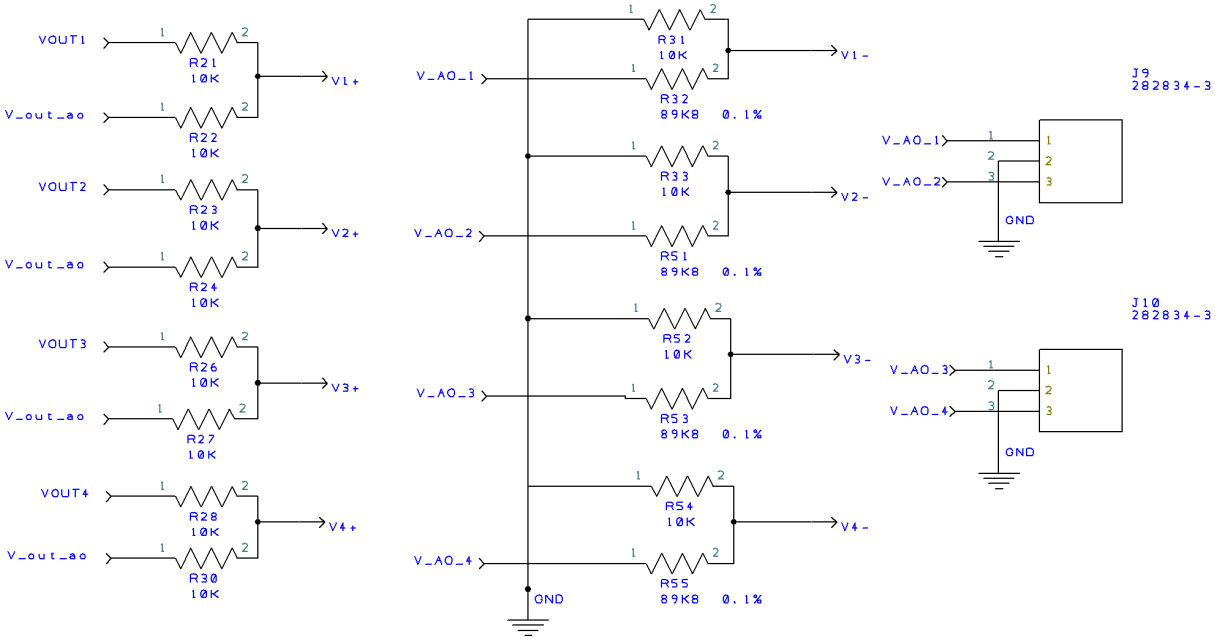


Figura 8: Circuitería de las salidas analógicas en DesignSpark

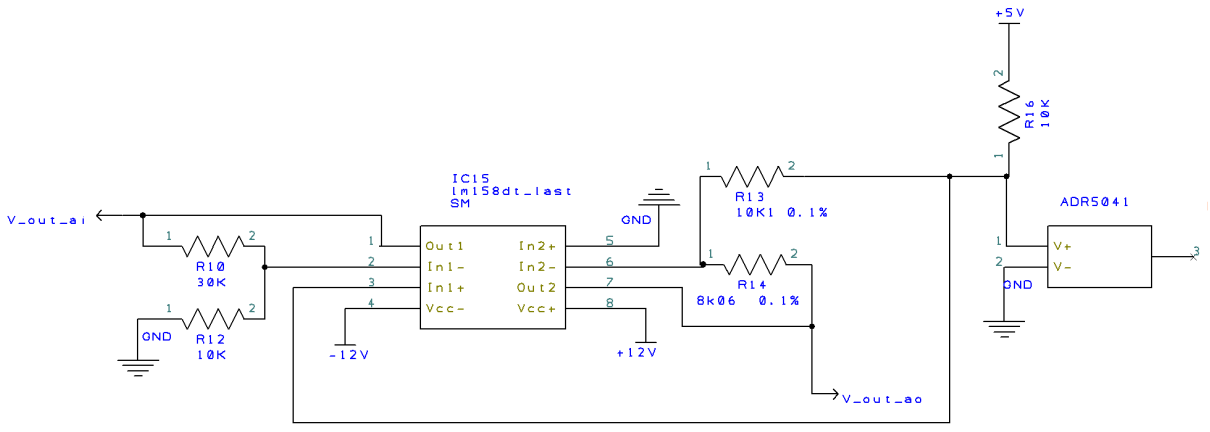


Figura 9: Circuitería del LM158

En la Figura 9, se observa la circuitería del LM158 que proporciona las tensiones de referencia de las entradas (V_out_ai) y salidas (V_out_ao) analógicas. Asimismo, se observa que este operacional se alimenta a ± 12 V y el diodo zener (ADR5041) [10] que proporciona la tensión de referencia de 2,5 V.

Como se puede comprobar en los datasheet de los amplificadores operacionales, ambos necesitan una tensión de alimentación de tipo diferencial, concretamente de ±12 V. Para ello se ha instalado la fuente de tensión TMA 1205 [7], cuyo voltaje de entrada es de 5 V y de salida es de ±12 V.

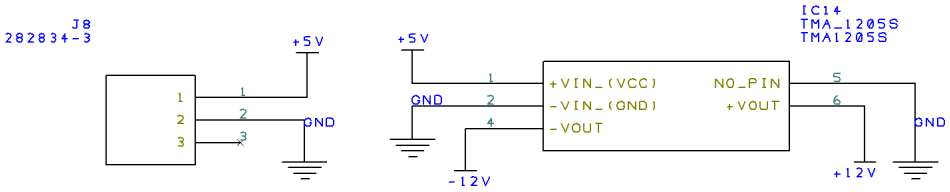


Figura 10: Diagrama del terminal y de la fuente de alimentación

En lo que concierne al MCP23008 (Figura 11), los terminales SCK y SI se conectan a los SCL y SDA del bus a 5 V y los terminales Vdd y Vss a la fuente de 5V. Por otro lado, los terminales que se refieren al direccionamiento del dispositivo (S0, A0 y A1) se han conectado a masa poniendo estos bits a nivel bajo. Finalmente, cada una de las entradas/salidas digitales se ha establecido con sus respectivos terminales (J1, J2, J3 y J4) que se muestran en la Figura 12. En estos terminales también se ha establecido el criterio de una masa común cada dos entradas/salidas digitales.

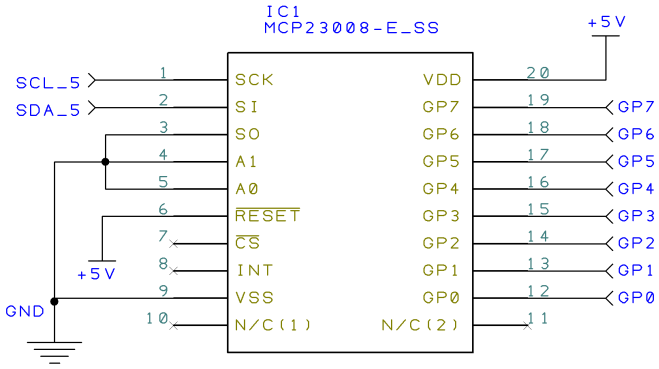


Figura 11: Diagrama del MCP23008

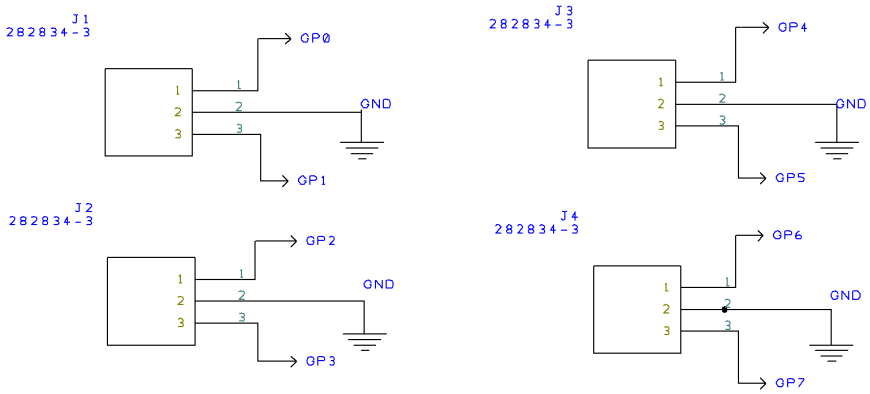


Figura 12: Terminales de las entradas/salidas digitales

Seguidamente, se va a describir el circuito que se encarga de adaptar las tensiones del bus I^2C de 3.3 a 5 V y viceversa. Para ello es necesario un dispositivo dual de comunicación bidireccional, concretamente se ha empleado el PCA9306 [4] de la compañía Texas Instruments.

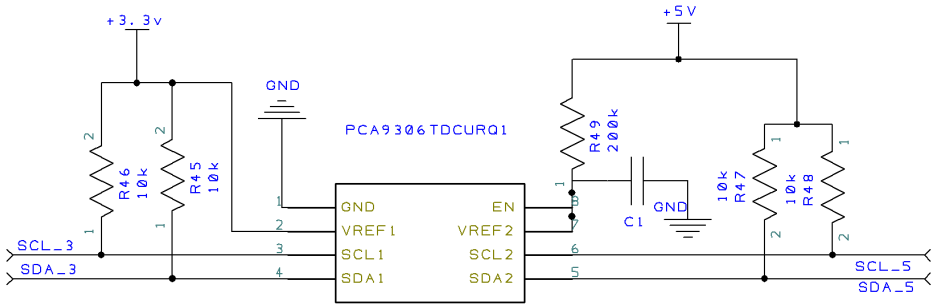


Figura 13: Circuitería del adaptador de tensión del bus I^2C

En la Figura 13, se observa que a ambos lados del adaptador de tensión se dispone de unas resistencias en configuración pull up en ambos canales del bus. El condensador y la resistencia restantes se colocan como se indica en el datasheet del componente.

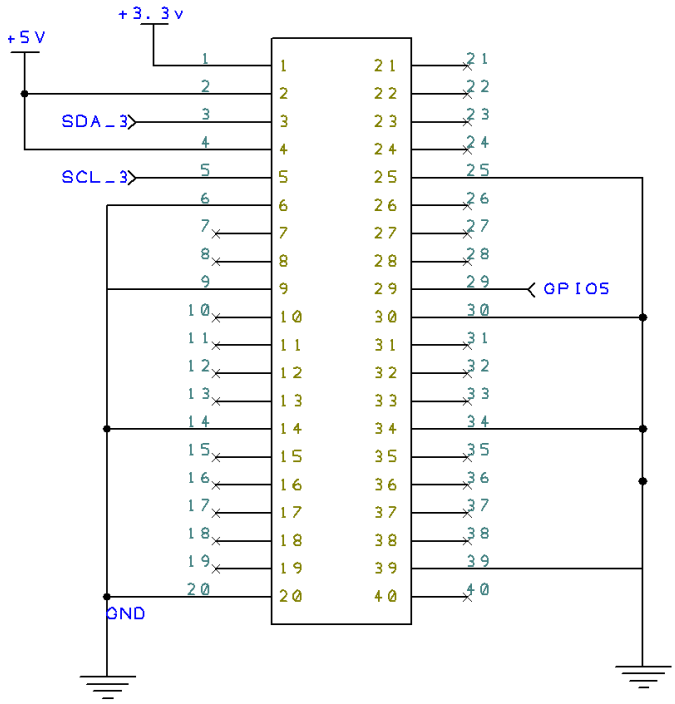


Figura 14: Diagrama del puerto de expansión de la TAD

Por último, el esquema del conector de expansión de la TAD, donde se han empleado los terminales de la alimentación de 5 V y 3.3 V, los terminales del protocolo I^2C (SDA y SCL) pero a una tensión de 3.3 V, la salida digital 5 (pin 29) y los terminales de masa.

6. DISEÑO DE LA PCB

En este capítulo se va a relatar cómo se ha realizado el diseño de la PCB a partir de la circuitería anteriormente explicada. Para este caso se ha empleado una PCB de 2 capas para que el coste de realizarla fuese el menor posible. Una PCB de 2 caras significa que la placa únicamente posee dos láminas de cobre, una en la parte superior y otra en la parte inferior de la misma, las cuales están separadas por un material aislante, que además proporciona solidez a la PCB. En caso de que se quisieran unir dos pistas que se encuentran en caras diferentes, es necesario realizar una vía. Esta consiste en realizar un agujero pasante para conectar ambas pistas. Por otro lado, el ancho de las pistas de cobre se ha establecido en el valor por defecto del programa 0.254mm, ya que es un ancho suficiente para las corrientes bajas que circularán por la TAD.

A la hora de realizar el diseño de cualquier PCB es necesario seguir algunas recomendaciones:

- Agrupar cada una de las partes necesarias que realizan una determinada función.
- Establecer un orden (orientación y alineación) tanto a la hora de establecer los componentes, como a la hora de colocar las pistas. Este hecho se realiza para mejorar la apariencia visual de la TAD.

- Evitar realizar ángulos agudos o rectos a lo largo de la trayectoria de las pistas o en el lugar donde se unen varias de ellas. Esto se realiza para evitar que alguna pista se dañe al circular por ellas una gran cantidad de corriente.
- Respetar en todo momento la distancia mínima entre pistas para evitar posibles fallos durante la fabricación de la PCB.

Para este proyecto en concreto, los conectores de las entradas/salidas analógicas y digitales debían de poseer una posición determinada para poder aprovechar las carcasas de protección que se encuentran disponibles en el mercado actualmente. Otra restricción es la ubicación del conector de expansión de la RPi, ya que debe de encajar perfectamente con esta. Finalmente destacar que ha sido necesario la realización de diversos agujeros, uno en cada una de las esquinas de la TAD para que se pueda sujetar con la RPi mediante la tornillería correspondiente.

Con todas estas recomendaciones y las especificaciones de diseño se ha obtenido el siguiente esquema de la PCB (Figura 15).

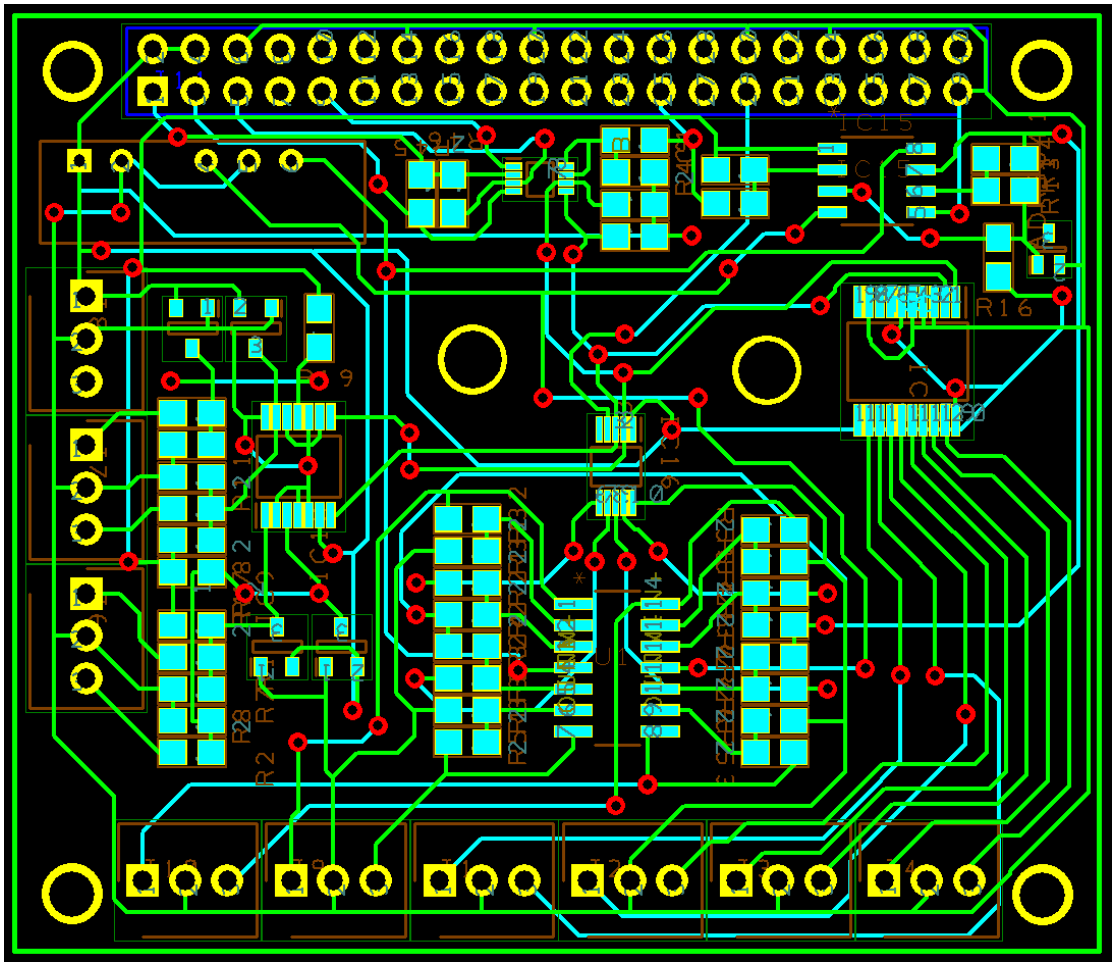


Figura 15: Vista de la PCB desde la ventana de diseño de DesignSpark

En la figura anterior se muestra el diseño final de la TAD, donde se muestran las pistas, vías, agujeros y componentes que la conforman. Las pistas de color verde representan aquellas que se

encuentran en la cara superior de la PCB y las de color azul, representan las que se encuentran por la cara inferior de la misma. Por otro lado, todos aquellos orificios de color amarillo, corresponde con agujeros pasante de sujeción y los de color rojo, con vías que unen pistas de las dos caras de la PCB. En Figura 16, se pueden observar por separado la cara superior e inferior de la TAD respectivamente.

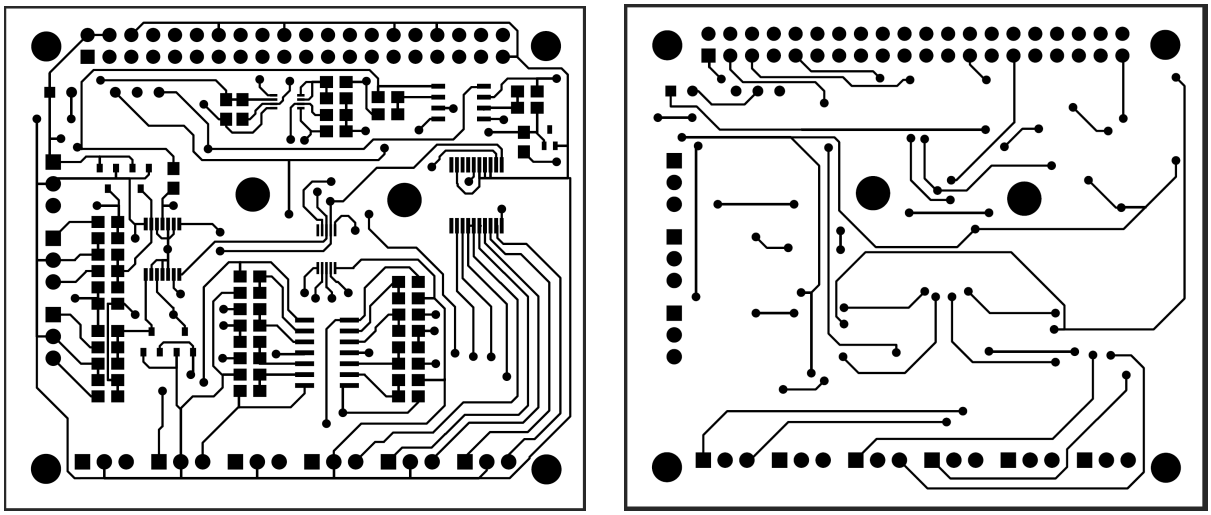


Figura 16: Cara superior (izquierda) e inferior (derecha) de la TAD

En la Figura 17 se puede observar una vista 3D del diseño definitivo de la tarjeta de adquisición de datos con todos sus componentes.

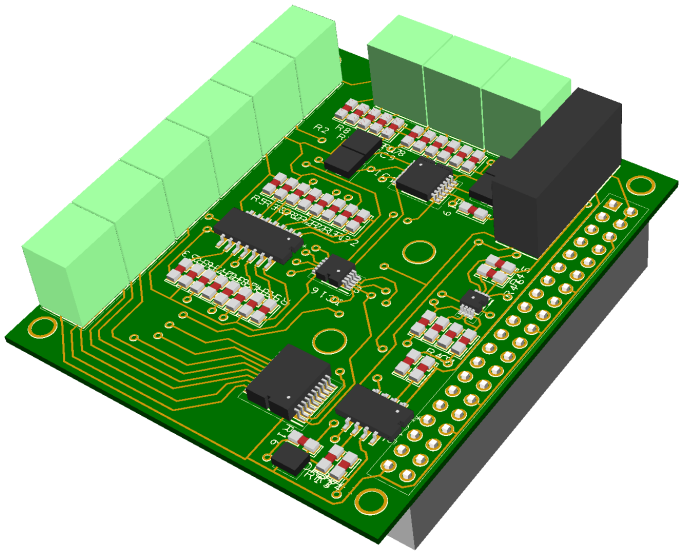


Figura 17: Vista 3D de la TAD

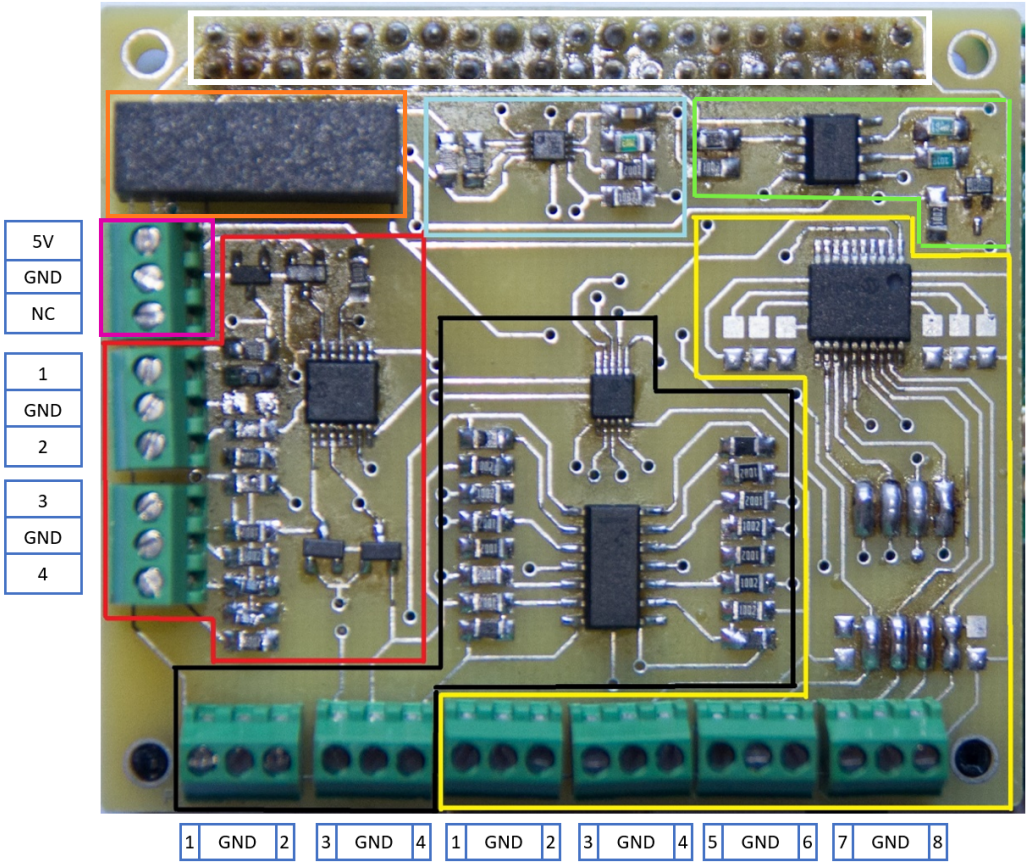


Figura 18: Vista en planta del primer prototipo de la TAD

A continuación, se van a explicar cómo están organizadas cada una de las partes que conforman la TAD y a que corresponde cada uno de los terminales de la misma. En la Figura 18 se puede observar una vista cenital del primer prototipo de la TAD, donde se han resaltado cada una de las partes de la misma, que son:

- Zona naranja: representa la fuente de alimentación TMA 1205.
- Zona rosada: representa el terminal por el que se suministra la tensión de alimentación de 5V a la TAD y la Raspberry Pi.
- Zona roja: representa las entradas de tipo analógico. Está formada por los terminales, las resistencias, el ADC (MCP3424) y los diodos de protección (MMBD1203) descritas anteriormente.
- Zona negra: representa las salidas analógicas. Está formado por el DAC (MCP 4728), el amplificador operacional (LM124), las resistencias y terminales expuestos anteriormente.
- Zona amarilla: representa las entradas/salidas digitales. Está formado por el Digital I/O expander (MCP23008)
- Zona verde: representan los componentes necesarios para obtener las tensiones de referencias tanto para las entradas como para las salidas analógicas. Está formado por el amplificador operacional (LM 158), un diodo zener (ADR5041) y un conjunto de resistencias.

- Zona azul: representa el adaptador de tensión del protocolo I^2C, formado por el chip PCA9306 y un conjunto de resistencias.
- Zona blanca: representa la parte posterior del conector de expansión hembra, encargado de establecer conexión con la RPi.

Finalmente, en la Figura 19 se observa el primer prototipo de la TAD con todos los componentes montados y ensamblada en la RPi.

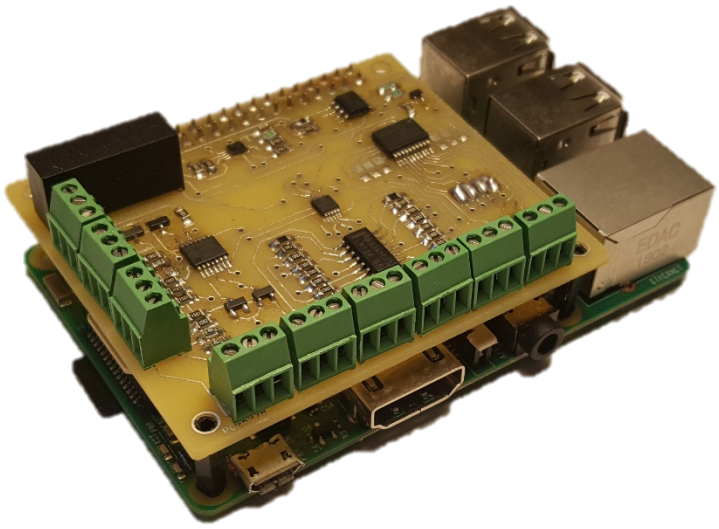


Figura 19: Primer prototipo de la TAD, ensamblada en la RPi

Como se ha podido observar entre el primer prototipo y el diseño definitivo de la TAD, se han producido algunos cambios. El primero de ellos corresponde con las entradas/salidas digitales, que en el primer prototipo se introdujeron una serie de resistencias que finalmente no eran necesarias para su funcionamiento. Por otro lado, al eliminar estas resistencias se ha permitido introducir dos orificios en la parte central de la TAD, para que haya un mayor flujo de aire entre la PCB de la tarjeta de adquisición de datos y la RPi. Este hecho solucionará en parte, los problemas de calentamiento que se han producido durante el ensayo de la misma, con se ha relatado en la memoria principal del documento.

7. COMPONENTES UTILIZADOS

En la siguiente tabla están recogidos todos los componentes electrónicos empleados en la TAD. Estos están ordenados por orden alfabéticos del identificador que aparece en los esquemas eléctricos. Asimismo, también aparece el valor, el encapsulado y una breve descripción de cada uno de ellos.

Tabla 1: Listado de componentes electrónicos de la TAD

Identificador	Valor	Encapsulado	Descripción
ADR5041	-	ADR5041	Diodo zener
C1	-	0805 (SMD)	Condensador
IC1	-	MCP23008	Digital I/O expander
IC2	-	PCA9306	Adaptador de tensión I2C
IC9	-	MMBD1203	Diodo
IC10	-	MMBD1203	Diodo
IC11	-	MMBD1203	Diodo
IC12	-	MMBD1203	Diodo
IC13	-	MCP3424	ADC
IC14	-	TMA_1205	Fuente alimentación
IC15	-	LM158	Amplificador operacional
IC16	-	MCP4728	DAC
J1	-	282834-3	Terminal entradas/salidas digitales
J2	-	282834-3	Terminal entradas/salidas digitales
J3	-	282834-3	Terminal entradas/salidas digitales
J4	-	282834-3	Terminal entradas/salidas digitales
J6	-	282834-3	Terminal entradas analógicas
J7	-	282834-3	Terminal entradas analógicas
J8	-	282834-3	Terminal alimentación
J9	-	282834-3	Terminal salidas analógicas
J10	-	282834-3	Terminal salidas analógicas
J11	-	GPIO	Conector RPi
R1	10k 1%	0805 (SMD)	Resistencia
R2	10k 1%	806 (SMD)	Resistencia
R3	10k 1%	807 (SMD)	Resistencia
R6	10k 1%	808 (SMD)	Resistencia
R7	1.24k 1%	809 (SMD)	Resistencia
R8	1.24k 1%	810 (SMD)	Resistencia
R9	1.24k 1%	811 (SMD)	Resistencia
R10	30K 1%	812 (SMD)	Resistencia
R11	1.24k 1%	813 (SMD)	Resistencia
R12	10k 1%	814 (SMD)	Resistencia
R13	10.1k 0.1%	815 (SMD)	Resistencia
R14	8.06k 0.1%	816 (SMD)	Resistencia
R16	10k 1%	817 (SMD)	Resistencia

R17	10k 1%	818 (SMD)	Resistencia
R18	10k 1%	819 (SMD)	Resistencia
R19	10k 1%	820 (SMD)	Resistencia
R20	10k 1%	821 (SMD)	Resistencia
R21	10k 1%	822 (SMD)	Resistencia
R22	10k 1%	823 (SMD)	Resistencia
R23	10k 1%	824 (SMD)	Resistencia
R24	10k 1%	825 (SMD)	Resistencia
R26	10k 1%	826 (SMD)	Resistencia
R27	10k 1%	827 (SMD)	Resistencia
R28	10k 1%	828 (SMD)	Resistencia
R30	10k 1%	829 (SMD)	Resistencia
R31	10k 1%	830 (SMD)	Resistencia
R32	10k 1%	831 (SMD)	Resistencia
R33	10k 1%	832 (SMD)	Resistencia
R45	10k 1%	833 (SMD)	Resistencia
R46	10k 1%	834 (SMD)	Resistencia
R47	10k 1%	835 (SMD)	Resistencia
R48	10k 1%	836 (SMD)	Resistencia
R49	10k 1%	837 (SMD)	Resistencia
R51	10k 1%	838 (SMD)	Resistencia
R52	10k 1%	839 (SMD)	Resistencia
R53	10k 1%	840 (SMD)	Resistencia
R54	10k 1%	841 (SMD)	Resistencia
R55	10k 1%	842 (SMD)	Resistencia
U1	-	LM124AD	Amplificador operacional

8. BIBLIOGRAFÍA

[1] Microchip (2008) *Datasheet MCP3424.* Obtenido el 1 de Mayo de 2018 de http://ww1.microchip.com/downloads/en/DeviceDoc/22088b.pdf

[2] Microchip (2010) *Datasheet MCP4728.* Obtenido el 1 de Mayo de 2018 de http://ww1.microchip.com/downloads/en/DeviceDoc/22187E.pdf

[3] Microchip (2007) *Datasheet MCP23008.* Obtenido el 5 de Mayo de 2018 de http://ww1.microchip.com/downloads/en/DeviceDoc/21919e.pdf

[4] Texas Instruments (2016) *Datasheet PCA9306.* Obtenido el 1 de Junio de 2018 de http://www.ti.com/lit/ds/symlink/pca9306.pdf

[5] Texas Instruments (2015) *Datasheet LM124.* Obtenido el 15 de Mayo de 2018 de http://www.ti.com/lit/ds/snosc16d/snosc16d.pdf

[6] Texas Instruments (2014) *Datasheet LM158.* Obtenido el 15 de Mayo de 2018 de http://www.ti.com/lit/ds/symlink/lm158-n.pdf

[7] Traco Power (2017). *Datasheet TMA1205S.* Obtenido el 2 de Mayo de 2018 de https://www.tracopower.com/products/tma.pdf

[8] RS-online (s.f.) *DesignSpark.* Obtenido el 1 de Junio de 2018 de https://www.rs-online.com/designspark/home

[9] National Instruments (s.f.) *¿Qué es Multisim?* Obtenido el 29 de abril de 2018 de http://www.ni.com/es-es/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-multisim.html

[10] Analog Devices (2018) *Datasheet ADR5041.* Obtenido el 18 de Mayo de http://www.analog.com/media/en/technical-documentation/data-sheets/ADR5040_5041_5043_5044_5045.pdf

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC:

ANEXO 2: MANUALES DE USUARIO DE LAS APLICACIONES DISEÑADAS

ÍNDICE:

ÍNDICE DE FIGURAS:

1. INTRODUCCIÓN

A lo largo de este documento se van a recoger los requisitos necesarios y las instrucciones a seguir para el correcto funcionamiento de la aplicación de control y de la interfaz gráfica remota.

2. REQUISITOS DE LAS APLICACIONES

Para que toda la aplicación de control desarrollada funcione de forma correcta es necesario que tanto la RPi como el ordenador donde se vaya ejecutar la interfaz gráfica, deben cumplir ciertos requisitos.

En lo que se refiere al controlador de la RPi, esta debe tener instaladas las correspondientes librerías empleadas a lo largo del programa y además, debe tener instalado el programa que se ha empleado para programar dicha aplicación, Python.

Por otro lado, el ordenador donde se ejecute la interfaz gráfica debe tener instalado el programa LabVIEW para poder ejecutarla. Si no se puede obtener una licencia LabVIEW, es necesario disponer de la aplicación en formato .EXE donde no es necesario poseer dicha licencia.

Asimismo, para establecer la comunicación entre ambas aplicaciones, es necesario disponer de una conexión tipo Ethernet, donde se debe conocer la dirección y el puerto de acceso del servidor, en este caso de la RPi.

Finalmente, destacar que para poder usar toda la aplicación es necesario el proceso a controlar, que si no se dispone de ninguno real, como es este caso, es necesario un HIL para poder simular el comportamiento del proceso y para ello es necesario sus correspondientes TADs para medir y mandar la información de las variables pertinentes. En el caso de que se tenga un proceso real, los sensores y actuadores del mismo se deberían conectar con las entradas y salidas de la TAD diseñada respectivamente. En el caso de que se posea un HIL, es necesario conectar las entradas y salidas de las TADs del HIL a las salidas y entradas de la TAD que posee la RPi respectivamente, como se ha mencionado en la memoria principal (ver Figura 1). Además tanto el proceso real como el HIL deben estar en funcionamiento antes de ejecutar la aplicación de control de la RPi.

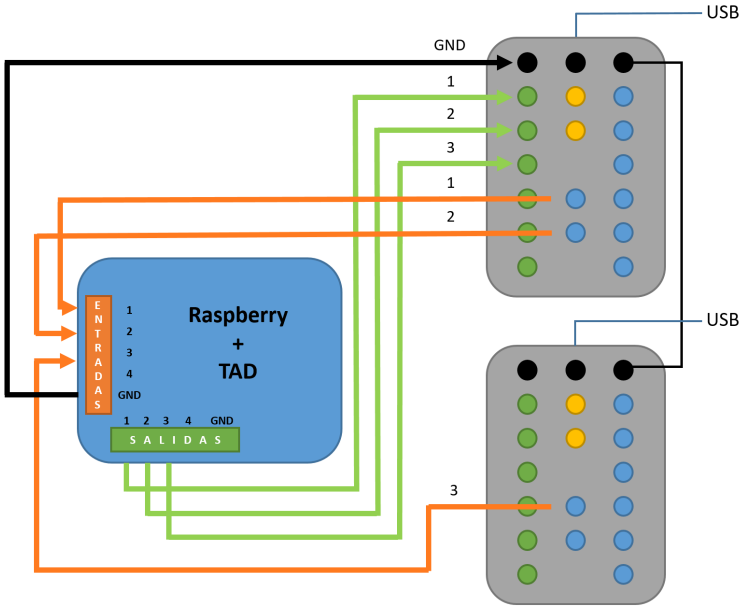


Figura 1: Diagrama de conexión de la RPi con el HIL a través de sus correspondientes TADs de un proceso de 3 entradas y 3 salidas

3. MANUAL DE USUARIO DE LA APLICACIÓN DE CONTROL EN LA RPI

3.1. Aplicación servidor Modbus TCP

La primera aplicación que es necesario ejecutar para que el controlador funcione de forma correcta es la aplicación *Servidor_Modbus.py*, que configura el servidor Modbus TCP. Esta aplicación siempre se encuentra en ejecución de forma paralela con la aplicación de control y se debe ejecutar en la consola de comandos de la RPi.

Los únicos parámetros configurables de esta aplicación son la dirección IP del servidor, que coincide con la dirección IP que posea la RPi, y el puerto de conexión que debe ser superior al 1024, ya que la RPi no permite emplear puertos de un valor inferior.

3.2. Aplicación controlador DMC

Una vez ejecutado el servidor, se puede ejecutar la aplicación que actúa como controlador en la consola de comandos de la RPi. No obstante, para su funcionamiento son necesarios los archivos de texto que describen el comportamiento ante una entrada de tipo escalón en un determinado proceso y el archivo de configuración pertinente, que deben de estar ubicados en la carpeta raíz del programa. Los archivos con la respuesta del modelo se deben nombrar de una forma específica, indicando el proceso a controlar y si se trata de la respuesta del modelo ante una variable de control o una perturbación medible. Para ello, se ha establecido el siguiente formato, *step_pX_yX_uX.txt* (acciones de control) o *step_pX_yX_dX.txt* (perturbaciones mediables), donde se establece el identificador del proceso, el número de la salida y el número de la acción de control o perturbación medible respectivamente. Cabe destacar que los procesos que el usuario desee implementar deben de poseer un número del identificador superior al uno, ya que este se reserva para el proceso de prueba que

incluye la aplicación. Asimismo, los valores de la respuesta del modelo se deben encontrar en formato columna.

Cuando se ejecuta, el controlador comienza a operar en función de los archivos de configuración que se disponga en la carpeta raíz del programa. En esta carpeta se puede disponer de hasta dos archivos de configuración, el primero de ellos consiste en una configuración predeterminada por el programador (*Configuracion_inicial.txt)* donde la aplicación carga un determinado proceso, cuya finalidad es que el usuario puede probar que la aplicación funciona de forma correcta. No obstante, se puede disponer de otro archivo que contenga la configuración del proceso que desea controlar el usuario (*Configuración_actual.txt*) que previamente ha sido realizado por el usuario. Si al ejecutarlo existe el fichero de *Configuración_actual.txt*, la aplicación empleará esta configuración y si no existe, se cargará la información del fichero *Configuracion_inicial.txt*. El archivo de configuración contiene la siguiente información:

- Identificador del proceso
- Modo de funcionamiento (Modo desactivado = 0; modo manual = 1 y modo automático = 2)
- Número de entradas
- Número de salidas
- Número de perturbaciones
- Acciones de control manuales (%)
- Horizonte de control (c)
- Periodo de muestreo (Ts)
- Valor de las referencias
- Los parámetros de las rectas de conversión de las salidas
- Alfas
- Lambdas
- Restricciones del QDMC (Δu, U_max, U_min, Y_max, Y_min)
- Los parámetros de las rectas de conversión de las entradas

Cabe destacar, que para realizar el archivo *Configuracion_actual.txt* de forma manual, se parte de una copia del archivo *Configuración_inicial.txt*.

Una vez ya establecidos todos los archivos de texto necesarios, la aplicación funciona de forma totalmente autónoma con la configuración establecida, a la espera de las órdenes de la aplicación que actúa como interfaz gráfica a través del protocolo Modbus. Asimismo, cuando esta se encuentra en funcionamiento, muestra por pantalla (ver Figura 2) el valor de las referencias, de las salidas y las entradas del proceso. Finalmente, muestra el tiempo sobrante en cada una de las iteraciones, una vez ya realizadas las operaciones pertinentes.

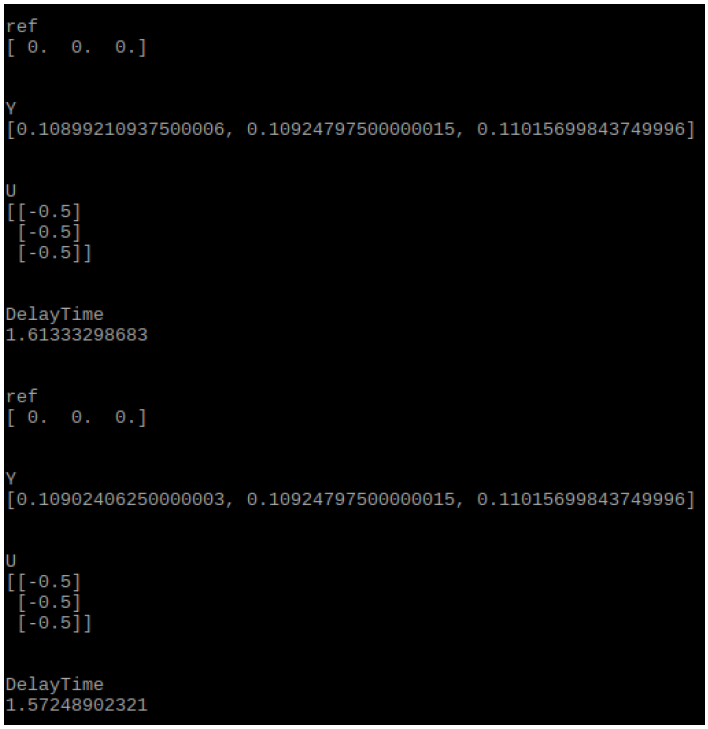


Figura 2: Consola de comandos de la RPi

4. MANUAL DE USUARIO DE LA INTERFAZ GRÁFICA REMOTA

Una vez ejecutada las aplicaciones de la RPi, se procede con la ejecución de la interfaz gráfica. Como requisito para su funcionamiento, las aplicaciones de la RPi deben estar en funcionamiento. En la Figura 3, se muestra el panel de configuración del QDMC con todos los elementos que lo componen enumerados. Una vez ejecutada se configura la dirección IP (1) y el puerto de conexión (2) del servidor, y seguidamente se activa el botón *Conectar* (3). Si durante la conexión se produce algún tipo de error, este se muestra por pantalla, especificando el tipo de error producido. Una vez establecida la conexión, ya se pueden configurar cada uno de los elementos del panel, que son los siguientes:

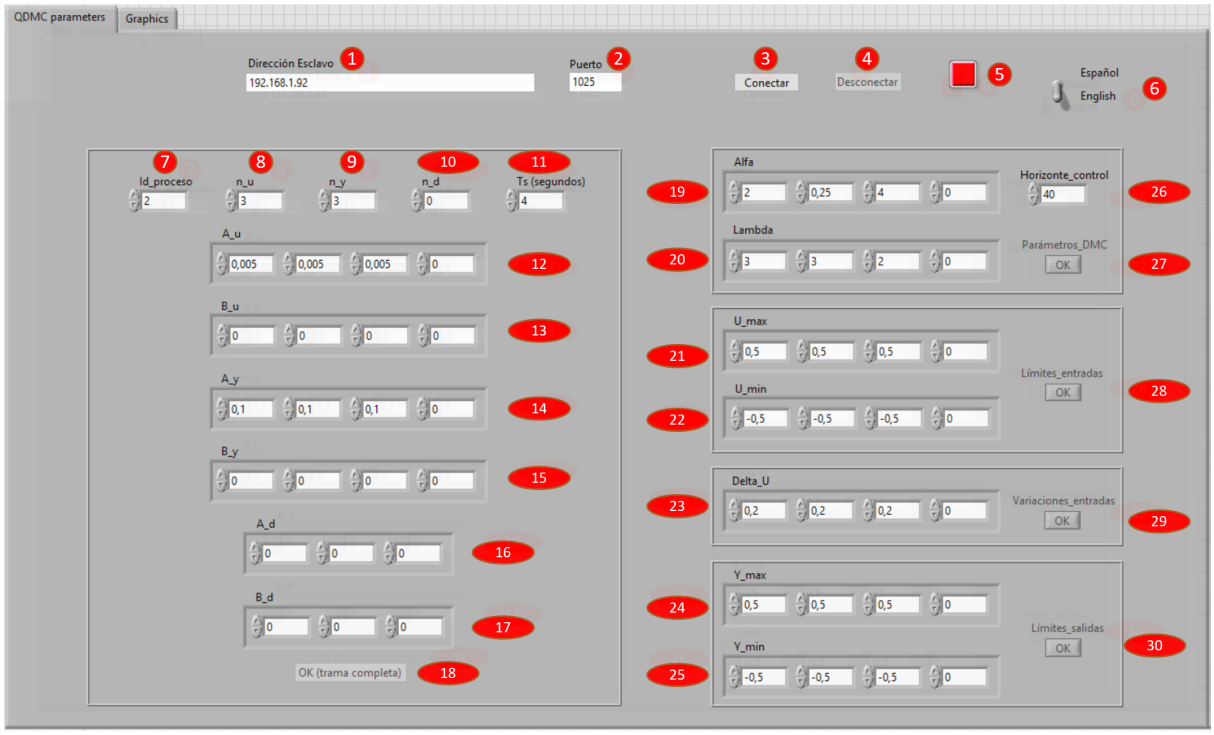


Figura 3: Panel de configuración del QDMC

- **Id_process (7):** Identificador del proceso.
- **N_u (8):** Número de entradas del proceso.
- **N_y (9):** Número de salidas del proceso.
- **N_d (10):** Número de perturbaciones medibles.
- **Ts (11):** Periodo de muestreo, en segundos.
- **A_u (12) y B_u (13):** Parámetros de las rectas de conversión de unidades a voltios de las entradas.
- **A_y (14) y B_y (15):** Parámetros de las rectas de conversión de voltios a unidades de las salidas.
- **A_d (16) y B_d (17):** Parámetros de las rectas de conversión de unidades a voltios de las perturbaciones.
- **OK (trama completa) (18):** Pulsador para actualizar todas las variables en la trama del Modbus. Se activa cuando se produce algún cambio en estas variables.
- **Alfa (19):** Parámetros alfa de cada una de las variables de salida.
- **Lambda (20):** Parámetros lambda de cada una de las variables de entrada.
- **Horizonte_control (26):** Horizonte de control.
- **Parámetros_DMC (27):** Pulsador para actualizar las variables alfa, lambda y horizonte_control en los bloques de memoria de Modbus.
- **U_max (21):** Parámetros de los valores máximos de las acciones de control.
- **U_min (22):** Parámetros de los valores mínimos de las acciones de control.

- **Delta_U (23):** Parámetros de los valores máximos en la variación por iteración de las acciones de control.
- **Y_max (24):** Parámetros de los valores máximos de las variables a controlar.
- **Y_min (25):** Parámetros de los valores máximos de las variables a controlar.
- **Límites_entradas (28):** Pulsador para actualizar las variables U_max y U_min en los bloques de memoria de Modbus.
- **Variaciones_entradas (29):** Pulsador para actualizar las variables Delta_U en los bloques de memoria de Modbus.
- **Limites_salidas (30):** Pulsador para actualizar las variables Y_max y Y_min en los bloques de memoria de Modbus**.**

Por otro lado, en esta misma pestaña se encuentran los pulsadores *Desconectar* (4), para realizar la desconexión de la red Modbus, *Pulsador rojo* (5), que se encarga de detener la aplicación y el interruptor (6), que se encarga de cambiar el idioma de la aplicación de inglés a español y viceversa.

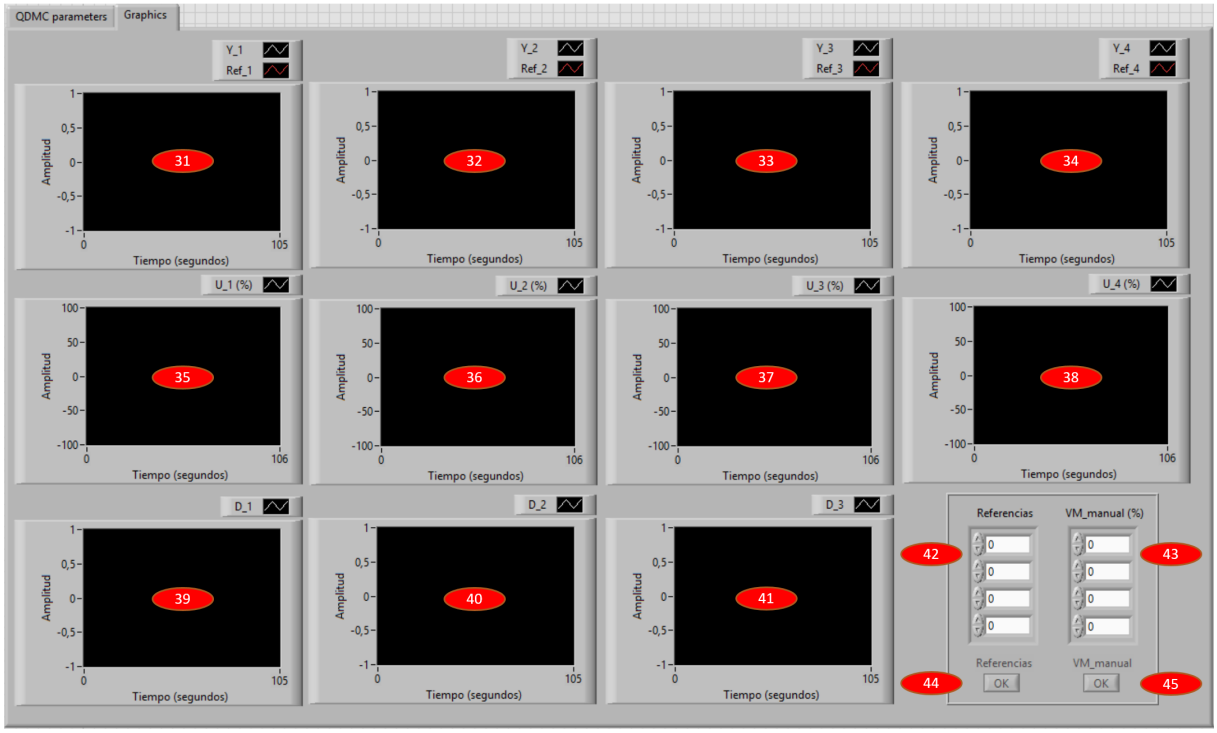


Figura 4: Panel de monitorización de las variables deseadas

En la Figura 4, se muestra el panel de monitorización que está compuesto por los siguientes elementos (de izquierda a derecha):

- **Gráficas 1 (31), 2 (32), 3 (33) y 4 (34):** Muestran el comportamiento de las salidas y sus correspondientes referencias.
- **Gráficas 5 (35), 6 (36), 7 (37) y 8 (38):** Muestran el comportamiento de las acciones de control.

- **Gráficas 9 (39), 10 (40) y 11 (41):** Muestran el comportamiento de las perturbaciones medibles.
- **Referencias (42):** Valores de cada una de las referencias del proceso a seguir.
- **Pulsador Referencias (44):** Pulsador para actualizar las referencias en los bloques de memoria de Modbus.
- **VM_manual (%) (43):** Valores de cada una de las acciones de control cuando se desea realizar el control de forma manual. Se activan cuando se produce algún cambio en estas variables.
- **Pulsador VM_manual (45):** Pulsador para actualizar las variables VM_manual en los bloques de memoria de Modbus. Se activan cuando se produce algún cambio en estas variables.

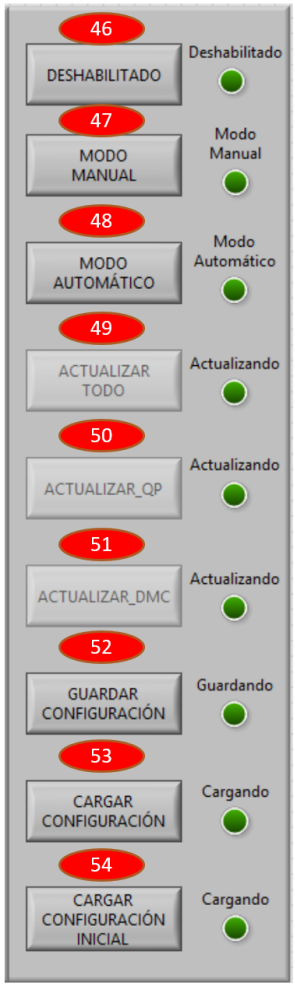


Figura 5: Panel de control de los estados del controlador

Finalmente en la Figura 5, se muestra el panel de control de estados del controlador, el cual siempre se encuentra a la vista del usuario. Este panel está formado por una serie de pulsadores con sus correspondientes indicadores que se describen a continuación:

- **Deshabilitado (46):** Desconecta el controlador y pone las acciones de control a cero.
- **Modo manual (47):** Habilita el modo manual de controlador donde se introducen las acciones de control de forma manual a través de los controles *VM_manual*.
- **Modo automático (48):** Activa el modo automático del controlador, es decir activa el algoritmo de control QDMC.
- **Actualizar todo (49):** Con este pulsador se da la orden a la aplicación de control de la RPi para actualizar todas las variables locales de la misma, a partir de la información que se encuentra en los bloques de memoria de Modbus.
- **Actualizar_QP (50):** Con este pulsador se da la orden a la aplicación de control de la RPi para actualizar las variables locales de la misma referentes al algoritmo optimizador QP, a partir de la información que se encuentran en los bloques de memoria de Modbus.
- **Actualizar_DMC (51):** Con este pulsador se da la orden a la aplicación de control de la RPi para actualizar las variables locales de la misma referentes al algoritmo DMC, a partir de la información que se encuentran en los bloques de memoria de Modbus.
- **Guardar configuración (52):** Con este pulsador se da la orden a la aplicación de la RPi para almacenar la última configuración de la aplicación en el fichero de texto *Configuración_actual.txt.*
- **Cargar configuración (53):** Con este pulsador se carga la configuración que se encuentra en el fichero *Configuración_actual.txt* en la aplicación de control de la RPi.
- **Cargar configuración inicial (54):** Con este pulsador se carga la configuración que se encuentra en el fichero *Configuración_inicial.txt* en la aplicación de control de la RPi.

Cada uno de estos interruptores posee un indicador, que en el caso de los modos de funcionamiento, indica en qué modo se encuentra la aplicación de la RPi. Por otro lado, en los indicadores que corresponde con los interruptores de actualizar variables, estos permanecen activos hasta que se efectúen los cambios pertinentes, debido a que el controlador se puede encontrar en un modo de funcionamiento donde no se pueden actualizar ciertas variables y para realizar la actualización, es necesario cambiar el modo de funcionamiento. El indicador de guardar la configuración actual, se activa en el momento que se está guardando la información en el fichero y se desactiva una vez ya guardada. Finalmente, los indicadores de cargar las configuraciones, poseen un funcionamiento similar a los indicadores de actualizar variables.

Esta aplicación puede conectarse o desconectarse en cualquier momento de la red Modbus, sin afectar en ningún momento a la aplicación de control que se está ejecutando continuamente en la RPi, con la última configuración establecida por el usuario.

5. MANUAL DE USUARIO DE LA APLICACIÓN HIL

La aplicación HIL se ha programado en LabVIEW al igual que la interfaz gráfica mencionada en el apartado anterior. Se trata de una aplicación que posee un sencillo panel frontal (ver Figura 6) donde se encuentran los siguientes elementos:

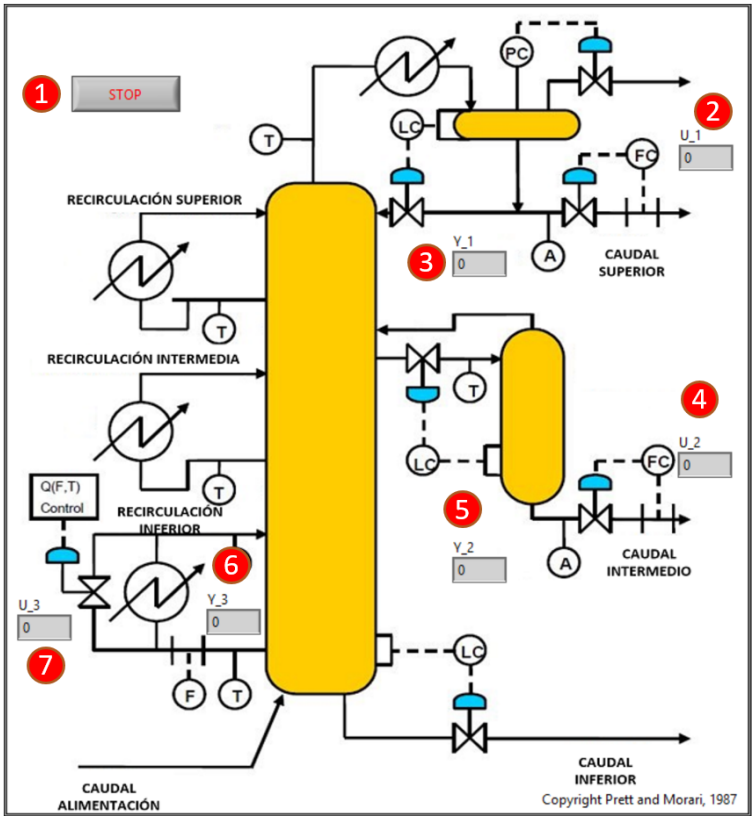


Figura 6: Panel frontal de la aplicación HIL (Hardware-in-the-loop)

- **Stop (1):** Pulsador para detener la aplicación.
- **U_1 (2), U_2 (4) y U_3 (7):** Indicadores que muestran los valores de las acciones de control.
- **Y_1 (3), Y_2 (5) y Y_3 (6):** Indicadores que muestran los valores de las salidas del proceso.

Esta aplicación una vez iniciada, se ejecuta de forma cíclica hasta que el usuario acciona el pulsador *Stop*. El proceso que simula este HIL, se describe con un mayor nivel de detalle en la memoria principal del documento.

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC:

ANEXO 3: MANUALES DE PROGRAMACIÓN DE LA INTERFAZ GRÁFICA Y DEL HARDWARE-IN-THE-LOOP

ÍNDICE

ÍNDICE DE FIGURAS:

1. INTRODUCCIÓN

A lo largo de este documento se va explicar la programación empleada para realizar la aplicación que actúa como interfaz gráfica tanto para la configuración del controlador DMC como para la monitorización básica de las variables de interés. Asimismo, también se recoge la programación de la aplicación que actúa como Hardware-in-the-Loop.

2. ESPECIFICACIONES DE DISEÑO DE LA INTERFAZ GRÁFICA

Las especificaciones de diseño de la aplicación son las siguientes:

- Establecer comunicación con la aplicación que se ejecuta en la RPi mediante el protocolo de comunicación Modbus TCP.
- Poder establecer conexión con la RPi en cualquier momento mientras el controlador se encuentra en funcionamiento.
- Manipular todos los parámetros del controlador DMC desde la interfaz gráfica.
- Modificar el estado en el que se encuentra el controlador.
- Gestionar cada uno de los modos de funcionamiento del controlador.
- Monitorizar todas las variables de interés del proceso a controlar.
- Emplear el lenguaje de programación de LabVIEW.

Figura 1: Vista general de la interfaz gráfica

3. PROGRAMACIÓN DE LA INTERFAZ GRÁFICA

Esta aplicación está compuesta por una serie de bucles y elementos que han sido cruciales para el desarrollo de la interfaz gráfica. En primer lugar, se va a destacar la programación referente a la comunicación Modbus TCP, que está compuesta por diferentes partes. Para la programación de todo lo referente al protocolo Modbus, se ha partido de diversos ejemplos que National Instruments posee en web [2]. La primera de estas partes, hace referencia al objeto (API) [2] necesario para generar una conexión de tipo Modbus (ver Figura 2).

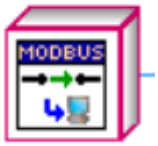


Figura 2: API Modbus perteneciente a la librería de LabVIEW

Seguidamente, para escribir y leer en cada uno de los bloques de memoria de Modbus, se ha creado una estructura *tipo event case* [1], que consiste en un tipo de estructura que únicamente se ejecuta cuando ocurre una determinada condición o evento, la cual se encuentra dentro de un bucle *while* [1]. En este caso, cada evento va asociado a un determinado booleano (pulsador) que en el momento que se activa, determina un estado u otro de la estructura *case*. Para cambiar el estado de la estructura a partir de una serie de booleanos, es necesario otro bucle, en este caso de tipo *while*, que a su vez alberga una estructura *event*, que se encarga de estar pendiente de si se produce el accionamiento de un booleano. Una vez accionado uno de estos booleanos, la estructura *event* envía un dato de tipo *string* que contiene la información necesaria para activar alguno de los eventos del bucle que gestiona la comunicación Modbus. En la Figura 3, se puede observar que en el momento que cambia de valor el booleano *Connect*, la estructura enviará el dato de tipo cadena de caracteres, en este caso la palabra *Connect.*

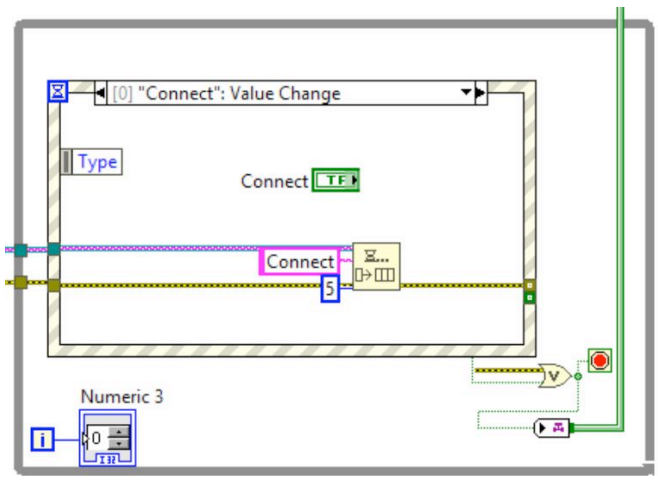


Figura 3: Código encargado de gestionar los eventos del bucle principal del programa

Cuando la estructura *case* detecta esta palabra, activa el estado *Connect* y ejecuta el código de programación que alberga, en este caso (ver Figura 4) la conexión a la red Modbus.

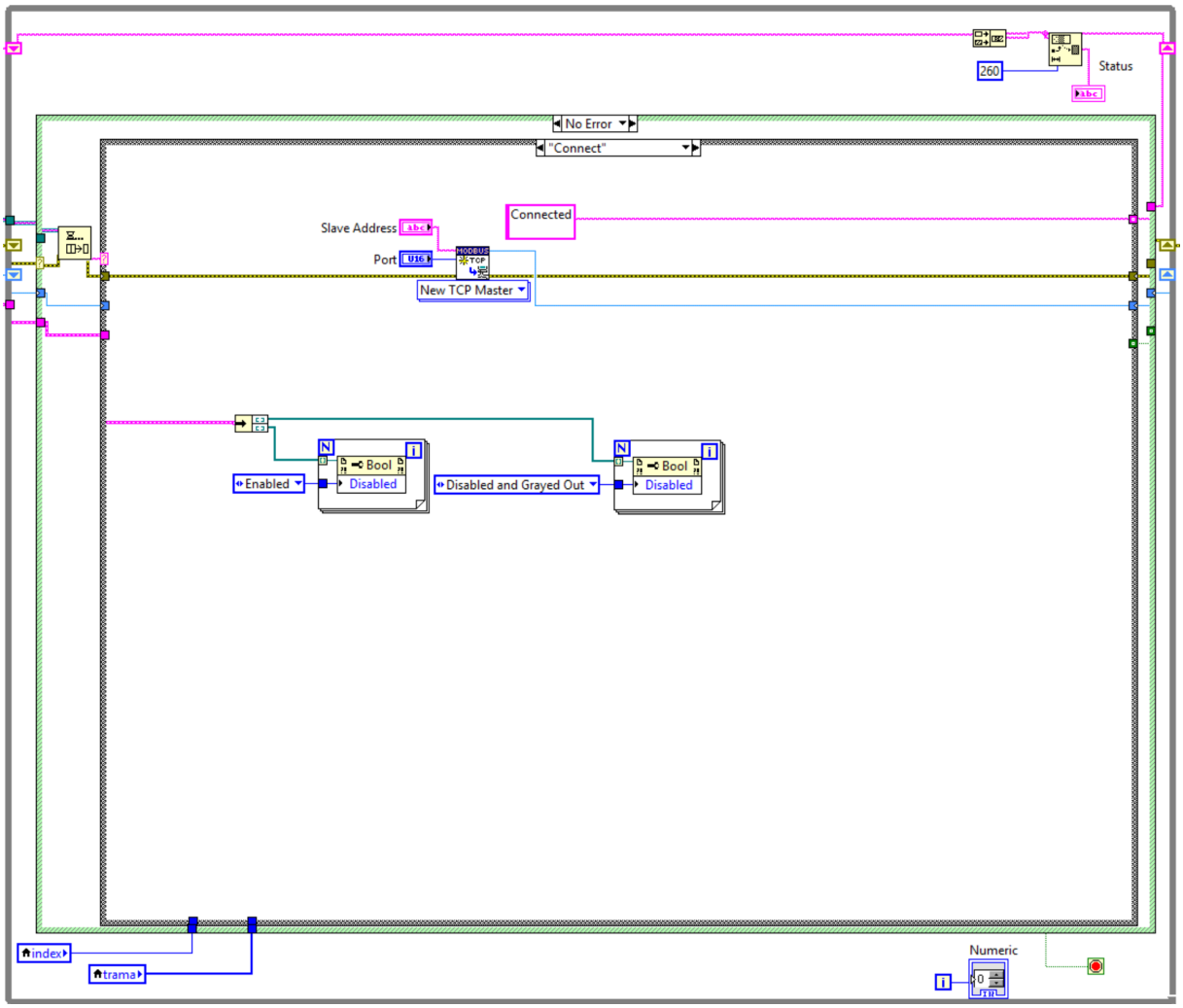


Figura 4: Bucle que gestiona la comunicación Modbus

Una vez explicado cómo funcionan ambos bucles, se va a proceder a explicar cada uno de los casos de mayor importancia que contiene la estructura *case.* El siguiente caso posee la funcionalidad de escribir los valores de las variables deseadas en sus correspondientes bloques de memoria. En el caso de los registros de retención (U16), se emplea la función que se observa en la Figura 5 de la librería Modbus, a la cual únicamente es necesario pasar la dirección del primer bloque de memoria (*index)* desde donde se desea comenzar a realizar la escritura, y seguidamente la información (*trama)* a almacenar.

Figura 5: Función de la librería Modbus (*Set Holding Registers*)

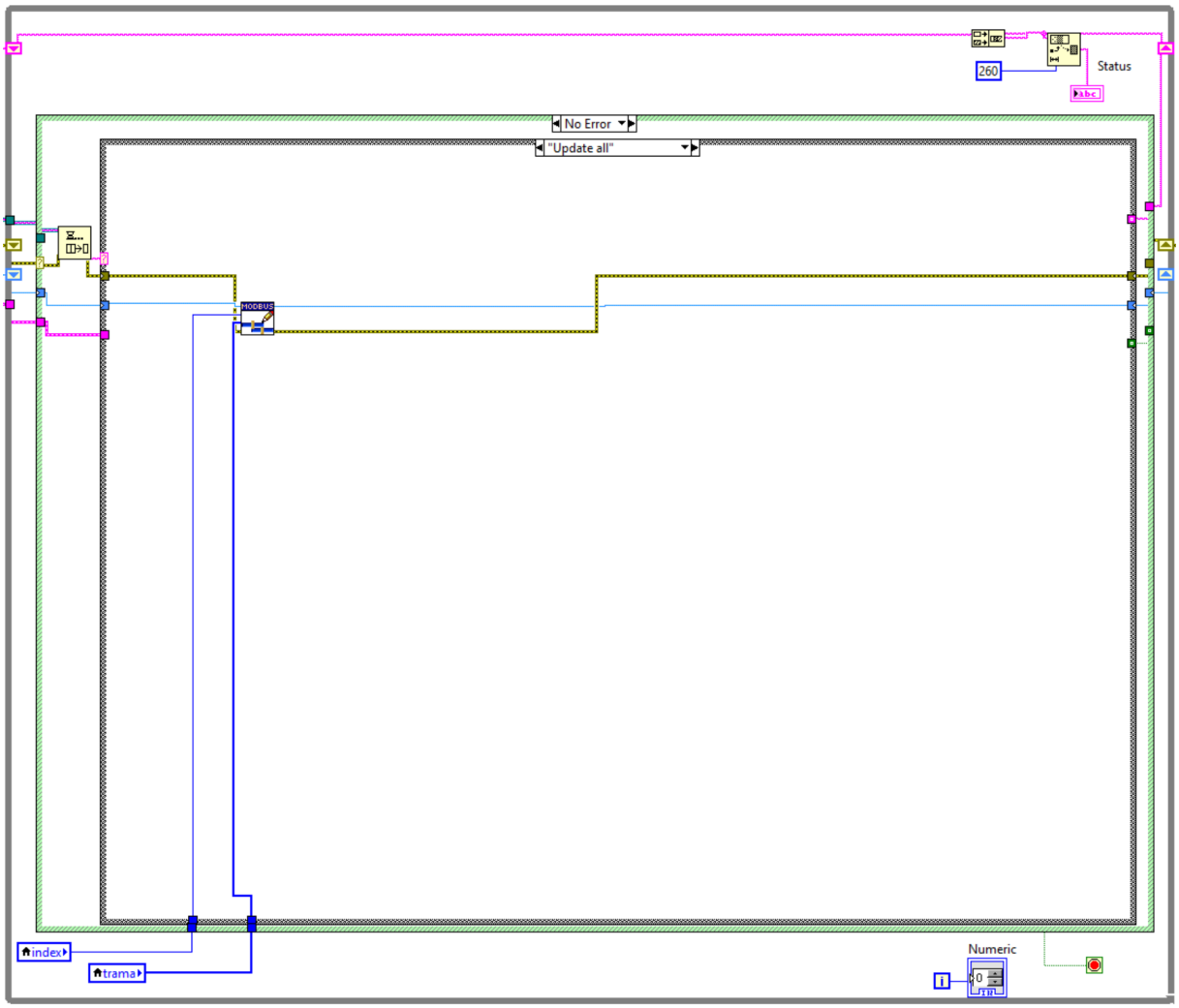


Figura 6: Programación de la escritura en los registros de retención

No obstante, en este caso la información no se puede almacenar directamente en los bloques de memoria de Modbus, ya que estos únicamente admiten datos de tipo U16, y en esta aplicación es necesario enviar datos de tipo flotante (U32). Por ello, es necesario un pre procesamiento de los datos a enviar, el cual debe convertir un número de tipo flotante, en dos de tipo entero. Para ello se ha realizado la siguiente función (ver Figura 7) a la que se le introducen el nombre y el valor de la variable y devuelve las dos posiciones de los bloques de memoria y el valor de cada uno de ellos. El código de la programación de esta función se observa en la Figura 8.

Figura 7: Función *FLOAT*

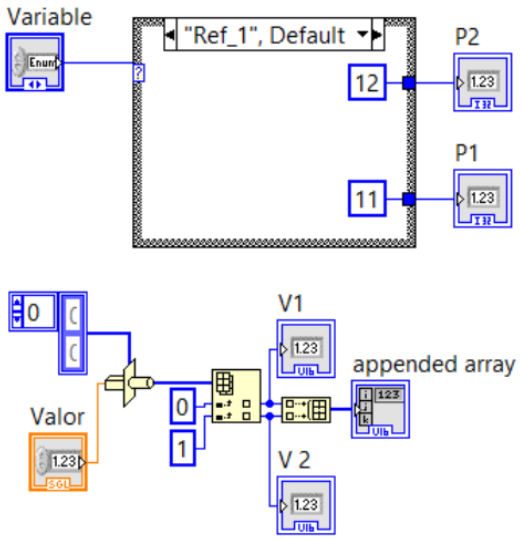


Figura 8: Código de la función *FLOAT*

Para las variables de tipo entero se ha realizado otra función muy similar a la anterior, pero sin la parte que se encarga de transforma un flotante en dos enteros. Con estas dos funciones ya se puede generar la trama (vector) que contiene toda la información que se debe almacenar en los bloques de memoria. En la Figura 9, se puede observar el código que realiza este vector y lo almacena en la variable *trama.*

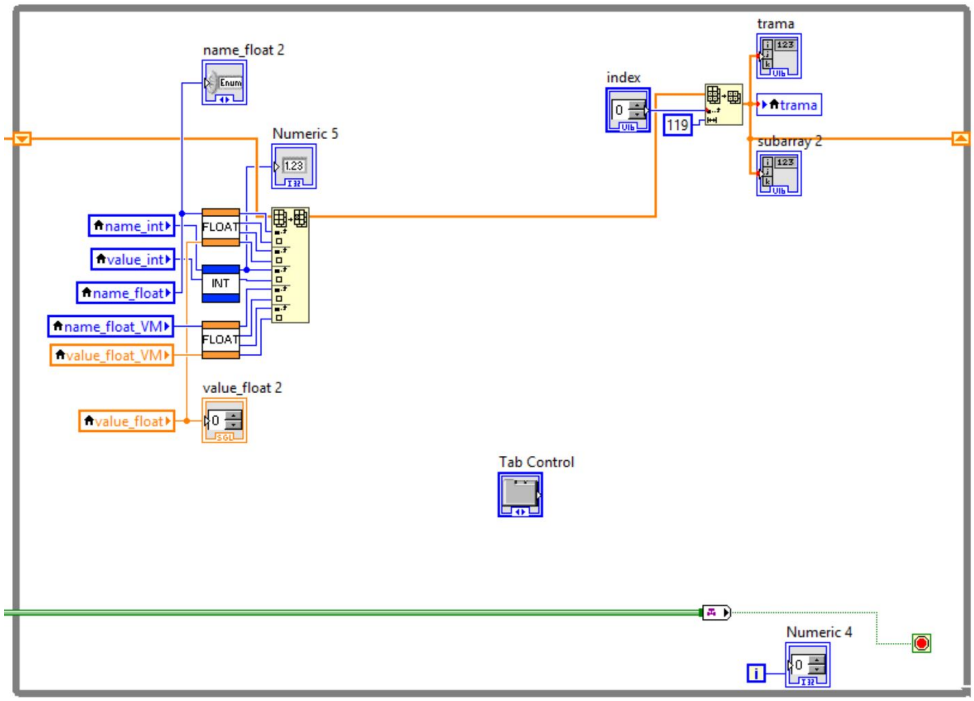


Figura 9: Bucle que gestiona la trama (vector) de los bloques de registros de retención

Por otro lado, el vector *trama* únicamente actualiza sus valores en el momento que se ha producido algún cambio en alguna de las variables de la interfaz gráfica. Para ello, se ha realizado una estructura de tipo *event* (ver Figura 10) que proporciona el nombre y el valor de la variable que ha modificado su valor. Cabe destacar que se ha realizado este tipo de estructura para cada tipo de variable, una para las variables de tipo entero y otra, para las de tipo flotante.

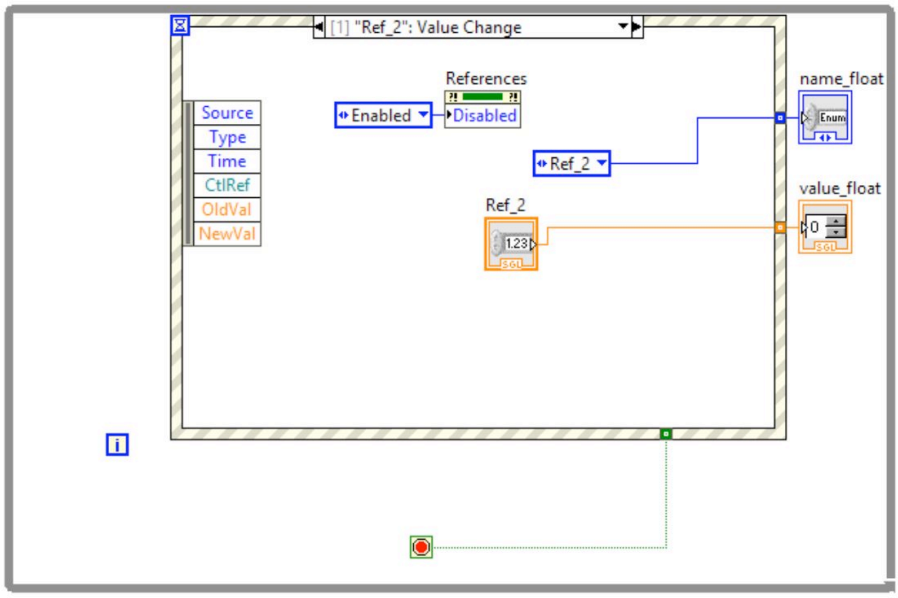


Figura 10: Bucle que gestiona los cambios de los valores de las variables

Finalmente en lo que concierne a esta parte de la programación, se va a mencionar que posición posee cada una de las variables dentro de los registros de retención. En la Tabla 1, se observan las posiciones de las variables de tipo entero y en la Tabla 2, las de tipo flotante.

Tabla 1: Bloques de memoria de los registros de retención de las variables de tipo entero

Variable	Bloque de memoria
Id_process	0
Modo_funcionamiento	1
N_u	2
N_y	3
N_d	4
U_1_manual	5
U_2_manual	6
U_3_manual	7
U_4_manual	8
Horizonte control	9
Ts	10

Tabla 2: Bloques de memoria de los registros de retención de las variables de tipo flotante

Variable	Bloque de memoria
Ref_1	11
	12
Ref_2	13
	14
Ref_3	15
	16
Ref_4	17
	18
A_y1	19
	20
A_y2	21
	22
A_y3	23
	24
A_y4	25
	26
B_y1	27
	28
B_y2	29
	30
B_y3	31
	32
B_y4	33
	34
A_d1	35
	36
A_d2	37
	38
A_d3	39
	40
B_d1	41
	42
B_d2	43
	44
B_d3	45
	46
Alfa_1	47
	48
Alfa_2	49
	50
Alfa_3	51
	52
Alfa_4	53
	54
Lambda_1	55
	56

Variable	Bloque de memoria
Lambda_2	57
	58
Lambda_3	59
	60
Lambda_4	61
	62
ΔU1	63
	64
ΔU2	65
	66
ΔU3	67
	68
ΔU4	69
	70
U1_max	71
	72
U2_max	73
	74
U3_max	75
	76
U4_max	77
	78
U1_min	79
	80
U2_min	81
	82
U3_min	83
	84
U4_min	85
	86
Y1_max	87
	88
Y2_max	89
	90
Y3_max	91
	92
Y4_max	93
	94
Y1_min	95
	96
Y2_min	97
	98
Y3_min	99
	100
Y4_min	101
	102

Variable	Bloque de memoria
A_u1	103
	104
A_u2	105
	106
A_u3	107
	108
A_u4	109
	110
B_u1	111
	112
B_u2	113
	114
B_u3	115
	116
B_u4	117
	118
Y1_L	119
	120
Y2_L	121
	122
Y3_L	123
	124
Y4_L	125
	126
D1_L	127
	128
D2_L	129
	130
D3_L	131
	132
U1_L	133
	134
U2_L	135
	136
U3_L	137
	138
U4_L	139
	140
Ref1_L	141
	142
Ref2_L	143
	144
Ref3_L	145
	146
Ref4_L	147
	148

A continuación, se va explicar la parte del código de programación que se encarga de la lectura/escritura de los datos de tipo booleano. A diferencia de los bloques de registro de retención, estos actualizan su valor en los bloques de memoria de Modbus de forma periódica, concretamente

cada un segundo. Para ello, se ha empleado un bucle temporizado (ver Figura 11) que activa el caso de la estructura principal que gestiona el Modbus.

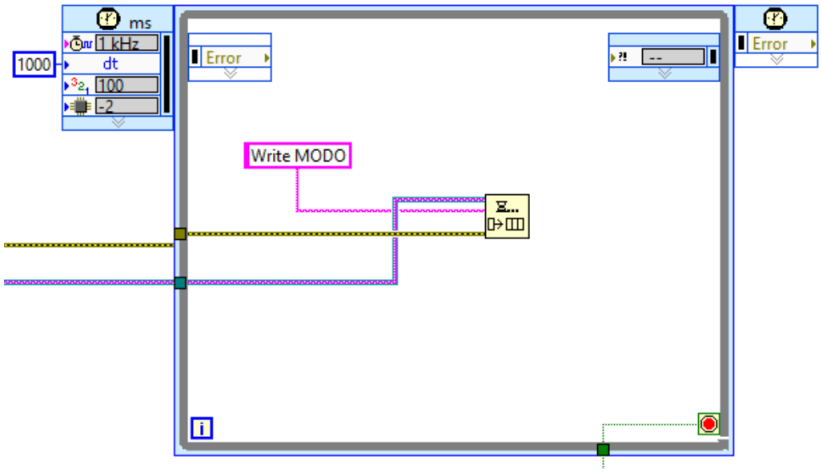


Figura 11: Bucle que gestiona el caso de las variables booleanas y la variable modo funcionamiento

El motivo por el que este caso se debe ejecutar de forma periódica es porque incluye la lectura de los indicadores de tipo booleano, que muestran si se ha realizado o no una determinada operación en la aplicación de control de la RPi, que ha solicitado el usuario a través de la interfaz gráfica.

En la Figura 13, se observa el caso en cuestión donde se realiza la lectura (ver Figura 12) de las variables de tipo booleano que actúan como indicadores en la interfaz gráfica, y la escritura (ver Figura 12) de los controles de tipo booleana que gestionan cada una de las funciones de la aplicación de control.

Figura 12: Funciones de la librería Modbus para la lectura (izquierda) y la escritura (derecha) en los registros de tipo booleano

Como se observa en esta misma figura, también se realiza la gestión de la variable modo de funcionamiento. Esto es debido a que el usuario de la interfaz gráfica siempre tiene prioridad para establecer el modo de funcionamiento de la aplicación. Es decir, si el usuario ha establecido una configuración en la interfaz gráfica y se conecta a la aplicación de la RPi, y esta se encuentra en un modo de funcionamiento diferente al que se encuentra la aplicación de control, entonces el modo de funcionamiento cambia automáticamente al establecido por el usuario. Finalmente, las posiciones que ocupan cada una de las variables booleanas en los bloques de memoria se pueden observar en la Tabla 3.

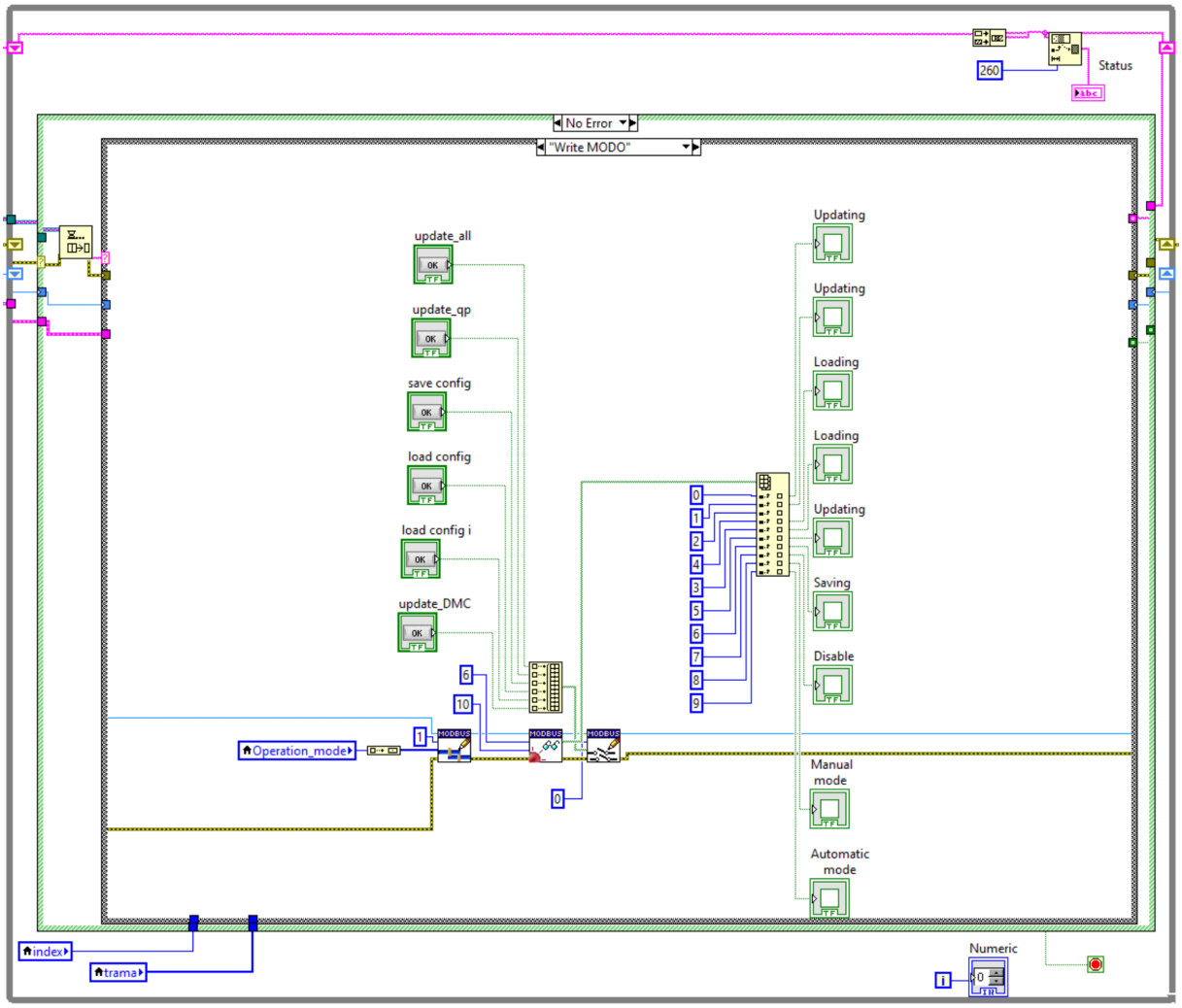


Figura 13: Caso que gestiona las variables booleanas y la variable modo de funcionamiento

Tabla 3: Bloques de memoria de tipo bobina de las variables booleanas

Variable	Bloque de memoria
Update_all_data	0
Update_qp	1
Guardar_configuracion_txt	2
Cargar_configuracion_txt	3
Cargar_conf_inicial_txt	4
Update_dmc	5
Update_all_data_L	6
Update_qp_L	7

Variable	Bloque de memoria
Guardar_configuracion_txt_L	8
Gargar_configuracion_txt_L	9
Cargar_conf_inicial_txt_L	10
Update_dmc_L	11
Guardado_L	12
Disable	13
Manual	14
Auto	15

Otro de los casos de mayor importancia en la programación de la interfaz gráfica, es el que se encarga de gestionar las gráficas para realizar la monitorización de las variables de interés, que se ejecuta cada un segundo mediante el mismo código que se emplea para los bloque de tipo booleano. Este caso (ver Figura 14) se compone de la función de lectura de los registros de retención de Modbus que devuelve un vector con una serie de datos de tipo entero. No obstante, la información que se desea gestionar en las gráficas es de tipo flotante y por tanto, es necesario realizar la conversión de dos datos de tipo entero a uno de tipo flotante. Para ello, se ha realizado la función *int to float* que realiza esta función. Una vez obtenido los flotantes se representan en sus correspondientes gráficas.

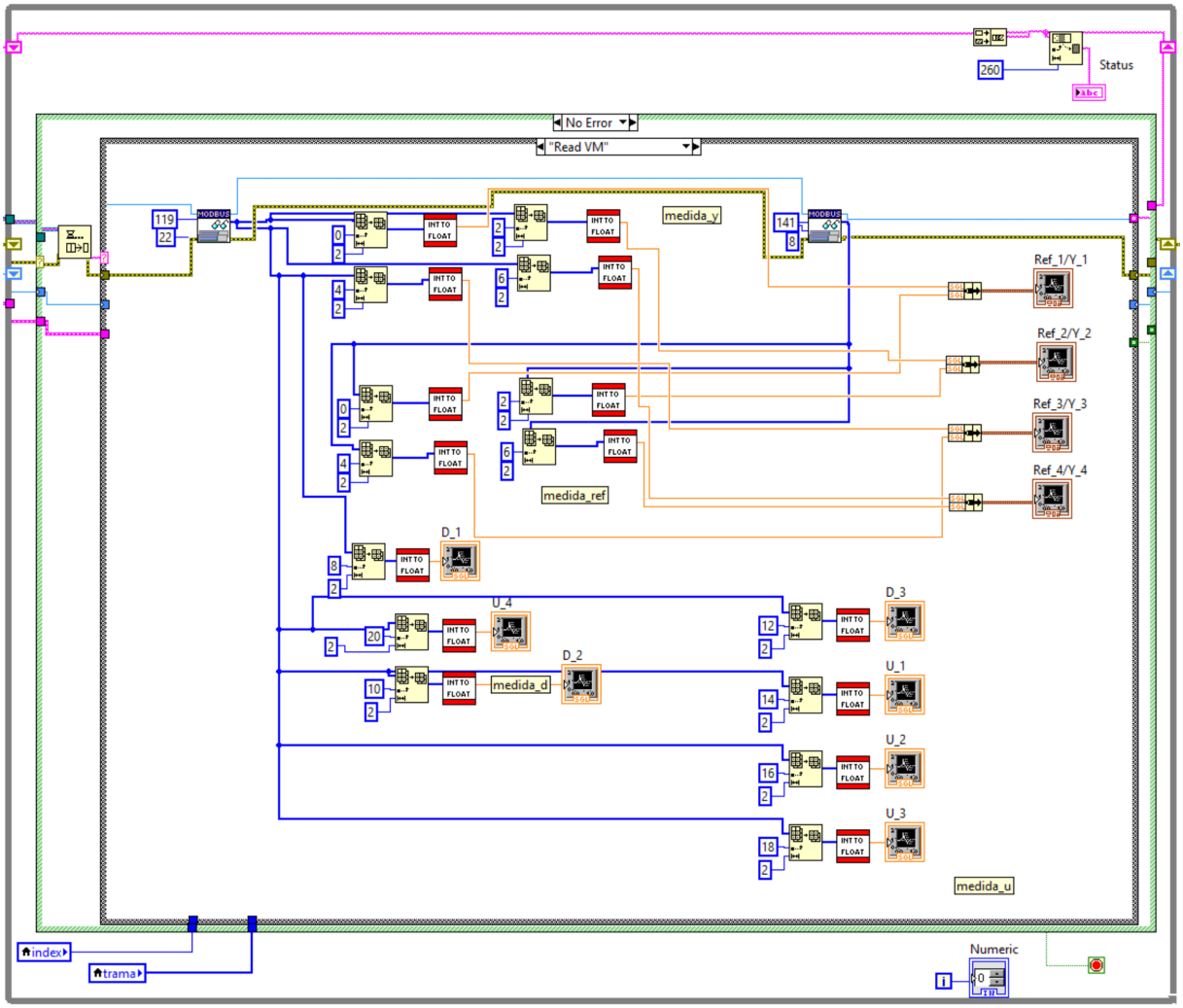


Figura 14: Caso que gestiona las lecturas de las variables de interés

Finalmente, el último caso de gran interés del bucle que gestiona la comunicación Modbus es el que se encarga de realizar la desconexión de la red (ver Figura 15). Como se ha podido observar en algunas de las imágenes, el bucle principal está compuesto por dos estructuras *case,* donde una se encuentra dentro de la otra. Esto es debido a que si se produce algún tipo de error en alguna de las funciones empleadas de la librería Modbus, muestre por pantalla el valor que se ha producido para conocer donde se encuentra el error. Este hecho se puede observar en la Figura 16.

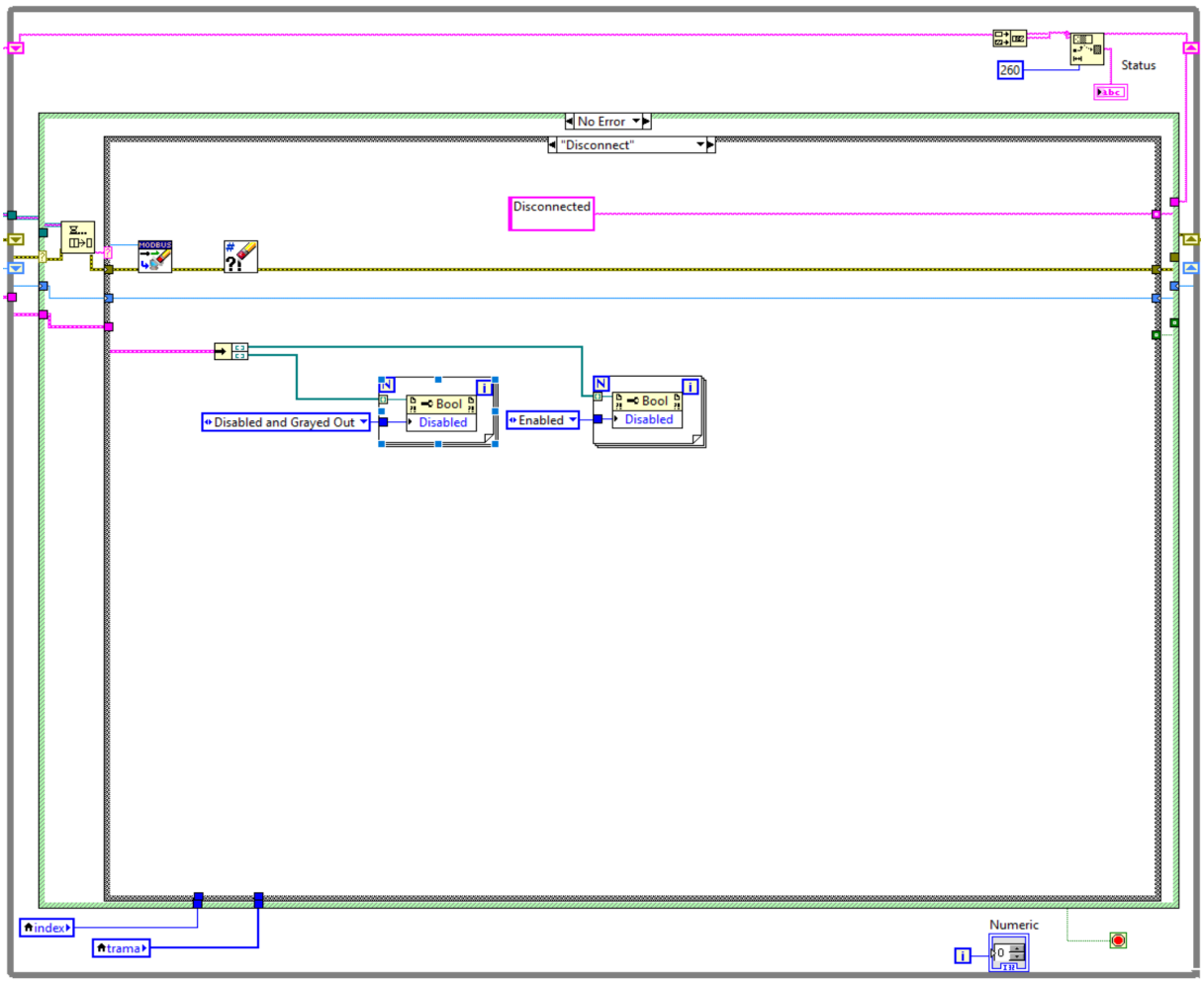


Figura 15: Caso que gestiona la desconexión de la red Modbus

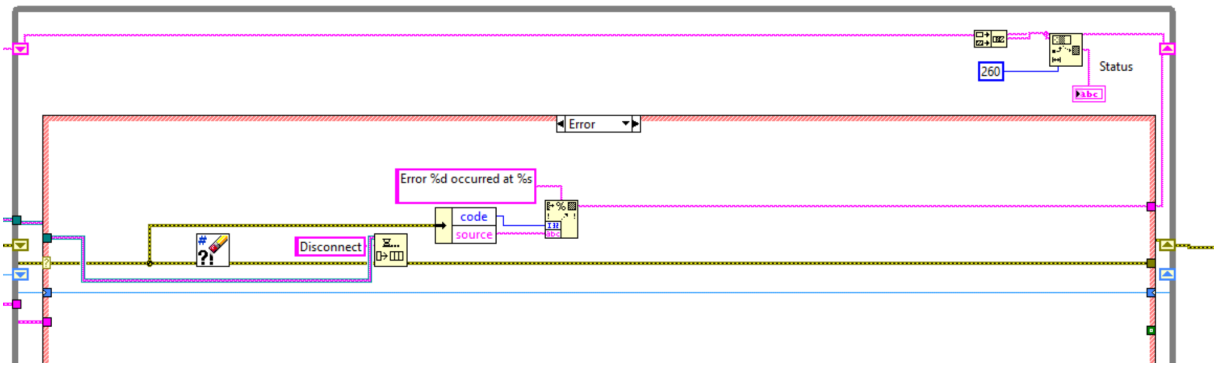


Figura 16: Caso que gestiona si se produce algún error durante la ejecución del programa

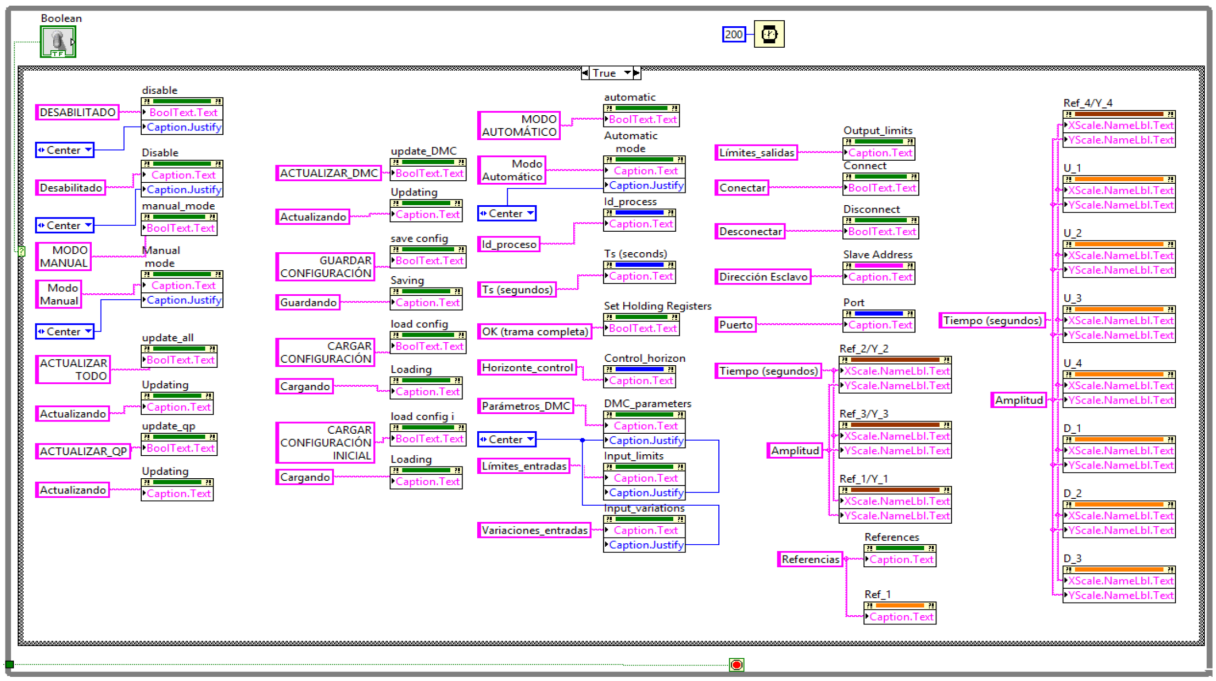


Figura 17: Bucle que gestiona el idioma de la aplicación

Otra de las partes que conforman el código de la interfaz gráfica, es la que realiza la función de cambiar el idioma de la aplicación mediante un interruptor de tipo booleano. Para cambiar el idioma se ha accedido a cada uno de los títulos de todas las variables y gráficas mediante la herramienta *Property Node* de LabVIEW.

La última parte del código de esta aplicación es la que se encarga de gestionar habilitar y deshabilitar ciertos pulsadores dependiendo de las condiciones en las que se encuentre la aplicación (ver Figura 18). El funcionamiento de esta parte de la aplicación se explica con mayor profundidad en el manual de usuario de la interfaz gráfica.

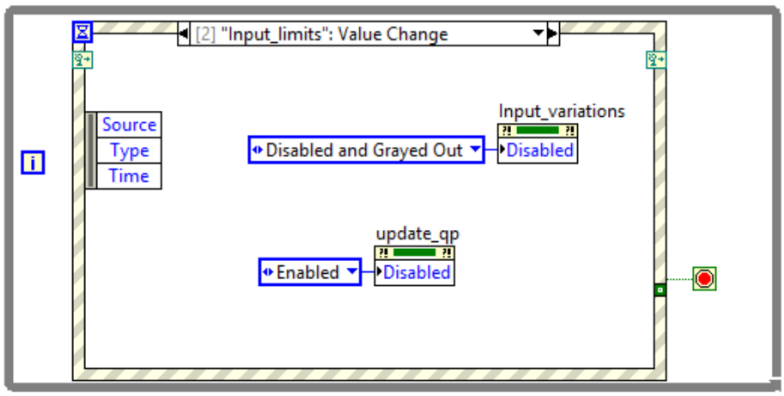


Figura 18: Bucle que gestiona el estado de los pulsadores de la aplicación

4. MANUAL DE PROGRAMACIÓN DEL HARDWARE-IN-THE-LOOP

En este apartado se va abordar la programación realizada para la aplicación que realiza el papel de HIL. Para esta aplicación también se ha empleado el lenguaje de programación de LabVIEW. En este caso, es necesario tener instaladas las librerías de las tarjetas de adquisición de datos (NI 6001) [3] y la librería LabVIEW 2017 Control Design and Simulation Module [1].

El primer paso en esta aplicación consiste en configurar las entradas y salidas de cada una de las TADs, estableciendo el dispositivo empleado y el rango de tensión de cada una de ellas (ver Figura 19).

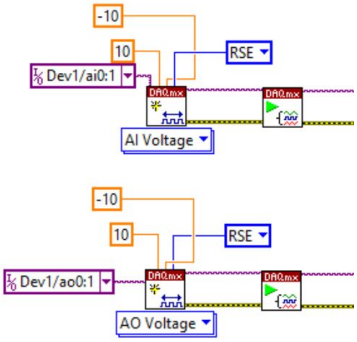


Figura 19: Configuración de las tarjetas de adquisición de datos

Seguidamente se establece el modo de adquisición y escritura de la información (una muestra por iteración del bucle temporizado) de las TADs y la función de transferencia que describe el comportamiento del proceso, que al estar dentro de un bucle temporizado se ha tenido que implementar en discreto a un periodo Ts = 1 segundo (ecuación 1).

$$\mathbf{H(z)} = \begin{bmatrix} \frac{0{,}0802}{z^{28} - 0{,}9802z^{27}} & \frac{0{,}02926}{z^{29} - 0{,}9835z^{28}} & \frac{0{,}1164}{z^{28} - 0{,}9802z^{27}} \\ \frac{0{,}1067}{z^{19} - 0{,}9802z^{18}} & \frac{0{,}09454}{z^{15} - 0{,}9835z^{14}} & \frac{0{,}1704}{z^{16} - 0{,}9753z^{15}} \\ \frac{0{,}1307}{z^{21} - 0{,}9702z^{20}} & \frac{0{,}09932}{z^{23} - 0{,}9775z^{22}} & \frac{0{,}3691}{z - 0{,}9487} \end{bmatrix} \quad (1)$$

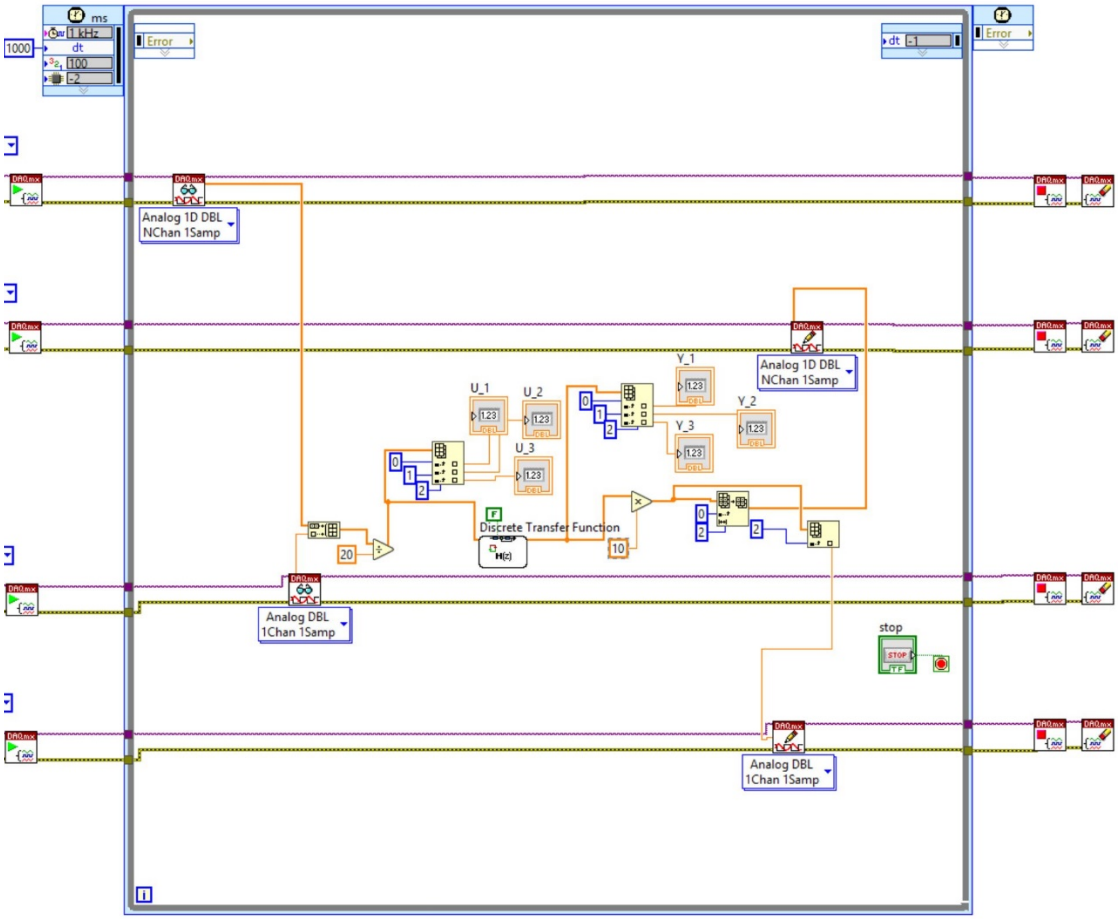


Figura 20: Bucle temporizado que gestiona el HIL

En la Figura 20, se observa cada una de las funciones empleadas. Cabe destacar que tanto a la entrada como a la salida de la función de transferencia, se ha revertido la recta de conversión aplicada en la aplicación de control de la RPi, para que concuerden las unidades que se recibe/manda de las TADs con las unidades de la función de transferencia.

5. BIBLIOGRAFÍA

[1] National Instruments (s.f.) *Context Help de LabVIEW.*

[2] National Instruments (s.f.) *Modbus Library for LabVIEW.* Obtenido el 14 de abril de 2018 de http://www.ni.com/example/29756/en/

[3] National Instruments (s.f.) *USB-6001.* Obtenido el 15 de Julio de 2018 de http://www.ni.com/es-es/support/model.usb-6001.html

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC:

ANEXO 4: MANUALES DE PROGRAMACIÓN DE LAS APLICACIONES IMPLEMENTADAS EN LA RASPBERRY PI

ÍNDICE:

ÍNIDICE DE CÓDIGOS:

1. INTRODUCCIÓN

En el presente documento el manual de programación de las dos aplicaciones que se ejecutan en la RPi. La primera de ellas corresponde con el servidor Modbus y la segunda, con la aplicación de control QDMC.

2. ESPECIFICACIONES DE DISEÑO DE LA APLICACIÓN DE CONTROL

Las principales especificaciones de diseño de la aplicación de control son las siguientes:

- Programación de un algoritmo de control QDMC mediante el lenguaje de programación de Python.
- Establecer conexión con diferentes dispositivos, empleando el protocolo Modbus TCP.
- Establecer diferentes modos de funcionamiento en la aplicación.
- Programación del automatismo que se encarga de gestionar los modos de funcionamiento, la actualización de las variables locales del programa y de los bloques de memoria de Modbus.
- Establecer el algoritmo de seguridad (watchdog) que en caso de fallo de la aplicación, se reinicia la RPi y se vuelve a ejecutar las aplicaciones.
- Capacidad de cargar varios procesos/configuraciones desde la memoria interna de la RPi.
- Capacidad de almacenar la última configuración establecida por el usuario.
- Gestión de las entradas/salidas de la TAD diseñada.

3. FICHEROS QUE CONSTITUYEN LAS APLICACIONES.

Las aplicaciones desarrolladas han sido programadas en diferentes ficheros Python que a continuación se enumeran:

- **Globals.py:** contiene las funciones programadas explícitamente para esta aplicación, que poseen una funcionalidad en el apartado de gestión de datos y conversión de variables.
- **Funciones_dmc:** contiene las funciones encargadas para realizar las diversas matrices que constituyen el algoritmo QDMC. Estas han sido programadas explícitamente para esta aplicación.
- **Aplicación_de_control:** posee el código referente a la aplicación principal de la RPi. Hace uso de las funciones que se encuentran en los ficheros anteriormente mencionados.
- **Aplicación_Modbus:** es una aplicación independiente que su funcionalidad es crear un servidor Modbus TCP en la RPi.

4. PROGRAMACIÓN DEL SERVIDOR MODBUS

Para la programación de la aplicación que actúa como servidor del protocolo Modbus (ver Código 1) es necesario instalar previamente la librería *pyModbusTCP* [3] e importar las funciones correspondientes. Para el funcionamiento de la aplicación únicamente es necesario la dirección IP del servidor y el puerto de conexión. Una vez ejecutada, esta aplicación se ejecuta de forma continua, en segundo plano, y simultáneamente a la aplicación de control.

```
# Servidor Modbus/TCP
import numpy as np
import time

import argparse
from pyModbusTCP.server import ModbusServer
from pyModbusTCP.server import DataBank

parser = argparse.ArgumentParser()
parser.add_argument('-H', '--host', type=str, default='192.168.1.92', help='Host')
parser.add_argument('-p', '--port', type=int, default=1025, help='TCP port')
args = parser.parse_args()
server = ModbusServer(host=args.host, port=args.port)
server.start()
```

Código 1. Aplicación que actúa como servidor del protocolo Modbus TCP

5. PROGRAMACIÓN DE LA APLICACIÓN PRINCIPAL

La aplicación principal está estructura fundamentalmente en tres partes, cada una de ellas corresponde a un bucle temporizado que se ejecuta cíclicamente con un periodo de tiempo determinado. Asimismo, para que el programa funcione, es necesario instalar previamente las librerías *Scipy* [1], *Cvxopt* (optimizador QP) [2] y *pyModbusTCP* [3]. Asimismo, se importan una serie de funciones de los ficheros *globals.py* y *funciones_dmc.py* que se han programado explícitamente para esta aplicación y se explicarán a lo largo de este documento.

```
from scipy import signal
from scipy.linalg import block_diag,inv
import itertools
import numpy as np
from cvxopt import matrix, solvers
import time
from threading import Timer
from pyModbusTCP.client import ModbusClient
import globals
import funciones_dmc
import glob, os
```

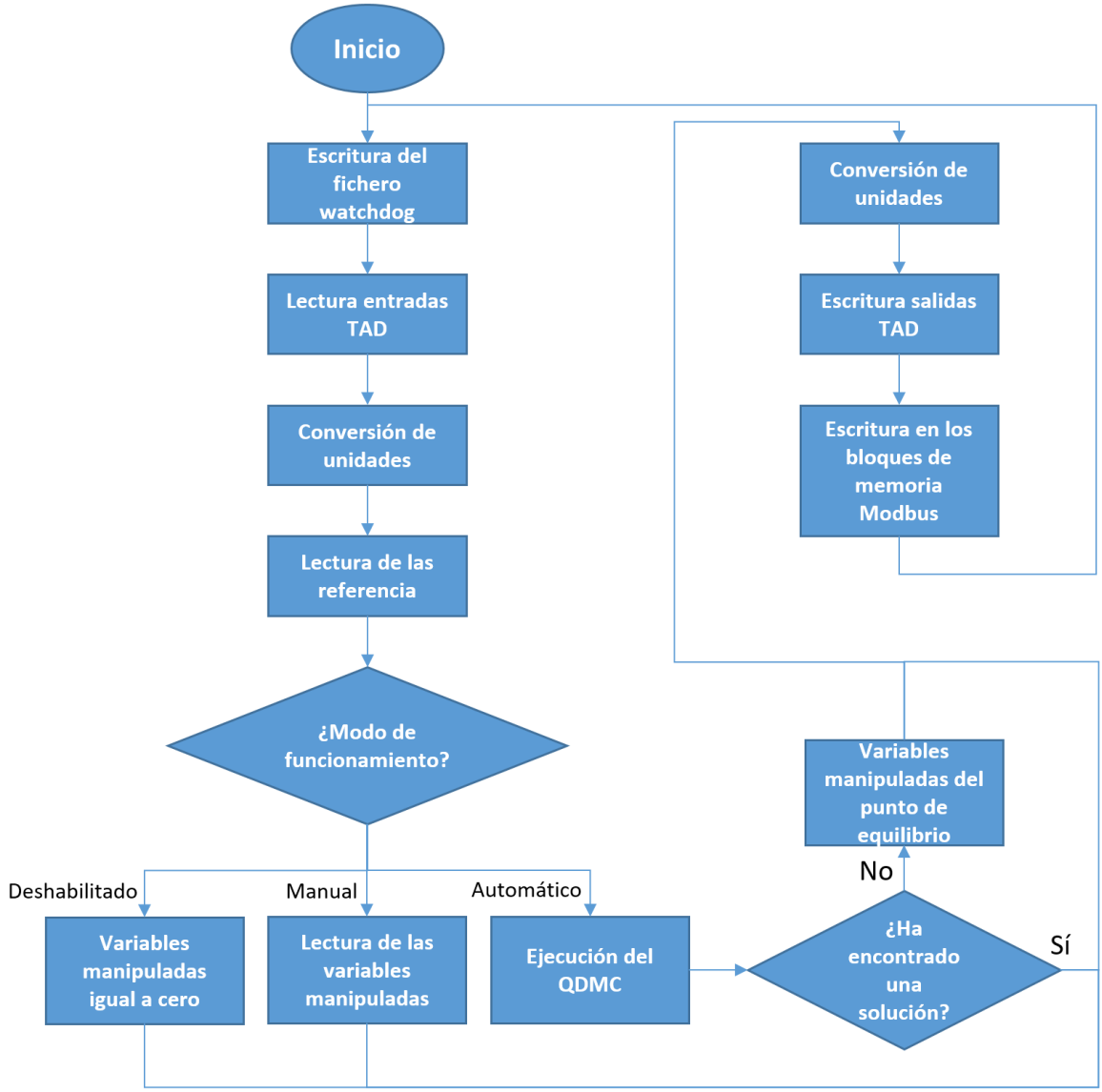
Código 2: Importación de las librerías necesarias para la aplicación

El primer bucle temporizado desempeña la función del controlador de la aplicación, que según el modo de funcionamiento en el que se encuentre, el bucle ejecuta una determinada parte del código. La primera parte del bucle se encarga de la inicialización de una serie de variables y de la activación de la función watchdog. Esta función actúa como medida de seguridad en caso de que la aplicación se detuviese de forma repentina, reiniciando la RPi y volviendo a ejecutar la aplicación. Seguidamente, se

realiza la lectura de las entradas analógicas de la TAD, que reciben la información correspondiente de los sensores del proceso industrial a controlar, empleando la función *adc.read_voltage()* [4]. No obstante, como los componentes electrónicos empleados en el montaje de la TAD no poseen exactamente los valores que se han calculado, es necesario realizar una corrección de las medidas obtenidas para cada una de las entradas (ver Código 3). Una vez corregidas, se almacenan los valores en los vectores *y* o *d*, dependiendo de si se trata de las salidas o las perturbaciones del proceso respectivamente. Posteriormente, se realiza la lectura de las referencias recibidas mediante el protocolo Modbus (*globals.lectura_trama_var_float()*) y se realiza la conversión de voltios a unidades del proceso (*funciones_dmc.c_datos()*) (ver Código 4) de las medidas obtenidas de la TAD. El comportamiento general de este bucle temporizado, esta descrito en el flujograma 1.

```
38  def DMCTask():
39      time_in = time.time()
40      global c,p,n,n_salidas,n_entradas,n_referencia,n_perturbaciones,delta_u_max,delta_u_min
41      global T_m,M_m, G_m, lambda_t, alfa_t, f_m, hessian, esc_p_ext_m,esc_p_m, esc_m, A_qdmc,H
42      global  bias_m, y_free, e, deltau, u, u_ant, d, d_ant, ref_m, deltau_recortado
43      global trama_ref,trama_y, Ts, u_max, u_min, y_max, y_min, ro
44      global U_send, u_ant, Y_send, Ref_send
45      global A_y,B_y,A_d,B_d, A_u, B_u
46      global Modo_funcionamiento
47      global f_watchdog
48
49      #Escritura WATCHDOG
50      os.write(f_watchdog,'watchdog')
51      #Inicializacion de variables
52      bias_m=0
53      ref_m=0
54      #########LECTURA DATOS TAD###########
55      lectura_y=[]
56      lectura_d=[]
57
58      for x in range(0,n_salidas):
59          voltage = adc.read_voltage(x+1)
60          if (x==0):          ##Corrección de los voltajes
61              voltage = 1.0225*(voltage + 0.18)
62          if (x==1):
63              voltage = 1.0246*(voltage + 0.15)
64          if (x==2):
65              voltage = 1.02145*(voltage + 0.16)
66          if (x==3):
67              voltage = voltage
68          lectura_y.append(voltage)
69      if n_perturbaciones > 0:
70          for y in range(n_salidas,n_perturbaciones):
71              voltage=adc.read_voltage(x+1)
72              lectura_d.append(voltage)
73      else:
74          lectura d=0
```

Código 3: Gestión de las entradas de la TAD y el watchdog en el bucle del controlador



Flujograma 1: Bucle del controlador

```
75      #Lectura de las referencias
76      Ref = globals.lectura_trama_var_float(trama_ref,2*n_salidas)
77      Y = lectura_y
78      D = lectura_d
79
80      print("ref")
81      print(Ref)
82      print("\n")
83
84      ref=Ref
85
86      #Conversión de unidades de las salidas de proceso
87      y=funciones_dmc.c_datos(A_y,B_y,n_salidas,Y)
88
89      print("Y")
90      print(y)
91      print("\n")
92
93      if n_perturbaciones>0: #Medida de las perturbaciones medibles
94          D=np.zeros((n_perturbaciones,1))
95          d_ant = np.zeros((n_perturbaciones,1))
96          d=funciones_dmc.c_datos(A_d,B_d,n_perturbaciones,D)
97
98      else:
99          d=0
100         d_ant=0
```

Código 4: Gestión de las referencias y la conversión de unidades

Una vez realizada las lecturas de la TAD, dependiendo del modo de funcionamiento en el que se encuentre el controlador, se ejecuta una parte u otra del código. Si el modo de funcionamiento es igual a cero, se ejecuta el controlador en modo deshabilitado, estableciendo las acciones de control a cero. Si el modo es igual a uno, se ejecuta el controlador en modo manual, estableciendo las acciones de control a través de la interfaz gráfica remota y del protocolo Modbus. Puesto que, Modbus únicamente dispone de bloques de memoria de hasta 16 bits, es decir solo se pueden almacenar datos de tipo de entero, es necesario realizar la conversión de dos enteros a un flotante (*globals.int2tofloat()*). Toda esta parte del código se puede observar a continuación.

```
102     if (Modo_funcionamiento == 0): #Modo deshabilitado
103         u=np.zeros((4,1))
104         u_porcentaje=u
105         u_ant=u[0:n_entradas]
106
107     if (Modo_funcionamiento == 1): #Modo manual
108         U_1_porcentaje = ([trama_1[5],0])
109         U_2_porcentaje = ([trama_1[6],0])
110         U_3_porcentaje = ([trama_1[7],0])
111         U_4_porcentaje = ([trama_1[8],0])
112         u_porcentaje = ([U_1_porcentaje,U_2_porcentaje,U_3_porcentaje,U_4_porcentaje])
113         #Conversión de 2 enteros a 1 flotante
114         u_porcentaje = globals.int2tofloat(u_porcentaje)
115         #Conversión de unidades de las acciones de control
116         u=funciones_dmc.c_datos(A_u,B_u,n_entradas,u_porcentaje)
117
118         u_ant=u[0:n_entradas]
119         u_ant=np.array(u_ant)
120         u_ant.shape=(n_entradas,1)
```

Código 5: Modos de funcionamiento deshabilitado y manual

Cuando el modo es igual a dos, se ejecuta la parte cíclica del controlador QDMC, realizando el cálculo de una serie de variables propias del algoritmo DMC (bias, respuesta libre, error,…) (ver Código 6) y el optimizador QP (ver Código 7), para el cálculo de las acciones de control óptimas en cada una de las iteraciones del bucle.

```
122     if (Modo_funcionamiento == 2): #Modo automático (QDMC)
123         delta_d = d - d_ant
124         for j in range(0,n_salidas):
125             bias = (y[j] - np.dot(f_m[(n*(j+1))-n][0],np.ones((p,1))))
126             bias_m = np.vstack((bias_m,bias))
127 
128         bias_m=np.delete(bias_m,0,axis=0)
129         y_free= np.dot(T_m,f_m) + bias_m  + np.dot(esc_p_ext_m,delta_d)
130         B_qdmc = funciones_dmc.matriz_B(delta_u_max,delta_u_min,u_max,u_min,y_max,
131                 y_min,u_ant,y_free,c,p,n_entradas,n_salidas)
132 
133         for k in range(0,n_referencia):
134             ref_aux = ref[k]*np.ones((p,1))
135             ref_m = np.vstack((ref_m,ref_aux))
136 
137         ref_m = np.delete(ref_m,0,axis=0)
138         e = ref_m -y_free
139 
140         C_qdmc = funciones_dmc.matriz_C(G_m, alfa_t,e)
```

Código 6: Modo de funcionamiento automático (QDMC)

Asimismo, en la parte del algoritmo QP se ha establecido otra medida de seguridad mediante el comando *try except* (ver Código 7), que en caso de fallo durante el cálculo de las acciones de control mediante el optimizador QP, este devuelve las acciones de control del punto de equilibrio del proceso. El comando *solvers.qp()* [2], proporciona la variación de las variables de control para esa iteración del bucle mediante dicho optimizador, con respecto a las acciones de control aplicadas en la iteración anterior.

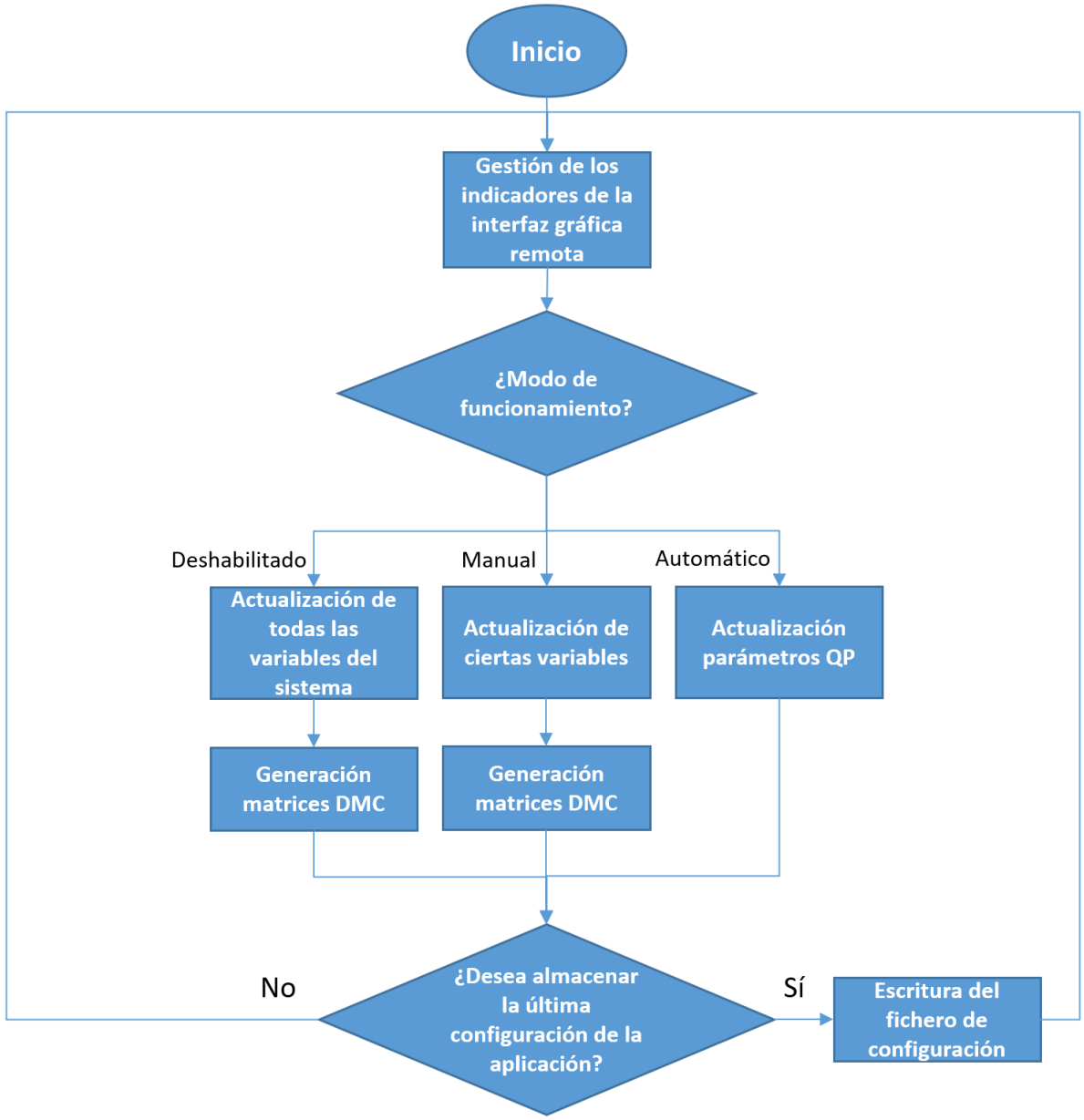
```
141         '''
142         En caso de que el algoritmo QP no funcione, establece las acciones
143         de control del punto de equilibrio
144         '''
145         try:
146             solvers.options['show_progress'] = False  #muestra por pantalla las interaciones del QP
147             sol = solvers.qp(P=matrix(hessian),q=matrix(C_qdmc.T),G=matrix(A_qdmc),h=matrix(B_qdmc))
148             deltau_x = list(sol['x'])
149             s = sol['status']
150             j = sol['primal objective']
151             end_1 = len(deltau_x)
152             deltau = np.asarray(deltau_x[0:end_1-1])
153             deltau.shape = (end_1-1,1)
154             u, deltau_recortado = funciones_dmc.vector_u(u_ant,deltau,c,n_entradas)
155 
156         except Exception as e:
157             u=np.zeros(n_entradas,1) #accion de control correspondiente al punto de equilibrio
158             deltau_recortado = u - u_ant
159 
160         u_ant = u
161         d_ant = d
162         f_m = np.dot(M_m,f_m) + np.dot(esc_m,deltau_recortado) + np.dot(esc_p_m,delta_d)
```

Código 7: Optimizador QP

Cuando se obtienen las acciones de control, estas deben comunicarse a las salidas de la TAD para establecer dichas acciones a los actuadores del proceso a controlar (*dac.single_internal()*) [5] (ver Código 8). Para ello hay que convertir las unidades del proceso a voltios y a porcentaje (*funciones_dmc.c_datos_inv()*) para la monitorización de estas variables en la interfaz gráfica, ya que en los procesos industriales normalmente estas se establecen en porcentaje. En la parte final del bucle se establece el tiempo restante entre el periodo del bucle y el tiempo que se ha consumido en los cálculos pertinentes. Seguidamente, se realiza una espera con el resultado obtenido para cumplir con el periodo establecido.

```
163     u = funciones_dmc.c_datos_inv(A_u,B_u,n_entradas,u)
164     for x in range(0,n_entradas): #Escritura en el DAC
165         if(x==0):
166             dac.single_internal(3,u[x])
167         if(x==1):
168             dac.single_internal(4,u[x])
169         if(x==2):
170             dac.single_internal(1,u[x])
171         if(x==3):
172             dac.single_internal(2,u[x])
173         time.sleep(0.1)
174 
175     u_porcentaje=funciones_dmc.c_datos_inv(A_u1,B_u1,n_entradas,u)
176     U_send = u_porcentaje
177     Y_send = y
178     Ref_send = ref
179 
180     print("U")
181     print (u)
182     print("\n")
183 
184     tnow=time.time()
185     ts = Ts
186     delaytime = ts-(tnow-time_in)
187 
188     print("DelayTime")
189     print(delaytime)
190     print("\n")
191     timer = Timer(delaytime, DMCTask)
192     timer.start()
```

Código 8: Conversión y escritura de las acciones de control en las salidas de la TAD



Flujograma 2: Bucle que gestiona el automatismo de la aplicación

El segundo bucle que forma la aplicación, es el automatismo que gestiona la actualización de las variables locales del programa a partir de los bloques de memoria del protocolo Modbus y los modos de funcionamiento del controlador. Asimismo, gestiona los indicadores de la interfaz gráfica que muestra el estado del controlador en cada iteración (ver Código 9). Según el modo en el que se encuentra el controlador se actualizan una serie de variables y según las variables actualizadas se vuelven a calcular las matrices del algoritmo DMC (*DMC_offline()*). La función que se encarga de actualizar las variables locales a partir de los bloques de memoria Modbus es *globals.lectura_trama_var_float()* y la encargada de actualizar las variables a partir de los ficheros de texto de configuración es *globals.lectura_fichero_configuración()* (ver Código 10, Código 11 y Código 12). Para conocer que variables se pueden modificar en cada uno de los modos de funcionamiento de

la aplicación, consultar el anexo 2: manuales de usuario de las aplicaciones diseñadas. Por otro lado, se puede visualizar el comportamiento general del bucle en el flujograma 2.

```
194  def GrafcetTask():
195      global Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control, Ts, Ref
196      global A_y, B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max, Y_min, A_u, B_u
197      global update_all_data, update_qp, save_config, load_config, load_config_i, update_DMC
198      global update_all_data_updating, update_qp_updating, save_config_updating, load_config_updating
199      global load_config_i_updating, update_DMC_updating, guardado, disable_L, manual_L, auto_L
200      global n_salidas, n_entradas, Modo_funcionamiento
201      global trama_1
202      time_in = time.time()
203      #Código que gestiona los indicadores de la interfaz gráfica
204      if (update_qp==1):
205          update_qp_updating=1
206          time.sleep(0.5)
207      else:
208          update_qp_updating=0
209
210      if (load_config==1):
211          load_config_updating=1
212          time.sleep(0.5)
213      else:
214          load_config_updating=0
215
216      if (load_config_i==1):
217          load_config_i_updating=1
218          time.sleep(0.5)
219      else:
220          load_config_i_updating=0
221
222      if(update_DMC==1):
223          update_DMC_updating=1
224
225          time.sleep(0.5)
226      else:
227          update_DMC_updating=0
228
229      if (update_all_data==1):
230          update_all_data_updating=1
231          time.sleep(0.5)
232      else:
233          update_all_data_updating=0
234
235      if (save_config==0):
236          guardado = 0
```

Código 9: Gestión de los indicadores del estado del controlador

```
238     if (Modo_funcionamiento == 0): #Deshabilitado
239 
240         disable_L = 1
241         manual_L = 0
242         auto_L = 0
243 
244         if (update_all_data == 1)
245             update_all_data=0
246             (Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control,
247              Ts, Ref, A_y, B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max,
248              Y_min, A_u, B_u) = globals.lectura_trama_inicial(trama_1)
249             DMC_offline()
250 
251         if (update_qp == 1):
252             update_qp=0
253             delta_u = globals.lectura_trama_var_float(trama_1[63:71],8)
254             U_max = globals.lectura_trama_var_float(trama_1[71:79],8)
255             U_min = globals.lectura_trama_var_float(trama_1[79:87],8)
256             Y_max = globals.lectura_trama_var_float(trama_1[87:95],8)
257             Y_min = globals.lectura_trama_var_float(trama_1[95:103],8)
258 
259         if (update_DMC == 1):
260             update_DMC=0
261             Horizonte_control = trama_1[9]
262             alfa_d = globals.lectura_trama_var_float(trama_1[47:55],8)
263             lambda_d = globals.lectura_trama_var_float(trama_1[55:63],8)
264             DMC_offline()
265 
266         if (load_config == 1):
267             load_config=0
268             (Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control,
269             Ts, Ref, A_y, B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max,
270              Y_min, A_u, B_u) = globals.lectura_fichero_configuracion("Configuracion_actual.txt")
271             DMC_offline()
272 
273         if (load_config_i==1):
274             load_config_i=0
275             (Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control,
276             Ts, Ref, A_y, B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max,
277             Y_min, A_u, B_u) = globals.lectura_fichero_configuracion("Configuracion_inicial.txt")
278             DMC_offline()
```

Código 10: Actualización de las variables locales en el modo deshabilitado

```
280     if (Modo_funcionamiento == 1): #MODO MANUAL
281         disable_L = 0
282         manual_L = 1
283         auto_L = 0
284         if (update_DMC == 1):
285             update_DMC=0
286             Horizonte_control = trama_1[9]
287             alfa_d = globals.lectura_trama_var_float(trama_1[47:55],8)
288             lambda_d = globals.lectura_trama_var_float(trama_1[55:63],8)
289             DMC_offline()
290 
291         if (update_qp == 1):
292             update_qp=0
293             delta_u = globals.lectura_trama_var_float(trama_1[63:71],8)
294             U_max = globals.lectura_trama_var_float(trama_1[71:79],8)
295             U_min = globals.lectura_trama_var_float(trama_1[79:87],8)
296             Y_max = globals.lectura_trama_var_float(trama_1[87:95],8)
297             Y_min = globals.lectura_trama_var_float(trama_1[95:103],8)
```

Código 11: Actualización de las variables locales en el modo manual

```
    if (Modo_funcionamiento == 1): #MODO MANUAL
        disable_L = 0
        manual_L = 1
        auto_L = 0
        if (update_DMC == 1):
            update_DMC=0
            Horizonte_control = trama_1[9]
            alfa_d = globals.lectura_trama_var_float(trama_1[47:55],8)
            lambda_d = globals.lectura_trama_var_float(trama_1[55:63],8)
            DMC_offline()

        if (update_qp == 1):
            update_qp=0
            delta_u = globals.lectura_trama_var_float(trama_1[63:71],8)
            U_max = globals.lectura_trama_var_float(trama_1[71:79],8)
            U_min = globals.lectura_trama_var_float(trama_1[79:87],8)
            Y_max = globals.lectura_trama_var_float(trama_1[87:95],8)
            Y_min = globals.lectura_trama_var_float(trama_1[95:103],8)

    if (Modo_funcionamiento == 2): #MODO AUTOMATICO DMC
        disable_L = 0
        manual_L = 0
        auto_L = 1

        if (update_qp == 1):
            update_qp=0
            delta_u = globals.lectura_trama_var_float(trama_1[63:71],8)
            U_max = globals.lectura_trama_var_float(trama_1[71:79],8)
            U_min = globals.lectura_trama_var_float(trama_1[79:87],8)
            Y_max = globals.lectura_trama_var_float(trama_1[87:95],8)
            Y_min = globals.lectura_trama_var_float(trama_1[95:103],8)

    if (save_config==1):
        save_config=0
        guardado = globals.guardado_configuracion_actual(Id_process, Modo_funcionamiento, n_u,
        n_y, n_d, U_manual, Horizonte_control, Ts, Ref, A_y, B_y, A_d, B_d, alfa_d, lambda_d,
        delta_u, U_max, U_min, Y_max, Y_min, A_u, B_u)

    tnow=time.time()
    delaytime = 2.0-(tnow-time_in)
    timer = Timer(delaytime, GrafcetTask)
    timer.start()
```

Código 12: Actualización de las variables locales en el modo automático y guardado de la configuración del controlador

Finalmente, en la última parte de este bucle (ver Código 12) se encuentra la parte de código que almacena la configuración actual del controlador en un fichero de texto (*globals.guardado_configuración_actual()*) y el cálculo del tiempo restante de espera del bucle (*timer()*) a partir del periodo del bucle y el tiempo consumido en las operaciones que se encuentran en dicho bucle.

La siguiente parte del código corresponde con la generación de las matrices del DMC que únicamente es necesaria su ejecución una vez, si no se varía ninguno de los parámetros mediante la interfaz gráfica. En esta parte, se establecen los diferentes parámetros de configuración del DMC (número de entradas, salidas, perturbaciones,…), y se generan las matrices para realizar la ejecución del algoritmo. El código del DMC se establece en las siguientes imágenes.

```
def DMC_offline():
    global Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control, Ts, Ref
    global A_y, B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max, Y_min
    global c,p,n,n_salidas,n_entradas,n_referencia,n_perturbaciones,delta_u_max
    global delta_u_min,u_max, u_min, y_max, y_min, ro
    global T_m,M_m,  G_m, lambda_t, alfa_t, f_m, hessian, esc_p_ext_m,esc_p_m, esc_m, A_qdmc,H

    n_1= 100 #Instantes en que se estabiliza el sistema
    n= n_1 -1
    c = int(Horizonte_control)
    p = (n_1-1) + c

    n_entradas=int(n_u)
    n_salidas=int(n_y)
    n_perturbaciones=int(n_d)
    n_referencia=int(n_y)

    lambdai=lambda_d[0:n_entradas]
    alfai=alfa_d[0:n_salidas]

    ro=100000000

    delta_u_max=delta_u[0:n_entradas]
    delta_u_min=delta_u[0:n_entradas]
    u_max=U_max[0:n_entradas]
    u_min=U_min[0:n_entradas]
    y_max=Y_max[0:n_salidas]
    y_min=Y_min[0:n_salidas]

    M, T = funciones_dmc.matriz_T(n,p)
    f = np.zeros((n,1))

    M_m=M
    T_m=T
    f_m=f
```

Código 13: Configuración del algoritmo DMC

```
    for i in range(0,n_salidas-1):
        M_m=block_diag(M_m,M)
        T_m=block_diag(T_m,T)
        f_m=np.vstack((f_m,f))

    lambda_t=[]
    alfa_t=[]

    G_m=np.zeros((1,c*n_entradas))
    esc_m=np.zeros((1,n_entradas))

    for i in range(0,n_salidas):

        G_t=np.zeros((p,1))
        esc_t=np.zeros((n,1))
        lambda_t=[]

        for j in range(0,n_entradas):
            esc = funciones_dmc.esc(j,i,n,Id_process)
            G=funciones_dmc.matriz_G(c,esc,p)
            length = len(esc)
            esc=np.array(esc)
            esc.shape = (length,1)
            esc_t = np.hstack((esc_t,esc[0:n]))
            G_t=np.hstack((G_t, G))
            lambda_aux=funciones_dmc.matriz_lambda(lambdai[j],c)
            lambda_t=block_diag(lambda_t,lambda_aux)
```

Código 14: Cálculo de las matrices del algoritmo DMC

```
            G_t=np.delete(G_t,0,axis=1)
            esc_t=np.delete(esc_t,0,axis=1)
            alfa_aux=funciones_dmc.matriz_alfa(alfai[i],p)
            alfa_t=block_diag(alfa_t,alfa_aux)
            G_m = np.vstack((G_m,G_t))
            esc_m = np.vstack((esc_m,esc_t))

        G_m = np.delete(G_m,0,axis=0)
        esc_m = np.delete(esc_m,0,axis=0)
        #alfa_t = np.delete(alfa_t,0,axis=0)
        #lambda_t=np.delete(lambda_t,0,axis=0) #comentarlo en la raspberry
        H, hessian = funciones_dmc.matriz_H(G_m,alfa_t,lambda_t)
```

```
        if n_perturbaciones>0:
            esc_p_m = np.zeros((1,n_perturbaciones))
            esc_p_ext_m = np.zeros((1,n_perturbaciones))

            for i in range(0,n_salidas):
                esc_p_t = np.zeros((n,1))
                esc_p_ext_t = np.zeros((p,1))

                for j in range(0,n_perturbaciones):
                    esc = funciones_dmc.esc_p(j,i,n,Id_process)
                    esc_p=np.array(esc[0:n])
                    esc_p.shape=(n,1)
                    esc_p_ext=esc_p

                    for ij in range(0,p-n):
                        esc_p_ext=np.vstack((esc_p_ext,esc_p[n-1]))

                    esc_p_t = np.hstack((esc_p_t, esc_p))
                    esc_p_ext_t = np.hstack((esc_p_ext_t,esc_p_ext))

                esc_p_t = np.delete(esc_p_t,0,axis=1)
                esc_p_ext_t = np.delete(esc_p_ext_t,0,axis=1)

                esc_p_m = np.vstack((esc_p_m,esc_p_t))
                esc_p_ext_m = np.vstack((esc_p_ext_m,esc_p_ext_t))

            esc_p_m = np.delete(esc_p_m,0,axis=0)
            esc_p_ext_m = np.delete(esc_p_ext_m,0,axis=0)

        else:
            esc_p_m = 0
            esc_p_ext_m = 0

        A_qdmc=funciones_dmc.matriz_A(c,n_entradas,G_m)

        hessian=block_diag(hessian,ro)
```

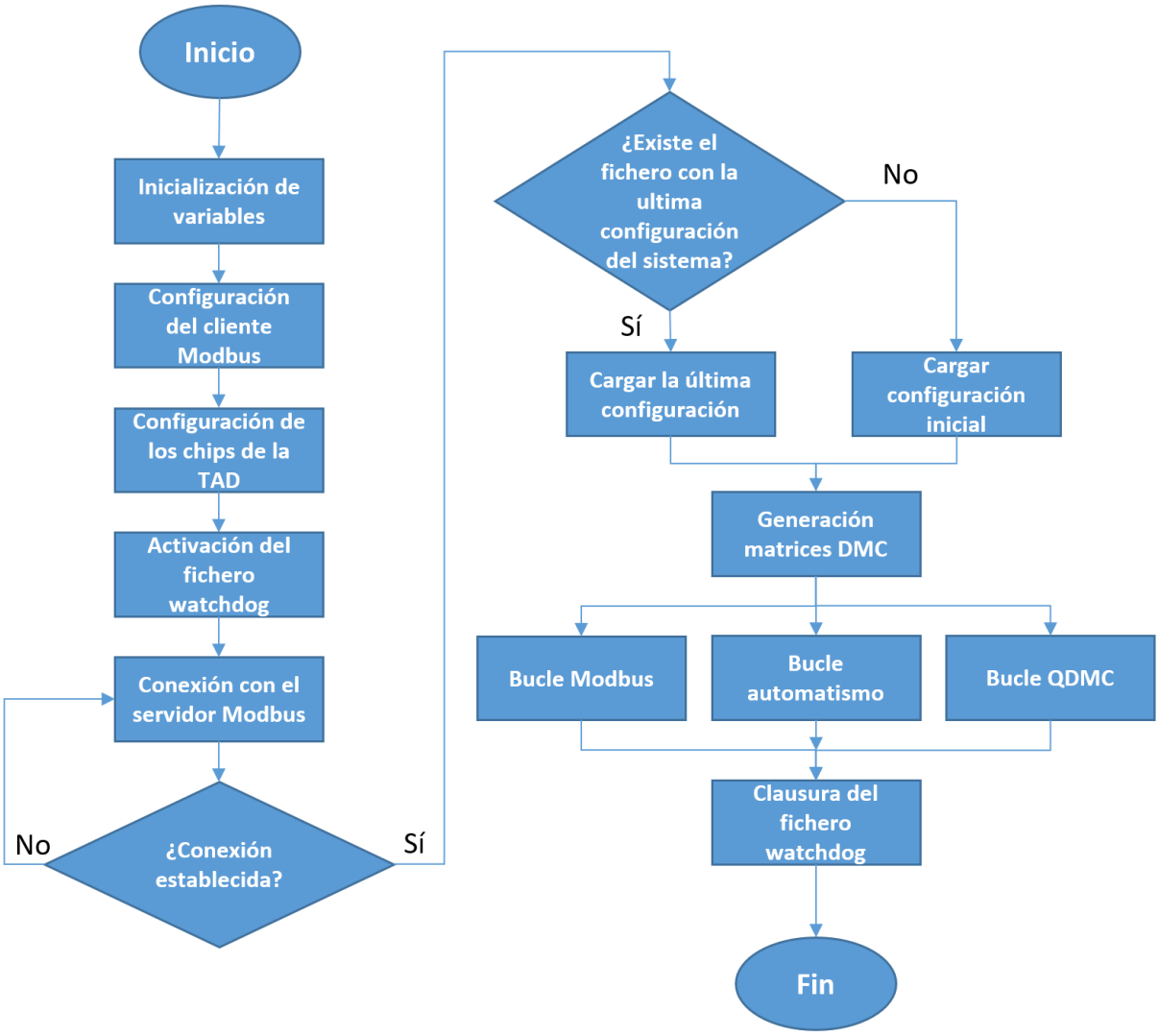
Código 15: Cálculo de las matrices del algoritmo DMC

En la siguiente parte del código, se establece los parámetros de configuración de los chips que forman la TAD (ver Código 16), estableciendo las direcciones de cada uno de los convertidores [4] [5] y la configuración de cada una de las entradas/salidas digitales [6]. A continuación, se establece la dirección del servidor Modbus (dirección IP de la RPi) y el puerto de conexión (ver Código 17), realizando la conexión mediante el servidor. Seguidamente, dependiendo de los archivos de configuración existentes en la carpeta raíz de la aplicación, se carga una configuración u otra, estableciendo la configuración correspondiente (*globals.lectura_fichero_configuración()*). Por otra parte, se verifica si la conexión Modbus se ha establecido y si no se vuelve a intentar establecer conexión con el servidor Modbus (*MB_C.open()*) [3]. Seguidamente, se realiza una primera

actualización de las variables en los bloques de memoria Modbus, cuando se ejecuta por primera vez la aplicación de control.

```
438  ##############Configuración direcciones chips TAD################
439
440  bus=SMBus(1)
441  adc = MCP3424(0x68, 0x68, 18)
442  dac = MCP4728(0x60)
443  address_dac=0x60
444  address_adc=0x68
445  GPIO.setup(5, GPIO.OUT, initial=GPIO.LOW) #dac en modo continuo
446  dig = Adafruit_MCP230XX(address = 0x20, num_gpios = 8) # MCP23008
447
448  #Configuración de las I/O digitales
449  dig.config(0, dig.OUTPUT)
450  dig.config(1, dig.OUTPUT)
451  dig.config(2, dig.OUTPUT)
```

Código 16: Configuración de las direcciones de los ADC, DAC y Digital Expander



Flujograma 3: Parte inicial de la aplicación

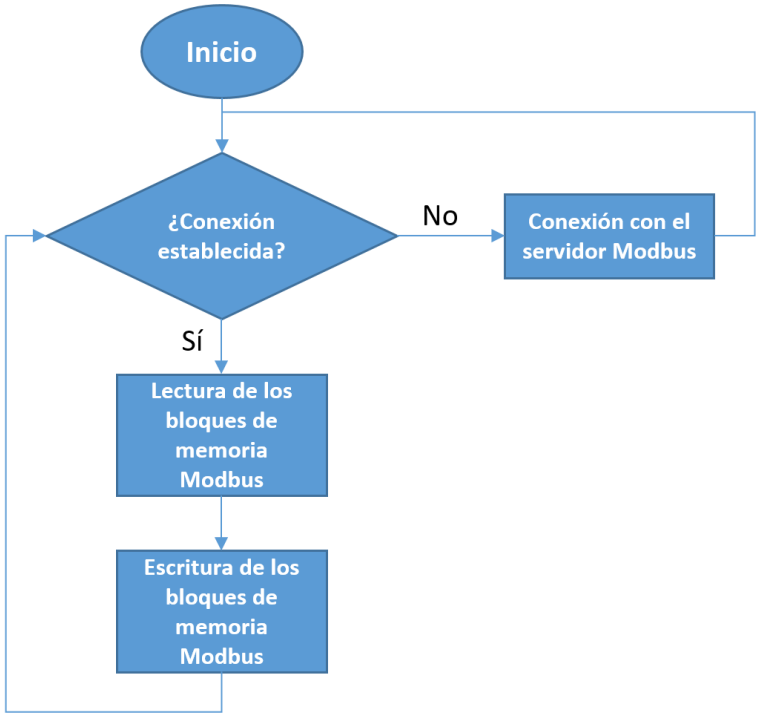
Las funcionalidades de las primeras líneas de código, cuando se inicia por primera vez la aplicación, se pueden observar en el flujograma 3, ya que estas líneas son las encargadas de iniciar cada uno de los bucles que constituyen esta aplicación.

```
#Dirección IP y puerto del servidor
SERVER_HOST = "192.168.1.92" #192.168.1.137   158.42.206.6
SERVER_PORT = 1025

MB_C = ModbusClient(host=SERVER_HOST, port=SERVER_PORT)

fichero_inicial = "Configuracion_inicial.txt"
fichero_actual = "Configuracion_actual.txt"

#Lectura del fichero de configuración correspondiente

initial_file_load = 0

for file in glob.glob("Configuracion_actual.txt"):
    initial_file_load = 1

if(initial_file_load == 0):
    (Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control, Ts, Ref, A_y,
     B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max, Y_min, A_u, B_u) = (
     globals.lectura_fichero_configuracion(fichero_inicial))
    print ("inicial")
else:
    (Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual, Horizonte_control, Ts, Ref, A_y,
     B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min, Y_max, Y_min, A_u, B_u) = (
     globals.lectura_fichero_configuracion(fichero_actual))
    print ("actual")

if not MB_C.is_open():
    MB_C.open()

trama_ref = MB_C.read_holding_registers(11,8) #Lectura trama referencias
trama_1 = MB_C.read_holding_registers(0,119) #Lectura trama completa
MB_C.write_single_register(1,Modo_funcionamiento) #Escritura del modo de funcionamiento
trama_bool = MB_C.read_coils(0,6) #Lectura de los booleanos
'''
Actualizamos las variables locales que gestionan el actualización de las variables
locales del DMC
'''
update_all_data, update_qp, save_config, load_config, load_config_i, update_DMC = (
    globals.lectura_trama_bool(trama_bool,update_all_data, update_qp, save_config,
    load_config, load_config_i, update_DMC))
```

Código 17: Configuración del protocolo Modbus y lectura de los ficheros de configuración del QDMC

```
DMC_offline() #Generación de las matrices del QDMC

u_ant = np.zeros((n_entradas,1)) #Inicialización del vector u_ant

GrafcetTask() #Ejecución de la gestión del controlador
DMCTask() #Ejecución del controlador QDMC

if n_perturbaciones>0: #Inicialización del vector de perturbaciones
    d_MB = globals.floattoint2(d)
else:
    d_MB = [0,0,0,0,0,0]

while 1:
    if not MB_C.is_open(): #Comprobamos si está abierto el servidor Modbus
            MB_C.open()

    trama_ref = MB_C.read_holding_registers(11,8)
    trama_1 = MB_C.read_holding_registers(0,119)

    Modo_funcionamiento = trama_1[1]

    trama_bool = MB_C.read_coils(0,6)
    update_all_data, update_qp, save_config, load_config, load_config_i, update_DMC =(
        globals.lectura_trama_bool(trama_bool,update_all_data, update_qp, save_config,
        load_config, load_config_i, update_DMC))

    u_MB = globals.floattoint2(U_send)
    y_MB = globals.floattoint2(Y_send)
    ref_MB = globals.floattoint2(Ref_send)

    MB_C.write_multiple_registers(133,u_MB)
    MB_C.write_multiple_registers(119,y_MB)
    MB_C.write_multiple_registers(141,ref_MB)


    updating_indicator=(update_all_data_updating, update_qp_updating, save_config_updating,
     load_config_updating, load_config_i_updating, update_DMC_updating, guardado, disable_L,
      manual_L, auto_L)
    MB_C.write_multiple_coils(6,updating_indicator)

    time.sleep(0.5)

#Paramos el watchdog y cerramos el archivo
os.write(f_watchdog,'V')
os.close(f_watchdog)
```

Código 18: Ejecución del algoritmo DMC, de los bucles de control y gestión del controlador, gestión de los bloques de memoria de Modbus y del watchdog



Flujograma 4: Bucle que gestiona la conexión Modbus

El último bucle temporizado gestiona la comunicación y los bloques de memoria Modbus, realizando la gestión de los datos de la aplicación de control con la interfaz gráfica remota. Para ello, cada 0.5 segundos se realizan las lecturas (*read_holding_registers()* y *globals.lectura_trama_bool()*) y escrituras (*write_multiple_registers()* y *write_multiple_coils()*) [3] en los pertinentes bloques de memoria de Modbus. Finalmente, cuando se detiene la aplicación de control se detiene la función watchdog (*os.write()*) y se cierra el correspondiente archivo (*os.close()*). El funcionamiento básico del bucle está recogido en el flujograma 4.

A continuación se van a explicar las funciones realizadas explícitamente para esta aplicación. En primer lugar se va a comenzar con las funciones que contiene el fichero *funciones_dmc.py*, que son las siguientes:

- **Matriz_T (n, p):** proporciona las matrices T y M necesarias en el algoritmo DMC, a partir del horizonte de control (c) y el horizonte de predicción (p).
- **Matriz_G (c, esc, p):** proporciona la matriz de ganancias del proceso, a partir del horizonte de control (c), la respuesta escalón del sistema (esc) y el horizonte de predicción (p).
- **Matriz_lambda (lambdai, c):** proporciona la matriz diagonal con las lambdas del sistema mediante los valores de lambda (lambdai) y el horizonte de control (c).
- **Matriz_lambda (alfai, c):** proporciona la matriz diagonal con los alfas del sistema mediante los valores de alfa (alfai) y el horizonte de predicción (p).

- **Matriz_H (G, alfa_1, lambda_1):** proporciona la matriz H y el hessiano para el algoritmo DMC, a partir de la matriz de ganancias (G), la matriz de alfas (alfa_1) y lambdas (lambda_1).
- **Matriz_A (c, n_entradas,G):** proporciona una de las matrices necesarias para establecer las restricciones del proceso, a partir del horizonte de control (c), el número de entradas del proceso (n_entradas) y la matriz de ganancias (G).
- **Matriz_B (delta_u_max, delta_u_min, u_max, u_min, y_max, y_min, u_ant, y_free, c, p, n_entradas, n_salidas):** proporciona la matriz que contiene parte de cada una de las restricciones del proceso. Para ello es necesario, las restricciones de las acciones de control (u_max, u_min, delta_u_max y delta_u_min), de las salidas del proceso (y_max y y_min), las acciones de control de la iteración anterior (u_ant), la respuesta libre del sistema (y_free), el horizonte de control (c), el horizonte de predicción (p), el número de entradas (n_entradas) y salidas (n_salidas) del proceso.
- **Matriz_C (G, alfa, e):** proporciona la matriz que contiene parte de las restricciones del optimizador QP. Se obtiene a partir de la matriz de ganancias (G), matriz de alfas (alfa) y el error (e) entre las referencias y las salidas del proceso.
- **Esc (entrada, salida, n, id_process):** proporciona la respuesta escalón para cada entrada y salida del proceso, a partir de la entrada y salidas correspondientes, de los instantes en que se estabiliza la respuesta del sistema (n) y el identificador del proceso que se desea cargar (id_process).
- **C_datos (A_y, B_y, n_var, var):** conversión de unidades a partir de una recta de conversión, formada por la pendiente (A_y) y el término independiente (B_y). Asimismo, es necesario el número de variables a convertir (n_var) y el vector con dichas variables (var). Devuelve un vector con la correspondiente conversión.
- **C_datos_inv (A_y, B_y, n_var, var):** exactamente lo mismo que la función anterior pero de forma inversa.

Para ver el código de programación de estas funciones con un mayor nivel de detalle, este se puede observar en las siguientes partes del mismo (19 a 23).

```
def matriz_T(n,p):
    M=np.eye(n-1)
    zero = np.zeros((n-1,1))
    M = np.concatenate((zero,M),1)
    one = M[[n-2]]
    M = np.concatenate((M,one))
    T = M
    if p>n:
        for x in range(0,(p-n)):
            T=np.concatenate((T,one))
    else:
        T=M
    return M, T

def matriz_G(c,esc,p):
    esc = np.transpose(esc)
    n=len(esc)
    g1 = np.zeros((c,1))
    G=np.transpose(g1)
    if p>n:
        for x in range(0,p-n):
            esc=np.append(esc,esc[n-2])
    for i in range(0,p):
        end = len(g1)
        g1=np.append(esc[i],g1[0:end-1])
        G=np.vstack((G,g1))
    G=np.delete(G,0,axis=0)
    return G

def matriz_lambda(lambdai,c):
    l=lambdai*np.eye(c)
    return l

def matriz_alfa(alfai,p):
    a=alfai*np.eye(p)
    return a

def matriz_H(G,alfa_1,lambda_1):
    hessian=np.dot(np.dot(G.T,alfa_1),G)+lambda_1
    H = np.dot(np.dot(np.linalg.inv(hessian),G.T),alfa_1)
    return H, hessian
```

Código 19: Funciones del archivo *funciones_dmc.py* (1)

```
def matriz_A(c,n_entradas,G):
    I=np.eye((c))
    L=np.tril(np.ones((c,c)))
    ceros = np.zeros((4*n_entradas*c,1))
    M, N= G.shape
    unos=-1*np.ones((2*M,1))
    I_1=I
    L_1=L
    for i in range(0,n_entradas-1):
        I_1=block_diag(I_1,I)
        L_1=block_diag(L_1,L)
    I_2=-1*I_1
    L_2=-1*L_1
    G_2=-1*G
    A_1=np.vstack((I_1,I_2,L_1,L_2,G,G_2,np.zeros((1,c*n_entradas))))
    A_2=np.vstack((ceros,unos,-1))
    A=np.hstack((A_1,A_2))
    return A
```

Código 20: Función del archivo *funciones_dmc.py* (2)

```
66  def matriz_B(delta_u_max,delta_u_min,u_max,u_min,y_max,y_min,u_ant,y_free,c,p,n_entradas,
67      n_salidas):
68      u_max_m=0
69      u_min_m=0
70      delta_u_max_m=0
71      delta_u_min_m=0
72      y_max_m=0
73      y_min_m=0
74      for i in range(0,n_entradas):
75          delta_u_max_t=np.dot(np.ones((c,1)),delta_u_max[i])
76          delta_u_min_t=-1*(np.dot(np.ones((c,1)),-1*(delta_u_min[i])))
77          u_max_t=np.dot(np.ones((c,1)),(u_max[i]))-u_ant[i]
78          u_min_t=-1*(np.dot(np.ones((c,1)),(u_min[i])))+u_ant[i]
79          delta_u_max_m=np.vstack((delta_u_max_m,delta_u_max_t))
80          delta_u_min_m=np.vstack((delta_u_min_m,delta_u_min_t))
81          u_max_m=np.vstack((u_max_m,u_max_t))
82          u_min_m=np.vstack((u_min_m,u_min_t))
83      delta_u_max_m=np.delete(delta_u_max_m,0,axis=0)
84      delta_u_min_m=np.delete(delta_u_min_m,0,axis=0)
85      u_max_m=np.delete(u_max_m,0,axis=0)
86      u_min_m=np.delete(u_min_m,0,axis=0)
87      for i in range(0,n_salidas):
88          y_max_t=np.dot(np.ones((p,1)),(y_max[i]))-y_free[i*p:((i+1)*p)]
89          y_min_t=(-1*(np.dot(np.ones((p,1)),(y_min[i]))))+y_free[i*p:((i+1)*p)]
90          y_max_m=np.vstack((y_max_m,y_max_t))
91          y_min_m=np.vstack((y_min_m,y_min_t))
92      y_max_m=np.delete(y_max_m,0,axis=0)
93      y_min_m=np.delete(y_min_m,0,axis=0)
94      B=np.vstack((delta_u_max_m, delta_u_min_m, u_max_m, u_min_m, y_max_m, y_min_m,0))
95      return B
96
97  def matriz_C(G,alfa,e):
98      c1=np.transpose(-1*(np.dot(np.dot(np.transpose(G),alfa),e)))
99      a = np.zeros((1,1))
100     c=np.hstack((c1,a))
101     return c
102
103 def vector_u(u_ant,deltau,c,n_entradas):
104     deltau_u = deltau[0]
105     for i in range(0,n_entradas-1):
106         deltau_u_n=deltau[((i+1)*c)][0]
107         deltau_u=np.vstack((deltau_u,deltau_u_n))
108     u_vector = u_ant + deltau_u
109     return u_vector, deltau_u
```

Código 21: Funciones del archivo *funciones_dmc.py* (3)

```
152 def c_datos(A_y,B_y,n_var,var): #Voltios a unidades
153     U_c =[]
154     for x in range(0,n_var):
155         U = A_y[x]*var[x] + B_y[x]
156         U_c.append(U)
157     return U_c
158
159 def c_datos_inv(A_y,B_y,n_var,var): #Unidades a voltios
160     U_c =[]
161     for x in range(0,n_var):
162         U = (1/A_y[x])*(var[x] - B_y[x])
163         U_c.append(U)
164     return U c
```

Código 22: Funciones del archivo *funciones_dmc.py* (4)

```
112  def esc(entrada,salida,n,id_process):
113      esc=[]
114      entrada = entrada + 1
115      salida = salida + 1
116      nombre_fichero = 'step_p%d_y%d_u%d.txt'% (id_process,salida,entrada)
117      archivo = open(nombre_fichero,"r")
118      parametros = archivo.read()
119      parametros = str(parametros)
120      archivo.close()
121      parametros = parametros.split("\n")
122      longitud = len(parametros)
123      parametros = parametros[0:longitud-1]
124      for x in range (0,longitud-1):
125          dato = parametros[x]
126          dato = dato.replace(',','.',1)
127          dato = float(dato)
128          esc.append(dato)
129      esc = esc[0:n]
130      return esc
```

Código 23: Función del archivo *funciones_dmc.py* (5)

El segundo fichero que contiene funciones programadas para esta aplicación es *Globals.py*. Las funciones que contiene son las siguientes:

- **Guardado_configuración_actual ():** función que almacena en un fichero de texto toda la información que se le introduce a la misma. Esta almacena el nombre y el valor de cada una de estas variables.
- **Lectura_fichero_configuración (nombre_fichero):** función que obtiene la información de los parámetros de configuración de la aplicación de un fichero de texto (nombre_fichero) y los almacena en sus correspondientes variables.
- **Lectura_trama_inicial (trama_1):** función que devuelve cada una de las variables de la aplicación a partir de la lectura de la trama recibida (trama_1) de los bloques de memoria de Modbus.
- **Lectura_trama_var_float (trama, n_bits):** función que devuelve los valores de una variable de tipo flotante en un vector, a partir del fragmento de la trama Modbus (trama) y del número de bits que se desea leer (n_bits).
- **Int2tofloat (vector):** función que convierte los dos enteros procedentes de los bloques de memoria de Modbus que se encuentra en un vector (vector) a flotante y lo almacena en otro vector.
- **Floatto2int (vector):** lo mismo que la función anterior pero de forma inversa.
- **Selección_i_o (vector):** cambia el orden de las entradas o salidas de la TAD, vía software.
- **Lectura_trama_bool (trama, update_all_data, update_qp, save_config, load_config, load_config_i, update_DMC):** función que actualiza las variables locales que gestionan el automatismo que gestiona el controlador y la actualización de las variables del mismo a partir de la información de Modbus (trama).

Para ver el código de programación de estas funciones con un mayor nivel de detalle, este se puede observar en las siguientes partes del mismo (24 a 28).

```
#Escritura del fichero de la configuracion actual
def guardado_configuracion_actual(Id_process, Modo_funcionamiento, n_u, n_y, n_d, U_manual,
 Horizonte_control, Ts, Ref, A_y, B_y, A_d, B_d, alfa_d, lambda_d, delta_u, U_max, U_min,
 Y_max, Y_min, A_u, B_u):

    archivo_escritura = open ("Configuracion_actual.txt","w")
    print (Modo_funcionamiento)
    archivo_escritura.write("Id_process=%.f\n" % Id_process)
    archivo_escritura.write("Modo_funcionamiento=%.f\n" % Modo_funcionamiento)
    archivo_escritura.write("n_u=%.f\n" % n_u)
    archivo_escritura.write("n_y=%.f\n" % n_y )
    archivo_escritura.write("n_d=%.f\n" %  n_d)
    archivo_escritura.write("U_1_manual=%.2f\n" %  U_manual[0])
    archivo_escritura.write("U_2_manual=%.2f\n" % U_manual[1] )
    archivo_escritura.write("U_3_manual=%.2f\n" % U_manual[2])
    archivo_escritura.write("U_4_manual=%.2f\n" %  U_manual[3])
    archivo_escritura.write("Horizonte_control=%.f\n" % Horizonte_control)
    archivo_escritura.write("Ts=%.2f\n" %  Ts)
    archivo_escritura.write("Ref_1=%.2f\n" % Ref[0])
    archivo_escritura.write("Ref_2=%.2f\n" %  Ref[1])
    archivo_escritura.write("Ref_3=%.2f\n" % Ref[2])
    archivo_escritura.write("Ref_4=%.2f\n" %  Ref[3])
    archivo_escritura.write("A_y1=%.4f\n" %  A_y[0])
    archivo_escritura.write("A_y2=%.4f\n" %  A_y[1])
    archivo_escritura.write("A_y3=%.4f\n" %  A_y[2])
    archivo_escritura.write("A_y4=%.4f\n" %  A_y[3])
    archivo_escritura.write("B_y1=%.4f\n" %B_y[0])
    archivo_escritura.write("B_y2=%.4f\n" %B_y[1])
    archivo_escritura.write("B_y3=%.4f\n" %B_y[2])
    archivo_escritura.write("B_y4=%.4f\n" %B_y[3])
```

Código 24: Función del archivo *globals.py* (1)

```
#Lectura del fichero txt que contiene la configuracion
def lectura_fichero_configuracion(nombre_fichero):
    archivo = open(nombre_fichero,"r")
    parametros = archivo.read()
    parametros = str(parametros)
    archivo.close()
    parametros = parametros.split("\n")
    for elemento in parametros:
        elemento = elemento.split("=")
        #Almacenamos el valor deseado en el diccionario
        lista[elemento[0]] = float(elemento[1])

    Id_process = lista['Id_process']
    Modo_funcionamiento = lista['Modo_funcionamiento']
    n_u = lista['n_u']
    n_y = lista['n_y']
    n_d = lista['n_d']
    U_1_manual = lista['U_1_manual']
    U_2_manual = lista['U_2_manual']
    U_3_manual = lista['U_3_manual']
    U_4_manual = lista['U_4_manual']
    U_manual = ([U_1_manual,U_2_manual,U_3_manual,U_4_manual])

    Horizonte_control = lista['Horizonte_control']
    Ts = lista['Ts']
    Ref_1 = lista['Ref_1']
    Ref_2 = lista['Ref_2']
    Ref_3 = lista['Ref_3']
    Ref_4 = lista['Ref_4']
    Ref = ([Ref_1,Ref_2,Ref_3,Ref_4])
```

Código 25: Función del archivo *globals.py* (2)

```
195  def lectura_trama_inicial(trama_1):
196      Id_process = trama_1[0]
197      Modo_funcionamiento = trama_1[1]
198      n_u = trama_1[2]
199      n_y = trama_1[3]
200      n_d = trama_1[4]
201      U_1_manual = ([trama_1[5],0])
202      U_2_manual = ([trama_1[6],0])
203      U_3_manual = ([trama_1[7],0])
204      U_4_manual = ([trama_1[8],0])
205      U_manual = ([U_1_manual,U_2_manual,U_3_manual,U_4_manual])
206      #Conversión de enteros a flotante
207      U_manual = int2tofloat(U_manual)
208      Horizonte_control = trama_1[9]
209      Ts = trama_1[10]
210      Ref_1 = ([trama_1[11], trama_1[12]])
211      Ref_2 = ([trama_1[13], trama_1[14]])
212      Ref_3 = ([trama_1[15], trama_1[16]])
213      Ref_4 = ([trama_1[17], trama_1[18]])
214      Ref = ([Ref_1,Ref_2,Ref_3,Ref_4])
215      Ref = int2tofloat(Ref)
```

Código 26: Función del archivo *globals.py* (3)

```
308  def lectura_trama_var_float(trama,n_bits):
309      variable=[]
310      for x in range(0,n_bits,2):
311          variable_2 = ([trama[x], trama[x+1]])
312          variable.append(variable_2)
313      variable = int2tofloat(variable)
314      return variable
315
316  def int2tofloat(vector):
317      f=0
318      longitud = len(vector)
319      for i in range(0,longitud):
320          aux = vector[i]
321          f_2 = pack('>HH',aux[0],aux[1])
322          f_1 = unpack('>f',f_2)
323          f=np.hstack((f,f_1))
324      f=np.delete(f,0,axis=0)
325      return f
326
327  def floattoint2(vector):
328      int_3=[]
329      longitud = len(vector)
330      for i in range(0,longitud):
331          f=pack('>f',vector[i])
332          int_2 = unpack('>HH',f)
333          for j in range(0,2):
334              int_3.append(int_2[j])
335      return(int_3)
336
337  def seleccion_i_o(vector):
338      vector_ordenado = []
339      vector_ordenado.append(vector[2])
340      vector_ordenado.append(vector[3])
341      vector_ordenado.append(vector[0])
342      vector_ordenado.append(vector[1])
343      return vector_ordenado
```

Código 27: Funciones del archivo *globals.py* (4)

```
345  def lectura_trama_bool(trama,update_all_data, update_qp, save_config, load_config,
346   load_config_i, update_DMC):
347      if (trama[0]==1):
348          update_all_data = trama[0]
349      if (trama[1]==1):
350          update_qp = trama[1]
351      if (trama[2]==1):
352          save_config = trama[2]
353      if (trama[3]==1):
354          load_config = trama[3]
355      if (trama[4]==1):
356          load_config_i = trama[4]
357      if (trama[5]==1):
358          update_DMC = trama[5]
359      return update_all_data, update_qp, save_config, load_config, load_config_i, update_DMC
```

Código 28: Función del archivo *globals.py* (5)

6. BIBLIOGRAFÍA

[1] Scipy (2018) *Librerías para Python.* Obtenido el 25 de Mayo de 2018 de https://www.scipy.org/

[2] Cvxopt (2018) *Librerías del optimizador QP para Python.* Obtenido el 15 de Junio de 2018 de https://cvxopt.org/

[3] PyModbusTCP (2017) *Librerías del Modbus TCP para Python.* Obtenido el 27 de Mayo de https://pymodbustcp.readthedocs.io/en/latest/

[4] Github (2015) *Librerías del convertidor MCP3424 para Python.* Obtenido el 4 de Junio de 2018 de https://github.com/kleebaum/bayeosraspberrypi/blob/master/bayeosraspberrypi/mcp3424.py

[5] Pinteric (2015) *Librerías del convertidor MCP4728 para Python.* Obtenido el 4 de Junio de 2018 de http://www.pinteric.com/data/MCP4728.py

[6] Github (2014) *Librerías del convertidor MCP23008 para Python.* Obtenido el 5 de Junio de 2018 de https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/blob/legacy/Adafruit_MCP230xx/Adafruit_MCP230xx.py

DESARROLLO DE UN DISPOSITIVO BASADO EN RASPBERRY PI PARA EL CONTROL MULTIVARIABLE DE PROCESOS MEDIANTE TÉCNICAS DMC:

PRESUPUESTO DEL PROYECTO

ÍNDICE:

ÍNDICE DE TABLAS:

1. INTRODUCCIÓN

En esta parte de la memoria se va a recoger el coste de la realización del proyecto. Se va a incluir desde la mano de obra de cada uno de los actores que participan en el proyecto hasta todo el software y hardware empleado durante el desarrollo del mismo. El presupuesto tiene como objeto el montaje y puesta en marcha de un prototipo completamente funcional.

Este presupuesto se ha dividido en doce unidades de obra que describen todo el trabajo realizado. Las unidades de obras son las siguientes:

- **Investigación preliminar:** se refiere a la parte de búsqueda de información de cada una de las partes que comprenden el proyecto, como el aprendizaje de los programas a emplear o la elección de cada uno de los componentes de la TAD.
- **Diseño de la circuitería de la TAD:** comprende el diseño de los circuitos de cada una de las partes que forman la TAD.
- **Diseño de la placa de circuito impreso de la TAD:** se refiere al diseño integro de la PCB.
- **Fabricación y montaje de la TAD:** comprende desde la fabricación de la PCB hasta el montaje de los componentes electrónicos en la misma.
- **Programación de la interfaz gráfica:** comprende la programación íntegra de la interfaz gráfica remota en LabVIEW.
- **Programación del HIL:** comprende la programación de la aplicación que simula el proceso a control mediante un Hardware-in-the-Loop.
- **Ensayo del proceso elegido mediante Matlab:** comprende el control del proceso mediante simulación con Matlab.
- **Programación de la aplicación servidor Modbus:** se refiere a la programación de la aplicación que inicia la comunicación mediante el protocolo Modbus TCP en la RPi.
- **Programación de la aplicación de control:** comprende la programación de todas las partes que conforman dicha aplicación mediante Python en la RPi.
- **Ensayo completo de todo el sistema de control implementado:** comprende la prueba de todas las aplicaciones diseñadas desde la interfaz gráfica y monitorización de todas las variables de interés del proceso, hasta la aplicación de control implementada en la RPi.
- **Documentación del proyecto:** se refiere a la generación de todos los documentos que recogen el trabajo realizado durante el desarrollo del proyecto.
- **Montaje y mantenimiento de los equipos:** comprende el trabajo realizado por los técnicos de laboratorio para que todos los equipos se encuentren siempre en perfecto funcionamiento.

2. CUADROS DE PRECIOS UNITARIOS

En este apartado se va a obtener los precios unitarios de cada una de las unidades que han contribuido a realizar el proyecto. Estas se pueden clasificar en dos grandes tipos, la primera referente a la mano de obra y la segunda, a la parte hardware/software.

Para la primera parte se ha considerado dos unidades, un ingeniero recién graduado en la titulación Máster Universitario en Ingeniería Industrial (MUII) y un técnico de laboratorio. Para realizar el cálculo del coste por hora, se ha tenido en cuenta 220 días laborables al año y una jornada laboral de 8 horas diarias (ver Tabla 1).

Tabla 1: Precios unitarios referentes a la mano de obra

Costes	Graduado en el Máster Universitario en Ingeniería Industrial (€) (MO.1)	Técnico de laboratorio (€) (MO.2)
Salario base (220 días/ año)	24000.00	16000,00
Seguridad social	9600.00	4800,00
Pagas extra	3000.00	2000,00
Total al año	36600.00	22800,00
Coste por hora (€ /hora)	20.80	12,95

La segunda parte se ha dividido en una parte software, que representa todos aquellos programas necesarios para desarrollar y emplear la aplicación diseñada, y en una parte hardware, donde se describe todo el material empleado. Para la parte software se ha considerado que todos los programas poseen una licencia anual y que se han amortizado durante la jornada laboral (220 días laborables al año y jornada laboral de 8 horas diarias) (ver Tabla 2).

Tabla 2: Precios unitarios de la parte software del proyecto

Código	Unidad	Número de unidades	Precio(€)	Horas de funcionamiento	Coste (€/h)
MAT.S1	NI LabVIEW 2018 (+ librerías)	1.00	1080.00	1760.00	0.60
MAT.S2	DesignSpark 8.0	1.00	0.00	1760.00	0.00
MAT.S3	Matlab 2017b	1.00	800.00	1760.00	0.44
MAT.S4	NI Multisim 13.0	1.00	685.00	1760.00	0.38
MAT.S5	Python 2.7	1.00	0.00	1760.00	0.00
MAT.S6	Windows 10	1.00	145.00	1760.00	0.08
MAT.S7	Raspbian	1.00	0.00	1760.00	0.00
MAT.S8	Microsoft office 2017	1.00	69.00	1760.00	0.04

Para la parte del hardware, se ha considerado una vida útil de 4 años, teniendo en cuenta los mismos días que para unidades de mano de obra (220 días laborables al año y jornada laboral de 8 horas diarias) (ver Tabla 3).

Tabla 3: Precios unitarios de la parte hardware del proyecto

Código	Unidad	Número de unidades	Precio(€)	Horas de funcionamiento (h)	Coste (€/h)
MAT.H1	Ordenador portátil	1.00	1190.00	7040.00	0.17
MAT.H2	Multímetro	1.00	85.00	7040.00	0.01
MAT.H3	Estación de soldadura	1.00	435.60	7040.00	0.06
MAT.H6	TAD NI 6001	2.00	432.00	7040.00	0.06
MAT.H7	Raspberry Pi	1.00	39.99	7040.00	0.01

Por otro lado, los precios unitarios referentes al hardware desarrollado, se han considerado el coste de fabricar un prototipo (ver Tabla 4).

Tabla 4: Precios unitarios del hardware desarrollado

Código	Unidad	Número de unidades	Coste (€/unidad)
MAT.H4	PCB prototipo (Modelo NAKED)	1.00	27.00
MAT.H5	Componentes electrónicos	1.00	36.83

3. CUADROS DE PRECIOS DESCOMPUESTOS

En este apartado se van a exponer los costes de cada una de las unidades de obra que recogen el desarrollo íntegro del proyecto.

Tabla 5: Unidad de obra nº1

Investigación preliminar					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	35.00	20.80	727.84
MAT.H1	h	Ordenador portátil	35.00	0.17	5.92
MAT.S6	h	Windows 10	35.00	0.08	2.82
				Importe total	736.58

Tabla 6: Unidad de obra nº2

Diseño de la circuitería de la TAD					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	25.00	20.80	519.89
MAT.H1	h	Ordenador portátil	25.00	0.17	4.23
MAT.S6	h	Windows 10	25.00	0.08	2.01
MAT.S4	h	NI Multisim 13.0	25.00	0.38	9.51
				Importe total	535.64

Tabla 7: Unidad de obra nº3

Diseño de la placa de circuito impreso de la TAD					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	60.00	20.80	1247.73
MAT.H1	h	Ordenador portátil	60.00	0.17	10.14
MAT.S6	h	Windows 10	60.00	0.08	4.83
MAT.S2	h	DesignSpark 8.0	60.00	0.00	0.00
				Importe total	1262.70

Tabla 8: Unidad de obra nº4

Fabricación y montaje de la TAD					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	2.00	20.80	41.59
MO.2	h	Técnico de laboratorio	10.00	12.95	129.55
MAT.H2	h	Multímetro	2.00	0.01	0.02
MAT.H3	h	Estación de soldadura	8.00	0.06	0.50
MAT.H4	u	PCB de pruebas (Modelo NAKED)	1.00	27.00	27.00
MAT.H5	u	Componentes electrónicos	1.00	36.83	36.83
				Importe total	235.49

Tabla 9: Unidad de obra nº5

Programación de la interfaz gráfica					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	65.00	20.80	1351.70
MAT.H1	h	Ordenador portátil	65.00	0.17	10.99
MAT.S6	h	Windows 10	65.00	0.08	5.24
MAT.S1	h	NI LabVIEW 2018	65.00	0.60	39.00
				Importe total	1406.93

Tabla 10: Unidad de obra nº6

Programación del HIL					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	5.00	20.80	103.98
MAT.H1	h	Ordenador portátil	5.00	0.17	0.85
MAT.S6	h	Windows 10	5.00	0.08	0.40
MAT.S1	h	NI LabVIEW 2018	5.00	0.60	3.00
				Importe total	108.23

Tabla 11: Unidad de obra nº7

Ensayo del proceso elegido mediante Matlab					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	3.50	20.80	72.78
MAT.H1	h	Ordenador portátil	3.50	0.17	0.59
MAT.S6	h	Windows 10	3.50	0.08	0.28
MAT.S3	h	Matlab 2017b	3.50	0.44	1.56
				Importe total	75.21

Tabla 12: Unidad de obra nº8

Programación de la aplicación servidor Modbus					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	4.00	20.80	83.18
MAT.H1	h	Ordenador portátil	4.00	0.17	0.68
MAT.S6	h	Windows 10	4.00	0.08	0.32
MAT.S5	h	Python 2.7	4.00	0.00	0.00
MAT.S7	h	Raspbian	4.00	0.00	0.00
MAT.H7	h	Raspberry Pi	4.00	0.01	0.02
				Importe total	84.20

Tabla 13: Unidad de obra nº9

Programación de la aplicación de control					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	100.00	20.80	2079.55
MAT.H1	h	Ordenador portátil	100.00	0.17	16.90
MAT.S6	h	Windows 10	100.00	0.08	8.06
MAT.S5	h	Python 2.7	100.00	0.00	0.00
MAT.S7	h	Raspbian	100.00	0.00	0.00
MAT.H7	h	Raspberry Pi	100.00	0.01	0.57
				Importe total	2105.07

Tabla 14: Unidad de obra nº10

Ensayo completo de todo el sistema de control implementado					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	24.00	20.80	499.09
MAT.H1	h	Ordenador portátil	24.00	0.17	4.06
MAT.S6	h	Windows 10	24.00	0.08	1.93
MAT.S5	h	Python 2.7	24.00	0.00	0.00
MAT.S7	h	Raspbian	24.00	0.00	0.00
MAT.S1	h	NI LabVIEW 2018	24.00	0.60	14.40
MAT.H6	h	TAD NI 6001	24.00	0.06	1.47
MAT.H7	h	Raspberry Pi	24.00	0.01	0.14
				Importe total	521.09

Tabla 15: Unidad de obra nº11

Documentación del proyecto					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.1	h	Graduado en Máster Universitario Ingeniería Industrial	100.00	20.80	2079.55
MAT.H1	h	Ordenador portátil	100.00	0.17	16.90
MAT.S6	h	Windows 10	100.00	0.08	8.06
MAT.S8	h	Microsoft office 2017	100.00	0.04	3.83
				Importe total	2108.34

Tabla 16: Unidad de obra nº12

Montaje y mantenimiento de los equipos					
Código	**Unidades**	**Descripción**	**Rendimiento**	**Precio (€)**	**Importe (€)**
MO.2	h	Técnico de laboratorio	24.00	12.95	310.91
				Importe total	310.91

4. CUADROS DE MEDICIONES Y PRESUPUESTO

En este apartado se va determinar la cantidad necesaria de cada una de las unidades de obra descritas en el apartado anterior y el presupuesto final del proyecto aplicando los porcentajes de gastos generales, beneficio industrial e IVA.

Tabla 17: Cuadro de mediciones

Unidades de obra	Coste (€/u)	Medición (u)	Importe (€)
Investigación preliminar	736.58	1.00	736.58
Diseño de la circuitería de la TAD	535.64	1.00	535.64
Diseño de la placa de circuito impreso de la TAD	1262.70	1.00	1262.70
Fabricación y montaje de la TAD	235.49	1.00	235.49
Programación de la interfaz gráfica	1406.93	1.00	1406.93
Programación del HIL	108.23	1.00	108.23
Ensayo del proceso elegido mediante Matlab	75.21	1.00	75.21
Programación de la aplicación servidor Modbus	84.20	1.00	84.20
Programación de la aplicación de control	2105.07	1.00	2105.07
Ensayo completo de todo el sistema de control implementado	521.09	1.00	521.09
Documentación del proyecto	2108.34	1.00	2108.34
Montaje y mantenimiento de los equipos	310.91	1.00	310.91

Tabla 18: Presupuesto de ejecución del proyecto

Presupuesto de ejecución del proyecto	Importe (€)
Desarrollo de un dispositivo basado en Raspberry Pi para el control multivariable de procesos mediante técnicas DMC	9490.38

Tabla 19: Presupuesto base de licitación

Presupuesto de ejecución del proyecto	9490.38
13% Gastos generales	1233.75
6% Beneficio industrial	569.42
Presupuesto de ejecución por contrata	11293.56
21% IVA	2371.65
Presupuesto base de licitación	13665.20

El presupuesto de ejecución del proyecto asciende a la cantidad en Euros:

NUEVE MIL CUATROCIENTOS NOVENTA CON TREINTA Y OCHO

El presupuesto de ejecución por contrata asciende a la cantidad en Euros:

ONCE MIL DOSCIENTOS NOVENTA Y TRES CON CINCUENTA Y SEIS

El presupuesto base de licitación asciende a la cantidad en Euros:

TRECE MIL SEISCIENTOS SESENTA Y CINCO CON VEINTE