Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería electrónica industrial y automática.

Laboratorio remoto. Control de un sistema Bola Balancín de forma remota vía web.

Autor:

Sánchez Santiago, Mario

Tutor:

García Ruíz, Francisco Javier

Departamento de ingeniaría de sistemas y automática.

Valladolid, febrero de 2022.

RESUMEN

En el presente proyecto se pretende realizar una aplicación para controlar un sistema bola-viga de forma remota. Esta aplicación deberá servir a los alumnos de la Escuela para hacer prácticas sin estar presentes en el laboratorio.

En el sistema se debe controlar la posición en la que se encuentra una bola que puede rodar sobre una viga lineal. Para ello se modificará la inclinación de dicha viga mediante un servomotor.

El manejo de la planta se realiza desde una Raspberry Pi, con la que podremos trabajar conectando una pantalla, un teclado y un ratón o utilizando el escritorio remoto. Gracias a la segunda opción y a la instalación de una cámara que reproduzca en streaming, se podrá trabajar con el sistema a distancia.

Se ha desarrollado además un sitio web con información sobre el sistema y desde el cual se puede descargar la aplicación para graficar los datos.

PALABRAS CLAVE

PID, Raspberry Pi, Comunicación, Sensor, Servomotor.

ABSTRACT

The objective of this project is to develop an application for controlling a beam-ball system remotely. This application must serve to the students of the school to do the practices without being presently in the laboratory.

The system must control the position of a ball which can move along a linear beam. The way of doing it will be varying the inclination of the beam, thing which is done with a servomotor.

The management of the plant is done from a Raspberry Pi. To work with it we could connect a screen, a keyboard and a mouse or we could use the remote desktop. Thanks to the second option and to the installation of a camera that reproduces streaming, it will be possible to work with the system remotely.

In addition, a web site has been developed with information about the system and the possibility of downloading the application to plot data.

KEY WORDS

PID, Raspberry Pi, Comunication, Sensor, Servomotor.

ÍNDICE.

1. Introducción.

El presente proyecto se plantea como un sistema que pueda ser empleado por alumnos de ingeniería para el estudio de los controles PID. Para ello se dispone de una planta que está formada por un sistema bola-viga. Este sistema consiste en una viga horizontal sobre la que reposa la bola, según se varíe la inclinación de la viga, la bola se desplaza hacia uno u otro lado.

La variación de la inclinación de la viga se realiza, como veremos en detalle más adelante, mediante un servomotor que mueve un mecanismo biela manivela. El sistema incluye además dos sensores de posición que nos permiten saber en todo momento cuál es la posición de la bola.

El objetivo final es, que indicando una posición objetivo, por ejemplo el centro de la viga, el sistema sea capaz de llevar la bola a ese punto lo más rápidamente posible y que permanezca en él.

Para ello se implementará un control PID, que es un control en lazo cerrado. Mediante la medición de la posición real de la bola y comparándola con la posición deseada, genera una señal para el actuador que permitirá llevar la bola al lugar deseado. Esta señal generada depende de tres términos, el proporcional, el integral y el derivativo, cada uno de los cuales tiene su propia función y características.

La lectura de los sensores, el manejo del servomotor y la implementación del control PID se realizan en una tarjeta Raspberry Pi, que es un minicomputador que incluye un procesador, memoria RAM, periféricos de entrada y salida, pines de conexión..., en ella se llevará a cabo la aplicación principal. Esta aplicación ofrece además un entorno gráfico con el que se puede interactuar para cambiar los valores del PID y comenzar y parar los diferentes procesos.

Debido a las limitaciones de la tarjeta, ha sido necesario sacar de ella ciertos procesos para liberarla así de una carga computacional excesiva. Estos procesos han sido la visualización del vídeo en straming, que se hace a gracias a un servidor web, y la realización de la gráfica con los datos obtenidos, que se hace en una aplicación que se ejecuta en el ordenador del alumno.

La comunicación entre la aplicación principal y la destinada a graficar se puede realizar de dos maneras, mediante puerto serie en caso de estar trabajando presencialmente y mediante sockets si se trabaja a distancia.

Para poder descargar la aplicación graficar se ha desarrollado una página web a la que podrán acceder los alumnos. En este sitio web podrán

encontrar además información relativa al proyecto y a su modo de funcionamiento.

Tras esta breve introducción, en el capítulo 2 enumeraremos los objetivos del proyecto.

En el capítulo 3 se realizará una descripción más detallada sobre la planta y el funcionamiento de un sistema bola-viga.

En el capítulo 4 veremos uno por uno cuáles son los elementos que componen el sistema.

A continuación, en el quinto capítulo, se explicarán ciertos fundamentos teóricos que han sido necesarios para realizar los programas.

Una vez hecho esto, podemos en el capítulo 6 explicar el funcionamiento de la aplicación principal y la aplicación graficar.

En el capítulo 7 se incluye una guía de instalación, en la que se explica cómo acceder a la Raspberry y como descargar y ejecutar la AppGraficar, y un manual de uso para familiarizar a los alumnos en cómo se debe trabajar con el sistema.

En el capítulo 8 se habla acerca de la página web creada para brindar una base de apoyo a los alumnos el trabajar con la práctica.

En el capítulo 9 se muestra cómo funciona el sistema y se prueban diferentes ajustes del control PID, para mostrar el efecto de cada uno de los términos. Además se encuentran unos valores que permiten a la bola alcanzar la posición deseada rápidamente y sin error estacionario, incluso ante perturbaciones.

Para acabar, en el apartado 10 se encuentran las conclusiones finales sobre la realización del proyecto y las posibles líneas futuras que tiene.

Y por último, el apartado 11 recoge la bibliografía que se ha utilizado durante la realización del proyecto

2. Objetivos.

1 – Realizar el control en lazo cerrado de la posición de la bola sobre la viga.

2 – Crear una aplicación desde la que se introduzcan los parámetros del control PID.

3 – Crear una aplicación que permita graficar la situación del sistema, es decir, la posición objetivo de la bola y la real.

4 – Comunicar ambas aplicaciones mediante puerto serie y sockets para que se pueda trabajar presencial o remotamente.

5 – Encontrar los parámetros adecuados para el PID.

6 – Crear una página web desde la que se pueda acceder a la aplicación.

7 – Realizar un vídeo en directo de la planta para que se pueda manipular a distancia.

3. Descripción del sistema.

El sistema bola-viga con el que vamos a tratar es un caso de estudio típico dentro de la ingeniería de sistemas. Se compone, tal y como se observa en la siguiente imagen, de una barra sobre la que la que se encuentra una bola. Dicha barra tiene un punto fijo y otro que puede subir o bajar mediante un mecanismo biela-manivela o similar.

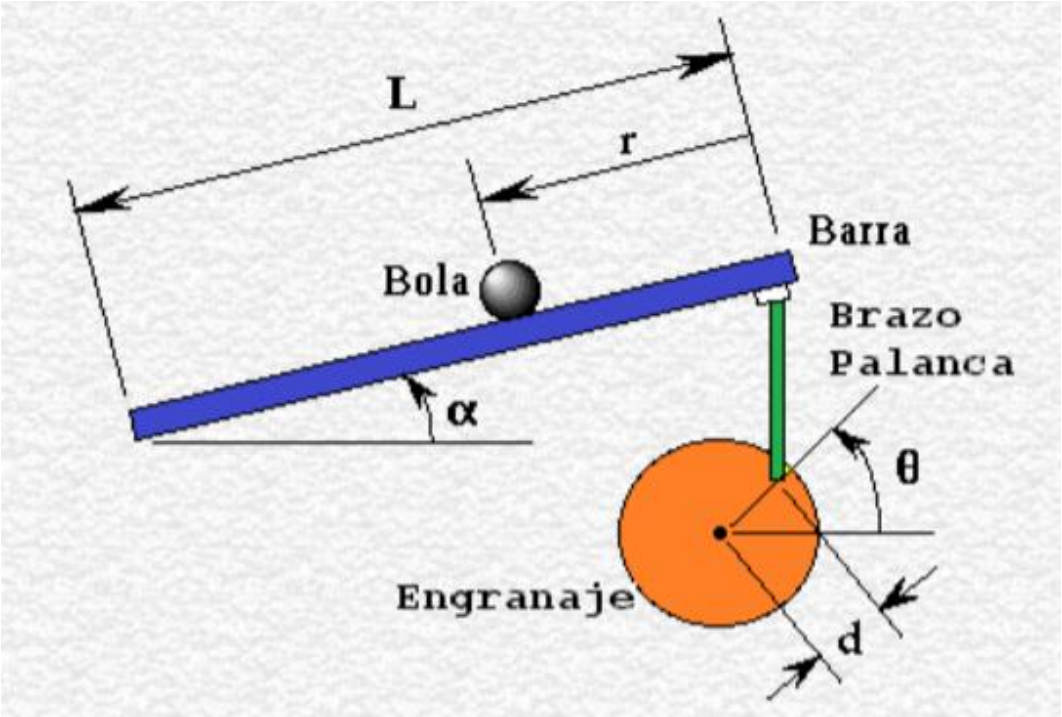


Figura 3.1 - Esquema sistema bola-viga.

Mediante la acción de subir o bajar este segundo punto de apoyo de la barra, podemos variar la inclinación de la misma, y de esta manera haremos que la bola se desplace, debido a la gravedad, hacia la derecha, la izquierda o permanezca quieta.

El sistema biela-manivela que hemos mencionado estará controlado por un servomotor, por lo que según el movimiento que haga este motor variará la inclinación de la barra.

Para indicar a dicho motor cuál es el movimiento que debe hacer en cada momento se va a implantar un control PID, que calculará cuál debe ser la salida partiendo del dato de la posición actual de la bola. Es necesario por tanto medir cuál es esa posición, tarea que realizaremos gracias a dos sensores de distancia situados uno a cada lado de la barra, de forma que su rango de medida cubra toda la longitud requerida.

Además de estos elementos, la maqueta cuenta también con una fuente de alimentación, una seta de emergencia, una cámara para retransmitir vídeo... componentes cuyas características y funciones veremos en el siguiente apartado.

En la siguiente imagen se muestra la maqueta real ya construida:

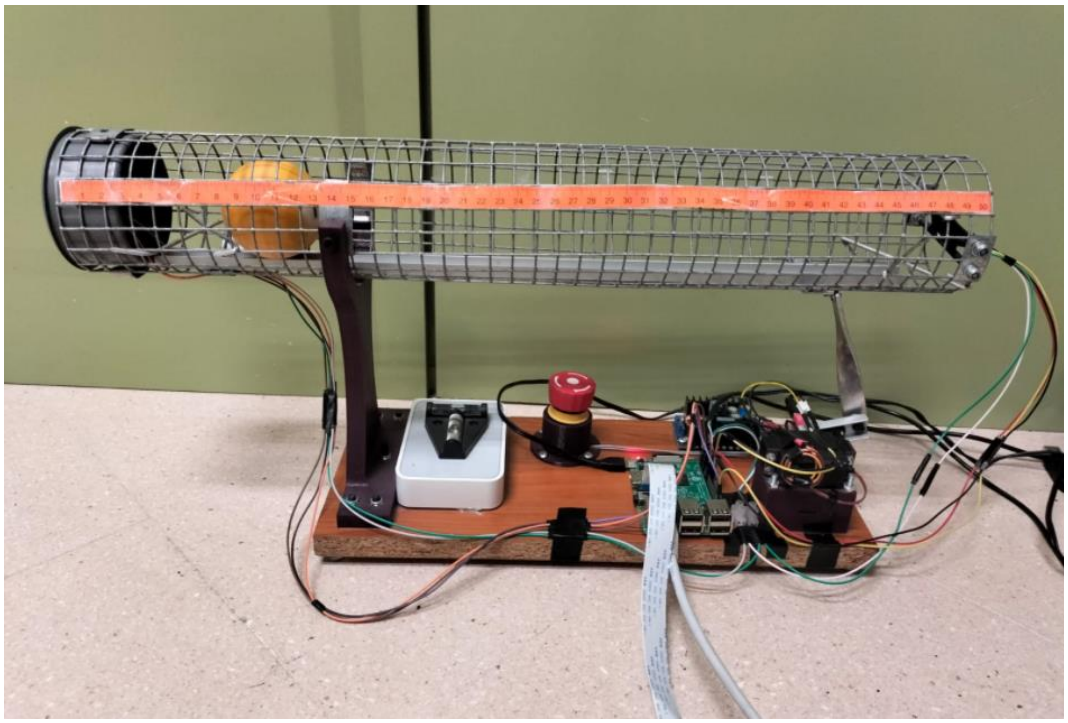


Figura 3.2 - Imagen maqueta real del sistema.

4. Componentes de la planta.

En este apartado hablaremos de los componentes del sistema más allá de los elementos mecánicos ya comentados como son la viga, el sistema biela-manivela, la base y los soportes. Lo haremos uno por uno comentando cuáles son sus características y la función que desempeñan dentro del equipo.

Además de los elementos mecánicos comentados, como son la viga, el sistema biela-manivela, los soportes, la base… el sistema necesita para realizar el control los elementos de los que hablamos a continuación:

4.1 Raspberry Pi

4.1.1 Introducción

La Raspberry Pi es una placa de ordenador simple desarrollada por la Fundación Raspberry Pi, fundación que nace en Reino Unido en el 2009 con el objetivo de animar a los niños a aprender electrónica e informática en las escuelas. Más allá de su objetivo inicial, es ampliamente utilizada también para el desarrollo de proyectos de electrónica, debido a su pequeño tamaño y bajo precio. Además, conectando a la placa una pantalla, teclado, ratón y demás periféricos se convierte en un ordenador que puede realizar multitud de tareas básicas [1].

La placa Raspberry Pi está compuesta por un SoC, CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, ranura microSD para almacenamiento, conectividad de red, reloj, conexiones para periféricos de bajo nivel y muchos elementos más. Sus características van variando dependiendo del modelo que elijamos, en la siguiente tabla se muestran las principales características de los modelos existentes:

	SoC	FRECUENCIA DE RELOJ	RAM	PUERTOS USB	ETHERNET	WIRELESS/ BLUE-TOOTH
Raspberry Pi Modelo A+	BCM2835	700Mhz	512MB	1	No	No
Raspberry Pi Modelo B+	BCM2835	700Mhz	512MB	4	Si	No
Raspberry Pi 2 Modelo B	BCM2836 o BCM2837	900Mhz	1 GB	4	Si	No
Raspberry Pi 3 Modelo B	BCM2837	1200Mhz	1 GB	4	Si	Sí

Raspberry Pi 3 Modelo B+	BCM2837B0	1500Mhz	1 GB	4	Gigabit Ether-net over USB 2.0	Sí
Raspberry Pi Zero	BCM2835	1000Mhz	512MB	1	No	No
Raspberry Pi Zero W	BCM2835	1000Mhz	512MB	1	No	Sí

El modelo elegido para el proyecto ha sido la Raspberry Pi 3 modelo B, que como vemos en la tabla es una de las más completas. Cuenta con 4 puertos USB y uno RJ45 para conexión mediante Ethernet, una CPU BCM2897, el procesador funciona a una velocidad de 1,2 GHz, tiene 1Gb de memoria RAM y la posibilidad de conectarse a ella de forma inalámbrica.

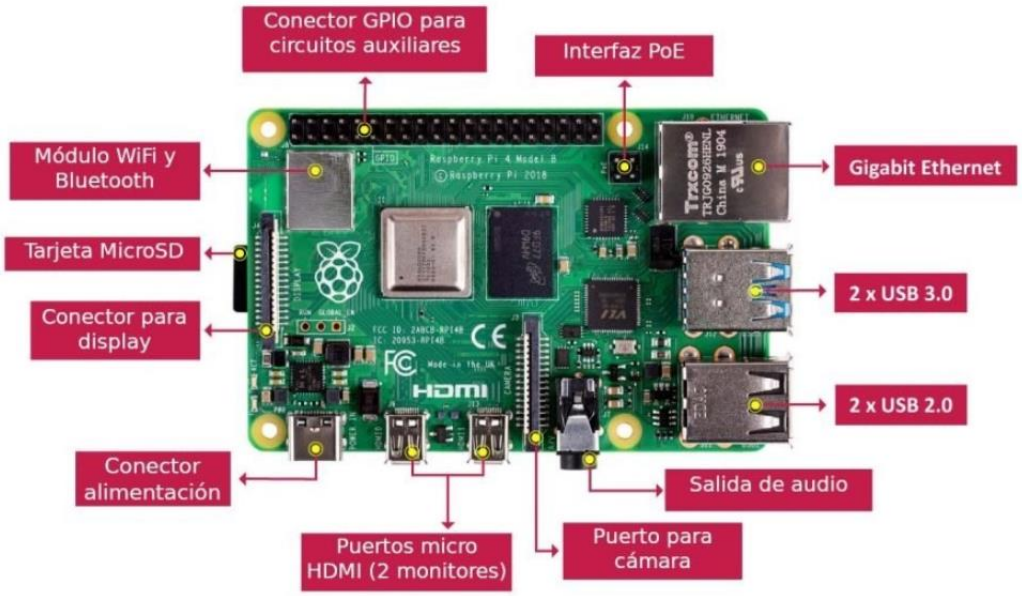


Figura 4.1 - Componentes Raspberry Pi.

4.1.2 Sistema Operativo

La ranura SD de la que dispone sirve para introducir una tarjeta microSD que es donde se instala el sistema operativo. El sistema operativo que utiliza Raspberry Pi es Raspbian, un S.O. libre y gratuito basado en Linux, en su distribución Debian, y que además está optimizado para ser usado en el hardware de Raspberry Pi [2].

Hay dos versiones de Raspbian que podemos instalar en nuestra tarjeta:

- Raspbian Pixel: Es una versión completa que incluye entorno gráfico, es decir, un escritorio con menús, ventanas, fondos de pantalla, iconos,

etc. Es el adecuado si se va a emplear la tarjeta como ordenador de sobremesa.

- Raspbian Lite: Es una versión reducida de la anterior en la que se prescinde del entorno gráfico. Esto la hace mucho menos pesada, pero a cambio hay que interactuar con ella a través de la línea de comandos. Por tanto, es adecuada si se tienen ciertos conocimientos básicos sobre Linux y se quiere utilizar la Raspberry Pi como servidor.

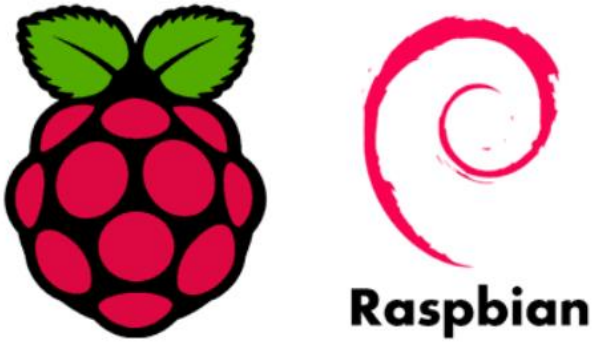


Figura 4.2 - Logos Raspberry y Raspbian.

En un primer momento se trató de usar la segunda opción, puesto que usando la Raspberry como servidor podíamos ahorrar bastante carga computacional si prescindíamos del escritorio. Sin embargo, posteriormente se cambió a la versión completa de Raspbian con escritorio. Esto fue porque de esta forma podíamos desarrollar una aplicación en la Raspberry Pi que tuviera un entorno gráfico con el que el alumno pudiera interactuar, lo cual la hace mucho más agradable e intuitiva a la hora de trabajar con ella.

Es importante señalar que la Raspberry Pi no tiene botón de encendido o apagado, por lo que para trabajar con ella basta con alimentarla o no a través de su puerto microUSB mediante una fuente de alimentación de 5V y 2,5A.

4.1.3 Puerto GPIO.

Los pines GPIO dotan a la Raspberry de capacidades especiales que no tiene un ordenador, y es que al igual que los que tiene Arduino, te permiten conectar dispositivos electrónicos que posteriormente controlaremos mediante códigos en algún lenguaje como Python. Su número varía entre los diferentes modelos, la Raspberry Pi 3 modelo B tiene 40 pines, los cuales tienen funciones específicas [3].

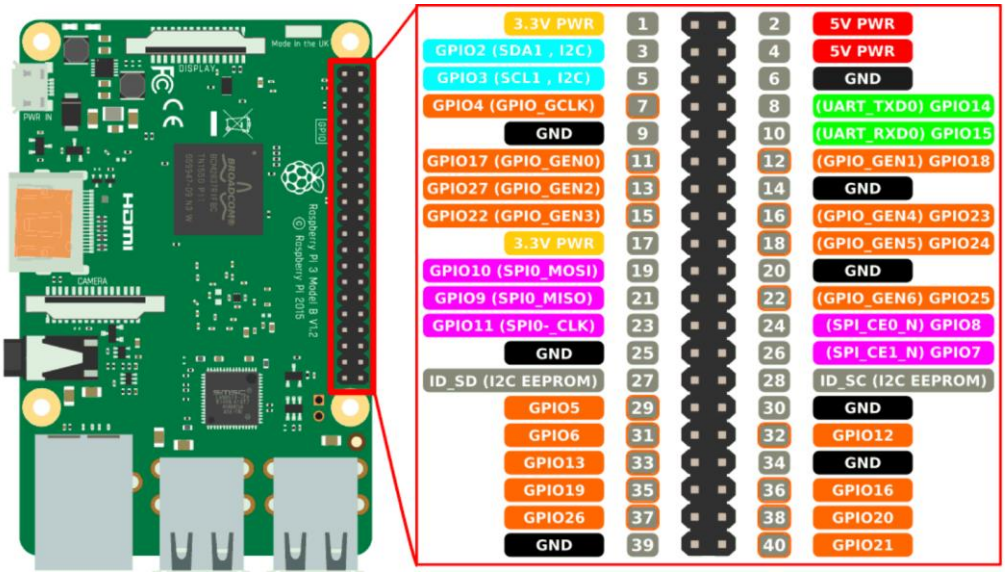


Figura 4.3 - Pines Raspberry Pi

En la imagen superior se puede ver la función de cada pin, según su color su funcionalidad es:

- Amarillo: Alimentación a 3,3V.
- Rojo: Alimentación a 5V.
- Naranja: Son los pines GPIO, que significa Entrada/Salida de Propósito General. Son pines digitales, lo que quiere decir que solo manejan dos niveles de tensión, el bajo que son 0V y el alto que son 3,3V. Se pueden usar para interactuar con otros dispositivos electrónicos, para escribir o para leer datos, según se configuren como salida o entrada respectivamente. Algunos de ellos tienen además la capacidad de funcionar como PWM.
- Negros: Son pines que están conectados a toma tierra. Son muy importantes debido a que todos los dispositivos que conectemos deben tener una tierra común.
- Azules: Son pines GPIO que tienen además la capacidad de funcionar como bus I2C. El pin 3 funciona como línea de datos (SDA) y el pin 5 como línea de reloj (SCL). Estos son los dos pines principales para la comunicación I2C, y a ellos se pueden conectar múltiples dispositivos, no obstante, de ser necesario, hay más pares de pines GPIO que podrían configurarse como líneas SCL y SDA configurando ciertos archivos.
- Morados: Son pines GPIO con la funcionalidad adicional de permitir la comunicación mediante el protocolo SPI. Esta utilidad no la usaremos en nuestro proyecto.
- Verdes: Son también pines de entrada salida de propósito general que tienen además una función única especial. En este caso, son los pines

que permiten llevar a cabo la comunicación serial. El pin 8 funciona como TX, para la transmisión de datos, y el pin 10 como RX, para la recepción.

- Grises: Son pines reservados, normalmente se usan para la comunicación I2C con una EEPROM.

Para referirse a los pines hay dos nomenclaturas distintas. La nomenclatura BOARD se refiere a los pines por su posición física en la tarjeta. En la imagen superior se corresponde con los números en gris. La nomenclatura BCM se refiere a los pines por su número de "Broadcom SOC channel" y no se corresponden con la anterior nomenclatura. Por ejemplo, el GPIO 5 según BCM es el pin 29 en la nomenclatura BOARD.

Otros factores a considerar son que los pines funcionan a 3,3V, por lo que si queremos conectar dispositivos que funcionen a 5V es necesario un adaptador de voltajes. Además, son todos digitales, por lo que si se quieren manejar señales analógicas como por ejemplo para leer un sensor, será necesario también un dispositivo auxiliar.

Por último, es importante tener en cuenta que los pines de la Raspberry no tienen ningún tipo de protección, por lo que un mal uso de los mismos podría dañar la placa.

4.2 Adaptador USB-TTL.

Como explicaremos más adelante cuando hablemos de la comunicación serial, el tipo de puerto serie que se ha impuesto a los demás y está presente en todos los ordenadores es el USB. Por tanto, para poder llevar a cabo la comunicación serie entre un ordenador portátil y la Raspberry, va a ser este el puerto que debamos utilizar.

En el lado de la Raspberry la comunicación serial será a través de los pines específicos para esta función, que se comportan con Rx y Tx. Esto hace que para poder conectar el ordenador a los pines necesitemos un adaptador USB-ttl.

En un extremo tenemos un conector USB macho, que, al conectarlo, es visto desde el ordenador como un puerto COM. En el otro extremo tenemos cuatro pines hembra, que se corresponden con Vcc, Gnd, Rx y Tx.

Figura 4.4 - Adaptador USB-ttl

Gnd es la línea de tierra común que hay que conectar a un pin Gnd de la tarjeta. Tx es la línea de escritura de datos, y por tanto debe conectarse al pin de lectura UART de la Raspberry, Rx, que es el pin 10. El cable de Rx debe ir a su vez al Tx de la placa, que es el pin 8. Y por último tenemos el cable Vcc que suministra 5V, de conectarlo al pin 5V de la tarjeta no sería necesaria la fuente de alimentación. Del mismo modo, si ya tenemos la fuente de alimentación conectada no es necesario utilizar este cable.

Para que el adaptador sea compatible con la Raspberry Pi el nivel lógico de los cables Rx y Tx debe ser de 3,3V.

4.3 Sensores VL53L0X.

El sensor VL53L0X es un sensor de distancia por infrarojo láser de última generación, que nos permite medir de forma precisa distancias entre 50 y 2000mm [4]. La función de este sensor en nuestra maqueta será medir la posición de la bola a lo largo de la viga. Para ello estará ubicado en un extremo de la viga, como esta tiene una longitud de 50cm, el rango que alcanza es suficiente.

Figura 4.5 - Sensor VL53L0X.

4.3.1 Características del sensor.

El VL53L0X es un sensor ToF (Time of Flight). Su forma de obtener la medición de distancia es enviar un pulso láser de luz infraroja, esperar a que este rebote en algún objeto y medir el tiempo que ha tardado en volver. Con esta medida y sabiendo la velocidad de la luz podemos obtener la distancia. Esta técnica de medida hace que el sensor no sea vulnerable al ruido u otras condiciones del ambiente como ecos o la reflectancia de los objetos. Por ello, tiene una precisión superior a los sensores de ultrasonidos.

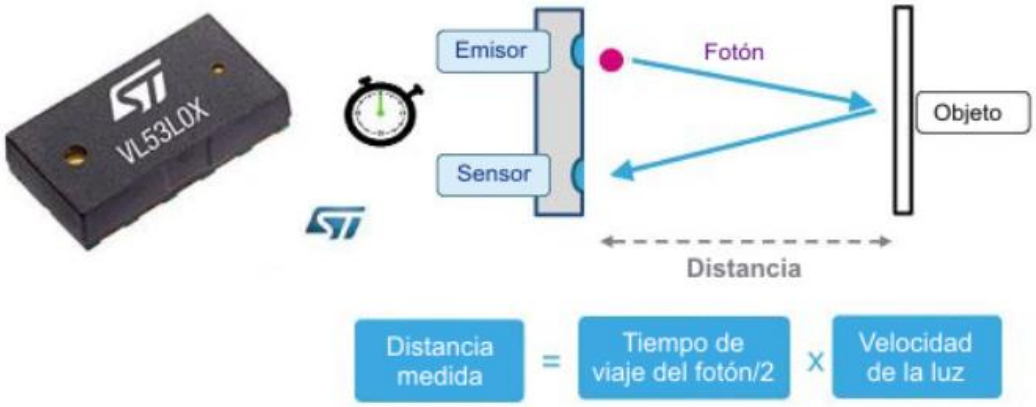


Figura 4.6 - Método de medida de los sensores ToF

.

Sus principales características son [5]:

- Emisor infrarrojo: 940 nm
- Distancia medible: hasta 2000 mm
- Fuente de luz VCSEL (Vertical-cavity surface-emitting laser = Láser de emisión de superficie de cavidad vertical)
- Pequeño tamaño del chip: 4,4 x 2,4 x 1,0 mm
- Medición de distancia rápida y precisa.

- El rango reportado es independiente de la reflectividad del objetivo
- Compensación óptica cruzada integrada avanzada para simplificar la selección del vidrio de cobertura.
- Seguro para el ojo humano.
- Regulador de voltaje integrado en la plaqueta
- Interfaz I2C para control de dispositivos y transferencia de datos
- Pines de entrada salida de uso general Xshutdown (para reinicio) e Interrupt (interrupción)

4.3.2 Conexionado del VL53L0X.

El módulo en el que va integrado el sensor cuenta con seis pines de conexión, cuyas funciones son las siguientes:

- VIN: Es la fuente de alimentación, acepta de 3 a 5 V, por lo que los pines de alimentación de 3,3V de la Raspberry serán suficientes. El chip VL53L0X funciona a 2,8V, por lo que el propio módulo se encarga de convertir el voltaje que llega en este valor.
- 2,8V: Este pin está por si se desea alimentar al módulo directamente con 2,8V, de forma que no tenga que hacer ninguna transformación de voltajes.
- GND: La toma de tierra común que conectaremos con un pin GND de la Raspberry.
- SCL: La línea de reloj para la comunicación I2C.
- SDA: La línea de datos para la comunicación I2C, tanto este como el anterior deben conectarse a su homónimo de la tarjeta.
- GPIO: La función de este pin es indicar si están listos los datos para ser leídos. Puede ser útil cuando se realiza detección continua, aunque no es muy utilizado. Como además nuestro proyecto necesita mediciones discretas, no lo usaremos.
- SHON: Es el pin de apagado/reinicio del sensor. Dispone de un diodo de cambio de nivel para que también se pueda usar con lógica de 3,3V (nuestro caso) o 5V. Por defecto se encuentra a nivel alto y el sensor está encendido, pero cuando le llega un nivel bajo se apaga. Esta función sí que será necesaria en nuestro proyecto

4.3.2 Modos de funcionamiento.

El sensor tiene cuatro modos distintos de funcionamiento:

- Defaolut: Es el modo por defecto del sensor. Tiene un rango máximo de medida de 1200mm, un tiempo entre medidas de 30ms y una precisión aceptable.

- Alta precisión: Este modo nos permite tener una mayor precisión en las medidas tomadas, por el contrario, el tiempo que requiere hacer cada medida es mucho mayor, de 200ms. El rango de alcance sigue siendo 1200mm.
- Largo alcance: Este modo nos permite medir distancias de hasta 2000mm, manteniendo un periodo entre medidas aceptable, de 33ms. Como desventaja tiene que la precisión de las medidas baja respecto a los anteriores modos.
- Alta velocidad: Con este modo pueden obtenerse medidas más seguidas en el tiempo que en los anteriores modos, ya que tiene un periodo de medida de 20ms. Mantiene el rango de medida de 1200mm pero pierde algo de precisión respecto a los casos anteriores.

Analizando las ventajas y desventajas y experimentando con cada modo de funcionamiento se determinó que el que más se adecuaba a nuestros requisitos era el modo por defecto. El modo de largo alcance no era necesario porque tan sol necesitamos medir 500mm. El modo de alta velocidad tampoco porque los 30ms de periodo del modo por defecto son suficientes para los requisitos de control PID. Y por último, el modo de alta precisión se tuvo que descartar debido a su gran tiempo entre medidas, de 200ms, ya que si nos limitaría el tiempo de muestreo para el control.

No obstante, el modo por defecto tiene una precisión aceptable, que podemos ver en la siguiente tabla sacada de su datasheet:

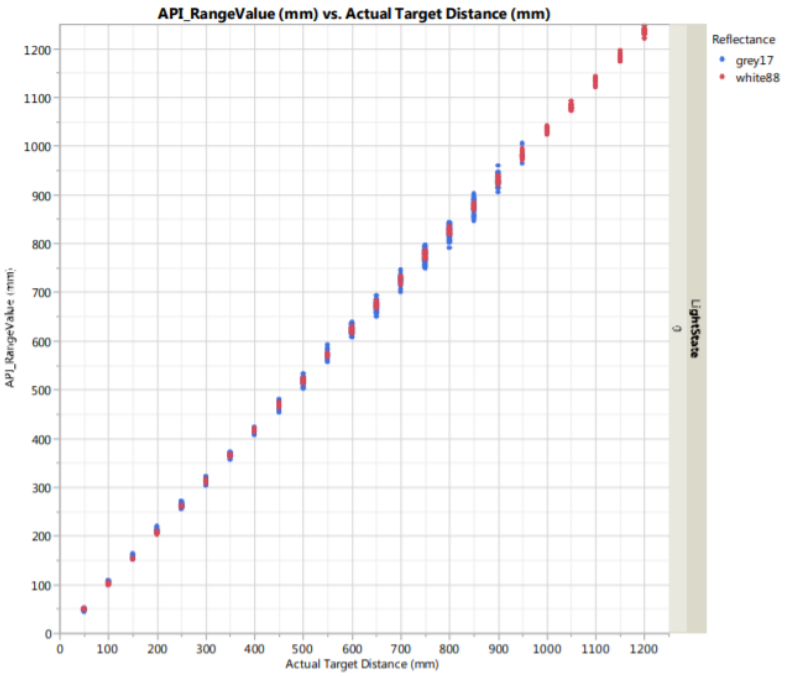


Figura 4.7 - Precisión del sensor frente a la distancia.

4.3.3 Problemas en la obtención de medidas.

Pese a que el rango teórico de medida del sensor era suficiente para nuestros requisitos, durante el montaje y pruebas de la maqueta se vio que no era así. Al colocar el sensor en un extremo de la viga, apuntando hacia dentro del cilindro de alambre que recubre esta, se observó que la máxima distancia que podía medir el sensor eran unos 27cm aproximadamente. Esto se debe a que el área de medición del sensor es de 25º, lo que provoca que a partir de cierta distancia los rayos de luz choquen con las paredes del cilindro e imposibiliten la medida.

Viendo la imposibilidad de medir la longitud entera de la viga con este sensor se trató de probar con el sensor ultrasónico HC-SR04, que tiene un rango de apertura menor, de 15º [6]. Su rango de medición va de 2cm a 450cm, lo que nuevamente es más que suficiente, y su precisión de ±3mm, completamente aceptable. Su técnica de medición es similar a la del VL53L0X pero en vez de infrarojos emplea ultrasonidos para medir el tiempo que tardan en regresar y calcular la velocidad, lo que hace que sea más vulnerable al ruido.

Sin embargo, al hacer las pruebas en la maqueta con este sensor, si bien si se vio que alcanzaba una mayor distancia, a esta aún le quedaban algunos centímetros para cubrir la longitud de la viga por completo. Además, se producían errores en muchas medidas, en las cuales no detectaba la bola, y esto habría sido un problema a la hora de implementar el PID.

Esta prueba sirvió para ver que el problema del ángulo de apertura era común a todos los sensores, y se planteó otra solución al problema usando el VL53L0X, que era el sensor que mejores resultados había dado en cuanto a precisión y robustez en las medidas.

La solución fue colocar dos sensores, uno a cada extremo de la viga, de forma que cada uno de ellos cubriera la mitad más cercana a él, 25 cm de distancia. Para acoplarlos al resto del proyecto, hubo que hacer el programa de tal manera que según en qué mitad de la viga estuviera la bola leyera la medición de un sensor o el otro.

4.3.4 Comunicación con la Raspberry.

Como ya hemos dicho anteriormente, la comunicación entre el módulo del sensor y la Raspberry se realiza mediante I2C. Para poder conectar los dos sensores al mismo bus I2C, será necesario modificar la dirección de uno de estos dispositivos para que esta no coincida en ambos. La dirección que usan por defecto es la 0x29, y en la que pondremos el segundo sensor será la 0x2B.

De esta forma, el conexionado de la Raspberry Pi con los dos sensores sería como se muestra en el siguiente esquema:

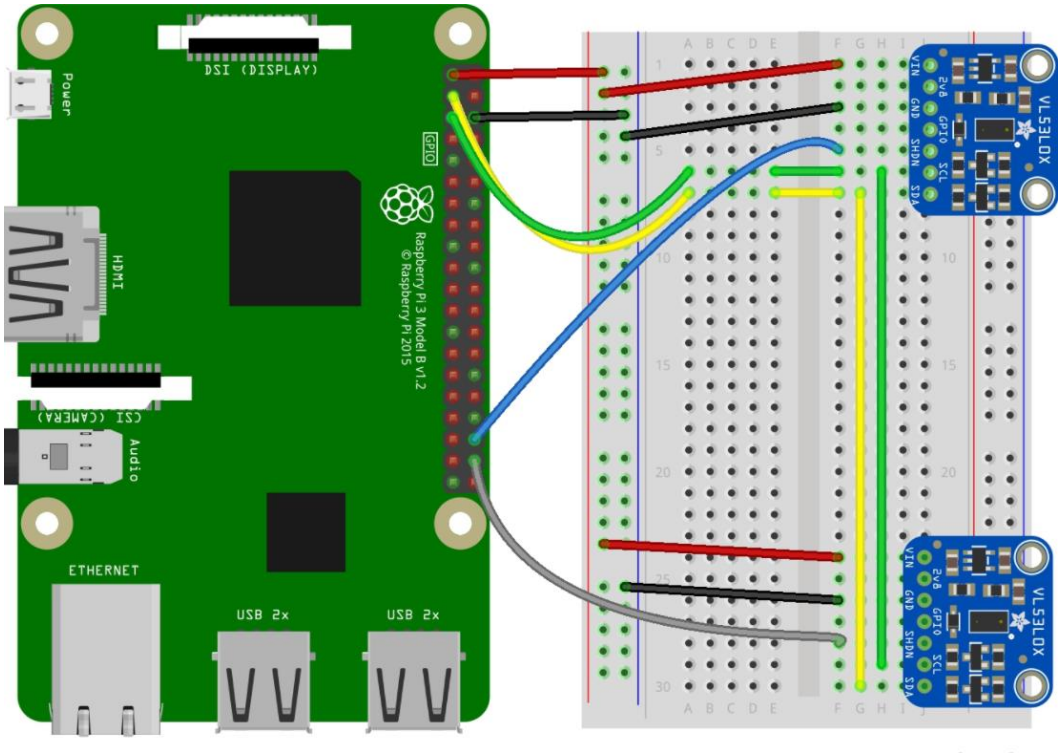


Figura 4.8 - Conexión de los sensores a la Raspberry Pi

Como vemos, la señal de datos y la de reloj de ambos sensores se juntará en el mismo bus e irá a los pines 3 y 5 de la Raspberry respectivamente. No obstante, como los sensores están incorporados a la maqueta y no puede hacerse uso de una protoboard, ha habido que soldar los cables para hacer esta unión.

Para la señal de alimentación y de tierra no ha sido necesario soldar cables puesto que la tarjeta cuenta con más pines de 3,3V y de GND. Por tanto para el primer sensor se ha utilizado el pin 1 para alimentación y el pin 6 para tierra, y en el segundo sensor se ha usado el pin 17 para alimentación y el 14 para tierra.

Por último, como pin de apagado (SHON) se ha configurado el 36 para el sensor uno y el 38 para el sensor dos.

(Para nombrar a los pines se ha utilizado la nomenclatura BOARD)

4.4 Servomotor.

Un servomotor es un actuador rotativo que a diferencia de otros tipos de motores permite un control preciso de la posición angular que alcanza, debido a lo cual es ideal para nuestro proyecto.

Figura 4.9 - Servomotor.

La gran precisión que tiene se debe a su composición, ya que está formado, entre otros componentes, por un motor eléctrico y un sensor que sirve para medir su posición y velocidad. De esta forma el servomotor funciona como un mecanismo en lazo cerrado, cuya retroalimentación es el valor de su posición. La posición final se compara con la deseada, y si hay diferencia se mueve el motor, de ahí su gran precisión [7].

Otra de las características de estos motores es su buena resistencia frente a pares externos, lo cual le permite mantener su posición, aunque esté recibiendo una fuerza externa.

Las partes principales de un servomotor son [8]:

- Un motor eléctrico de corriente continua, que es el encargado de generar el movimiento a través de su eje.
- Un sistema de regulación, formado por engranajes que permiten modificar la velocidad y el par con el que queremos trabajar.
- Un sistema de control que estará formado por el sensor. Sirve para controlar el movimiento del motor mediante el envío de pulsos eléctricos.

- Un potenciómetro (una resistencia eléctrica variable), que conectado al eje central del motor nos permite saber en qué posición angular se encuentra este en todo momento.

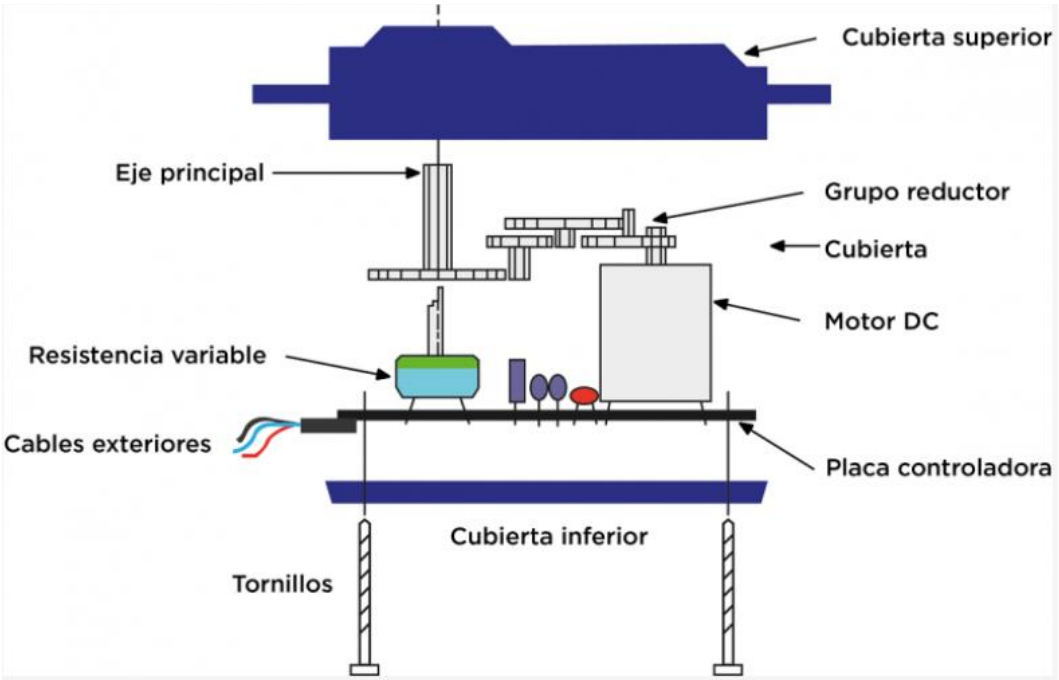


Figura 4.10 – Partes de un servomotor.

El servomotor tiene tres cables, uno de tierra, uno de alimentación (que conectaremos a 5V) y uno de control. El control se hace mediante una señal PWM, que es la que en nuestro caso enviará la Raspberry. El pulso eléctrico de ancho variable o modulación de ancho de pulso (por sus siglas en inglés PWM), consiste en una señal digital periódica, en la que variando el tiempo de cada que está a nivel alto podemos variar su tensión nominal.

La señal tiene una frecuencia de repetición, que para el control del servo será de 50Hz, lo que quiere decir que se enviará un pulso periódicamente cada de 20 milisegundos. Dicho pulso tendrá una duración mínima de unos 0,5ms y una máxima de unos 2,5ms. Estas anchuras de pulso supondrán respectivamente el margen izquierdo y derecho del rango de giro máximo del servomotor. Este rango que se puede alcanzar es aproximadamente de 180º, sin embargo, para este proyecto será suficiente con bastante menos.

En la siguiente imagen se muestra gráficamente como variando la anchura del pulso que se genera mandamos al servomotor a una posición angular u otra:

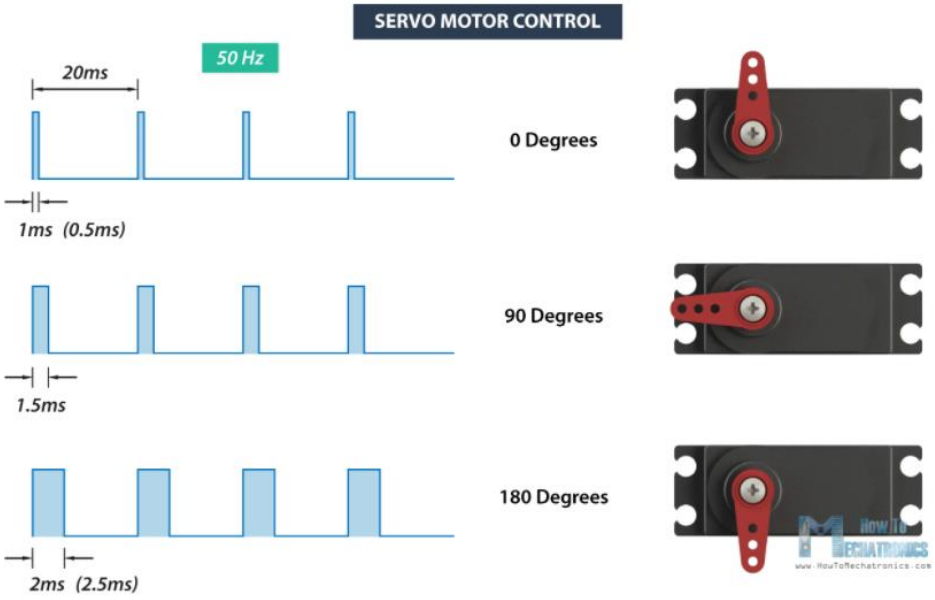


Figura 4.11 – Control Servo mediante PWM.

La anchura de pulso exacta necesaria para cada posición depende del servomotor que estemos utilizando, por lo que es necesario calibrarlo mediante prueba y error para suministrarle la señal adecuada.

El modelo que se ha utilizado en el proyecto es el Power HD Standard Servo 3001HB, que es uno de los más populares debido a sus buenas cualidades y precio económico.

Especificaciones técnicas:

- Modulación analógica.
- 43g de peso.
- Dimensiones: 40.7 x 20.5 x 39.5 mm
- Para máximo (alimentado a 4.8V): 3.5kg·cm
- Velocidad (alimentado a 4.8V): 0.15s/60º

En cuanto a las conexiones, se ha decidido suministrar los 5V de alimentación desde una fuente externa en vez de usar el pin de 5V de la Raspberry Pi. La razón ha sido no sobrecargar en exceso las capacidades de la tarjeta, puesto que los pines tienen una corriente limitada y ya se están usando para alimentar a dos sensores. De esta forma podemos además separar el circuito de potencia de la Raspberry y del motor, lo que nos permite incluir una seta de emergencia. La fuente de alimentación y el adaptador de tensión utilizados los veremos con más detalle en el siguiente apartado.

Para la señal de control se ha utilizado el pin 22 de la Raspberry, configurado para que sea una salida PWM, y para señal de tierra se ha utilizado el pin 22. Esta tierra está conectada al mismo tiempo a la tierra del servomotor y a la de la fuente de alimentación, para que todas tengan un atierra común.

4.5 Fuentes de alimentación.

Para el funcionamiento del proyecto son necesarias dos fuentes de alimentación, una para la Raspberry y otra para el servomotor. Vamos a explicar las dos detalladamente:

4.5.1 Alimentación Raspberry Pi.

La Raspberry Pi 3 utiliza una fuente de alimentación micro USB que suministra 5,1V y 2,5A. Por tanto, para alimentarla podría utilizarse un cargador de móvil común cuya salida sea de este tipo. No obstante, para este proyecto utilizaremos la fuente de alimentación oficial de la Raspberry Pi.

Figura 4.12 - Fuente de alimentación Raspberry Pi

La fuente es la que se ve en la imagen, y viene con distintos adaptadores de modo que se pueda utilizar en distintas partes del mundo. De ellos se ha utilizado el tipo F.

La entrada de la fuente de alimentación es entre 100 y 240 Voltios de corriente alterna a una frecuencia de entre 50 y 60 Hz y una intensidad de 0,5A. Estas son condiciones que cumple cualquier enchufe corriente de un hogar o centro de trabajo.

Figura 4.11 - Especificaciones fuente de alimentación.

4.5.2 Alimentación servomotor.

Como vimos en el apartado anterior, el servomotor necesita una alimentación de 5V. Esta tensión la obtiene en nuestra maqueta mediante una fuente de alimentación de 12V y un adaptador de voltaje que los convierte en 5. La fuente de alimentación es la que se ve en la siguiente imagen, su entrada es entre 110 y 240V de corriente alterna (un enchufe normal) y su salida son 12V a 2A de intensidad. La forma del puerto de salida es un conector DC, también llamado conector de cilindro.

Figura 4.12 - Fuente de alimentación de 12V.

La fuente se conecta a un conector DC hembra, de donde los 12V de corriente continua irán al adaptador de tensión, pero pasando antes por un botón de emergencia. La función de este es poder cortar la alimentación al servomotor en cualquier momento que sea necesario, de esta forma parará de inmediato en caso de haber algún problema que pudiera causar peligros.

El botón está normalmente cerrado, por lo que en su situación normal deja pasar la corriente. De ser pulsado es cuando se cortará, y para volver a dejar pasar la corriente habrá que tirar del botón hacia arriba al tiempo que se gira. El objetivo de este mecanismo es que no pueda reanudarse el movimiento por error al apoyarse sobre el botón.

Figura 4.13 - Seta de emergencia.

Después de esto llegan los 12V al convertidor reductor. Este tiene una entrada donde pueden llegar entre 7 y 32 V DC y una salida regulable mediante dos potenciómetros que puede ser desde 0,8 a 28V. Una vez ajustado para tener 5 V a la salida, ya podemos alimentar desde ahí al servomotor.

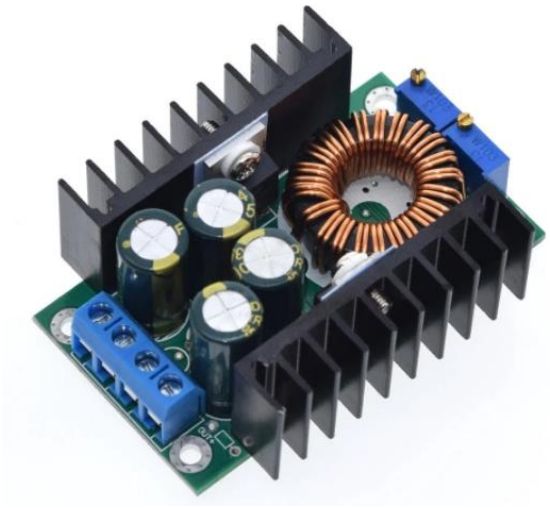


Figura 4.14 - Convertidor reductor de tensión.

4.7 Raspicam.

Raspicam es el módulo oficial de cámara de Raspberry Pi. Existen dos tipos de modelos, el básico y el Raspberry Pi NoIR Módulo de cámara que incorpora la capacidad de tomar fotografías por infrarrojos. Al no ser necesaria esta funcionalidad para nuestro proyecto usaremos el modelo de cámara básico [9].

Este módulo es una cámara de alta calidad con un sensor de 8 megapíxeles que permiten obtener imágenes HD (3280x2464píxeles) y vídeos de alta resolución (1080p máx). Todo esto en un pequeño dispositivo de dimensiones 23.86x25x9mm y 3 gramos de peso.

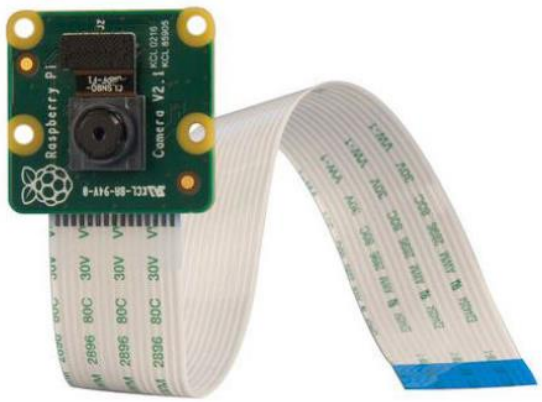


Figura 4.15 - Módulo de cámara de Raspberry Pi

En la imagen además de la cámara vemos el cable de cinta o plano que se utiliza para la conexión. Este cable debe conectarse a uno de los dos puertos CSI (Camera Serial Iterface) de los que dispone la Raspberry Pi 3.

La función de la cámara en el proyecto es reproducir un vídeo de la maqueta en directo, de forma que si un alumno trabaja con el sistema remotamente pueda ver el funcionamiento del mismo.

5. Fundamentos teóricos.

En este apartado vamos a repasar y explicar detalladamente los aspectos teóricos sobre los que se apoya la realización del proyecto.

Primero abordaremos el control PID, que es el fin último de este trabajo. Después seguiremos con la comunicación, donde explicaremos algunas ideas básicas de comunicación y posteriormente nos centraremos en los tres tipos que se utilizan en este proyecto, el puerto serie, la comunicación mediante sockets y el protocolo I2C. Por último, hablaremos sobre el lenguaje de programación Python, que es el que se ha utilizado para desarrollar las dos aplicaciones.

5.1 Fundamentos teóricos del control PID.

Un controlador PID es un dispositivo que permite regular un sistema en lazo cerrado para que alcance una salida deseada. Está compuesto por la suma de tres términos, el proporcional, el integral y el derivativo, cada uno con su función. Su uso comenzó a principios del siglo XX, y pese a los avances tecnológicos que se han sucedido desde entonces el control PID sigue siendo el más ampliamente utilizado en la industria, ya que más del 95% de los controladores son de este tipo. En la siguiente imagen podemos ver el esquema de un PID típico [10]:

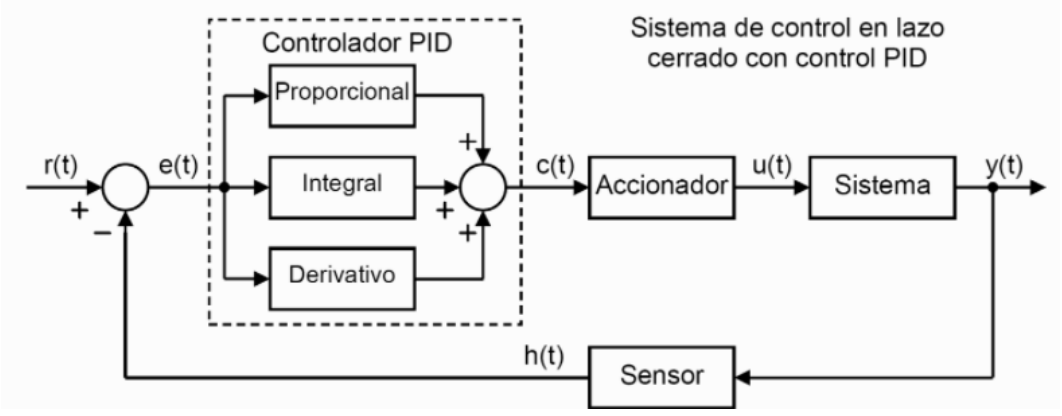


Figura 5.1 - Esquema Control PID.

La señal de entrada o de referencia, en el esquema r(t), es la que se quiere conseguir, y la de salida, en el esquema y(t), es la que tenemos en realidad. Restando la señal de salida a la de referencia obtenemos el error, e(t), esta señal será la que entre al controlador PID. Cuanto mayor sea la señal de error más lejos estaremos de la posición deseada, y cuanto menor sea más

cerca, por lo que el objetivo del control es reducir la señal del error al máximo, y hacerlo lo más rápidamente posible y sin oscilaciones.

Para ello cuenta con las acciones que hemos mencionado, la proporcional, que actúa sobre el error que hay en el momento preciso, la integral, que actúa en base al error acumulado en el pasado y sirve para eliminar el error estacionario, y la derivativa, que se anticipa a lo que va a hacer el sistema y actúa sobre errores futuros. El resultado de estas tres acciones se suma y se adapta a la entrada que necesita el actuador, que en función de esta entrada influirá en el sistema y hará que la salida se acerque más al valor de la entrada de referencia.

La siguiente ecuación muestra el efecto de cada término:

$$c(t) = Kp \cdot e(t) + Ki \cdot \int e(t)dt + Kd \cdot \frac{\partial e(t)}{\partial t}$$

A continuación, procedemos a explicar más detalladamente cada uno de los tres términos y los efectos que tienen en el sistema.

5.1.1 Término proporcional.

Debe su nombre a que su acción es proporcional al valor de la señal de error en el presente. Lo que hace es multiplicar este valor por una constante llamada ganancia proporcional, Kp.

Es la más simple de las tres acciones y su función es reducir el error, pero no puede eliminarlo en estado estacionario. Cuanto mayor sea el valor de la ganancia proporcional más rápida es la respuesta del sistema y más disminuye el error del sistema en régimen permanente. Sin embargo, al aumentar Kp también lo hace la inestabilidad del sistema, por lo que valores demasiado grandes de esta ganancia pueden provocar que la respuesta oscile demasiado o incluso que el sistema sea inestable.

Hay por tanto que encontrar un punto medio que logre una reducción del error y una velocidad de respuesta adecuadas, pero sin llevar el sistema a la inestabilidad.

5.1.2 Término integral.

Para solucionar el problema de que haya un error estacionario que el termino proporcional no puede eliminar se introduce el término integral. Lo que hace esta acción es calcular la integral del error, o lo que es lo mismo, sumar todos los errores pasados. De esta forma, aunque el error que haya sea muy

pequeño, conforme pasa el tiempo y estos se van sumando la acción integral es cada vez mayor. Gracias a esto se consigue eliminar el error en estado estacionario.

Además, este término nos permite también que el sistema sea capaz de responder ante perturbaciones externas, ya que tras sufrir una la compensa y devuelve al sistema a la posición deseada.

La consecuencia negativa de este término es que, al acumular los errores del pasado, crea una cierta inercia, y de ser muy grande puede hacer al sistema inestable. Por ello una vez más hay que encontrar el punto de equilibrio entre sus ventajas e inconvenientes.

La forma de reducir o aumentar el efecto de esta acción es mediante la ganancia integral, Ki.

5.1.3 Término derivativo.

Y por último nos queda la acción derivativa, que como su nombre indica es proporcional a la derivada del error. La derivada es la forma de calcular la velocidad a la que varía el error, la cual es importante porque nos permite prever como actuará el sistema en el futuro y evitar errores antes de que se produzcan.

La situación que se desea evitar con este término es que el valor de la señal de salida se acerque tan rápido a la deseada que, al alcanzarla, pase de largo y se genere error en el sentido contrario. La acción derivativa frena esta velocidad de cambio, y consigue que el sobrepico que se produzca sea mucho menor, consiguiendo así que la salida oscile menos y alcance antes el punto deseado.

Al aumentar la ganancia derivativa, Kd, aumenta la estabilidad del sistema y reduce el tiempo de establecimiento, aunque por contra reduce la velocidad del sistema. Otro efecto perjudicial de la acción derivativa es que incrementa las señales de alta frecuencia (ruido).

5.2 Comunicación.

Para el correcto funcionamiento del sistema va a ser necesario que los distintos dispositivos que lo componen se comuniquen entre ellos de forma efectiva. Hay muchas maneras pasibles en las que se puede llevar a cabo el intercambio de información, por lo que hay que buscar cuál es la que mejor se adapta a nuestras necesidades en cualquier momento.

Pero antes de hablar sobre las formas de comunicación que se utilizan en este proyecto, es necesario aclarar ciertos conceptos generales sobre la comunicación entre dispositivos. Vamos a empezar por los componentes involucrados en el intercambio de datos entre dos dispositivos, que se puede luego extrapolar a una red con un número mayor.

La parte de cada dispositivo que se encarga de la transmisión de datos se llama Estación de Datos, y está formado a su vez por el Equipo Terminal de datos (EDT), cuya misión principal es actuar como fuente o destino de los datos para su almacenamiento y/o procesamiento, y el Equipo Terminal del Circuito de Datos (ETCD), que actúa como emisor o receptor, permitiendo la transmisión a través de la línea [11]. La Línea de Transmisión es el soporte para el intercambio de señales entre los ETCD, y en la práctica impondrá una serie de restricciones a dichas señales. Este canal podrá ser alámbrico o inalámbrico.

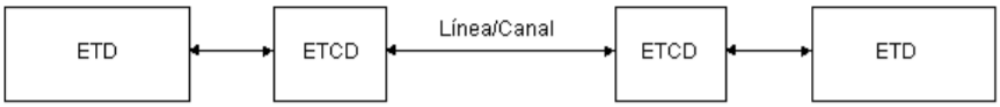


Figura 5.2 – Esquema transmisión de datos.

Además, hay una serie de características de la transmisión que permiten clasificar los diferentes tipos de comunicación, y que debemos comentar:

- Transmisión analógica o digital. La analógica consiste en transmitir la información por medio de ondas de las que cambiamos valores como la amplitud, frecuencia… mientras que la digital consiste en el envío de bits, que son valores lógicos de 0 o 1, nivel de tensión alto o nivel de tensión bajo.
- Trasmisión serie o paralelo. En la transmisión serie los datos se envían de forma secuencial, bit a bit, por un mismo conductor, y en la transmisión paralelo estos se envían de forma simultánea a través de un bus de datos, por lo que es necesario tener un conductor para cada bit.

Con el paso de los años, las formas de comunicación que se han impuesto mayormente han sido aquellas que utilizaban transmisión analógica

y en serie, que es el caso de los tres métodos que se utilizan en este proyecto. Nos centramos por tanto en estos tipos, y pasamos ahora a ver las dos técnicas básicas de funcionamiento que se pueden emplear en la transmisión de datos en serie:

- Transmisión asíncrona: Se envían los caracteres sueltos sin ninguna separación entre ellos. El receptor debe identificar el comienzo y el final de cada uno, para lo cual se utilizan bits de arranque y de parada.
- Transmisión síncrona: Se mantiene un sincronismo durante la transmisión de varios caracteres consecutivos. Los datos a enviar se agrupan en un bloque al que se añade una o dos cabeceras, en las que se incluye información sobre cómo debe interpretar los datos el receptor. Hay diferentes protocolos y cada uno indica un formato concreto que debe tener el mensaje.

Y por último, también dentro de la transmisión serie, tenemos tres posibles formas de comunicación:

- Simplex: En la cual la transmisión es unidireccional, es decir, hay un emisor y un receptor y la información va en un solo sentido.
- Dúplex: Cada extremo puede funcionar como emisor y como receptor, por lo que se utilizan diferentes cables para enviar y recibir.
- Semi-dúplex: Cada extremo puede funcionar como emisor y como receptor, pero mientras uno transmite el otro escucha.

Una vez comentadas las principales características de la transmisión, vamos a repasar brevemente los principales tipos de interfaces para la comunicación serie que existe. Y es que la comunicación entre los ETD de dos dispositivos requiere que haya intercambio de información entre el ETD y el ETCD, que para que se lleve a cabo correctamente debe producirse a través de un interfaz. Dada la importancia que tiene esta conexión, ha sido normalizada por varios organismos.

Uno de los primeros interfaces estandarizados, que fue un éxito y tuvo gran aceptación fue el RS-232-C, conocido como el RS-232. Se describe como un interfaz de propósito general entre Equipo Terminal de Datos y Equipo Terminal del Circuito de Datos, mediante un intercambio de datos binarios en serie. La norma especifica además los niveles lógicos, la velocidad y longitud máxima recomendada entre otras cosas. El conector más extendido era el DB-25, que posteriormente fue simplificado al DB-9 [12].

Otras normas que se fueron desarrollando para tratar de superar las limitaciones de este fueron el RS-442 y el RS-485. No obstante, estos interfaces están ya obsoletos y el más ampliamente utilizado en la actualidad

es el USB (Universal Serial Bus). La interfaz del USB dispone de tan solo cuatro conectores, uno para suministrar tensión de 5V, otro de tierra, y otros dos para el intercambio de datos. Aun así, algunas nuevas versiones cuentan con una quinta patilla. Hay varios tipos de conectores como el USB 2.0, el USB 3.0, el USB tipo C, ...

Vistos estos conceptos básicos, podemos pasar a hablar de los tipos concretos de comunicación que se utilizan en el proyecto, que son el puerto serie, la comunicación mediante sockets y el protocolo I2C, todos ellos como ya habíamos dicho emplean la transmisión de datos digitales en serie.

5.2.1 Puerto serie.

Este será uno de los dos métodos de comunicación que se podrá utilizar para enviar los datos desde la aplicación principal en la Raspberry Pi a la aplicación para graficar en el ordenador que esté usando el alumno. Aunque puertos serie son también los ejemplos mencionados antes y otros muchos más, se asocia este nombre al que veremos aquí en concreto. Esta opción solo es válida cuando se está trabajando con la práctica presencialmente, puesto que la conexión entre ambos dispositivos se realiza mediante cable (el adaptador USB-ttl que vimos en el apartado de los componentes del sistema).

El puerto serie o comunicación serial utiliza trasmisión de datos binarios en serie de forma asíncrona y en full-dúplex. Es una forma de conexión punto a punto, es decir, conecta dos dispositivos entre sí.

El puerto serie cuenta con dos líneas, Rx y Tx, que sirven para leer y para enviar datos respectivamente. Por tanto, al conectar dos dispositivos estas líneas deben estar cruzadas, para que la línea de envío de uno vaya a la de recepción del otro y viceversa. También hay que unir otro cable extra para que ambos tengan una tierra común.

El hardware encargado de controlar la comunicación serie es la UART (Universal Asynchronous Receiver-Transmiter), y entre sus funciones está enviar y leer datos, gestionar interrupciones y calcular tiempos de bit entre otras cosas [13]. La UART nos permite configurar las características de la transmisión, como son el número de bits de parada, si se utiliza bit de paridad o la velocidad de transmisión. Es importante que los dos dispositivos estén configurados igual, puesto que de lo contrario no se entenderán.

La velocidad de transmisión indica cómo de rápido se envían los datos a través de la línea y se mide en baudios (bits por segundo). Este valor determina cuanto tiempo debe mantenerse la línea en alto o en bajo para que se detecte como un bit. Hay muchas velocidades distintas que se pueden elegir,

nosotros utilizaremos 115200 bps, que es la que suelen utilizar la mayoría de microprocesadores.

También debemos indicar la estructura que tendrán los datos que enviemos. La estructura que suelen tener estos consta de un bit de arranque, que produce un cambio en la línea para que el receptor sepa que debe de recibir los datos siguientes, un bloque de datos o carácter, que es la información que queremos enviar, y los bits de parada, que son uno o más en nivel alto e indican el fin del mensaje. Además, antes del bit de parada se puede incluir un bit de paridad, cuya función es comprobar si se ha producido algún error. Su modo de funcionamiento consiste en que con este bit se indica si la suma de los bits del dato a enviar debe ser par o impar, al llegar al receptor este los cuenta, y si la paridad del número de bits recibidos no es la que debiera, detecta un error. Procedería entonces a solicitar el reenvío del dato. Sin embargo, este bit de paridad no es muy utilizado, salvo en casos en los que la comunicación se lleve a cabo en líneas muy ruidosas.

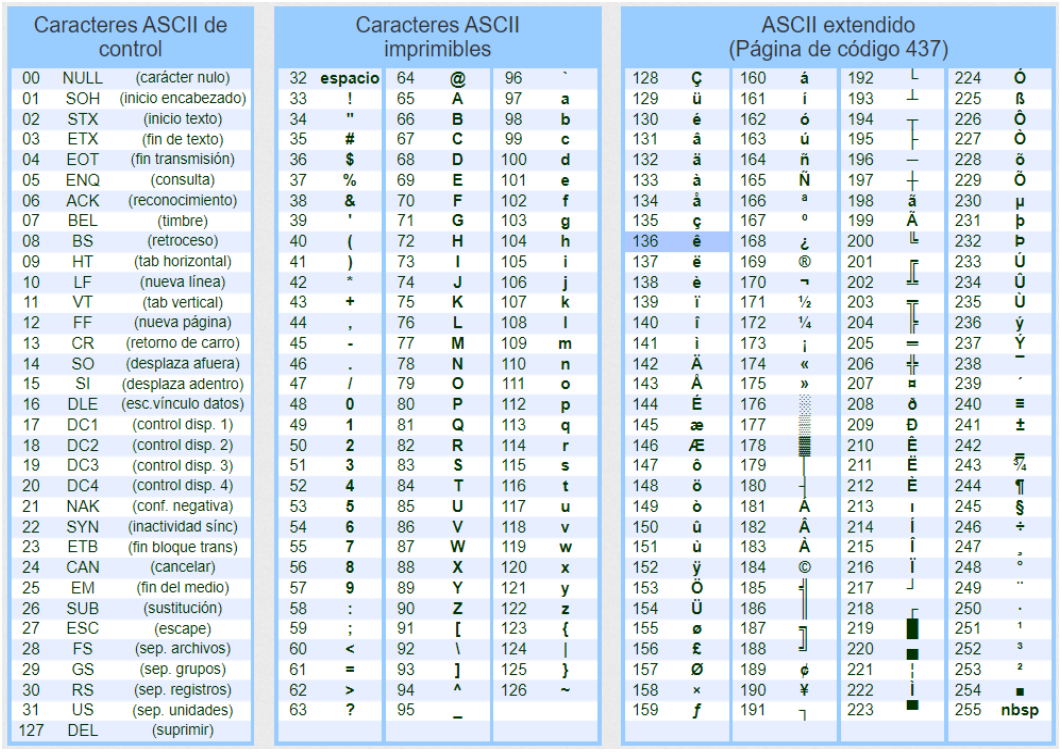
Para indicar la estructura que tendrán los datos se una el formato ABC, donde A es un entero que indica el número de bits de datos, B es una letra que indica la paridad (E: par, O: impar, N: sin paridad) y C indica los bits de parada que se utilizan. Teniendo esto en cuenta, nuestro formato será 8N1.

Por tanto, tenemos los siguientes parámetros de configuración:

- Velocidad de transmisión: 115200 bps
- Número de bits transmitidos: 8
- Bits de stop: 1
- Bits de paridad: Ninguno

El hecho de que en cada bloque de datos vayamos a enviar ocho bits se debe a que utilizaremos la codificación ASCII. ASCII es un código de caracteres basado en el alfabeto latino, que fue creado en 1963 por el comité estadounidense de estándares a partir de los códigos que se utilizaban hasta entonces en telegrafía. El código ASCII originalmente emplea 7 caracteres, pero hay diversas extensiones del mismo que utilizan 8, para incluir caracteres propios de otras lenguas como puede ser el español.

Si empleamos los ocho bits, se tiene 255 posibles caracteres, entre los que hay caracteres de control e imprimibles. Estos últimos incluyen los números, letras tanto en minúscula como en mayúscula, signos de exclamación, operadores y demás. A continuación, se muestra la tabla de cada carácter con su equivalencia [14]:



Caracteres ASCII de control		
00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles					
32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Figura 5.3 - Tabla de los caracteres ASCII.

5.2.2 Comunicación mediante sockets.

Además de mediante puerto serie, tendremos la posibilidad de conectar la Raspberry Pi (AppPrincipal) con el ordenador (AppGraficar) para el envío de los datos de la gráfica mediante sockets a través de la red Ethernet. De esta forma, no será necesario hacer la conexión con un cable y por tanto se podrá trabajar con el equipo remotamente.

Los sockets son un mecanismo de comunicación bidireccional entre procesos que se ejecutan en una misma máquina o en máquinas distintas, se definen como el punto final de una conexión [15]. Un socket se caracteriza por tres atributos, dominio, protocolo y tipo.

Existen diferentes tipos de sockets, los más famosos son:

- Socket de flujo (SOCK_STREAM): orientado a conexión, está asociado al protocolo TCP. Crea un camino entre el origen y el destino, por donde irán los datos, y lo mantiene durante toda la conexión. Son los que utilizaremos nosotros.
- Socket de datagrama (SOCK_DGRAM): no orientado a conexión, basado en el protocolo UDP. EN este caso no se fija un camino, cada paquete irá con por un camino y no se garantiza la recepción secuencial.

El dominio indica el medio de comunicación de la red que se utilizará. Los dos dominios más empleados son:

- AF_UNIX: El cliente y el servidor debe estar en la misma máquina. Cada socket debe tener una dirección única, que será el nombre de un archivo local.
- AF_INET: El cliente y el servidor pueden estar en cualquier máquina de la red Internet y la comunicación se basa en el conjunto de protocolos TCP/IP. La identificación del socket se realiza mediante el par dirección IP de la máquina y número de puerto. Es el que utilizaremos nosotros.

Y por último, el protocolo, utilizaremos el que viene por defecto.

Las funciones del API (Application programming interface) que utilizaremos para crear y controlar los sockets son las siguientes:

- socket(): crea un nuevo socket de un determinado tipo y le asigna recursos del sistema.
- bind(): Se usa en el lado servidor y sirve para asociar al socket a una estructura de dirección de socket, que incluye una dirección IP y número de puerto local.
- listen(): También en el lado servidor, pone el socket en escucha.
- connect(): Se utiliza en el lado cliente para asignar un número de puerto libre al socket y tratar de establecer una conexión TCP.
- accept(): Con esta función el servidor acepta crear una conexión TCP y se forma un socket asociado a los dos pares de direcciones (IP y puerto).
- write(): Se usa para enviar datos de un lado al otro.
- read(): Se usa para leer los datos que han sido enviados desde el otro extremo.
- close(): Cierra la conexión y hace que el sistema libere los recursos que estaban asignados al socket.

En el siguiente esquema podemos ver cómo se usan estas funciones para permitir la comunicación entre el servidor y el cliente. Como podemos observar algunas de estas funciones se usan exclusivamente en uno de los dos lados, y es importante el orden en el que se utilizan.

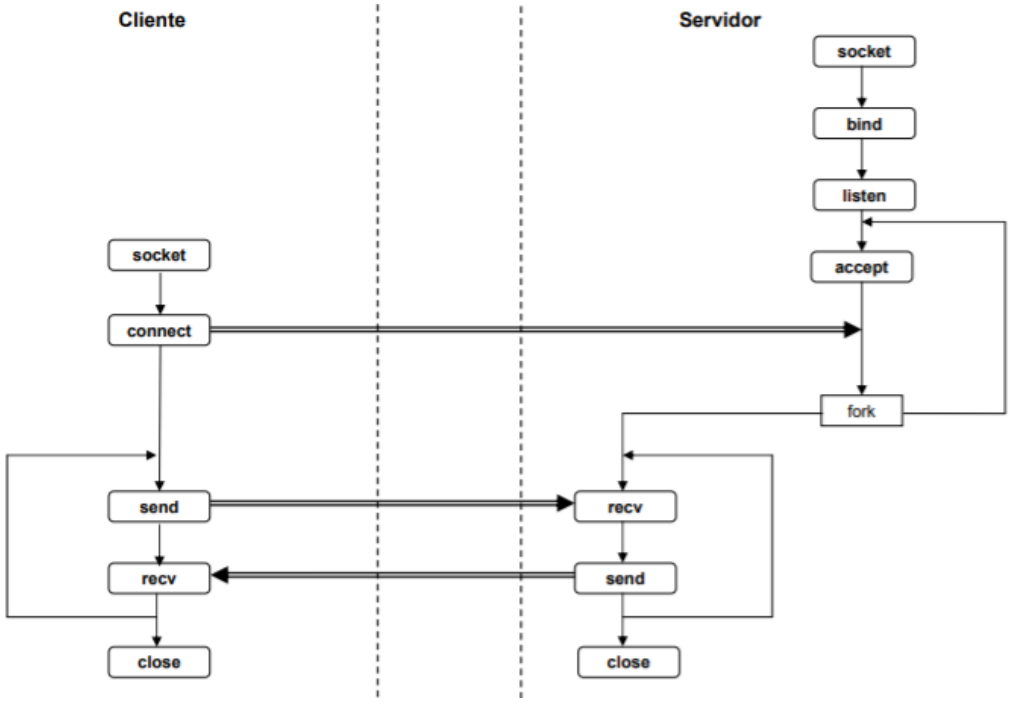


Figura 5.4 - Esquema comunicación mediante sockets.

Lo primero que hay que hacer es crear el socket mediante la función socket(), esto hay que hacerlo tanto en el servidor como en el cliente. Después de eso usamos la función bind() en el lado servidor para proporcionar al socket una dirección IP y número de puerto local, y seguidamente usamos listen(),que como ya hemos dicho pone el socket en escucha y determina el número de clientes a los que se podrá conectar. A continuación lanzamos la función accept(), que en cuanto llegue una petición de conexión por parte de un servidor la aceptará si procede y creará el socket asociado a los dos pares de direcciones. La función accept() tiene además la capacidad de bloquear el programa servidor hasta que no llegue una petición de conexión por parte de algún cliente.

Una vez que el servidor ha completado esos pasos, el cliente puede mandar una petición de conexión mediante la función connect(), si esta es aceptada el socket estará creado y cliente y servidor podrán empezar a intercambiar información. Este intercambio de datos lo haremos con las funciones write() y read(), que nos sirven para mandar y recibir datos respectivamente. Una vez que haya acabado la ejecución de los programas o simplemente no es necesario que se comuniquen más, podemos cortar la conexión y liberar los recursos empleados por el socket mediante la función close(), que se usa en ambos lados.

5.2.3 Protocolo I2C.

Y nos queda hablar de una última forma de comunicación entre dispositivos, el protocolo I2C, que en nuestro proyecto lo utilizan los sensores de distancia para comunicarse con el microprocesador de la Raspberry Pi.

I2C (inter integrated circuits) es un bus de datos diseñado por Philips Semiconductors a principios de los años 80 que sirve para conectar circuitos integrados [17]. La metodología de trasmisión de los datos es en serie, síncrona y full-duplex. Permite intercambiar información entre múltiples dispositivos a una velocidad aceptable y usando tan solo tres líneas, que son las siguientes:

- SCL (system clock): es la línea que se encarga de transmitir los pulsos del reloj, necesarios para sincronizar el sistema.
- SDA (system data): La línea que utilizan los dispositivos para enviar los datos.
- GND: La tierra común, a la que deben estar conectados todos los dispositivos.

Una de las grandes ventajas de este bus es que se pueden conectar tantos dispositivos como se deseen (siempre que no se superen ciertas limitaciones físicas), y que todos ellos pueden actuar como maestros simplemente iniciando la transmisión de datos. Sin embargo, al utilizar todos la misma línea de datos para comunicarse, debe de haber un protocolo bien claro para que no se produzcan colisiones en la línea, es decir, que más de un dispositivo transmita al mismo tiempo.

Dicho protocolo es el protocolo de comunicación I2C y se explican a continuación sus principales características.

La condición inicial de bus libre es cuando ambas señales, SDA y SCL, se encuentran a nivel alto. En este momento cualquier dispositivo maestro puede iniciar una transmisión, empleando para ello la condición de inicio, que consiste en poner en estado bajo la señal de datos manteniendo en alto la de reloj.

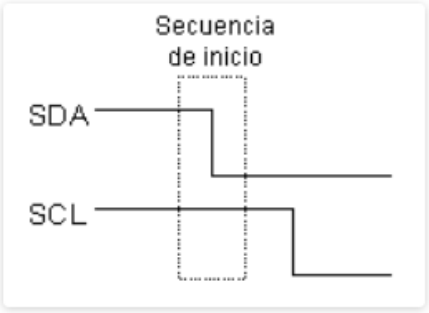


Figura 5.5 - Secuencia de inicio.

Una vez que un maestro ha enviado la condición inicial, los demás dispositivos, que ahora serán esclavos, se ponen alerta para recibir información. El primer byte que envía el dispositivo maestro incluye siete bits que indican la dirección con la que se quiere comunicar y un octavo bit que indica si la operación va a ser de lectura o escritura.

Si el dispositivo al que se ha hechor referencia con la dirección está en el bus, responde con un bit de reconocimiento (ACK), que consiste en un bit en nivel bajo. Esto significa que acepta la solicitud, y en ese momento se establece la comunicación entre esos dos dispositivos y los demás pasan a ignorar las líneas, hasta que haya otra condición de inicio.

Si el bit de lectura/escritura (R/W) era a nivel bajo, el dispositivo maestro enviará datos al esclavo (escritura), y si era a nivel alto, será el esclavo el que envíe datos al maestro (lectura).

A partir de este momento comienza la comunicación, donde el dispositivo al que le toque escribir irá mandando datos en forma de bytes. Por cada byte que lee, el otro dispositivo debe enviar un bit de reconocimiento como respuesta. Si este bit es bajo significa que ha leído el byte correctamente y se puede enviar el siguiente, y si es en estado alto indica que no puede recibir más bytes y el maestro debe finalizar la trasmisión. Por tanto, para cada 8 bits que se envían son necesarios 9 pulsos del reloj.

Si no hay ningún problema, la comunicación continuará hasta que se hayan enviado todos los datos que se querían, en ese momento, el maestro puede liberar el bus enviando una condición de parada. Esta condición de parada consiste en poner la línea de datos a nivel alto cuando la de reloj lo estaba también.

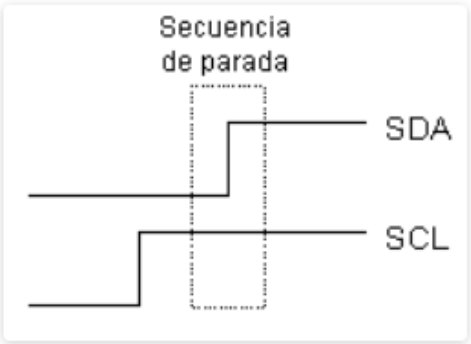


Figura 5.6 - Secuencia de parada.

Las condiciones de inicio y parada son las únicas veces en las que un dispositivo puede cambiar el estado de la línea de datos estando la de reloj a nivel alto. El resto de cambios en la línea SDA, como por ejemplo la escritura

de datos, debe producirse cuando la línea SCL está en bajo, para seguir así el sincronismo.

Si el dispositivo maestro ha terminado la operación que estaba realizando con un esclavo, pero en vez de liberar el bus quiere llevar a cabo otro tipo de operación, puede enviar de nuevo una condición de inicio. Esta condición de inicio se llama inicio reiterado, y puede servir para comunicarse con otro dispositivo diferente o seguir con el mismo, pero cambiando la condición de lectura/escritura.

5.3 Lenguaje Python.

Por último, vamos a explicar brevemente qué es el lenguaje de programación Python y cuáles son sus principales características y ventajas que han hecho que sea el lenguaje elegido para el desarrollo de las dos aplicaciones del proyecto.

Python es un lenguaje de programación de propósito general muy extendido, presente en múltiples aplicaciones y sistemas operativos. Esto se debe a que tiene una curva de aprendizaje moderada y su filosofía busca ofrecer una sintaxis de código clara y legible [18].

Nació a principios de los 90, cuando fue desarrollado por Guido Van Rossum, un ingeniero holandés que lo empezó como un hobby [19]. En sus inicios fue pensado para ser usado en pequeños proyectos, pero a día de hoy se usa también para programar grandes aplicaciones de estilo comercial. De hecho, es el lenguaje que utiliza el motor de búsqueda de Google, YouTube, el sistema de transacciones de la bolsa de Nueva York e incluso la NASA para programar sus equipos y maquinaria espacial. Todo esto nos da una idea del grado de desarrollo que ha alcanzado este lenguaje y las grandes posibilidades que nos brinda.

Figura 5.7 - Logo Python.

5.3.1 Características de Python.

Python es un lenguaje fácil de usar, que favorece el hecho de tener una sintaxis clara y legible. Se define como un lenguaje interpretado y multiparadigma [20].

Con interpretado nos referimos a la forma en que se traduje del lenguaje de alto nivel al de bajo nivel, el que entiende el procesador. Hay dos formas de llevar a cabo esta traducción:

- Compilado: Los lenguajes compilados son aquellos en los que el código es traducido por completo, para posteriormente poder ser ejecutado por el sistema. Ejemplos de lenguajes que utilicen este método son C, C++, Java…
- Interpretado: Los niveles interpretados son traducidos por un intérprete según va siendo necesario en su ejecución. Ejemplo de lenguajes son Javascript o el propio Python.

El hecho de que Python sea un lenguaje interpretado presenta ventajas, como un gran ahorro de tiempo en el desarrollo de los programas, puesto que no es necesario compilarlos por completo cada vez que quiera realizarse una prueba. El código fuente podrá ser ejecutado en cualquier sistema siempre y cuando disponga del intérprete.

Con multiparadigma hacemos referencia a que Python puede soportar más de un paradigma. Un paradigma de programación es un marco conceptual, un conjunto de ideas que describe una forma de entender la construcción de un programa. Los distintos que admite son:

- Imperativo. Los lenguajes de programación se pueden agrupar en declarativos o imperativos. Los primeros son aquellos que tan solo se encargan de describir el problema que se quiere solucionar, mientras que los segundos también permiten modificar el estado del programa mediante instrucciones de código que indican cómo realizar tareas. Python es de este segundo grupo.
- Funcional. La programación funcional se basa en el uso de funciones matemáticas que permiten la variación del programa mediante la mutación de variables. En definitiva, esto nos va a permitir trabajar con datos de entrada y salida. Además, el hecho de trabajar con funciones recursivas permite ahorrar gran cantidad de líneas de código.
- Orientada a objetos. Este paradigma se basa en el concepto de clases y objetos. Nos sirve para estructurar el programa en piezas simples y reutilizables para crear instancias individuales de objetos. Con la programación orientada a objetos se busca dejar de centrarnos únicamente en la lógica pura de los programas para tratar de buscar objetos que interactúen entre sí. Los objetos tienen una serie de atributos o características y una serie de funciones o acciones que pueden realizar.
- De tipado dinámico. Esto hace referencia a la forma que tiene Python de trabajar con variables. En este lenguaje las variables pueden almacenar datos de distintos tipos en diferentes momentos. Esto es

porque las variables son declaradas por su contenido en vez de por su contenedor. Mientras que en otros lenguajes es necesario especificar el tipo de la variable que queremos crear (entero, double, char, string, ...), en Python es suficiente con almacenar un dato en la variable, y el propio lenguaje se encarga de detectar el tipo que es.

Otras ventajas de programar en Python son:

- Programación simplificada y rápida.
- Es un lenguaje fácil para el programador porque favorece ser legible e interpretable.
- La curva de aprendizaje es moderada.
- Es un lenguaje muy portable, que puede usarse en casi todos los sistemas operativos.
- Su comunidad cuenta con un gran número de usuarios, que además participan activamente en el desarrollo del lenguaje.

6. Aplicaciones desarrolladas.

Como ya se ha explicado anteriormente, para trabajar con la prácrica se han desarrollado dos aplicaciones. Una es la aplicación principal, llamada por ello AppPrincipal, y se encarga controlar el sistema, realizando la lectura de los sensores, mandado la señal que determina su posición al servomotor, creando el streaming de vídeo en directo y diciendo cuando deben comenzar y acabar los diferentes procesos como por ejemplo graficar. Además, es la encargada de llevar a cabo el control PID, para lo cual necesita que se le indiquen los valores de los tres términos, el periodo y el setpoint. Para esto último, cuenta con una interfaz gráfica visual e intuitiva que permitirá al alumno interactuar con la aplicación.

La otra aplicación, AppGraficar, tiene como única misión graficar los datos que le son enviados desde la aplicación principal. Esta gráfica mostrará al mismo tiempo la posición de la bola y e setpoint frente al tiempo. La razón por la que la gráfica se realiza en otra aplicación distinta es que es un proceso demasiado costoso a nivel computacional, por lo que no podía llevarse a cabo en la Raspberry Pi simultáneamente al resto de procesos. Esta aplicación se ejecutará por tanto en el ordenador personal del alumno, y los datos enviados desde la Raspberry le llegarán mediante puerto serie o por sockets, dependiendo de si se está trabajando con la práctica presencial o remotamente.

A continuación, vamos a explicar con más detalle las funciones de cada aplicación y posteriormente a mostrar y comentar el código con el que se han creado.

6.1 AppPrincipal.

La aplicación principal se implementará en la tarjeta Raspberry Pi. La estructura del programa será un script principal llamado AppPrincipal que irá llamando a funciones presentes en otros scripts secundarios según vayan siendo necesarias. Las funciones básicas del programa serán:

- Pedir un usuario y contraseña, sin los cuáles no sea posible trabajar con la aplicación.
- Recoger los valores que introduzca el usuario para las ganancias del PID, el Setpoint donde deseemos situar la bola y el periodo de ejecución. Además, estos valores deben poder modificarse en cualquier momento, incluso durante la ejecución del PID.

- Determinar la posición de la bola en la viga, para lo cual será necesario leer las medidas de los dos sensores
- Una vez iniciada la ejecución del control, y utilizando el tiempo de muestreo indicado por el cliente, calcular los términos del PID y con ellos la salida necesaria para el servo en cada momento
- Comunicarse con la AppGraficar, mediante puerto serie o sockets dependiendo lo que elija el usuario.
- Enviar a la AppGraficar los datos de la posición de la bola e indicarla cuando debe comenzar y parar de graficar.
- Retransmitir vídeo en directo de la maqueta.

6.1.1 Módulo principal.

El módulo principal se llama AppPrincipal.py. En él se crea la interfaz gráfica que nos permite interactuar con el programa mediante botones, recuadros de texto y demás funcionalidades. Para crearla se utiliza la librería Tkinter, la cual está especializada en interfaces gráficas y viene por defecto con Python por lo que es ampliamente utilizada.

El módulo Tkinter está basado en widgets jerarquizados los cuales tienen cada uno una función específica. Los que se han utilizado para este programa han sido [21]:

- Tk: Es la raíz de la interfaz, sobre la que se colocan el resto de widgets.
- Frame: Es un marco que permite agrupar y colocar diferentes widgets. En nuestro caso solo se ha utilizado un Frame, que abarca toda la raíz.
- Label: Son etiquetas de texto estáticas que nos permiten mostrar texto e imágenes.
- Entry: Son recuadros que permiten escribir texto sobre ellos para luego guardarlo en una variable. Será muy útil en la aplicación para recoger el usuario y contraseña, así como los valores del PID que la persona quiera introducir.
- Button: Llama a una función cuando es pulsado.
- Radiobutton: Permite elegir entre varias opciones distintas. Tiene la característica de que solo permite seleccionar una de ellas, lo cual es beneficioso en casos en los que las opciones sean excluyentes.

Con el uso de estos seis tipos de widgets ha sido posible realizar una interfaz gráfica completa que cumpla con todas las funciones necesarias. Su apariencia se muestra en la siguiente imagen:

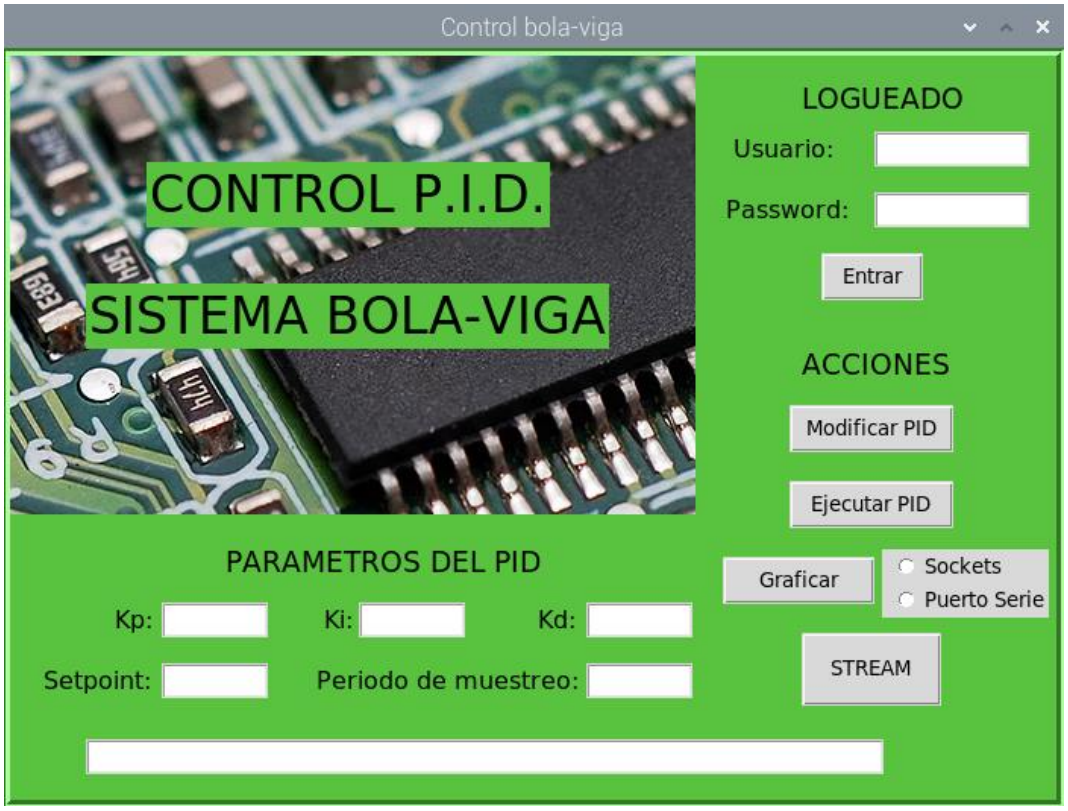


Figura 6.1 – Interfaz gráfica de la AppPrincipal

Vemos como el recuadro superior izquierdo está formado por el nombre del proyecto y una imagen decorativa de fondo. A la derecha de esto se encuentran los cajetines para introducir nombre y contraseña y debajo de estos el botón para confirmar el logueado. La zona debajo de la imagen se reserva para los recuadros que recogen el valor de setpoint, tiempo de muestreo y las ganancias del PID. En la parte inferior derecha se encuentran los botones que sirven para realizar las diferentes funciones de la aplicación. Por último, en la parte inferior, hay una barra en la que se hará aparecer texto que informa sobre el estado de la aplicación.

A continuación, vamos a mostrar y explicar brevemente el código desarrollado, tarea que haremos por partes debido a la longitud de este script.

En la primera parte que se muestra vemos como lo primero que se hace es importar los módulos que se utilizan a lo largo del código. Los seis primeros son librerías de Python que contienen alguna funcionalidad que necesitamos, y los cuatro restantes son los otros módulos de la aplicación, de los cuales vamos a importar alguna clase o función.

Seguidamente se declara el pin 22 de la tarjeta como una señal de salida PWM, y se asocia a la variable servo. Por lo tanto, cada vez que cambiemos el valor de la variable servo se modificará la anchura de pulso del

pin 22 y cambiará la posición del servomotor ahí conectado. También se declara un objeto de la clase PID y otro de la clase Tk, que como hemos dicho será la raíz de la interfaz gráfica. Tras configurar la raíz otorgándola un tamaño invariable, se declara también el frame que albergará todos los widgets y se le da un color mediante código de colores html, que como vimos en la anterior imagen es un verde claro.

```
 1  # Trabajo de fin de grado: Control sistema bola balancin
 2  # Alumno: Mario Sanchez Santiago
 3  # Tutor: Javier Garcia Ruiz
 4
 5  #Primero importamos los modulos necesarios
 6  import sys    #Permite usar funciones especificas del sistema
 7  import time          #Para la gestion de tiempos
 8  from Tkinter import *   #Para crear la interfaz grafica
 9  import RPi.GPIO as GPIO     #Para el uso de los pines
10  import threading    #Para el uso de hilos
11  import subprocess   #Para llamar a subprocesos
12  import FuncionesGrafica    #Modulo que contiene las funciones
que permiten graficar
13  import FuncionLoguin    #Modulo que contiene la funcion para
el logueado
14  import PID      #Modulo que contiene la clase pid
15  import FuncionesPID    #Modulo que contiene las funciones re-
lativas al control PID
16
17
18  GPIO.setwarnings(False)
19  a=22
20  GPIO.setmode(GPIO.BOARD)     #Vamos a usar la nomenclatura
board de los pines
21  GPIO.setup(a,GPIO.OUT)       #Usamos el pin 22
22  servo=GPIO.PWM(a,50)         #Lo ponemos modo PWM con una fre-
cuencia de 50
23  servo.start(7.9)             #Para colocar el servo centrado
24  servo.ChangeDutyCycle(7.9)
25
26  pid=PID.PID()    #Creamos un objeto de la clase PID
27
28  raiz = Tk()      #Sobre esta raiz construiremos la interfaz
grafica
29  raiz.title("Control bola-viga")
30  raiz.geometry("700x500")         #Damos un tamaño a la ventana
31  raiz.resizable(False, False)     #Impedimos que pueda modifi-
carse el tamaño
32
33  miFrame = Frame()
34  miFrame.pack()       #Creamos un frame que ocupara toda la
raiz
35  miFrame.config(width="700", heigh="500", bg="#59C33F",
bd="5", relief="groove")     #Configuramos el aspecto del frame
36
```

La segunda porción de texto que mostramos se corresponde con la declaración de las funciones que son llamadas cuando se pulsa alguno de los botones. Por tanto, para explicarlas vamos a mostrar el botón de la interfaz que las activa y explicar brevemente su funcionamiento.

La primera función que se usará será la función llamarLogueado, que es invocada cuando se pulsa el botón entrar.:

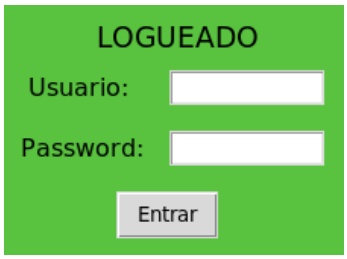


Figura 6.2 – Recuadros usuario y contraseña y botón entrar.

Su misión es recoger el valor del usuario y contraseña introducidos y crear una cadena con la que se llamará a la función Logueado, del módulo FuncionLoguin. Esta función determinará si los valores introducidos son correctos y si es así pone la variable acceso a True, lo cual permitirá acceder a todas las funcionalidades del programa. Sea cual sea el resultado, se utiliza una barra inferior para mostrar texto informativo sobre el estado del programa.

Figura 6.3 – Barra informativa

Esta barra se seguirá utilizando a lo largo de todo el programa para indicar en todo momento cuál es el estado de la aplicación y si ha habido algún error. Yendo ahora en orden descendente según su posición en la interfaz, el siguiente botón que nos encontramos es el Modificar PID:

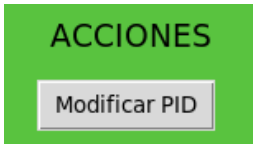


Figura 6.4 – Botón ModificarPID

Cuando se clica en este botón se invoca a la función llamaModificar-PID que, como indica su nombre, llama a la función ModificarPID, del módulo FuncionesPID, que se encargará de cambiar el valor a las ganancias del PID y al setpoint. Para ello, se le pasan por referencia a la función los valores que ha introducido el usuario en los lugares correspondientes.

En esta función, y en todas las que son invocadas por botones y que veremos a continuación, hay un condicional if, que tan solo si la variable acceso ha sido puesta a True permite llevar a cabo las acciones. En caso contrario, muestra en la barra inferior un menaje de que debe introducirse usuario y contraseña.

La siguiente función que veremos es invocada por el botón Ejecutar-PID:

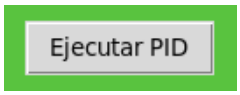


Figura 6.5 – Botón ejecutar.

El nombre de la función es llamaLanzarPID y su misión es llamar a su vez a la función LanzarPID, del módulo FuncionesPID. A esta función será necesario pasarle por referencia el periodo de muestreo y los objetos pid y servo. En la función llamaLanzarPID nos encontramos, además del condicional para saber si se ha introducido usuario y contraseña, otro condicional para actuar de distinta manera si la ejecución del PID ya ha sido lanzada o no. En caso de que el PID ya esté en ejecución, lo sabremos porque el valor de la variable ejecutando será True, y en este supuesto la función a la que se llamará será pararPID, perteneciente al mismo módulo que la anterior.

De esta forma, el mismo botón se usa para lanzar y para detener la ejecución del PID. Para recordar en todo momento en que situación nos encontramos, el texto del botón va cambiando, y si la ejecución del control está en marcha el texto que se mostrará será el siguiente:

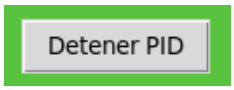


Figura 6.6 – Mismo botón ahora para detener el PID

El siguiente botón sobre el que hablaremos será el Graficar, que al igual que antes tiene dos misiones, lanzar y detener la gráfica, y por tanto el texto del botón también varía según cuál sea la que toque:

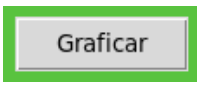


Figura 6.7 – Botón Graficar

Figura 6.8 – Mismo botón, pero para parar la gráfica

El objetivo de este botón es llamar a la función encargada de enviar los datos a la AppGraficar. Como este envío de datos se puede hacer mediante sockets o mediante puerto serie, se utiliza un radiobutton para seleccionar la opción deseada.

Figura 6.9 – Radiobutton

Si la opción que está seleccionada es Sockets, la variable modo valdrá uno, y si está seleccionado Puerto Serie, valdrá dos. Esto se utiliza para distinguir que función llamar, ya que en el primer caso será GraficaSockets y en el segundo GraficaSerial. En caso de pulsar el botón cuando el proceso de graficar ya estaba en marcha, la función a la que se llamará será Parar_grafica. Estas tres funciones se encuentran en el módulo FuncionesGrafica.

Como caso especial comentar que la función GraficaSockets devuelve el estado True en caso de que haya habido algún error a la hora de establecer la conexión con el servidor. En ese caso, muestra un mensaje en la barra inferior para indicar que no se pudo lanzar la gráfica.

Y por último, nos queda el botón STREAM, que llama a la función empezarStream:

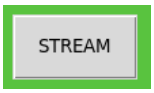


Figura 6.10 – Botón STREAM

Este botón, a diferencia de los dos anteriores solo nos permitirá empezar el stream, no pararlo, por lo que si se pulsa una segunda vez lo único que hará será mostrar en la barra inferior un mensaje de que ya se está retransmitiendo. La primera vez que se pulsa, lo que hace es crear y lanzar un hilo que ejecute la función Retransmitir, que está también declarada en este módulo, en la línea 38. Al llamarse esta función se crea un subproceso que ejecuta el script Stream.py, utilizando para ello el intérprete de python3. La razón por la que se crea un hilo para ejecutar este script es que este queda bloqueado retransmitiendo el vídeo, por lo que si lo hiciera el proceso principal la aplicación dejaría de responder.

Una vez lanzada la retransmisión es stream, esta no podrá detenerse hasta que se cierre la aplicación, y en la consola se introduzca una KeyboardInterrupt, Ctrl+C, para finalizar este proceso.

```
37  #Funciones
38  def Retransmitir():
39      subprocess.call(['python3', 'Stream.py'])    #Ejecuta el
script Stream.py utilizando el interprete de python3
40
```

```
 41  def llamarLogueado():
 42      UserPass = Usuario.get()+";"+Contrasena.get()    #Crea una
cadena con el usuario y contr. que se han introducido
 43      global acceso
 44      if(FuncionLoguin.Logueado(UserPass)==1):    #Si la fun-
cion logueado regresa un 1, es que ha sido correcto
 45          TextoInfo.set("Usuario y contrasena correctos")
#Muestra en la barra de informacion que el logueado fue correcto
 46          acceso=True         #Esta variable recoge que el
usuario ya dispone de acceso a la app
 47      else:
 48          TextoInfo.set("Usuario y contrasena incorrectos")
 49          acceso=False
 50
 51  def llamaGraficar():
 52      if(acceso==True):       #Solo puede lanzarse la grafica
si se introdujo correctamente Us. y cont.
 53          global graficando   #Esta variable indica si ya se
esta graficando o no
 54          if(graficando==False):
 55              if (modo.get()==1):     #hay que comprobar el
modo: sockets=1 y puerto serie=2
 56                  error=FuncionesGrafica.GraficaSockets(pid)
#La funcion GraficaSockets devuelve un 1 en caso de que haya un
error de comunicacion
 57                  if(error==True):
 58                      TextoInfo.set("No se pudo establecer co-
nexion con la AppGraficar")
 59                  else:
 60                      TextoInfo.set("Comenzamos a graficar")
 61                      BotonGraficar.config(text="Parar")
#Cambiamos el texto del boton, que ahora servira para pausar la
grafica
 62                      graficando=True         #Indicamos que
estamos graficando
 63              else:       #Si modo es distinto de 1, llamamos a
la funcion de graficar mediante puerto serie
 64                  FuncionesGrafica.GraficaSerial(pid)
 65                  BotonGraficar.config(text="Parar")
 66                  graficando=True
 67                  TextoInfo.set("Comenzamos a graficar")
 68          else:
 69              BotonGraficar.config(text="Graficar")
 70              graficando=False
 71              FuncionesGrafica.Parar_grafica()        #La fun-
cion Parar_grafica es igual para ambos modos
 72              TextoInfo.set("Paramos de graficar")
 73      else:
 74          TextoInfo.set("Introduce usuario y contrasena")
 75
 76  def llamaModificarPID():
 77      if(acceso==True):
 78          FuncionesPID.Modi-
ficar_pid(pid,Kp.get(),Ki.get(),Kd.get(),Setpoint.get(),Tm.get())
 79          TextoInfo.set("Parametros modificados")
 80      else:
 81          TextoInfo.set("Introduce usuario y contrasena")
 82
 83  def llamaLanzarPID():
```

```
 84      if(acceso==True):
 85          global ejecutando
 86          if(ejecutando==False):
 87              BotonEjecutarPID.config(text="Detener PID")
 88              ejecutando=True
 89              FuncionesPID.Lanzar_PID(pid,servo)
 90              TextoInfo.set("Comenzamos la ejecucion del PID")
 91          else:
 92              BotonEjecutarPID.config(text="Ejecutar PID")
 93              ejecutando=False
 94              FuncionesPID.pararPID()
 95              TextoInfo.set("Ejecucion del PID finalizada")
 96      else:
 97          TextoInfo.set("Introduce usuario y contrasena")
 98
 99  def empezarStream():
100      if(acceso==True):
101          global retransmitiendo
102          if(retransmitiendo==False):
103              TextoInfo.set("Comenzamos la retransmision del
video")
104              HiloRetransmitir=threading.Thread(target=Retrans-
mitir)
105              HiloRetransmitir.start()    #Creamos y lanzamos
un hilo que llama a la funcion retransmitir
106              retransmitiendo=True
107          else:
108              TextoInfo.set("retransmision en marcha!")   #Es-
cribimos en la barra de info. para indicar que ya se esta retrans-
mitiendo
109      else:
110          TextoInfo.set("Introduce usuario y contrasena")
111
112
```

La parte restante de código de este primer script se corresponde con la creación y configuración de los elementos de la interfaz gráfica. Primero se muestran los Labels, que como se dijo anteriormente son etiquetas de texto o imagen. De la línea 114 a la 118 se crea el Label que se corresponde con la imagen utilizada de fondo. Es importante destacar que el archivo de la imagen que utiliza debe estar en todo momento en la misma carpeta que el resto del programa.

A continuación, y hasta la línea 158 se declaran el resto de Labels, que serán el texto que vemos en la interfaz. Para cada uno de ellos indicamos el texto que muestra, tipo de letra, tamaño, color de fondo y posición que ocupa en el frame.

```
113  #Imagen
114  #Las siguientes lineas  permiten poner una imagen de fondo en
nuestra interfaz
115  miImagen = PhotoImage(file="Fondo.PNG") #El archivo debe es-
tar en la misma carpeta que el script
116  LabelImagenFondo = Label(miFrame, image=miImagen)
117  LabelImagenFondo.config(heigh="300", width="450")
118  LabelImagenFondo.place(x="0", y="0")
119
120
121  #Labels de texto
122  LabelTitulo1 = Label(miFrame, text="CONTROL P.I.D.",
font=("Comic Sans MS",25), bg="#59C33F")   #Creamos una etiqueta
de texto,
123  #indicamos el texto que muestra,el tipo de letra y tamaño y
el color de fondo
124  LabelTitulo1.place(x="90", y="70")      #Indicamos la posi-
cion que ocupara en la ventana
125
126  LabelTitulo2 = Label(miFrame, text="SISTEMA BOLA-VIGA",
font=("Comic Sans MS",25), bg="#59C33F")
127  LabelTitulo2.place(x="50", y="150")
128
129  LabelLogueado = Label(miFrame, text="LOGUEADO", font=("Comic
Sans MS",14), bg="#59C33F")
130  LabelLogueado.place(x="520", y="15")
131
132  LabelUsuario = Label(miFrame, text="Usuario:", font=("Comic
Sans MS",12), bg="#59C33F")
133  LabelUsuario.place(x="475", y="50")
134
135  LabelPassword = Label(miFrame, text="Password:", font=("Comic
Sans MS",12), bg="#59C33F")
136  LabelPassword.place(x="470", y="90")
137
138  LabelAcciones = Label(miFrame, text="ACCIONES", font=("Comic
Sans MS",14), bg="#59C33F")
139  LabelAcciones.place(x="520", y="190")
140
141  LabelParametros = Label(miFrame, text="PARAMETROS DEL PID",
font=("Comic Sans MS",14), bg="#59C33F")
142  LabelParametros.place(x="140", y="320")
143
144  LabelKp = Label(miFrame, text="Kp:", font=("Comic Sans
MS",12), bg="#59C33F")
145  LabelKp.place(x="67", y="360")
146
147  LabelKi = Label(miFrame, text="Ki:", font=("Comic Sans
MS",12), bg="#59C33F")
148  LabelKi.place(x="205", y="360")
149
150  LabelKd = Label(miFrame, text="Kd:", font=("Comic Sans
MS",12), bg="#59C33F")
151  LabelKd.place(x="346", y="360")
152
153  LabelSetpoint = Label(miFrame, text="Setpoint:", font=("Comic
Sans MS",12), bg="#59C33F")
154  LabelSetpoint.place(x="20", y="400")
155
```

```
156  LabelTm = Label(miFrame, text="Periodo de muestreo:",
font=("Comic Sans MS",12), bg="#59C33F")
157  LabelTm.place(x="200", y="400")
158
```

Otro widget que se utiliza en la interfaz son los Entrys, que como dijimos son recuadros en los que podemos introducir texto desde el teclado. Para cada uno de ellos hay que declarar una variable del tipo StringVar que será la que almacene el texto que se escriba en el recuadro. Además debemos indicar la posición que ocupa en el frame, la anchura y la posición del texto dentro del recuadro.

Para el Entry en el que se introduce la contraseña se configura un atributo a mayores. Poniendo show="*" hacemos que al escribir en este recuadro lo que se vea sean asteriscos, lo cual es muy útil para que terceras personas no puedan ver la contraseña que se está introduciendo.

```
161  #Entrys
162  Usuario=StringVar()     #Variable que almacenara la cadena
que se escriba en el recuadro
163  EntryUsuario = Entry(miFrame, textvariable=Usuario)     #Crea
un entry, que es un recuadro en el que se puede introducir y mos-
trar texto
164  EntryUsuario.place(x="570", y="50")
165  EntryUsuario.config(width="12", justify="center")       #Con-
figura el tamaño del recuadro y la posicion del texto en este
166
167  Contrasena=StringVar()
168  EntryPassword = Entry(miFrame, textvariable=Contrasena)
169  EntryPassword.place(x="570", y="90")
170  EntryPassword.config(width="12", justify="center", show="*")
171
172  Kp=StringVar()
173  EntryKp = Entry(miFrame, textvariable=Kp)
174  EntryKp.place(x="100", y="360")
175  EntryKp.config(width="8", justify="center")
176
177  Ki=StringVar()
178  EntryKi = Entry(miFrame, textvariable=Ki)
179  EntryKi.place(x="230", y="360")
180  EntryKi.config(width="8", justify="center")
181
182  Kd=StringVar()
183  EntryKd = Entry(miFrame, textvariable=Kd)
184  EntryKd.place(x="380", y="360")
185  EntryKd.config(width="8", justify="center")
186
187  Setpoint=StringVar()
188  EntrySetpoint = Entry(miFrame, textvariable=Setpoint)
189  EntrySetpoint.place(x="100", y="400")
190  EntrySetpoint.config(width="8", justify="center")
191
192  Tm=StringVar()
```

```
193  EntryTm = Entry(miFrame, textvariable=Tm)
194  EntryTm.place(x="380", y="400")
195  EntryTm.config(width="8", justify="center")
196
197  TextoInfo=StringVar()
198  EntryTextoInfo = Entry(miFrame, textvariable=TextoInfo)
199  EntryTextoInfo.place(x="50", y="450")
200  EntryTextoInfo.config(width="65")
201
```

Desde la línea 203 a la 225 se crean y configuran los botones, a los que se les indica el texto que muestran, su tamaño, la posición que ocupan en el frame y la función a la que se llama al pulsarlos.

Los Radiobuttons se declaran a continuación y hasta la línea 232. Además de las características habituales, hay que indicar a qué variable estarán asociados y cuál será su valor cuando esté seleccionado cada uno de ellos.

Finalmente, la última línea del programa sirve para crear un bucle en el que se encontrará atrapado la ejecución hasta que se cierre voluntariamente el programa. De no hacerse esto, al llegar al final el programa finalizaría y se cerraría la aplicación al instante. En este estado de bucle, la aplicación es capaz de responder cada vez que es pulsado un botón.

```
203  #Botones
204  BotonEntrar=Button(miFrame, text="Entrar", command=llamarLo-
gueado)       #Crea un boton que llamara a la funcion llamarLo-
gueado cuando se pulse
205  BotonEntrar.place(x="535", y="130")
206  acceso=False
207
208  BotonModificarPID=Button(miFrame, text="Modificar PID", com-
mand=llamaModificarPID)
209  BotonModificarPID.place(x="514", y="230")
210  BotonModificarPID.config(width="10")
211
212  BotonEjecutarPID=Button(miFrame, text="Ejecutar PID", com-
mand=llamaLanzarPID)
213  BotonEjecutarPID.place(x="514", y="280")
214  BotonEjecutarPID.config(width="10")
215  ejecutando=False
216
217  BotonGraficar=Button(miFrame, text="Graficar", command=llama-
Graficar)
218  BotonGraficar.place(x="470", y="330")
219  BotonGraficar.config(width="9")
220  graficando=False
221
222  BotonStream=Button(miFrame, text="STREAM", command=empezar-
Stream)
223  BotonStream.place(x="522", y="380")
```

```
224  BotonStream.config(width="8", height="2")
225  retransmitiendo=False
226
227
228  #Radiobuttom
229  modo=IntVar()
230  OpcionSockets=Radiobutton(miFrame, text="Sockets       ",
variable=modo, value=1).place(x="575", y="325")
231  OpcionSerial=Radiobutton(miFrame, text="Puerto Serie", varia-
ble=modo, value=2).place(x="575", y="347")
232  #Creamos un Radiobutton para elegir entre sockets y puerto
serie
233
234
235  raiz.mainloop()      #Hace un bucle en el que se mantendra el
programa para que no muera
```

6.1.2 Usuario y contraseña.

Para que la persona que quiera trabajar con la aplicación pueda acceder a todas las funcionalidades de esta es necesario que introduzca un usuario y contraseña correctos. Hasta que no lo haga, a cada botón que pulse se mostrará un mensaje en la barra informativa indicando que debe hacerlo. La comprobación de este hecho se lleva a cabo el mediante la función Logueado del módulo FuncionLoguin.py.

Para ello se importa el módulo haslib de la librería estándar, que permite hacer cifrados en Python con diferentes algoritmos [22]. Emplea el método hash, que consiste en un algoritmo matemático que transforma cualquier cadena de caracteres en una nueva serie de caracteres distinta y con una longitud fija. Esto se hace para el programa al ejecutarse no utilice la contraseña para hacer la verificación, si no que usa su encriptación. De esta manera en caso de robo de información, tan solo tendrán una serie de caracteres que no podrán relacionar con la contraseña real. Entre los diferentes algoritmos con que cuenta haslib se ha elegido el md5, que es uno de los más sencillos, pero cumple con los requerimientos de seguridad de este proyecto.

En el código podemos ver como se declaran unos usuarios y contraseñas que serán los válidos y se usa la función hexdigest() para codificar la contraseña en hexadecimal. A continuación, se declara la función Logueado, que compara el usuario y contraseña introducidos con los que estaban en la base de datos. Si la comprobación determina que son correctos la función devuelve un uno, y en caso contrario devuelve un cero.

```
 1 # MODULO DE AUTENTIFICACIÓN
 2
 3  import hashlib      #Biblioteca para realizar el cifrado
 4
 5  database={'Usuario1':'1234'}    #Aqui se añaden los usuarios y
sus respectivas contraseñas
 6  encrypted = dict([(name,hashlib.md5(pw).hexdigest()) for name,
pw in database.items()]) #Encriptamos la contraseña
 7
 8  def Logueado(UserPass):       #Funcion comprueba el usu. y
cont. que se han introducido
 9      print("Comprobando usuario y contrasena")
10      name, ask=UserPass.split(';')       #Convierte la cadena
en una lista
11      if name not in encrypted or hashlib.md5(ask).hexdigest()
!= encrypted.get(name):    #Compara el usu. y cont. introducidos
con las correctas
12          print("contrasena incorrecta")
13          return 0    #Si no es correcto
14
15      else:
16          print("contrasena correcta")
17          return 1    #Si es correcto
```

6.1.3 VL53L0X.

Los dos siguientes módulos que voy a comentar son los necesarios para trabajar con los sensores de distancia VL53L0X. Los presento ahora debido a que sus mediciones serán utilizadas en el resto de los scripts.

Para configurar los sensores se ha utilizado una librería ya existente, su nombre es VL53L0X-python y su autor la compañía Pimoroni [23]. Es de código libre, por lo que está disponible en internet y puede ser usada por cualquier persona sin ningún tipo de restricción.

La librería facilita una interfaz simple entre la Raspberry Pi y los sensores, comunicándose con ellos mediante el protocolo I2C. En su documentación además cuenta con ejemplos de uso para facilitar su entendimiento, y uno de ellos presenta la posibilidad de conectar dos sensores simultáneamente, que era una necesidad para este proyecto.

Del código podemos destacar entre las diferentes clases que crea la llamada VL53L0X, que será la que utilizaremos. Esta cuenta con funciones interesantes como son init y open para inicializar los sensores, close para apagarlos, start_ranging para comenzar a medir, get_distance para obtener la medida de ese instante y change_adress, que es la que nos permitirá utilizar los dos sensores.

```
  1 from ctypes import CDLL, CFUNCTYPE, POINTER, c_int, c_uint,
pointer, c_ubyte, c_uint8, c_uint32
  2  import sysconfig
  3  import pkg_resources
  4  SMBUS='smbus'
  5  for dist in pkg_resources.working_set:
  6      #print(dist.project_name, dist.version)
  7      if dist.project_name == 'smbus':
  8          break
  9      if dist.project_name == 'smbus2':
 10          SMBUS='smbus2'
 11          break
 12  if SMBUS == 'smbus':
 13      import smbus
 14  elif SMBUS == 'smbus2':
 15      import smbus2 as smbus
 16  import site
 17
 18
 19  class Vl53l0xError(RuntimeError):
 20      pass
 21
 22
 23  class Vl53l0xAccuracyMode:
 24      GOOD = 0        # 33 ms timing budget 1.2m range
 25      BETTER = 1      # 66 ms timing budget 1.2m range
 26      BEST = 2        # 200 ms 1.2m range
 27      LONG_RANGE = 3  # 33 ms timing budget 2m range
 28      HIGH_SPEED = 4  # 20 ms timing budget 1.2m range
 29
 30
 31  class Vl53l0xDeviceMode:
 32      SINGLE_RANGING = 0
 33      CONTINUOUS_RANGING = 1
 34      SINGLE_HISTOGRAM = 2
 35      CONTINUOUS_TIMED_RANGING = 3
 36      SINGLE_ALS = 10
 37      GPIO_DRIVE = 20
 38      GPIO_OSC = 21
 39
 40
 41  class Vl53l0xGpioAlarmType:
 42      OFF = 0
 43      THRESHOLD_CROSSED_LOW = 1
 44      THRESHOLD_CROSSED_HIGH = 2
 45      THRESHOLD_CROSSED_OUT = 3
 46      NEW_MEASUREMENT_READY = 4
 47
 48
 49  class Vl53l0xInterruptPolarity:
 50      LOW = 0
 51      HIGH = 1
 52
 53
 54  # Read/write function pointer types.
 55  _I2C_READ_FUNC = CFUNCTYPE(c_int, c_ubyte, c_ubyte,
POINTER(c_ubyte), c_ubyte)
 56  _I2C_WRITE_FUNC = CFUNCTYPE(c_int, c_ubyte, c_ubyte,
POINTER(c_ubyte), c_ubyte)
```

```
 57
 58  # Load VL53L0X shared lib
 59  suffix = sysconfig.get_config_var('EXT_SUFFIX')
 60  if suffix is None:
 61      suffix = ".so"
 62  _POSSIBLE_LIBRARY_LOCATIONS = ['VL53L0X-python-master/bin'] +
site.getsitepackages() + [site.getusersitepackages()]
 63  for lib_location in _POSSIBLE_LIBRARY_LOCATIONS:
 64      try:
 65          #_TOF_LIBRARY = CDLL(lib_location + '/vl53l0x_python'
+ suffix)
 66          _TOF_LIBRARY = CDLL(lib_location + '/vl53l0x_python'
+ suffix)
 67          break
 68      except OSError:
 69          pass
 70  else:
 71      raise OSError('Could not find vl53l0x_python' + suffix)
 72
 73
 74  class VL53L0X:
 75      """VL53L0X ToF."""
 76      def __init__(self, i2c_bus=1, i2c_address=0x29,
tca9548a_num=255, tca9548a_addr=0):
 77          """Initialize the VL53L0X ToF Sensor from ST"""
 78          self._i2c_bus = i2c_bus
 79          self.i2c_address = i2c_address
 80          self._tca9548a_num = tca9548a_num
 81          self._tca9548a_addr = tca9548a_addr
 82          self._i2c = smbus.SMBus()
 83          self._dev = None
 84          # Resgiter Address
 85          self.ADDR_UNIT_ID_HIGH = 0x16 # Serial number high
byte
 86          self.ADDR_UNIT_ID_LOW = 0x17 # Serial number low byte
 87          self.ADDR_I2C_ID_HIGH = 0x18 # Write serial number
high byte for I2C address unlock
 88          self.ADDR_I2C_ID_LOW = 0x19 # Write serial number low
byte for I2C address unlock
 89          self.ADDR_I2C_SEC_ADDR = 0x8a # Write new I2C address
after unlock
 90
 91      def open(self):
 92          self._i2c.open(bus=self._i2c_bus)
 93          self._configure_i2c_library_functions()
 94          self._dev = _TOF_LIBRARY.initialise(self.i2c_address,
self._tca9548a_num, self._tca9548a_addr)
 95
 96      def close(self):
 97          self._i2c.close()
 98          self._dev = None
 99
100      def _configure_i2c_library_functions(self):
101          # I2C bus read callback for low level library.
102          def _i2c_read(address, reg, data_p, length):
103              ret_val = 0
104              result = []
105
106              try:
```

```
107                     result = self._i2c.read_i2c_block_data(ad-
dress, reg, length)
108                 except IOError:
109                     ret_val = -1
110
111                 if ret_val == 0:
112                     for index in range(length):
113                         data_p[index] = result[index]
114
115                 return ret_val
116
117             # I2C bus write callback for low level library.
118             def _i2c_write(address, reg, data_p, length):
119                 ret_val = 0
120                 data = []
121
122                 for index in range(length):
123                     data.append(data_p[index])
124                 try:
125                     self._i2c.write_i2c_block_data(address, reg,
data)
126                 except IOError:
127                     ret_val = -1
128
129                 return ret_val
130
131             # Pass i2c read/write function pointers to VL53L0X
library.
132             self._i2c_read_func = _I2C_READ_FUNC(_i2c_read)
133             self._i2c_write_func = _I2C_WRITE_FUNC(_i2c_write)
134             _TOF_LIBRARY.VL53L0X_set_i2c(self._i2c_read_func,
self._i2c_write_func)
135
136         def start_ranging(self, mode=Vl53l0xAccuracyMode.GOOD):
137             """Start VL53L0X ToF Sensor Ranging"""
138             _TOF_LIBRARY.startRanging(self._dev, mode)
139
140         def stop_ranging(self):
141             """Stop VL53L0X ToF Sensor Ranging"""
142             _TOF_LIBRARY.stopRanging(self._dev)
143
144         def get_distance(self):
145             """Get distance from VL53L0X ToF Sensor"""
146             return _TOF_LIBRARY.getDistance(self._dev)
147
148         # This function included to show how to access the ST li-
brary directly
149         # from python instead of through the simplified interface
150         def get_timing(self):
151             budget = c_uint(0)
152             budget_p = pointer(budget)
153             status = _TOF_LIBRARY.VL53L0X_GetMeasurementTim-
ingBudgetMicroSeconds(self._dev, budget_p)
154             if status == 0:
155                 return budget.value + 1000
156             else:
157                 return 0
158
159         def configure_gpio_interrupt(
```

```
160             self, proximity_alarm_type=Vl53l0xGpioA-
larmType.THRESHOLD_CROSSED_LOW,
161             interrupt_polarity=Vl53l0xInterruptPolarity.HIGH,
threshold_low_mm=250, threshold_high_mm=500):
162         """
163         Configures a GPIO interrupt from device, be sure to
call "clear_interrupt" after interrupt is processed.
164         """
165         pin = c_uint8(0)  # 0 is only GPIO pin.
166         device_mode = c_uint8(Vl53l0xDeviceMode.CONTINU-
OUS_RANGING)
167         functionality = c_uint8(proximity_alarm_type)
168         polarity = c_uint8(interrupt_polarity)
169         status = _TOF_LIBRARY.VL53L0X_SetGpioCon-
fig(self._dev, pin, device_mode, functionality, polarity)
170         if status != 0:
171             raise Vl53l0xError('Error setting VL53L0X GPIO
config')
172
173         threshold_low = c_uint32(threshold_low_mm << 16)
174         threshold_high = c_uint32(threshold_high_mm << 16)
175         status = _TOF_LIBRARY.VL53L0X_SetInterruptThresh-
olds(self._dev, device_mode, threshold_low, threshold_high)
176         if status != 0:
177             raise Vl53l0xError('Error setting VL53L0X thresh-
olds')
178
179         # Ensure any pending interrupts are cleared.
180         self.clear_interrupt()
181
182     def clear_interrupt(self):
183         mask = c_uint32(0)
184         status = _TOF_LIBRARY.VL53L0X_ClearInterrupt-
Mask(self._dev, mask)
185         if status != 0:
186             raise Vl53l0xError('Error clearing VL53L0X inter-
rupt')
187
188     def change_address(self, new_address):
189         if self._dev is not None:
190             raise Vl53l0xError('Error changing VL53L0X ad-
dress')
191
192         self._i2c.open(bus=self._i2c_bus)
193
194         if new_address == None:
195             return
196         elif new_address == self.i2c_address:
197             return
198         else:
199             # read value from 0x16,0x17
200             high = self._i2c.read_byte_data(self.i2c_address,
self.ADDR_UNIT_ID_HIGH)
201             low = self._i2c.read_byte_data(self.i2c_address,
self.ADDR_UNIT_ID_LOW)
202
203             # write value to 0x18,0x19
204             self._i2c.write_byte_data(self.i2c_address,
self.ADDR_I2C_ID_HIGH, high)
```

```
205                 self._i2c.write_byte_data(self.i2c_address,
self.ADDR_I2C_ID_LOW, low)
206
207                 # write new_address to 0x1a
208                 self._i2c.write_byte_data(self.i2c_address,
self.ADDR_I2C_SEC_ADDR, new_address)
209
210                 self.i2c_address = new_address
211
212             self._i2c.close()
```

6.1.4 Sensores.

El módulo Sensores.py sirve solo para la inicialización y configuración de los dos sensores. Para ello se crean dos objetos de la clase VL53L0X vista antes, que nos permitirán obtener la distancia mediada por los sensores.

Lo primero que hacemos es declarar los pines 36 y 38 como shutdown pines, es decir, pines de apagado. Una vez hecho esto los ponemos a nivel bajo para apagar los dos sensores. Tras esto, procedemos a poner a nivel alto uno de ellos, con el fin de que este se encienda. Una vez encendido lo inicializamos y cambiamos su dirección en el bus de 0x29 que es la por defecto a 0x2B. A continuación ya podemos comenzar a medir con este sensor.

Ponemos ahora a nivel alto el otro shutdown pin, con lo que encendemos el segundo sensor, que cogerá la dirección por defecto 0x2B y la dejaremos tal cual. Si no hubiéramos cambiado la dirección del anterior, al conectar este habríamos tenido problemas. Tras esto, comenzamos a medir con este sensor también y ya están ambos preparados para ser utilizados.

Por último, creamos una función llamada Lectura_sensor cuyo objetivo será determinar la posición de la bola sobre la viga a partir de las lecturas de los dos sensores. Lo máximo que pueden medir los sensores es una distancia de aproximadamente 25cm, siendo las medidas cercanas a este valor algo deficientes, por tanto, primero se comprobará si la medida obtenida en el primer sensor está dentro de su rango bueno de medida, y si es así nos quedaremos con esta lectura como la distancia real. Si no es así se hace la misma comprobación para el segundo sensor, y en caso de que la bola esté más cerca de este daremos por válida su medida. Por último, en caso de que la bola esté entre los 22,5 y los 25,5 cm de la barra (que es el rango en el que se ha comprobado experimentalmente que las medidas comienzan a ser deficientes) se hará la media de la medida de ambos sensores. Con esto, se trata de minimizar el error, puesto que en este rango ambos sensores arrojaban medidas algo menores a la real, y de esta forma compensamos el efecto de uno con el del otro. Además, se suaviza el efecto de pasar del campo medido por un sensor al otro, ya que esta diferencia inevitablemente suponía cierta anomalía en los puntos cercanos a ella.

```
#Modulo sensores

  import sys
  import RPi.GPIO as GPIO     #Para el uso de los pines
  import time     #Para el uso de sleep
  import VL53L0X  #Para el uso de los sensores de distancia


  #Declaramos el pin que usaremos para el shutdown pin de cada sensor
  sensor1_shutdown = 36
  sensor2_shutdown = 38

  GPIO.setwarnings(False)
  GPIO.setmode(GPIO.BOARD)    #Elegimos la nomenclatura BOARD

  #Configuramos los pines elegidos antes como salidas
  GPIO.setup(sensor1_shutdown, GPIO.OUT)
  GPIO.setup(sensor2_shutdown, GPIO.OUT)

  #Ponemos los pines del shutdown a nivel bajo, con lo que apagamos los dos sensores
  GPIO.output(sensor1_shutdown, GPIO.LOW)
  GPIO.output(sensor2_shutdown, GPIO.LOW)
  time.sleep(0.5)     #Los mantenemos 500ms apagados para asegurarnos que se resetean

  #Ponemos en alto el shutdown pin de uno de los sensores, para poder inicializarlo
  GPIO.output(sensor1_shutdown, GPIO.HIGH)
  time.sleep(0.5)
  #Lo encontramos en la posición 0x29, pero lo cambiamos a la posicion 0x2B
  tof1 = VL53L0X.VL53L0X(i2c_address=0x29)
  tof1.change_address(0x2B)
  #Inicializamos y empezamos a medir con el sensor
  tof1.open()
  time.sleep(0.50)
  tof1.start_ranging(VL53L0X.Vl53l0xAccuracyMode.BETTER)

  #Ponemos en alto el segundo shutdown pin para inicializarlo tambien
  GPIO.output(sensor2_shutdown, GPIO.HIGH)
  time.sleep(0.50)
  #Este sensor lo dejamos en la posición 0x29
  tof2 = VL53L0X.VL53L0X(i2c_address=0x29)
  #Inicializamos y empezamos a medir con el sensor
  tof2.open()
  time.sleep(0.50)
  tof2.start_ranging(VL53L0X.Vl53l0xAccuracyMode.BETTER)

#Creamos una función que sera llamada desde otros módulos para obtener una lectura del sensor

  def Lectura_sensor():
      medidaS1 = (tof1.get_distance() +2)/10.0    #Leemos la medida del primer sensor
      medidaS2 = (500 - tof2.get_distance() +7)/10.0  #Leemos la medida del segundo sensor
```

```
50      if (medidaS2 > 25.5):
51          distancia=medidaS2  #Si la bola esta en los 24.5cm mas
próximos al sensor2, nos quedamos con la medida de ese sensor
52      elif(medidaS1 < 22.5):
53          distancia= medidaS1     #Si esta en los 22.5cm mas
próximos al sensor1, cogemos la de este
54      else:
55          distancia = (medidaS1 + medidaS2)/2     #En caso de
que este entre medias de esos dos puntos, hacemos la media de los
dos sensores
56      return distancia
```

6.1.5 FuncionesPID

El módulo FuncionesPID.py contiene las funciones necesarias para realizar la ejecución del control PID. Estas son:

- Modificar_pid: El cometido de esta función es actualizar los valores de las ganancias del control, del setpoint y del periodo de muestreo a partir de los valores que se le pasan por referencia, que son los que introdujo el usuario en las casillas adecuadas.
- Lanzar_PID: Su misión es crear y lanzar un hilo que ejecute la función Ejecutar_PID. Para ello utilizamos la clase Thread del módulo threading, que es la librería estándar de Python para trabajar con multiprocesamiento. El hecho de lanzar un hilo nos permite estar ejecutando esa función paralelamente a la ejecución del programa principal. De no usar hilos, el programa quedaría bloqueado en la ejecución de una sola función.
- Ejecutar_PID: Se encarga de actualizar la salida del sistema, que es el valor necesario para controlar el actuador, cada un determinado tiempo que viene dado por el periodo de muestreo elegido. Lo primero que hace es medir la posición de la bola, para lo cual llamamos a la función que creamos para ello en el módulo Sensores. Una vez hecho esto, y sólo si la medida está entre 0 y 500 (en caso contrario se considerará no válida), se llama con este valor a la función update de la clase pid, que es la encargada de realizar los cálculos para obtener la salida para el servo. Con el valor devuelto por update actualizamos la salida del PWM del servo. Todo este proceso está incluido dentro de un bucle while, que se repetirá constantemente mientras que la variable_parar_pid sea distinta de cero. Una vez pase eso, se pondrá la viga en posición vertical y acabará la ejecución del pid.

Hay que comentar, que utilizando una pausa del valor del periodo de muestreo después de cada ejecución, el periodo real no es exactamente el que debería, puesto que al tiempo de pausa se le suma el de procesamiento del código. No

obstante, se ha observado que este tiempo extra es completamente despreciable frente al valor del tiempo de muestreo, por lo que el método puede considerarse válido.

- pararPID: Como su nombre indica, es la función encargada de detener la ejecución del control pid. Para ello tan solo tiene que cambiar el valor de la variable_parar_pid a 1, y como vimos anteriormente, el bucle while de la función Ejecutar_PID dejará de repetirse.

```
#Modulo FuncionesPID

 import sys
 import PID  #Para el uso de la clase pid
 import time     #Para el uso de sleep
 import threading    #Para el uso de hilos
 import Sensores     #Para el uso de los sensores

 variable_parar_pid=0

 def Modificar_pid(pid,Kp,Ki,Kd,Setpoint,periodo):
     print("Modificamos los valores del control PID")
     pid.Kp=(float(Kp))    #modificamos la ganancia proporcional
     pid.Ki=(float(Ki))    #modificamos la ganancia integral
     pid.Kd=(float(Kd))    #modificamos la ganancia derivativa
     pid.setpoint=(float (Setpoint))  #modificamos el setpoint
     pid.muestreo=float(periodo)  #modificamos el periodo de muestreo
     print("Kp: %f"% pid.Kp)
     print("Ki: %f"% pid.Ki)
     print("Kd: %f"% pid.Kd)
     print("Setpoint: %f"% pid.setpoint)
     print("Periodo de muestreo: %f"% pid.muestreo)


 def Lanzar_PID(pid,servo):
     print ("Comenzamos la ejecucion del control PID")
     pid.ejecuta=1       #Variable par indicar que esta ejecutándose el pid
     hilopid = threading.Thread(target = Ejecutar_pid, args=(pid,servo,))
     hilopid.start()     #Creamos y lanzamos el hilo que ejecutara la función Ejecutar_pid


 def pararPID():
     global variable_parar_pid
     variable_parar_pid = 1      #Ponemos a 1 la variable que indica que hay que detener la ejecución del pid


 def Ejecutar_pid(pid, servo):
     global variable_parar_pid
     while (variable_parar_pid==0):
         distancia=Sensores.Lectura_sensor()     #Obtenemos la lectura de distancia del sensor

```

```
42          if(distancia>0 and distancia<50):      #Solo si la me-
dida esta en ese rango es valida
43              pid.update(distancia)    #Llamamos a la función up-
date de la clase pid, que calcula la salida necesaria para el
servo
44              servo.ChangeDutyCycle(pid.output)    #Llevamos al
servo a la posicion calculada
45          time.sleep(pid.muestreo)    #Esperamos el tiempo de
muestreo hasta la siguiente ejecucion
46
47      variable_parar_pid=0    #Ponemos la variable parar a 0 de
nuevo para la siguiente vez que se lance el pid
48      pid.ejecuta=0
49      print("Finalizamos la ejecucion del PID")
50      servo.ChangeDutyCycle(7.6)  #Al acabar dejamos la viga en
posición vertical
51      sys.exit()
```

6.1.6 FuncionesGráfica.

El módulo FuncionesGrafica.py recoge las funciones necesarias para llevar a cabo la gráfica de los datos obtenidos. Sin embargo, no se encargan directamente de ello, pues como ya hemos explicado, el proceso de generar la gráfica consume demasiados recursos y la Raspberry no podía con ello. Por tanto, esta tarea se realiza en un ordenador externo, para lo cual será necesario mandarle los datos que queremos graficar. Además del envío de datos, estas funciones nos permiten controlar desde la AppPrincipal cuando comienza y cuando se detiene la gráfica. Las funciones en cuestión son tres:

- GraficaSockets. Esta función es la que se usará para enviar mediante sockets los datos de la Raspberry a la AppGraficar que está alojada en otro ordenador. El papel que tomará en la comunicación será el de servidor, por lo que será la encargada de crear un socket y asociarlo a su dirección y a un puerto disponible, que será el 10000. En caso de que se produzca un error durante este proceso, la función terminará y devolverá el valor de True al programa principal para informar del error. Este fallo al crear el socket puede darse principalmente cuando se corta una transmisión de datos y se trata de crear una nueva al instante, puesto que el socket aún está cerrándose y no ha liberado los recursos por completo. Gestionando el error con un try except, evitamos que el programa finalice y puede intentarse de nuevo la conexión.
  Una vez hemos creado el socket y le hemos asignado una dirección IP y un puerto, el siguiente paso es ponerlo a escuchar posibles conexiones. Esto se hace con el comando listen, que nos permite además indicar cuantos clientes aceptar, en nuestro caso será uno solo. En el momento que llegue una solicitud de conexión por parte de un cliente, se acepta con el comando accept, y ya podemos empezar la comunicación.

Después de esto, se declara la función Mandar_datos, que será la encargada de enviar a la AppGraficar los datos de posición de la bola y el setpoint. Lo primero que debe hacer es hallar la posición de la bola haciendo uso de la función Lectura_sensor(). Además, en el probable caso de que paralelamente a este proceso se esté llevando a cabo la ejecución del control, se usará la medida de distancia hecha por este proceso. Con esto evitamos tener que hacer el doble de mediciones, lo cual es una tarea larga y costosa a nivel computacional. Sea cual sea la manera de obtener la distancia, hay que pasarla a milímetros, puesto que trabajar con números enteros facilita el envío de los datos.

A continuación, tenemos un condicional que, en caso de que el valor de la variable_parar_grafica sea 1, envía al cliente la orden de parar y sale de la función. Como veremos más adelante, el valor de esta variable es siempre cero hasta que desde la interfaz el usuario da la orden de parar la gráfica.

En caso de no haber entrado en ese if, tenemos uno nuevo que comprueba que la medida de distancia esté entre 0 y 500mm, puesto que es imposible que no y el hecho de que así fuera significaría que ha habido un error. Pasado este corte, se crea un mensaje que incluya el dato de la posición de bola y del setpoint, con un punto y coma entre medias para que sea posible separarlos en destino. Tras esto, el proceso duerme durante 200 milisegundos y comienza de nuevo, haciendo así que el periodo de muestreo sea aproximadamente 0.2 segundos.

Como la función Mandar_datos es un bucle que se repite continuamente, hay que crear un hilo para ejecutarla paralelamente al programa principal, pues de no ser así la aplicación se bloquearía.

- GraficaSerial. Esta es la homóloga de la anterior función, pero esta vez comunicándose con la AppGraficar mediante puerto serie en vez de sockets. Esta funcionalidad nos permitirá hacer la gráfica incluso si no tenemos conexión a la red, puesto que este método de comunicación es vía cable. Para empezar, debemos declarar un objeto serial, en el que indicaremos las características de la comunicación. Debemos indicar el puerto que utilizaremos (que serán los pines 8 y 10), la velocidad de transmisión que será de 115200 baudios), si utilizamos bit de paridad (no se usa), cuantos bits de parada usamos (uno), y el número de bits de datos, (ocho). Además indicamos una propiedad más que es el timeout, que determina cuantos segundos esperaremos a que lleguen los datos después de haber usado el comando recv. Si pasa más de ese tiempo y no han llegado datos el programa continúa.

  Una vez se ha configurado el puerto serie, se declara la función Mandar_datos, que es igual que en el caso anterior pero utilizando el puerto serie para transmitir los datos. Es importante comentar que la

AppGraficar lee la información que le llega al puerto serie en bloques de ocho caracteres, por lo que para que el análisis de los datos que llegan sea correcto, todos los mensajes que lleguen deben tener ese tamaño. En caso de que llegue un mensaje más corto, el programa leerá ese mensaje y el comienzo del siguiente hasta completar el número fijado de caracteres, haciendo así ambos mensajes indescifrables. Para evitar esto, la solución adoptada es añadir espacios en blanco hasta alcanzar la longitud de ocho caracteres. Tras la recepción del mensaje, esos espacios en blanco se eliminarán.

- Parar_grafica. Y por último tenemos la función utilizada para detener la gráfica. Cuando es llamada lo que hace es simplemente cambiar el valor de la variable varable_parar_grafica a 1. Como ya explicamos previamente, esto provocará que la función para enviar datos que esté ejecutándose en ese momento deje de hacerlo.

```
 1 #Modulo FuncionesGrafica
 2
 3  #La grafica se crea desde otra aplicacion, estas funciones
sirven para mandar a dicha app
 4  #los datos que debe graficar y controlar cuando empezar y pa-
rar
 5
 6  import sys
 7  import PID  #Para el uso de la clase pid
 8  import time     #Para el uso de sleep
 9  import threading    #Para el uso de hilos
10  import Sensores     #Para el uso de los sensores
11  import socket   #Para la comunicacion mediante sockets
12  import serial   #Para la comunicacion por puerto serie
13
14  variable_parar_grafica=0    #Variable para indicar cuando
queremos dejar de graficar
15
16
17  def GraficaSockets(pid):    #Esta funcion se llama cuando
queremos enviar los datos mediante sockets
18      print("Establecemos conexion mediante sockets con la
aplicacion graficar")
19      global variable_parar_grafica #Hay que indicar que la va-
riable que vamos a utilizar es global y no solo de esta funcion
20      variable_parar_grafica=0
21      HOST='157.88.201.42'     #Indicamos la dir IP que esta
utilizando la Raspberry
22      PORT=7000               #Indicamos el puerto que vamos a
usar
23      server_address = (HOST,PORT)
24      s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
#Creamos el socket
25      s.setblocking(True)         #Activamos el Set blocking
mode del socket, esto hace que se bloquee despues de listen
26      try:
27          s.bind(server_address)      #Asociamos el socket
creado a la direccion yu puerto especificados
```

```
 28      except socket.error as msg:     #Gestionamos la excepcion
en caso de error
 29          print("Conexion fallida")
 30          return True         #La funcion devuelve True y fina-
liza su ejecucion
 31      s.listen(1)         #Escuchamos peticiones de conexion,
solo queremos conectarnos a un cliente
 32      con, server_address = s.accept()       #Aceptamos la pe-
ticion que llega y creamos la conexion
 33      print('Conectando con ' + server_address[0] + ':' +
str(server_address[1]))
 34      time.sleep(0.5)
 35      print("Realizamos la grafica")
 36      def Mandar_datos(pid):#Funcion que se ejecutara en un
hilo paralelo
 37          try:
 38              while True:
 39                  if (pid.ejecuta == 0):
 40                      distancia_mm=int(Sensores.Lectura_sen-
sor()*10)     #Obtenemos la lectura de distancia del sensor
 41                  else:
 42                      distancia_mm=int(pid.distancia*1000)
#En caso de estar ejecutando el control PID, usaremos la medida de
los sensores que se ha utilizado para ello
 43                  if (variable_parar_grafica==1):
 44                      con.send("Parar")       #Mandamos un men-
saje a la AppGraficar para que detenga la grafica
 45                      break
 46                  #Tanto la medida obtenida como el setpoint
deben pasarse a mm, para facilitar su envÃo a la AppGraficar
 47                  if(distancia_mm>0 and distancia_mm<500):
#Si la medida esta fuera de ese rango no es valida, y por tanto no
se envia
 48                      setpoint_mm=int(pid.setpoint*1000)
 49                      mensaje = str(distancia_mm) + ";" +
str(setpoint_mm) +";"  #Cremos un mensaje con los dos datos, sepa-
rados por ";" para que el receptor pueda distinguirlos
 50                      con.send(mensaje)     #Se envia el mensaje
 51                      time.sleep(0.2)     #El periodo de mues-
treo sera de 0.2 segundos aprox
 52                  else:
 53                      print("Dato no valido")
 54          finally:
 55              print("Cerrando socket")
 56              con.close()     #En caso de fallo al enviar un
mensaje gestionamos la excepcion y cerramos el socket
 57
 58      HiloSockets = threading.Thread(target = Mandar_datos,
args=(pid,))
 59      HiloSockets.start()     #La funcion Mandar_datos la eje-
cutamos en un hilo para que no bloquee la aplicacion
 60      return False    #Devolvemos False para indicar que no
hubo ningun error al crear el socket
 61
 62
 63
 64  def GraficaSerial(pid):     #Esta funcion se llama cuando
queremos enviar los datos mediante puerto serie
```

```
 65      print("Nos comunicamos mediante Puerto serie con la apli-
cacion graficar")
 66      global variable_parar_grafica
 67      variable_parar_grafica=0
 68      #Declaramos el puerto serie y configuramos sus parametros
 69      puerto_serie = serial.Serial(
 70          port='/dev/ttyS0',
 71          baudrate = 115200,
 72          parity=serial.PARITY_NONE,
 73          stopbits=serial.STOPBITS_ONE,
 74          bytesize=serial.EIGHTBITS,
 75          timeout=15
 76      )
 77      print("Realizamos la grafica")
 78      def Mandar_datos(pid):#Funcion que se ejecutara en un
hilo paralelo
 79          while True:
 80              if (pid.ejecuta == 0):
 81                  distancia_mm=int(Sensores.Lectura_sen-
sor()*10)
 82              else:
 83                  distancia_mm=int(pid.distancia*1000)
 84
 85              if (variable_parar_grafica==1):
 86                  mensaje="Parar   "  #Para indicar a AppGrafi-
car que detenga la grafica. El mensaje debe tener 8 caracteres
para ser recibido correctamente
 87                  mensaje = mensaje.encode(encoding='UTF-
8',errors='strict')
 88                  puerto_serie.write(mensaje)
 89                  break   #Salimos del bucle
 90
 91              if(distancia_mm>0 and distancia_mm<500):
 92                  setpoint_mm=int(pid.setpoint*1000)
 93                  mensaje = str(distancia_mm) + ";" +
str(setpoint_mm) +";"
 94                  while(len(mensaje)<8):
 95                      mensaje=mensaje+" " #De esta forma nos
aseguramos de que el mensaje siempre tenga 8 caracteres
 96                  mensaje = mensaje.encode(encoding='UTF-
8',errors='strict')
 97                  puerto_serie.write(mensaje)
 98                  time.sleep(0.2)
 99              else:
100                  print("Dato no valido")
101
102      HiloSerial = threading.Thread(target = Mandar_datos,
args=(pid,))
103      HiloSerial.start()
104
105
106
107  def Parar_grafica():    #Esta funcion se llama cuando quere-
mos detener la grafica
108      global variable_parar_grafica
109      variable_parar_grafica=1    #Poniendo esta variable a 1
se detendran las otras funciones
110
```

6.1.7 Implementación control PID.

Los cálculos del control PID, cuyo estudio es el fin último de este proyecto, se llevan a cabo en el módulo PID. Para ello, se crea una clase con el mismo nombre que tendrá los atributos y funciones necesarios para implementar el control.

La primera función, _init_, es muy común verla en Python para la declaración de clases. En ella se declaran los atributos que tendrá la clase, y se les da un valor de inicialización, que más tarde podrá cambiarse. Cuando se cree un objeto de esta clase, se llamará a esta función para proveerla de atributos.

Después tenemos la función update, que será la que nos permita calcular la salida necesaria para el actuador partiendo de la distancia medida. Lo que se hace en estas líneas de código es desarrollar en Python los cálculos del controlador PID. Primero, en la línea 42, calculamos el error proporcional a partir de la distancia medida y la distancia objetivo, el setpoint. Este es el error que hay en el presente, y lo llamamos así porque es el que se utilizará más adelante para calcular el término proporcional.

$$E_P = setpoint - distancia$$

En la siguiente línea calculamos el error derivativo, que es el que se utilizará para obtener el término derivativo. Este término implementa la acción compensatoria en función de la derivada del error, es decir, la velocidad a la que varía, por lo que para calcularlo restamos al error en el presente el error que había en la ejecución anterior. De esta forma, cuando dividamos esta magnitud por el tiempo que ha pasado entre las dos medidas, tendremos la velocidad a la que está variando el error.

$$E_D = E_P - E_{P-Ant}$$

Tras estos cálculos podemos pasar a determinar el valor de cada uno de los tres términos. Empezamos por el proporcional, que es simplemente la ganancia proporcional multiplicada por el error.

$$T_P = K_P \times E_P$$

Continuamos con el término integral, que como sabemos actúa sobre los errores pasados. Por ello, va acumulándose y si el error es mantenido en el tiempo, aunque sea pequeño tendrá un gran efecto. Para llevar esto a la práctica, se multiplica el error actual por la ganancia integral, y esto se suma al término integral previo que hubiera hasta ese momento.

$$T_I = T_I - K_I \times E_P$$

Y por último hallamos el término derivativo. Para hacerlo tenemos que obtener la derivada del error, que como ya mencionamos antes la sacamos de dividir el error derivativo entre el tiempo de muestreo. Al valor que obtenemos le multiplicamos por su correspondiente ganancia, para tener así el término derivativo.

$$T_D = K_D \times E_D / T_m$$

A continuación se lleva a cabo el método anti-windup. El windup es un efecto que se produce a causa de la acción integral del control cuando este se satura. Este término, debido a su modo de funcionamiento, tiene memoria y va acumulando sobre su valor pasado los nuevos. Esto tiene la ventaja de que elimina los errores estacionarios por pequeños que sean, puesto que al pasar el tiempo el término va aumentando. Sin embargo, supone un problema cuando el sistema se satura en algún momento, puesto que la acción integral va creciendo y creciendo y se vuelve excesivamente grande. De esta forma cuando el sistema vuelve a un funcionamiento normal debe esperar hasta que el término integral se descargue para que el control comience a actuar de nuevo correctamente.

Para evitar este fenómeno, pueden implantarse diferentes soluciones. Por la que hemos optado es poner un límite superior e inferior al valor que puede alcanzar el término integral.

Tras este ajuste, pasamos a calcular la salida del actuador, que será la suma de los tres términos, más el valor de referencia que hace estar a la viga en posición horizontal. Por último, solo nos quedará limitar el valor máximo y mínimo que puede alcanzar la salida, con el objetivo de que no se someta a la planta a cambios o posiciones excesivamente bruscos.

```
1 #Modulo PID
 2
 3  class PID:      #Declaramos la clase PID
 4      #Controlador PID
 5      def __init__(self, P=1, I=0, D=0, muestreo=0.2):    #Cons-
tructos al que se llama cuando se crea un objeto de esta clase
 6          self.Kp = P
 7          self.Ki = I
 8          self.Kd = D
 9          self.muestreo = muestreo
10          self.setpoint = 0
11          self.bias = 7.8
12          self.ErrorPAnt = 0
13          self.ITerm = 0
14          self.output = 0
15          self.ejecuta = 0
16          self.MaximoI = 0.5
17          self.MinimoI = -0.5
18          self.distancia = 0
19
20
```

```
21      def update(self, feedback_value):       #Función pricipal,
que realiza el control PID
22          self.distancia = feedback_value/100.0   #Trabajamos
con la distancia en metros
23          print ("Valor de setpoint: %f" % self.setpoint)
24          print ("Distancia medida: %f" % feedback_value)
25
26          self.ErrorP = self.setpoint - self.distancia
#Cálculo error prop
27          self.ErrorDT =  self.ErrorP - self.ErrorPAnt    #
Cálculo error der
28          self.ErrorPAnt = self.ErrorP
29
30          # Cálculo componente proporcional
31          self.PTerm = self.ErrorP*self.Kp
32
33          # Cálculo componente integral
34          self.ITerm = self.ITerm + self.ErrorP*self.Ki
35
36          # Cálculo componente derivativa
37          self.DTerm = self.ErrorDT*self.Kd/self.muestreo
38
39          # Cálculo anti-windup
40          if (self.ITerm >= self.MaximoI):
41              self.ITerm = self.MaximoI
42          if (self.ITerm <= self.MinimoI):
43              self.ITerm = self.MinimoI
44
45          # Cálculo PID
46          self.output = self.bias + self.PTerm + self.DTerm +
self.ITerm
47
48          #Limitamos la salida max y min del actuador
49          if (self.output >= 10.8):
50              self.output = 10.8
51          if (self.output <= 4.8):
52              self.output = 4.8
53
54          print ("Error proporcional %f" % self.ErrorP)
55          print ("Error derivativo %f" % self.ErrorDT)
56          print ("    ")
57          print ("Termino proporcional %f" % self.PTerm)
58          print ("Termino integral %f" % self.ITerm)
59          print ("Termino derivativo %f" % self.DTerm)
60          print ("    ")
61          print ("Salida del actuador  %f" % self.output)
62
print("************************************************************
*************")
63
64          return(self.output)     #La funcion devuelve el valor
de salida del actuador
```

6.1.8 Stream.

La retransmisión de vídeo en directo de la planta está formada por dos procesos. Uno es el de grabar el vídeo y retransmitirlo, y otro el de visualizarlo. Como nuevamente estos son procesos de bastante carga computacional, se ha decidido realizar el segundo desde el ordenador personal del alumno también.

Para ello, se retransmitirá el streaming desde la Raspberry vía web, y podremos visualizarlo desde cualquier dispositivo que se encuentre en la misma red buscando en el navegador la dirección IP de la Raspberry y el puerto que esté utilizando.

Para crear el servidor web y retransmitir en él se utiliza el siguiente código, que se encuentra disponible en internet en el siguiente enlace [24]. El código está hecho para python3, y como el resto de la aplicación está programada en python2, para poder ejecutarlo hay que hacer uso de la librería subprocess, como vimos en el apartado del módulo principal. El inconveniente de este método es que no es posible parar la retransmisión desde el propio programa, tan solo desde la línea de comandos mediante una KeyboardInterrupt. Por tanto, una vez que se empiece la retransmisión del stream no podrá detenerse hasta que se cierre la aplicación.

```
 1 import io
 2  import picamera
 3  import logging
 4  import socketserver
 5  from threading import Condition
 6  from http import server
 7  import threading
 8
 9  PAGE="""\
10  <html>
11  <head>
12  <title>Streaming Sistema bola-viga controlado mediante
PID</title>
13  </head>
14  <body>
15  <h1>Streaming Sistema bola-viga</h1>
16  <img src="stream.mjpg" width="640" height="480" />
17  </body>
18  </html>
19  """
20
21  class StreamingOutput(object):
22      def __init__(self):
23          self.frame = None
24          self.buffer = io.BytesIO()
25          self.condition = Condition()
26
27      def write(self, buf):
28          if buf.startswith(b'\xff\xd8'):
```

```
29                # New frame, copy the existing buffer's content
and notify all
30                # clients it's available
31                self.buffer.truncate()
32                with self.condition:
33                    self.frame = self.buffer.getvalue()
34                    self.condition.notify_all()
35                self.buffer.seek(0)
36            return self.buffer.write(buf)
37
38    class StreamingHandler(server.BaseHTTPRequestHandler):
39        def do_GET(self):
40            if self.path == '/':
41                self.send_response(301)
42                self.send_header('Location', '/index.html')
43                self.end_headers()
44            elif self.path == '/index.html':
45                content = PAGE.encode('utf-8')
46                self.send_response(200)
47                self.send_header('Content-Type', 'text/html')
48                self.send_header('Content-Length', len(content))
49                self.end_headers()
50                self.wfile.write(content)
51            elif self.path == '/stream.mjpg':
52                self.send_response(200)
53                self.send_header('Age', 0)
54                self.send_header('Cache-Control', 'no-cache, pri-
vate')
55                self.send_header('Pragma', 'no-cache')
56                self.send_header('Content-Type', 'multipart/x-
mixed-replace; boundary=FRAME')
57                self.end_headers()
58                try:
59                    while True:
60                        with output.condition:
61                            output.condition.wait()
62                            frame = output.frame
63                        self.wfile.write(b'--FRAME\r\n')
64                        self.send_header('Content-Type', 'im-
age/jpeg')
65                        self.send_header('Content-Length',
len(frame))
66                        self.end_headers()
67                        self.wfile.write(frame)
68                        self.wfile.write(b'\r\n')
69                except Exception as e:
70                    logging.warning(
71                        'Removed streaming client %s: %s',
72                        self.client_address, str(e))
73            else:
74                self.send_error(404)
75                self.end_headers()
76
77    class StreamingServer(socketserver.ThreadingMixIn,
server.HTTPServer):
78        allow_reuse_address = True
79        daemon_threads = True
80
```

```
81  with picamera.PiCamera(resolution='640x480', framerate=24) as
camera:
82      output = StreamingOutput()
83      camera.start_recording(output, format='mjpeg')
84      try:
85          address = ('', 8080)
86          server = StreamingServer(address, StreamingHandler)
87          server.serve_forever()
88      finally:
89          camera.stop_recording()
```

6.2. Aplicación graficar.

La AppGraficar se ha desarrollado con el fin brindar el soporte necesario para realizar las gráficas en tiempo real sobre la variación de la posición de la bola y el setpoint a lo largo del tiempo. Para ello también se ha utilizado el lenguaje de Python, para que fuera más sencillo el proceso de delimitar las funciones de cada aplicación y cómo se compenetran entre ellas. No obstante, como esta aplicación debe ejecutarse en el ordenador personal del alumno, se ha querido evitar la obligación de tener un intérprete de Python instalado para poder trabajar con ella. En este sentido, se ha utilizado el software pyinstaller[25] para crear a partir del código desarrollado un archivo ejecutable único que pueda utilizarse en cualquier lugar, sin necesidad de tener nada instalado. Esto facilita además la labor de distribución de la aplicación, puesto que solo será necesario un archivo.

Cuando se ejecute dicho archivo, se abrirán dos ventanas. Una estará destinada a mostrar la gráfica y la otra será una línea de comandos desde la que se manejará la aplicación. En esta segunda ventana aparecerán mensajes informativos sobre el estado del programa y habrá que interactuar con ella para indicar aspectos como el tipo de comunicación que queremos utilizar y cuando queremos cerrar el programa.

Para la realización de la gráfica se ha utilizado el módulo pyqtgraph [26], que es una biblioteca de interfaz de gráfica de Python. Sus principales objetivos son proporcionar gráficos rápidos e interactivos para mostrar datos principalmente. Su carácter eminentemente científico y técnico ha hecho que la elija frente a otras opciones como matplotlib que pese a brindar mayores posibilidades en cuanto a tipo de gráficos es más limitada en el procesamiento de grandes cantidades de datos en tiempo real.

Centrándonos ya en el código, las primeras líneas como siempre son para importar los módulos necesarios para el programa. Seguidamente, de la línea 15 a 17 se crea la aplicación y una ventana asociada a ella en la que se representará la gráfica. Tras eso, se declara gData, matriz de tres dimensiones que se utilizará para almacenar los datos. Las tres dimensiones son para el tiempo, la posición de la bola y el setpoint respectivamente.

Entre la línea 25 y 32 nos dedicamos a configurar la ventana que muestra la gráfica, creando una leyenda, el gráfico de la posición de la bola y el gráfico del setpoint, los límites superior e inferior de los ejes, la rejilla, dar nombre a los ejes… Esta parte del código se ejecutará solo una vez, nada más abrir la aplicación, y lo que viene a continuación se hará en bucle hasta que se cierre.

La siguiente parte del código comienza con un menú que nos ofrece tres opciones:

- Opción 1, graficar los datos recibidos mediante sockets.
- Opción 2, graficar los datos recibidos desde el puerto serie.
- Opción 3, cerrar la aplicación.

Con el teclado habrá que introducir la opción que se quiera realizar y en caso de introducir una distinta se nos volverá a preguntar de nuevo. En este punto hay que comentar que, para iniciar la comunicación entre las dos aplicaciones, la principal debe dar primero la orden de graficar, puesto que si se lanza antes esta dará error porque aún no se han envido datos que poder leer. El error será gestionado para que la aplicación no se cierre, en su lugar se mostrará un mensaje de advertencia en la ventana de comandos y aparecerá el menú de nuevo.

Después de elegir una opción en el menú, tenemos un condicional con tres posibilidades distintas según la opción que se haya introducido. La primera, crea un socket para la comunicación e indica la dirección IP y el puerto al que quiere conectarse, que es el de la Raspberry Pi. Como este lado es el cliente de la comunicación, utilizamos el comando connect para solicitar al servidor la conexión. Si todo va bien y el servidor ya estaba esperando la solicitud, se aprobará y se creará la conexión. La segunda opción configura el puerto serie, indicando qué número de puerto usará y las características de la comunicación serial, que deberán ser las mismas que en la otra aplicación para que puedan entenderse. Recordamos que estas propiedades son:

- Velocidad de transmisión: 115200 baudios.
- Bit de paridad: Ninguno.
- Bits de parada: Uno.
- Bits de datos: Ocho.
- Timeout: Esta propiedad no tiene por qué ser igual en ambos lados. Aquí será None, lo que hará que tras el comando recv(), se espere indefinidamente hasta que llegue algún dato.

Y por último la tercera opción es cerrar la aplicación, lo cual se hará con el comando sys.exit().

Una vez hayamos configurado el método de comunicación elegido, se declara la función upDate, que será la encargada de recibir los datos, analizarlos y actualizar las gráficas. Dentro de la función lo primero que se hace es sumar a la variable tiempo 0.2, esto es porque este es el periodo de muestreo elegido, cada 0.2 segundos llegarán datos desde la Raspberry.

Seguidamente, ponemos un condicional para leer los datos de una forma u otra dependiendo de si estamos trabajando con sockets o con puerto serie. A continuación, un nuevo if comprueba si el mensaje recibido es la cadena “Parar”, lo cual significaría que hay que detener la gráfica. Esto lo

haríamos poniendo la variable_parar_grafica a 1, la razón la veremos más adelante.

En caso de que el mensaje recibido se corresponda con una recepción de datos, lo primero que se hará es separar el dato de la posición de la bola y el dato del setpoint. Tras esto, añadimos a la matriz de datos los nuevos valores obtenidos.

El siguiente paso es un condicional que comprueba si la longitud de la matriz es superior a 250 términos. En caso de ser así, elimina el primero, aquel que es más antiguo, para que la longitud máxima no pase de 250. Esto lo hacemos para que el eje X no se comprima en exceso según vamos recibiendo más y más datos. 250 valores representan los datos de 50 segundos de ejecución del control, que es una ventana bastante grande. El último paso de la función upDate es actualizar la gráfica de la posición y la del setpoint con los nuevos valores que se han añadido.

A la función upDate se la llama desde dentro de un bucle while que se repetirá constantemente siempre que la variable_parar_grafica sea cero. Por eso el método para detener la gráfica es poner esta variable a 1. Además, este bucle está a su vez dentro de un try, con el objetivo de que si se produce algún error debido a la comunicación pueda gestionarse la excepción y el programa no se cierre. La forma de gestionarlo es mostrar un mensaje de aviso y volver nuevamente al menú de selección, para comenzar de nuevo con el proceso.

```
 1 #Aplicacion que recibe datos por sockets o puerto serie
 2  #y realiza una gráfica con ellos.
 3
 4  import sys
 5  import time      #Para el uso de sleep
 6  import threading     #Para el uso de hilos
 7  from pyqtgraph.Qt import QtGui, QtCore
 8  import pyqtgraph as pg       #Para hacer la gráfica
 9  import numpy as np       #Para trabajar con matrices numericas
10  import socket    #Para la comunicacion mediante sockets
11  import serial    #Para la cmunicacion con puerto serie
12  import warnings
13  warnings.filterwarnings('ignore')    #Para que no se muestren
avisos por pantalla
14
15  app=QtGui.QApplication([])       #Creamos la aplicación
16  win=pg.GraphicsWindow(title="Sistema bola-viga. Gráfica posi-
ción y setpoint.")  #Creamos una ventana, que será lo que muestre
la App
17  p=win.addPlot()      #Añadimos una gráfica a la ventana
18
19  gData = []       #Creamos la matriz que vamos a graficar
20  gData.append([0])
21  gData.append([0])
22  gData.append([0])    #Será de tres dimensiones
```

```
 23  tiempo=0    #Inicializamos el temporizador
 24
 25  p.addLegend()   #Para añadir una leyenda a la gráfica
 26  Grafica1=p.plot(gData[0], gData[1], pen='y', name='Posicion
bola') #Creamos la primera gráfica, que será amarilla y represen-
tará la posición de la bola
 27  Grafica2=p.plot(gData[0], gData[2], pen='r', name='Setpoint')
#La segunda gráfica será roja y representará el setpoint
 28  p.setRange(yRange=[0,60])
 29  p.setRange(xRange=[0,50])   #Establecemos los límites de los
ejes
 30  p.showGrid(x=True, y=True)  #Mostramos una rejilla de fondo
 31  p.setLabel('left', 'Posicion sobre la viga', units='cm')
 32  p.setLabel('bottom', 'Tiempo de ejecucion', units='s')
#Ponemos nombre y unidades a los ejes
 33
 34  print("Aplicacion graficar")
 35  inicio=0
 36
 37  while True:
 38      #Menu para decidir si hacer gráfica y cómo o cerrar el
programa
 39      print("Elige el modo de conexión para recibir los da-
tos.")
 40      print("Opción 1 --> Conexión mediante sockets.")
 41      print("Opción 2 --> Conexión mediante puerto serie.")
 42      print("Opción 3 --> Cerrar la aplicación.")
 43      opcion=0
 44      while (opcion!=1 and opcion!=2 and opcion!=3):
 45          opcion = int(input('Introduce la opcion deseada: '))
 46
 47      if (opcion==1):
 48          print("Seleccionada la opción sockets.")
 49          try:
 50              s = socket.socket(socket.AF_INET, so-
cket.SOCK_STREAM)      # Creamos un socket TCP/IP
 51              HOST='157.88.201.42'      #Indicamos la dir IP que
esta utilizando la Raspberry
 52              PORT=7000               #Indicamos el puerto que
vamos a usar
 53              server_address = (HOST,PORT)
 54              s.connect(server_address)   #Solicitamos conec-
tarnos a un socket en la dirección y puerto especificados
 55          except socket.error:        #Para gestionar la excep-
ción en caso de error
 56              print("No se pudo conectar")    #Mostramos un
mensaje por pantalla, pero no cerramos la app
 57
 58      elif(opcion==2):
 59          print("Seleccionada la opción puerto serie.")
 60          if(inicio==0):  #esto solo se hará la primera vez
 61              puerto_serie = serial.Serial(       #Creamos un
objeto Serial, al que indicamos el puerto y las características de
transmisión
 62                  port='COM4',
 63                  baudrate = 115200,
 64                  parity=serial.PARITY_NONE,
 65                  stopbits=serial.STOPBITS_ONE,
 66                  bytesize=serial.EIGHTBITS,
```

```
 67                  timeout=None
 68              )
 69          inicio=1    #Para que en caso de parar y retomar la
gráfica, no vuelva a crearse otro objeto Serial
 70
 71      else:
 72          print("Cerrando aplicación...")
 73          sys.exit()      #Salimos de la aplicación
 74
 75      variable_parar_grafica=0
 76      print("Realizamos la gráfica")
 77
 78      def upDate():
 79          global Grafica1, Grafica2, gxData, tiempo, varia-
ble_parar_grafica
 80          tiempo+=0.2     #El periodo de muestreo será de 0.2s
 81          if(opcion==1):
 82              byt=s.recv(1024)    #Leemos los datos que lleguen
mediante sockets
 83              datos=byt.decode()
 84          else:
 85              byt = puerto_serie.read(8)      #Leemos los datos
que lleguen al puerto serie, siempre lo haremos en grupos de 8 ca-
racteres
 86              datos=byt.decode()
 87          datos = datos.strip()       #Elimina los espacios en
blanco iniciales y finales
 88          if(datos=="Parar"):     #Si recibimos la palabra
para, detenemos el proceso
 89              variable_parar_grafica=1
 90              print("Gráfica parada")
 91          else:
 92              vector_datos=datos.split(';') #para separar el
dato de la distancia del del setpoint
 93              distancia=int(vector_datos[0])
 94              dist_cm=distancia/10.0      #Pasamos el dato a
cm para la grafica
 95              Setpoint=int(vector_datos[1])
 96              setp_mm=Setpoint/10.0
 97              print("Distancia= %f "% dist_cm)
 98              print("Setpoint= %f "% setp_mm)
 99              #AÃ±adimos a la matriz un nuevo valor de tiempo,
distancia y setpoint
100              gData[0].append(tiempo)
101              gData[1].append(distancia)
102              gData[2].append(Setpoint)
103
104              if len(gData[1]) > 250:
105                  #Cuando las filas de la matriz superen los
250 datos, por cada uno que se añada se elimina el más antiguo
106                  gData[0].pop(0)
107                  gData[1].pop(0)
108                  gData[2].pop(0)
109                  p.setRange(xRange=[tiempo-50,tiempo])   #Ac-
tualizamos el eje X según avanza el tiempo
110
111              Grafica1.setData(gData[0],gData[1])     #Actuali-
zamos la gráfica de la distancia con el nuevo dato
```

```
112                 Grafica2.setData(gData[0],gData[2])      #Hacemos
lo mismo para la gráfica del setpoint
113
114             QtGui.QApplication.processEvents()
115
116         try:
117             while (variable_parar_grafica==0):
118                 upDate()    #Repetimos el proceso de lectura y
graficar datos hasta que la variable_parar_grafica cambie de valor
119         except:     #Si hay algún error al leer datos gestionamos
la excepción
120             print("Error en la recepcin de datos")      #Mandamos
mensaje de aviso y volvemos al menú principal
121  pg.QtGui.QApplication.exec_()
```

7. Guía de instalación y manual de uso.

Al estar este proyecto pensado para ser una práctica que en un futuro puedan realizar alumnos de la escuela de ingeniería, se ha realizado una guía de instalación y un manual de uso para dichos alumnos puedan acceder de forma sencilla a los programas necesarios y familiarizarse con el funcionamiento del sistema. Dichas guías se muestran a continuación:

7.1 Guía de instalación.

Para trabajar con el sistema son necesarios dos programas que se comunican entre sí. Uno es la aplicación principal, AppPrincipal, que se encuentra en la Raspberry Pi y maneja la ejecución del PID, además de controlar cuando comienzan y paran los procesos de graficar y visualizar el streaming. Por otro lado, tenemos la AppGraficar, que se ejecutará en el ordenador personal del alumno y se encargará de realizar una gráfica con los datos que se envían desde la Raspberry. A continuación, se explica detalladamente como descargar y lanzar cada una de las aplicaciones.

7.1.1 AppPrincipal.

Como ya hemos dicho, esta aplicación se encuentra en la Raspberry Pi, por lo que habrá que conectarse a ella para poder lanzarla. Además, como la app tiene entorno gráfico, va a ser necesario visualizar el escritorio, y no valdrá con tener acceso tan solo a la línea de comandos.

De las diferentes posibilidades que hay emplearemos el *Escritorio remoto*, por ser una funcionalidad que ya viene instalada en todos los ordenadores por defecto. Sin embargo, una limitación que tiene este método es que los dispositivos que se conectan por esta vía tienen que estar en la misma red VPN.

Para cumplir este requisito, y teniendo en cuenta que la maqueta está instalada en la escuela y la Raspberry Pi conectada a la VPN de la Uva, tendremos que conectarnos nosotros también a esta VPN. Esto lo conseguimos utilizando la aplicación *FortiClient VPN*. Los pasos que hay que seguir para instalarla y configurarla (esto solo habrá que hacerlo la primera vez) se encuentran en el siguiente enlace:

https://eii.uva.es/servicios/sinfor/index.php?tema=download

Una vez instalada y configurada, introduciremos nuestro usuario y contraseña de la Uva, y clicaremos en conectar:

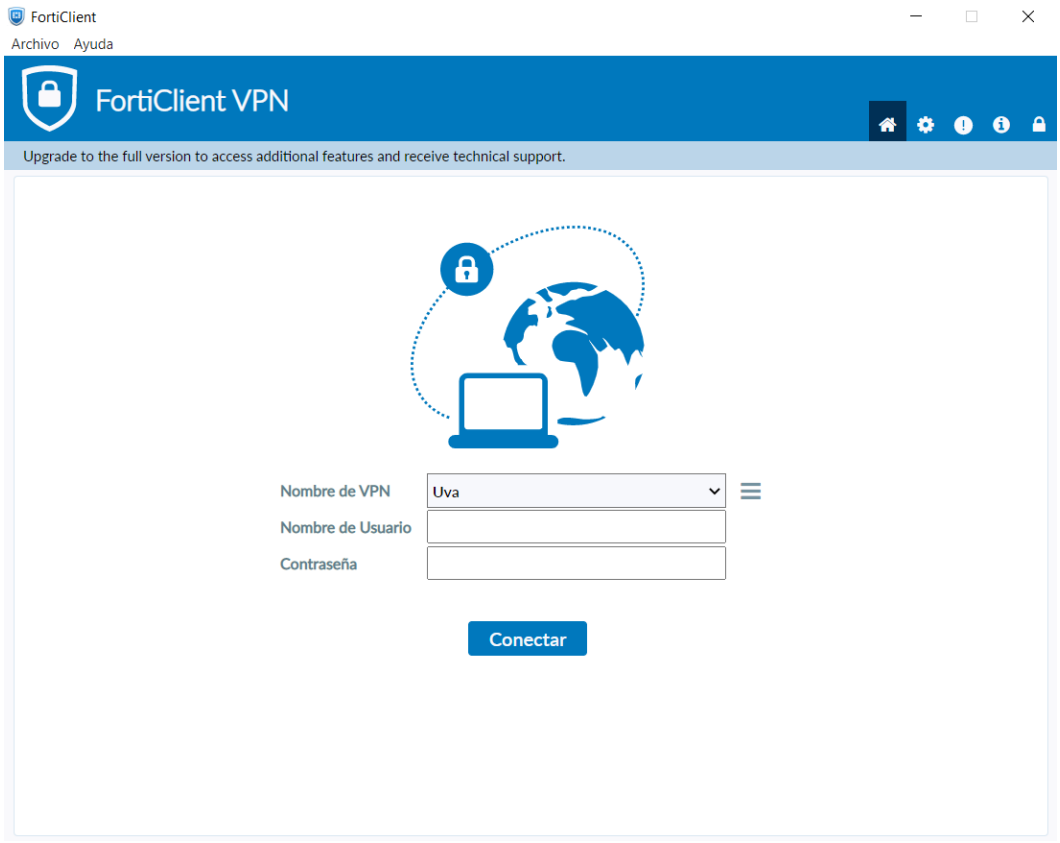


Figura 7.1 – Aplicación FortiClient VPN

El siguiente paso, será buscar en nuestro ordenador la aplicación *Escritorio remoto* y abrirla, veremos algo similar a esta ventana:

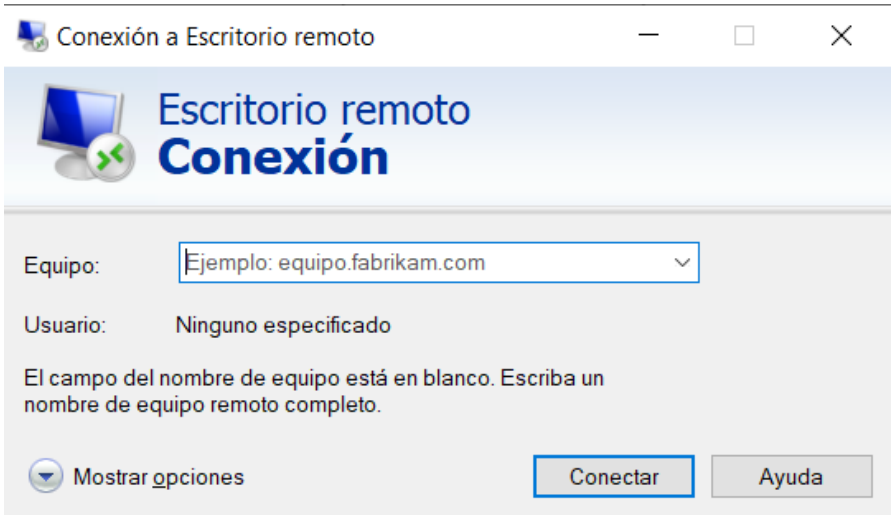


Figura 7.2 – Aplicación Escritorio Remoto

Para indicar el equipo al que queremos conectarnos introduciremos la dirección IP de la Raspberry Pi, que es **157.88.201.42**. Seguidamente daremos a conectar.

La siguiente ventana nos preguntará el usuario de la tarjeta al que deseamos conectarnos, introduciremos los siguientes datos:

Usuario → pi

Contraseña → balancin

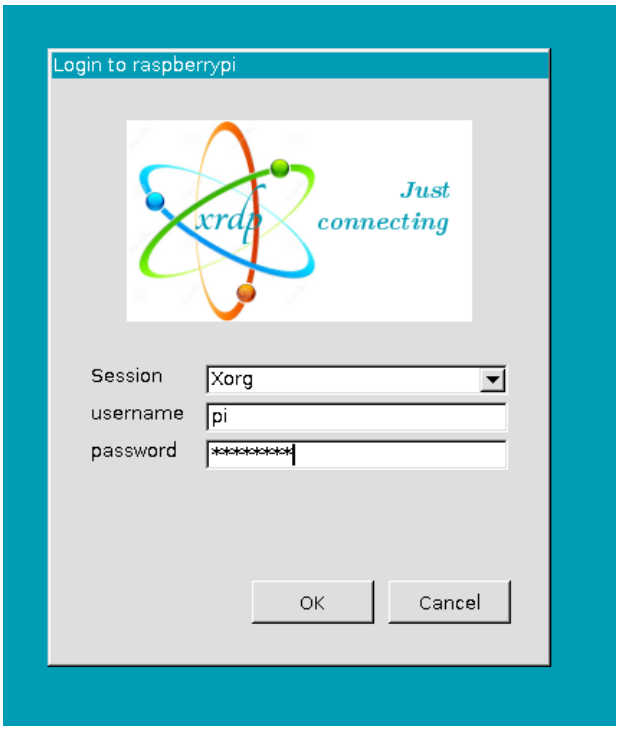


Figura 7.3 – Conexión a la Raspberry mediante Escritorio remoto

Tras esto, hemos completado el proceso y se deberá abrir una ventana en la que veremos el escritorio de la Raspberry, y podremos manejarla como si fuese nuestro ordenador. Para lanzar el programa AppPrincipal debemos abrir la línea de comandos, esto se hace clicando en el icono negro que se encuentra arriba a la izquierda.

Una vez abierta la línea de comandos iremos a la carpeta Desktop con el comando:

>> cd Desktop

Y lanzaremos el programa con el comando:

>> python AppPrincipal.py

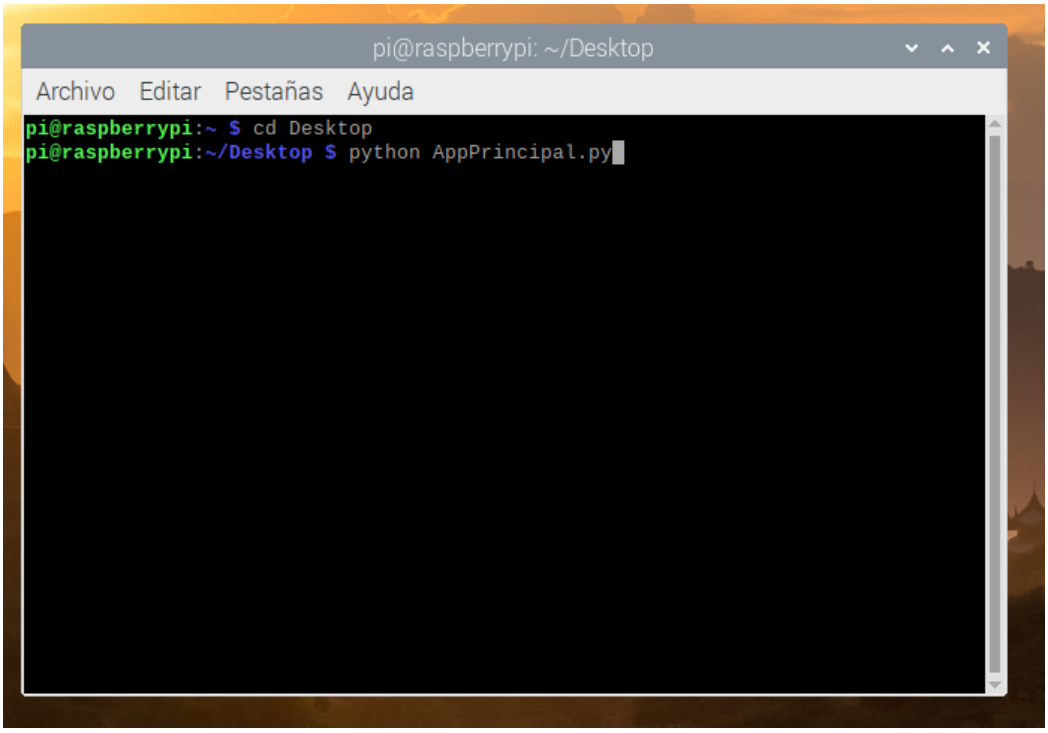


Figura 7.4 – Ventana de comandos de la Raspberry Pi

7.1.2 AppGraficar.

La aplicación AppGraficar es también un programa de Python, pero este deberemos lanzarlo desde nuestro ordenador. No obstante, no será necesario tener instalado un intérprete de Python, puesto que se ha creado un ejecutable del programa que incluye todo lo necesario para que funcione. Por tanto, será suficiente con descargar el ejecutable, lo cual podremos hacer en la siguiente página web.

https://controlbolaviga.wixsite.com/controlbolaviga/app-cliente

Yendo al anterior enlace iremos a un sitio web con información referente a la práctica, y en la que podremos descargar la aplicación AppGraficar clicando en el icono habilitado para ello.

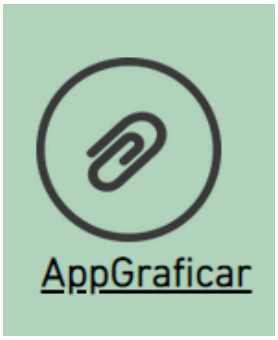


Figura 7.5 – Icono para descargar la AppGraficar

(Es posible que al tratar de descargarlo salga algún aviso de que el archivo puede ser peligroso, lo ignoramos y damos a continuar con la descarga)

Cuando la descarga haya finalizado tendremos un comprimido del cual podemos extraer la aplicación, que es un archivo .exe.

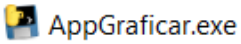


Figura 7.6 – Archivo ejecutable

Clicando dos veces sobre él se abrirá la aplicación, y podremos empezar a trabajar con ella.

7.2 Manual de utilización.

A continuación, explicaremos como trabajar correctamente con las aplicaciones. Ambas tienen un funcionamiento bastante intuitivo, pero teniendo en cuenta estos aspectos el trabajo con el sistema se hará aún más fácil.

Al abrir la AppPrincipal se mostrará la siguiente interfaz gráfica:

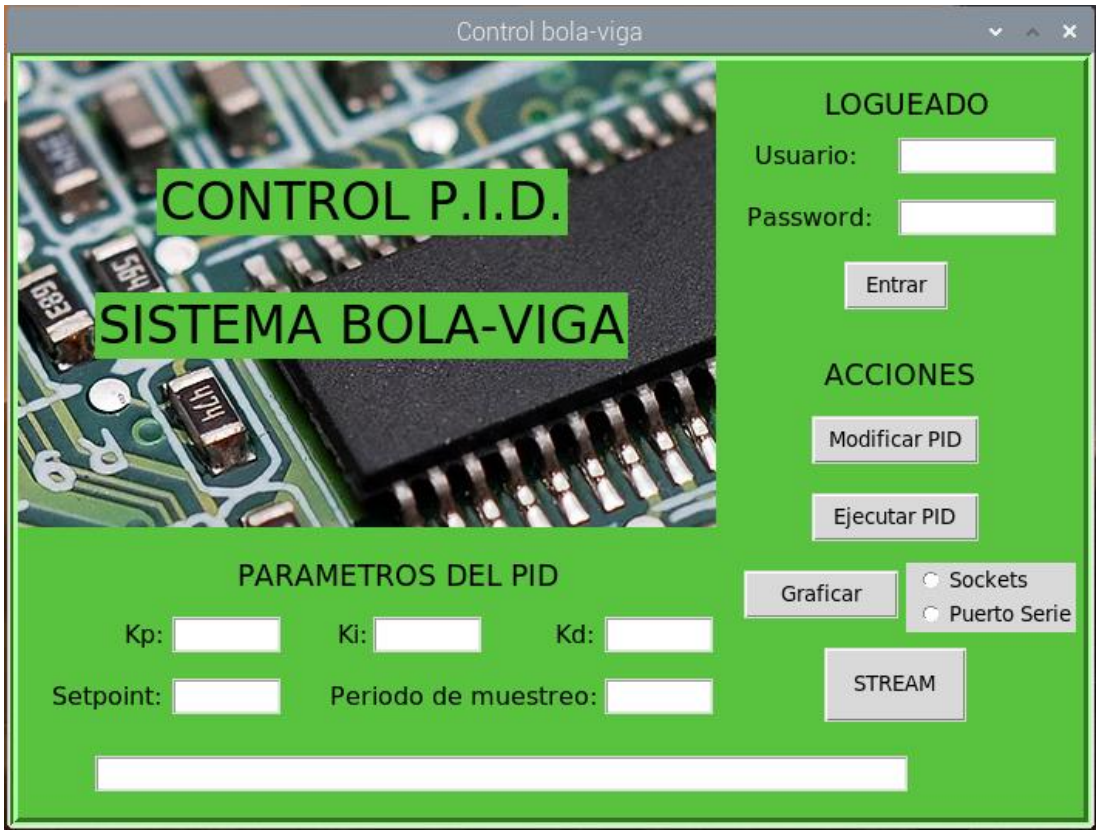


Figura 7.7 – Interfaz gráfica AppPrincipal

Con ella será con la que deberemos interactuar para controlar el sistema. No obstante, en la ventana de comandos se irá mostrando también información que puede ser interesante para el seguimiento del proceso, pero no habrá que interactuar con ella.

Lo primero que deberemos hacer será introducir el usuario y contraseña, pues de lo contrario no se podrá trabajar con la aplicación. Estos son:

Usuario → Usuario1

Contraseña → 1234

Tras este paso, ya tendremos acceso a todas las funcionalidades de la aplicación. No hay ningún orden en concreto en el que haya que hacer cada paso, pero seguiremos el que sería el más lógico para explicarlo.

En caso de estar trabajando con la práctica de forma remota lo primero que haremos será comenzar a visualizar la retrasmisión en directo del vídeo de la planta. Para ello en la aplicación pulsamos el botón *STREAM*, que comenzará este proceso. Para ver la retransmisión tendremos que ir a nuestro escritorio y abrir el navegador, en el que buscaremos la dirección IP de la Raspberry y el puerto en el que está retransmitiendo. Introducimos lo siguiente:

157.88.201.42:8080

Deberá abrirse una página como esta en la que se vea la maqueta:

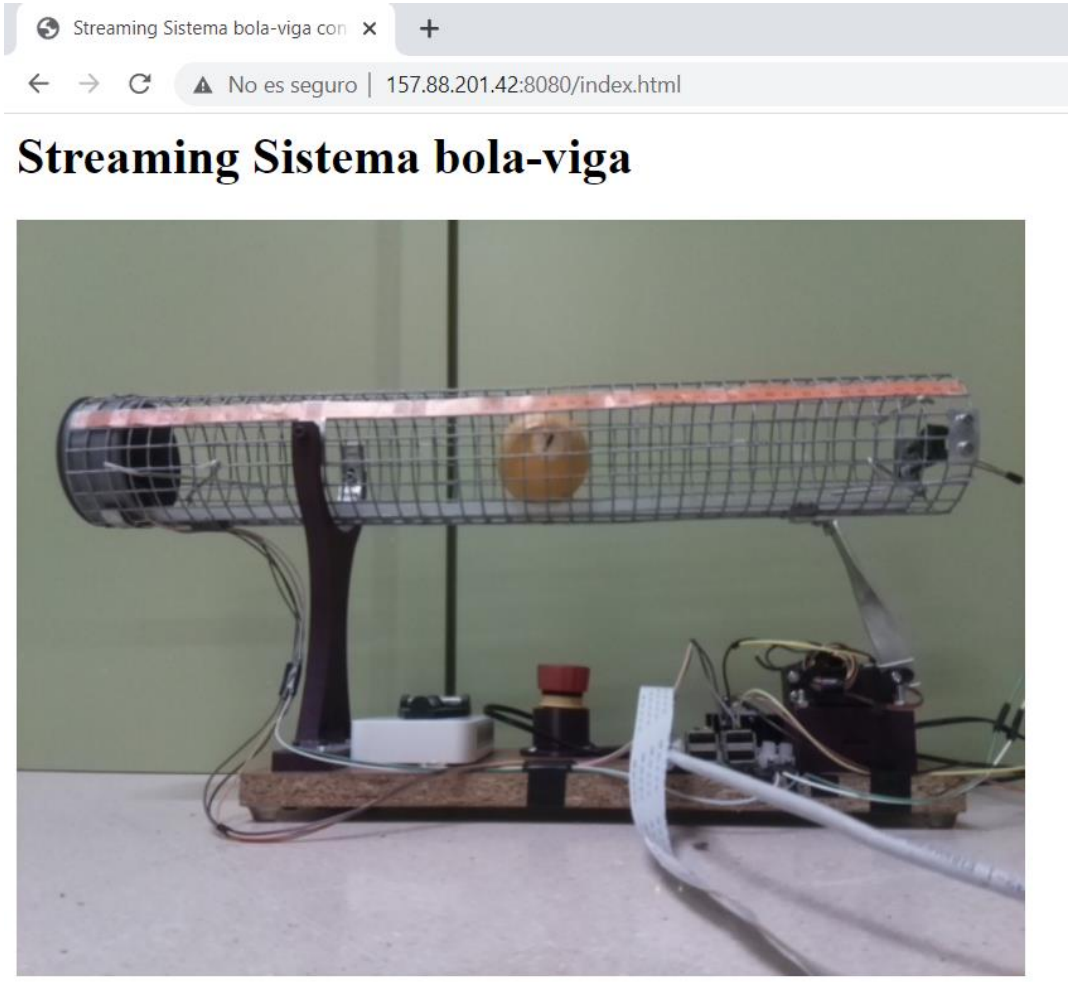


Figura 7.8 – Retransmisión vía web del streaming

Aprovechando que nos encontramos en nuestro escritorio, lo siguiente que haremos será abrir la Appgraficar, que nos mostrará un menú con tres opciones:

- Opción 1 → Graficar los datos que se reciban mediante sockets
- Opción 2 → Graficar los datos que se reciban por puerto serie.
- Opción 3 → Cerrar la aplicación.

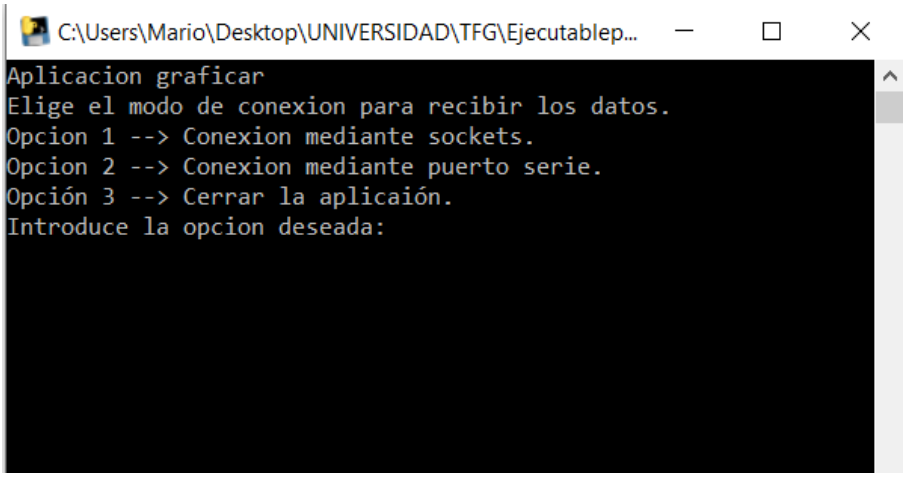


Figura 7.9 – Menú de la AppGraficar

No obstante, todavía no elegiremos ninguna opción, puesto que la comunicación debe empezarse desde el lado de la Raspberry. Volvemos por tanto a la AppPrincipal y seleccionamos el tipo de comunicación que queremos utilizar. Tras esto pulsaremos en el botón *Graficar*.

Figura 7.10 – Botones para el control de la gráfica

Una vez hecho esto volveremos a la AppGraficar y ahora sí introduciremos la opción deseada. Acto seguido en la ventana de la gráfica deberá empezar a verse la gráfica de la posición de la bola y la del Setpoint frente al tiempo. En caso de que se elija la opción mediante puerto serie, es importante asegurarse de que el ordenador está leyendo el puerto USB conectado y que lo hace en el COM4, pues en caso contrario no funcionará.

Al haber empezado a graficar, el botón que habíamos pulsado en la AppPrincipal ahora mostrará el texto de *Parar,* esto es porque cuando le pulsemos detendrá la gráfica. Tras esto, si se desea volver a graficar, habrá que seguir el mismo proceso de nuevo.

Con esto ya tendremos todo preparado para empezar a trabajar con el control PID. En la parte inferior de la interfaz tenemos unos recuadros que servirán para introducir los valores que queremos probar de ganancias de los

términos proporcional, derivativo e integral, de setpoint y de periodo de muestreo. Para hacerlos efectivos debemos pulsar en el botón *Modificar PID*.

Figura 7.11 – Botón Modificar PID

Este botón podrá pulsarse en cualquier momento, incluso durante la ejecución del control, y los valores se actualizarán al instante.

Para comenzar la ejecución del control PID, habrá que pulsar el botón *Ejecutar PID*. Al hacerlo el texto de este botón cambiará y mostrará "*Parar PID*", pulsándolo nuevamente la ejecución del control se detendrá.

Figura 7.12 – Mismo botón para ejecutar y parar el PID

Cuando hayamos terminado de trabajar con la práctica, detendremos los procesos de ejecución y graficar y cerraremos la aplicación clicando en la "x" situada arriba a la derecha. Para detener el subproceso que está retransmitiendo vídeo, iremos a la ventana de comandos e introduciremos Ctrl+C. En la AppGraficar introduciremos en el menú la tercera opción, cerrar aplicación.

8. Sitio web.

Uno de los objetivos que se planteaban al principio del proyecto, era la creación de una página web para facilitar a los alumnos el acceso y uso a las herramientas de la práctica. Se optó por este método debido al gran auge que ha tenido en los últimos años y las grandes posibilidades que ofrece, puesto que de esta forma es posible acceder a toda la información relativa al proyecto en cualquier momento, desde cualquier lugar y dispositivo, siempre y que se tenga conexión a internet.

Para ser correctos, lo que se ha creado es un sitio web, no una página. Una página web es tan solo un documento al que se accede en internet mediante una URL, mientras que el sitio web es un conjunto de varias páginas web. En concreto, nuestro sitio cuenta con cuatro páginas web diferentes. No obstante, coloquialmente está aceptado el uso de página web para denominar a un sitio web entero.

Para crear un sitio web, que no es otra cosa que un conjunto de archivos de código fuente alojados en un servidor, se utilizan habitualmente los lenguajes HTML, CSS y Javascript conjuntamente. Cada uno de ellos, se encarga de una parte del diseño de la página web. El proceso de aprendizaje de estos tres lenguajes y la creación desde cero de un sitio web es una labor muy pesada y que llevaría muchísimo tiempo, no obstante, existen ciertas plataformas que te permiten crear sitios web utilizando plantillas. Hay muchas de estas plataformas, como pueden ser Wordpress, Wix, Jimbo, Website Builder, Weebly… y muchas más. Gracias a ellas, es posible diseñar páginas web estáticas sin un conocimiento profundo de su programación y en un periodo de tiempo más asequible. Además, proporcionan un dominio gratuito de internet donde se alojará la web, en caso contrario sería necesario pagar por ello. No obstante esto último es a cambio de incorporar publicidad de la plataforma, tanto en el sitio web como en el propio nombre.

De todas las opciones disponibles, me decanté por Wix debido a las múltiples posibilidades de diseño que ofrecía, que incluía menos publicidad que otras similares, y sobre todo porque permite subir archivos que más tarde los usuarios puedan descargar, lo cual es necesario para hacerlo con la AppGraficar.

Figura 8.1 – Logo de la plataforma Wix.

El nombre del sitio web creado es controlbolaviga, y la dirección para encontrarla en internet es la siguiente:

https://controlbolaviga.wixsite.com/controlbolaviga

Al introducir dicha dirección en el buscador, nos lleva a la siguiente página:

Figura 8.2 – Página web: Inicio.

En ella podemos ver como la parte superior está destinada a un encabezado, en el cuál aparece el título del proyecto “Control PID de un sistema bola-viga”. El fondo de este encabezado en una imagen de un circuito electrónico, en referencia a la Raspberry Pi que utilizamos, y que es la misma que se utiliza en la AppPrincipal. Además, como color de fondo de la página se ha elegido el color verde para que está en concordancia también con la aplicación.

Esta primera página que vemos es el inicio, en ella se hace una descripción del proyecto y se incluyen imágenes del esquema del sistema y de la maqueta real.

Justo debajo del encabezado, vemos que hay un menú con cuatro opciones distintas que nos permitirán ir a las distintas páginas de las que está compuesto nuestro sitio web:

- INICIO
- COMPONENTES
- CONTROL PID
- APP GRAFICAR

La primera, Inicio, es la que hemos comentado anteriormente. La segunda, Componentes, luce como se muestra en la siguiente imagen:

Figura 8.3 – Página web: Componentes.

En ella se realiza una breve descripción de cada uno de los elementos que vimos en el apartado 4, incluyendo también imágenes de cada uno de ellos. El encabezado es el mismo que en la página principal, y lo será también en las dos siguientes puesto que es común para todas ellas. Pese que las capturas de pantalla mostradas no permiten ver las páginas al completo, no se incluirán más con el objetivo de no hacer excesivamente larga esta memoria, puesto que puede clicarse en el enlace de la dirección del sitio para verlo al completo.

La tercera página del menú es Control PID:

Figura 8.4 – Página web: Control PID.

Aquí se comentan los aspectos teóricos más relevantes de control PID, explicando los efectos de cada uno de los tres términos, de forma similar a como se hace en el apartado 5 de esta memoria.

Y por último, tenemos la página AppGraficar:

Figura 8.5 – Página web: AppGraficar.

En ella, además de mostrarse información relativa a la aplicación y la manera de usarla, se incluye un apartado en la parte inferior para su descarga:

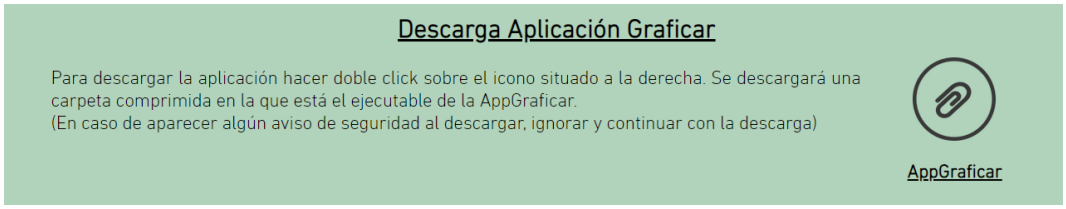


Figura 8.5 – Página web: AppGraficar.

Como la plataforma de diseño web no permitía incluir archivos ejecutables para ser descargados, ha sido necesario colocar el archivo dentro de una carpeta y comprimirla para poder incluirla. Por tanto, los alumnos tan solo tendrán que clicar en el icono que se ve a la derecha, descomprimir el archivo que se descargue, y dentro de la carpeta tendrán el ejecutable de la AppGraficar.

9. Pruebas de funcionamiento.

9.1 Introducción.

Por último, para finalizar la memoria vamos a hacer unas pruebas de funcionamiento de la práctica, simulando lo que en un futuro deberán hacer los alumnos con ella. Por tanto, veremos cómo funciona la ejecución del control PID, mandando la bola a diferentes posiciones a lo largo de la viga. Esto lo haremos con diferentes valores de los términos proporcional, derivativo e integral, para ver empíricamente cuál es el efecto de aumentar o disminuir cada uno de ellos. Para ilustrar en la memoria todas estas pruebas, haremos uso de las gráficas que realiza la AppGraficar, y de esta forma además se demostrará el funcionamiento del sistema al completo.

Existen diferentes métodos de ajuste de los controladores PID, tanto teóricos como prácticos. Los métodos teóricos requieren el desarrollo de un modelo dinámico del sistema con el que poder utilizar ciertos softwares diseñados para ayudarnos en esta labor. Por otro lado los métodos prácticos o empíricos son aquellos en los que se utilizan pruebas realizadas directamente con la planta real para encontrar los valores más adecuados de los parámetros. Como el enfoque del proyecto es el estudio práctico del control, se usará un método del segundo tipo.

A la hora de comenzar a buscar los valores adecuados, hay ciertos métodos pueden ayudarnos a encontrar unos valores próximos, aunque estos nunca serán los definitivos puesto que igualmente requieren de ajuste fino. Algunos de estos métodos son los de Zieglers y Nichols o el de Cohen-Coon. No obstante, para poder aplicar estos métodos es necesario que el sistema cumpla con determinadas características según el caso, como puede ser que el sistema sea estable en lazo abierto o llevar la planta al estado de estabilidad crítica. Como nuestro sistema no cumple estos requisitos, no es posible usar ninguno de estos métodos y el ajuste deberá hacerse completamente de forma manual. Sin embargo, debido a la sencillez de la planta, no habrá problemas para encontrar los valores adecuados.

Como último apunte teórico antes de comenzar con las pruebas, cabe recordar brevemente la forma típica de la respuesta de un proceso que emplea un controlador PID. Se muestra en la siguiente imagen:

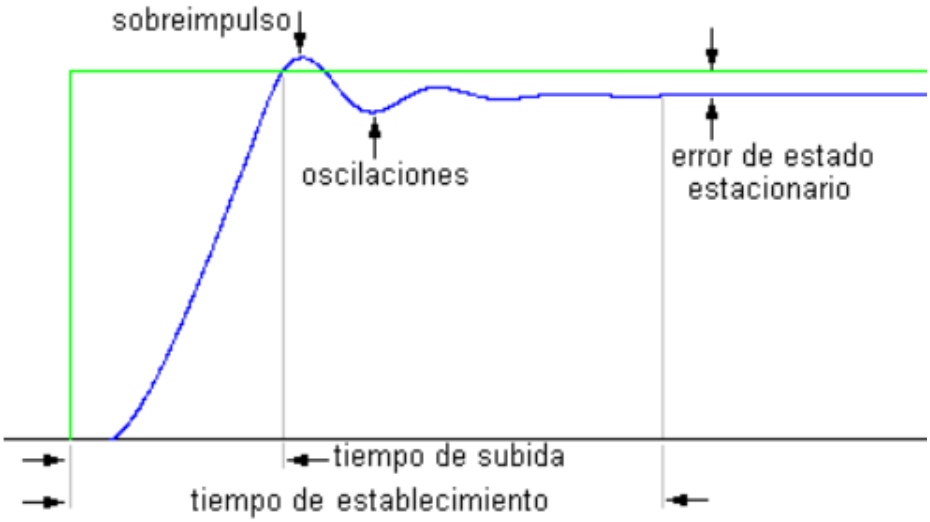


Figura 9.1 – Respuesta típica de control PID.

En ella podemos ver dos señales superpuestas representadas frente al tiempo. La verde es el setpoint, y representa un escalón, que es el tipo de señal de entrada que se va a estudiar con esta práctica. La entrada en escalón consiste en un cambio instantáneo del setpoint desde un valor constante hasta otro. La señal azul se corresponde con la salida del sistema, la cuál debe volver a ajustarse al valor de la señal de entrada en el menor tiempo posible.

De la señal de salida pueden estudiarse ciertas características que nos indicarán como de buena es la respuesta del sistema. Son las siguientes:

- Tiempo de subida: Representa el tiempo que tarda la señal en cambiar de valor. Existen diferentes criterios para medirlo, aunque el más extendido es tomar el tiempo de subida como el que pasa desde que la señal alcanza el 10% de su valor final hasta que llega al 90%. (En la figura anterior se toma el 0 y el 100% para medirlo).
- Tiempo de establecimiento: Se corresponde con el tiempo que tarda la señal en fijarse en el valor deseado. Para ello se toma un margen de error aceptable, en torno al 2 o 5% del valor del setpooint, y cuando la señal de salida no sale de ese rango, se considera la señal estabilizada.
- Sobepico: El sobrepico es el fenómeno que se produce cuando la señal de salida llega a la referencia deseada pero debido a la inercia del sistema lo sobrepasa, generando un error en el sentido contrario.
- Error estacionario: Se denomina así a la diferencia existente entre el valor de la señal de salida y el valor de la referencia cuando el sistema se ha estabilizado. El objetivo del control debe ser reducir esta diferencia al mínimo valor posible.

- Estabilidad: Es la característica principal del sistema. El sistema es estable cuando alcanza un valor final en el que se mantiene, e inestable cuando la salida tiende a infinito en algún momento.

9.2 Pruebas.

Para empezar, vamos a observar el efecto del controlador proporcional, que es el más simple de los tres. Para ello comenzamos poniendo el valor de los términos derivativo e integral a cero, para que solo actúe el término proporcional. Como señal de entrada se usará un escalón de 25 centímetros, por lo que el setpoint pasará de 10 a 35. Esta referencia es la que se usará para todas las pruebas.

Como en principio no sabemos de qué orden será este valor, empezamos con un valor bajo para evitar que el sistema sea inestable y haga movimientos demasiado bruscos que afecten a la planta. Tras probar con valores del orden de 0,01 se observa que el efecto que tiene el término proporcional sobre la señal de control es tan pequeña que es despreciable y no afecta al sistema. Por tanto, vamos aumentando la ganancia hasta que vemos que en valores superiores a 1 comienza a responder.

- Kp=1’2 -- Ki=0 -- Kd=0:

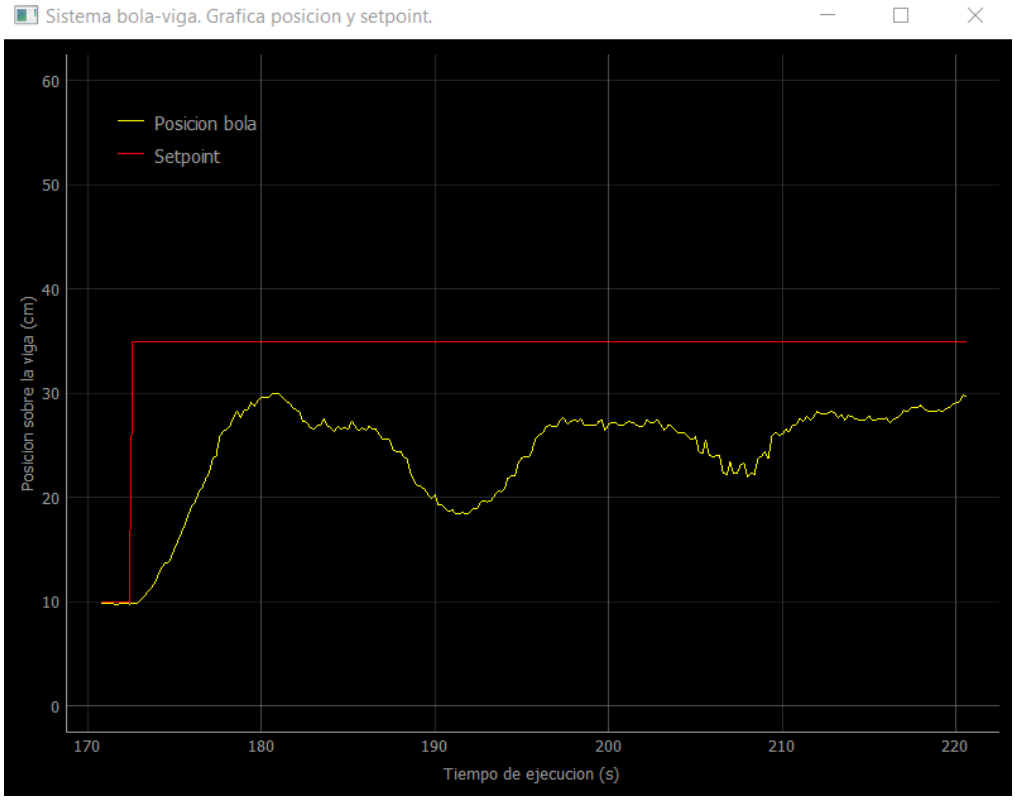


Figura 9.2 – Respuesta del sistema con Kp pequeña.

Como podemos ver en la gráfica, para una ganancia proporcional pequeña, de 1,2, la bola se desplaza lentamente y no llega a alcanzar el valor del setpoint deseado. Podemos apreciar también que, pese a ser estable, oscila demasiado y el tiempo de establecimiento es excesivamente grande. Probaremos por tanto a seguir aumentando la ganancia proporcional, con el objetivo de que la salida llegue a alcanzar el valor de la referencia.

- Kp=2'5 -- Ki=0 -- Kd=0:

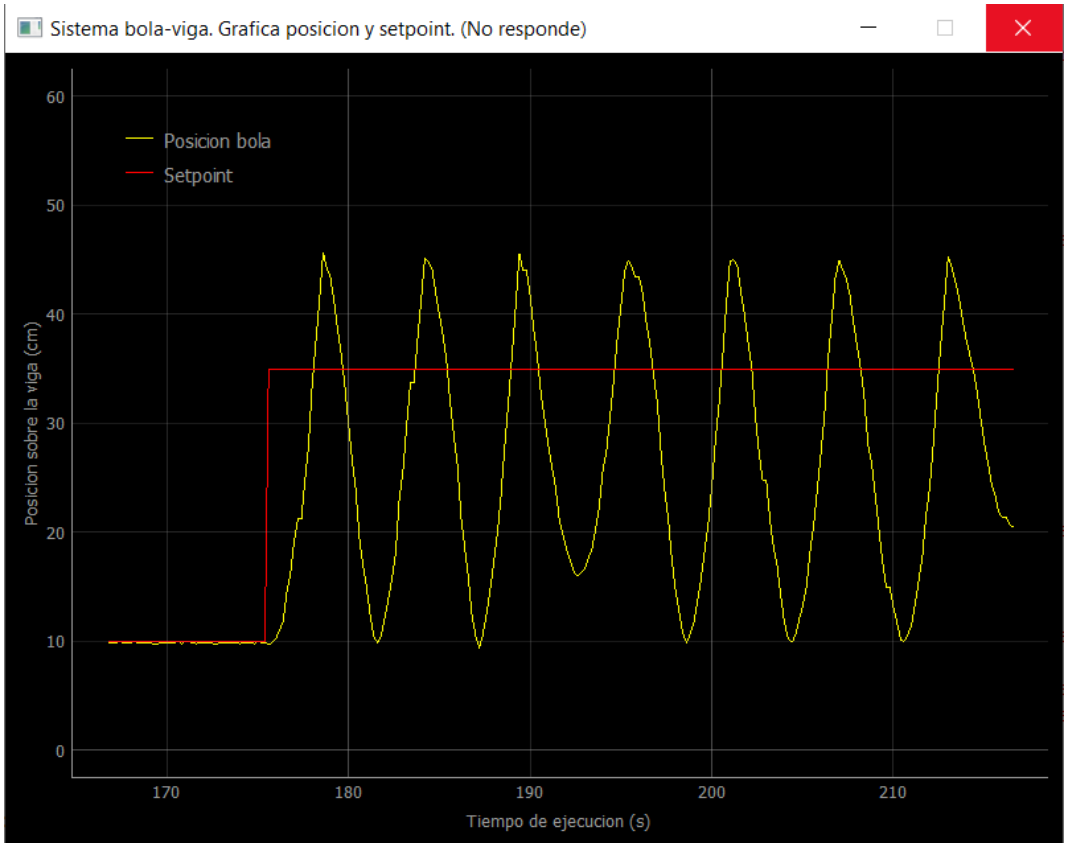


Figura 9.3 – Respuesta del sistema con Kp grande.

Ahora se ha utilizado un valor de Kp mayor, y efectivamente el sistema consigue alcanzar la referencia y lo hace en un tiempo más pequeño (disminuye el tiempo de subida), pero como consecuencia el sistema se ha vuelto inestable. Lo que sucede es que el término proporcional es mayor, y para llevar a la bola a la posición deseada, hace que esta coja una velocidad demasiado elevada y cuando llega al punto de referencia, se pasa de largo. En ese mismo momento, el término toma valores del signo contrario, que hacen que la viga se incline hacia el lado opuesto. El resultado es nuevamente el mismo y cuando la bola llega a la posición deseada se pasa de largo debido a la inercia. Lo único que evita que el valor de salida tienda a infinito son los límites físicos de la maqueta, ya que la bola queda rebotando de un lado al otro constantemente.

Para solucionar este problema debido a la inercia del sistema introducimos el término derivativo. Como ya habíamos comentado previamente, este término calcula la derivada del error, es decir, la velocidad a la que varía, para así poder predecir errores futuros y evitarlos.

- Kp=2'5 -- Ki=0 -- Kd=1:

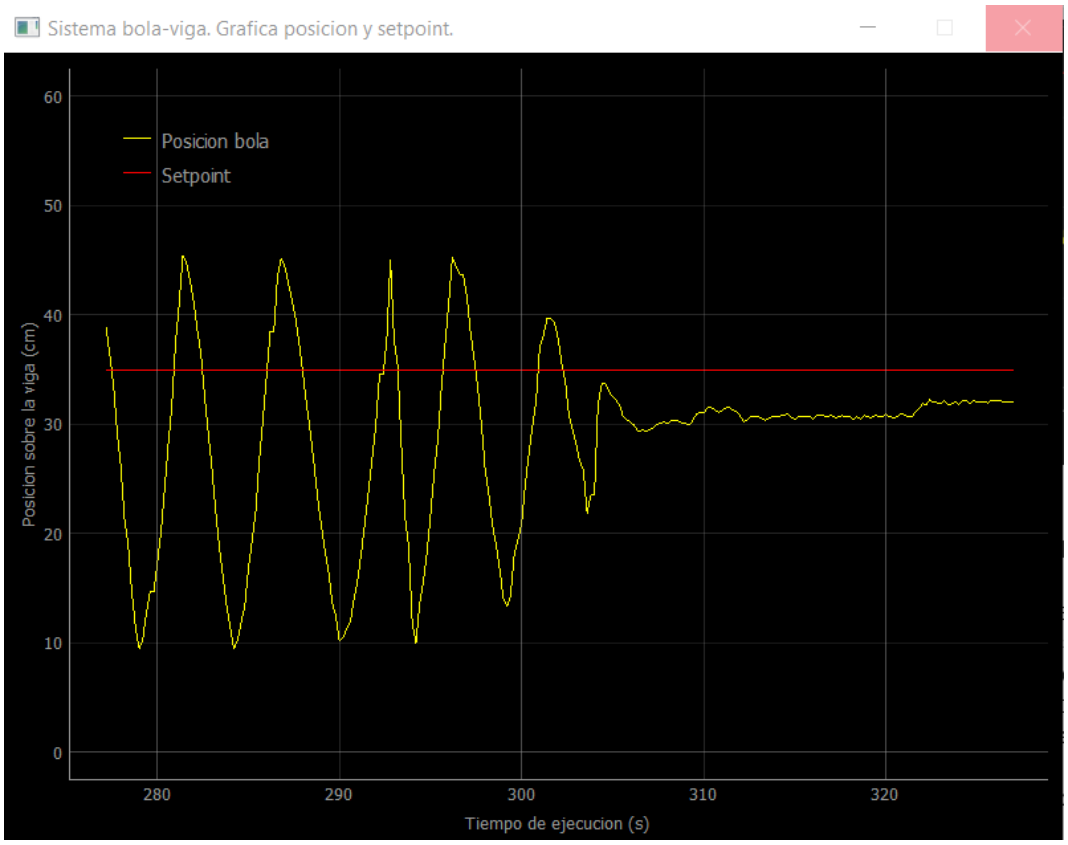


Figura 9.4 – Respuesta del sistema introduciendo Kd.

El resultado lo vemos en esta gráfica. Los nuevos valores del PID se introducen aproximadamente en el segundo 290, y a partir de ahí vemos como las oscilaciones que se estaban repitiendo previamente van disminuyendo su amplitud hasta que la bola se estabiliza en una posición fija. De esta forma hemos conseguido que el sistema vuelva a ser estable gracias al efecto de la acción derivativa, que en este caso actúa como un freno, impidiendo que la velocidad que alcance la bola sea excesivamente rápida.

No obstante, puede apreciarse que la respuesta continúa sin ser buena, puesto que aún no se alcanza la señal de referencia. Para conseguir que la bola llegue a la posición deseada, vamos a aumentar aún más la ganancia proporcional, y haremos lo mismo con la derivativa para que vayan compensadas.

- Kp=7 -- Ki=0 -- Kd=3:

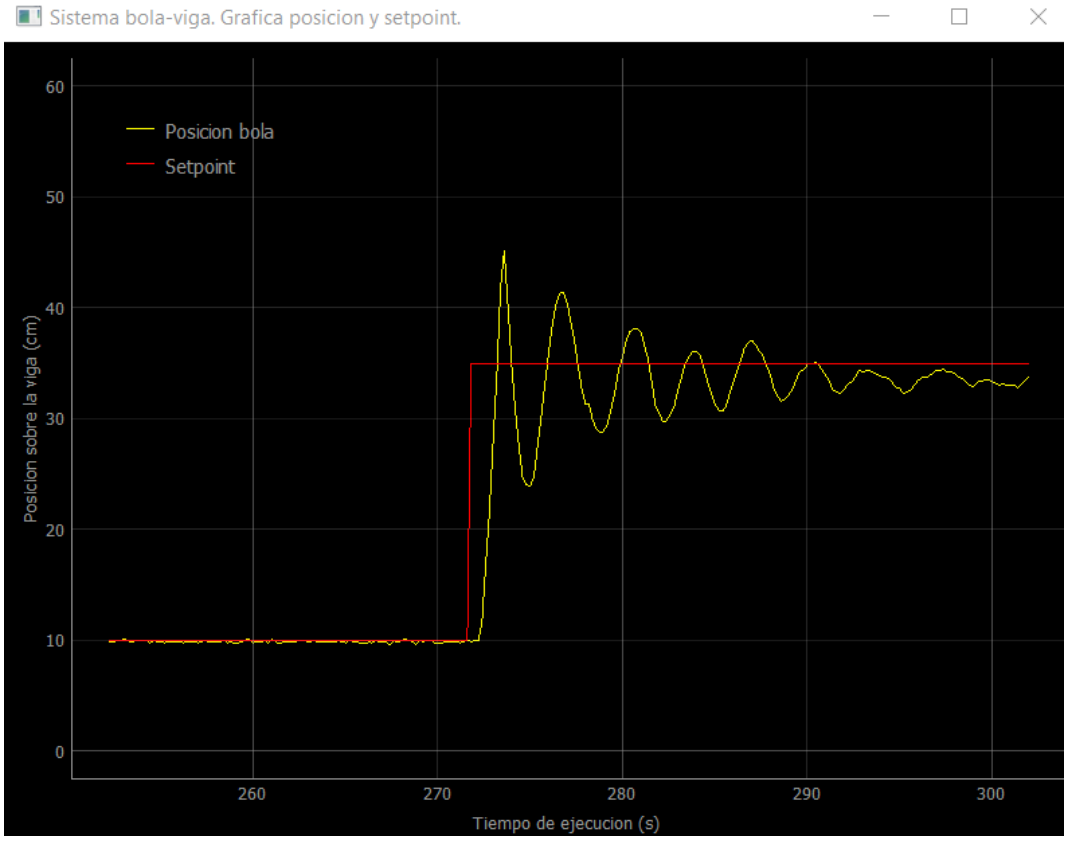


Figura 9.5 – Respuesta del sistema con aumentando Kp y Kd.

Ahora sí podemos observar cómo obtenemos una respuesta mucho más parecida a lo que estamos buscando. El sistema es estable y error estacionario se ha reducido hasta valores mucho más pequeños, además de que el tiempo de subida se ha reducido como consecuencia de que el sistema es más rápido.

Por contra a esta última característica, tenemos un sobrepico bastante grande, de unos 10 cm, lo que supone un 40% del valor del escalón. Además debido a esto se producen bastantes oscilaciones hasta que la posición de la bola se estabiliza, por lo que tenemos un tiempo de establecimiento bastante grande. Para reducir este efecto, habrá que incrementar aún más la ganancia derivativa, puesto que es la que se encarga de frenar la bola según se acerca a la posición deseada, con el fin de que no se pase al llegar.

- Kp=7 -- Ki=0 -- Kd=6:

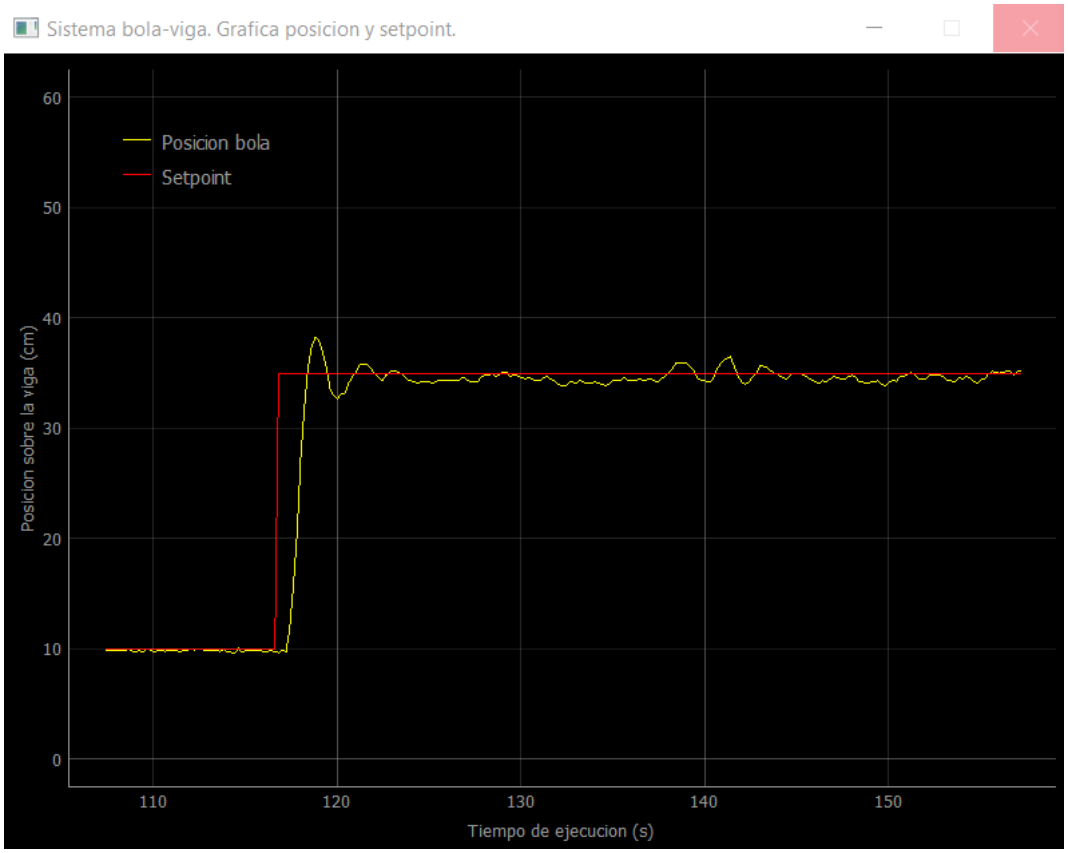


Figura 9.6 – Respuesta del sistema con aumentando Kd.

Al hacerlo vemos como resultado que el sobrepico se ha reducido considerablemente frente la prueba anterior, y ahora tan solo alcanza unos 3cm aproximadamente, lo que supone un 12% respecto al escalón. Como consecuencia de la reducción de este sobrepico también se nota cómo ha disminuido el tiempo de establecimiento, puesto que las oscilaciones son más pequeñas y duran menos.

Sin embargo, pese a estas mejoras en la respuesta obtenida puede observarse como aún se produce un pequeño error estacionario, puesto que la señal de salida queda ligeramente por debajo del setpoint establecido. Para corregir esta desviación se introduce el término integral. No obstante, antes de ello haremos una prueba más variando la ganancia derivativa, y posteriormente veremos el efecto de término integral introduciendo además una perturbación constante para que dicho efecto se aprecie mejor.

Para la prueba que se va a realizar acerca del término derivativo vamos a aumentar aún más el valor de su ganancia, para ver lo que sucede si se escoge un valor demasiado grande para esta constante.

- Kp=7 -- Ki=0 -- Kd=10:

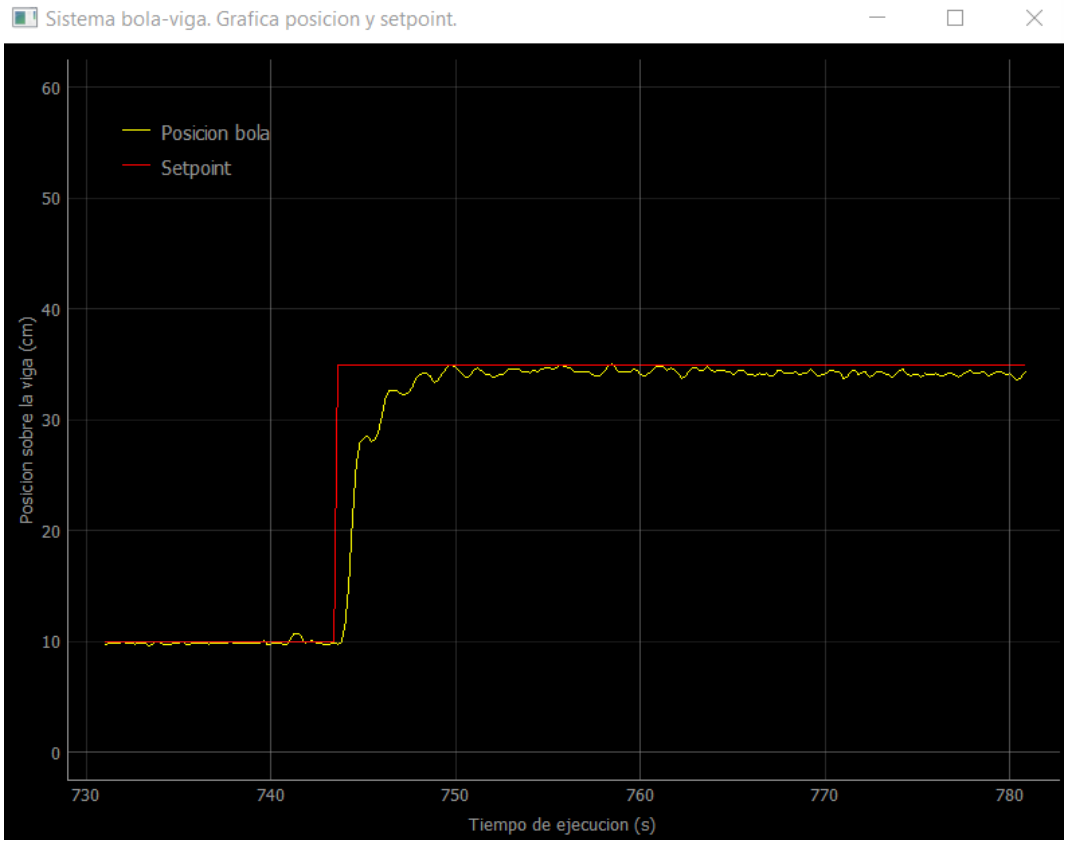


Figura 9.7 – Respuesta del sistema con Kd demasiado grande.

Como se puede observar, al escoger un valor de Kd aún mayor se acentúan las características que habíamos visto anteriormente. Al tener más peso el término proporcional, este consigue frenar mucho más la bola, hasta el punto de que la detiene por completo antes de llegar a la posición deseada. Como consecuencia de esto, la bola llega mucho más despacio al setpoint y el tiempo de subida y el de establecimiento aumentan. Por el contrario, tenemos la ventaja de que no se produce ningún tipo de sobrepico, lo cuál puede ser necesario en alguna ocasión, dependiendo del sistema con el que se esté trabajando. Comentar también, que si se toma un valor de la ganancia derivativa mucho mayor provocará que el sistema sea inestable.

Y ahora sí como última prueba vamos a ver el efecto y la utilidad que tiene la constante integral. Su principal función es eliminar el error estacionario, que es aquella diferencia constante entre la señal de salida y la deseada que se mantiene una vez que ha finalizado el periodo transitorio. Como el error que teníamos en el caso anterior era muy pequeño, vamos a introducir una perturbación en el sistema que lo incremente. Dicha perturbación va a consistir en desnivelar la maqueta, subiendo uno de los lados para que la viga pierda la horizontalidad en estado de reposo y se genere el error.

En la siguiente prueba se mostrará el funcionamiento del control que dimos por bueno antes (Kp=7, Ki=0 y Kd=6) ante la nueva perturbación, y se introducirá entonces una ganancia integral para ver cuál es su efecto.

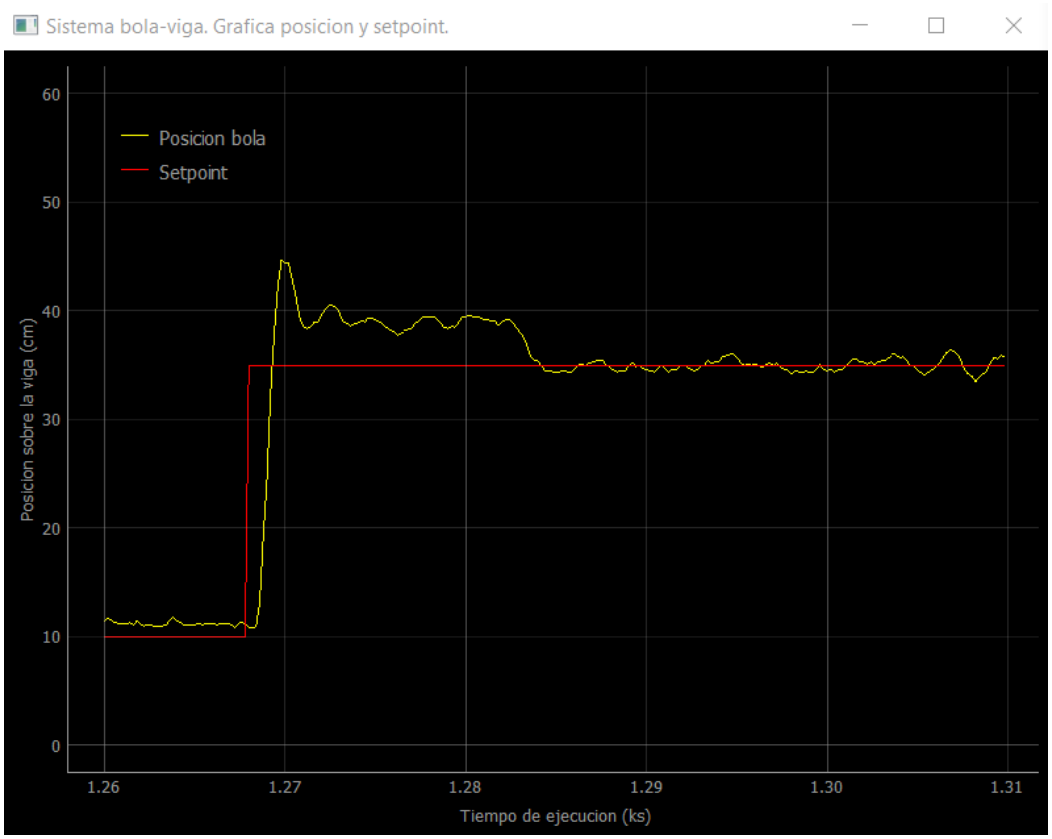


Figura 9.5 – Respuesta del sistema mostrando el efecto de Ki.

En la gráfica que obtenemos podemos ver perfectamente cómo al producirse el salto en el setpoint el ajuste anterior del PID continúa siendo capaz de seguir la señal y estabilizarse, pero ahora se genera un error estacionario mucho mayor. La bola se estabiliza entorno a los 39 cm, produciéndose una diferencia de 4 cm entre la señal de salida y la deseada.

Posteriormente, se modifican los valores del PID, manteniendo las ganancias proporcional y derivativa igual, pero asignando el valor de 1 a la ganancia integral. Como ya sabemos, el término integral actúa sobre los errores pasados, acumulando su nuevo valor a los antiguos en cada ejecución, por lo que pequeñas desviaciones respecto al setpoint acaban teniendo mucho peso y corrigiendo el error. Y eso es lo que podemos observar que se produce, en cuanto entra en juego el término integral la bola comienza a acercarse al valor deseado hasta que se reduce a cero el error estacionario.

El lado negativo de este término es que puede llevar al sistema fácilmente a la inestabilidad, por lo que su ganancia debe ser muy pequeña. Si por el ejemplo se le asigna el valor de 2, el sistema se vuelve inestable.

10. Conclusiones.

Finalmente, este último capítulo voy a dedicarlo a analizar cómo se ha llevado a cabo el desarrollo del proyecto y si se han logrado los objetivos planteados al inicio del mismo. Además, comentaré las posibles líneas futuras y ámbitos de mejora que creo que puede tener el proyecto.

10.1 Conclusiones técnicas.

Para comprobar si se han logrado los objetivos planteados al inicio del trabajo vuelvo a enumerar estos a continuación:

1 – Realizar el control en lazo cerrado de la posición de la bola sobre la viga.

2 – Crear una aplicación desde la que se introduzcan los parámetros del control PID.

3 – Crear una aplicación que permita graficar la situación del sistema, es decir, la posición objetivo de la bola y la real.

4 – Comunicar ambas aplicaciones mediante puerto serie y sockets para que se pueda trabajar presencial o remotamente.

5 – Encontrar los parámetros adecuados para el PID.

6 – Crear una página web desde la que se pueda acceder a la aplicación.

7 – Realizar un vídeo en directo de la planta para que se pueda manipular a distancia.

El primer objetivo que se planteaba, básico para la realización del resto del proyecto, era conseguir llevar a cabo el control en lazo cerrado de la posición de la bola sobre la viga. Esto se ha realizado satisfactoriamente tal y como se ha demostrado en el capítulo 9 de pruebas con la maqueta.

Como parte negativa en este punto, hay que comentar las limitaciones que se han encontrado respecto a los componentes del sistema. Se descubrió que los sensores de los que se disponía no trabajaban bien en las condiciones que ofrecía esta maqueta, puesto que al medir dentro de la malla su rango de medida se reducía a poco más de 20 cm. Esto se solucionó colocando un sensor a cada lado y acoplando la medida de ambos para que dieran como resultado una sola. Los resultados obtenidos finalmente fueron buenos teniendo en cuanta la problemática de la que partíamos, pero ha quedado una

zona central, de los 22 a 26 cm, en la que las medidas obtenidas son de peor calidad, lo cual afecta al sistema si se trata de estabilizar la bola en este punto.

Además, hay que mencionar también las vibraciones producidas debido al servomotor, que quizá por ser reutilizado y tener bastante tiempo de uso, su comportamiento no gozaba de la precisión requerida para tener una estabilidad mayor de la bola.

Los objetivos 2, 3 y 4, también se han llevado a cabo con éxito, y las aplicaciones desarrolladas funcionan y se comunican correctamente. Esto se ha demostrado también durante la realización del capítulo 9.

En el capítulo 9 sirvió además para lograr el objetivo 5, que era encontrar un ajuste del control PID que llevara la bola a la posición deseada sin error y lo más rápidamente posible.

El objetivo 6 se ha desarrollado correctamente y hay actualmente un sitio web al que los alumnos podrían acceder e informarse acerca de la práctica, así como descargar la AppGraficar. Debo decir que en un primer momento se planteó la idea de que las gráficas se realizaran en la página web, con el fin de simplificar el trabajo, pero esto no fue posible debido al alto nivel de conocimiento de programación web que se requiere para desarrollar algo de estas características. No obstante, el hecho de descargar un simple ejecutable con el que el alumno pudiera trabajar fue también fue una solución simple y cómoda para el alumno.

El último objetivo, el de realizar un vídeo en directo para trabajar con la práctica remotamente fue uno de los puntos más complicados, debido a la falta de conocimiento previa sobre el tipo de programación necesaria para llevarlo a cabo. La solución conseguida tiene por tanto la pega de que no es posible cortar la retransmisión desde la aplicación, es necesario cerrarla. No obstante, este detalle no afecta de ninguna manera al trabajo de los alumnos con la práctica, por lo que el objetivo 7 se ha cumplido también satisfactoriamente.

En resumen, se han logrado todos los objetivos planteados al principio del trabajo de fin de grado, y además se han englobado adecuadamente para conseguir hacer una práctica amena con la que los alumnos puedan trabajar y aprender a cerca del control PID.

10.2 Conclusiones personales.

Una vez analizado el trabajo desarrollado, comentaré brevemente lo que ha supuesto para mí y lo que ha aportado a mi aprendizaje.

Lo primero que hay que comentar es lo diferente que es este trabajo frente al resto que se realizan durante el grado, puesto que por muy extenso y difícil que pueda llegar a ser alguno no alcanzan la complejidad y necesidad de trabajo autónomo de este.

Creo que ha sido positivo la cantidad de áreas que he tocado para conseguir llevar a cabo todo el proyecto. Los que más saltan a la vista son la automática para llevar a cabo el control PID y la electrónica para trabajar con equipos como la Raspberry Pi, el servomotor o los sensores. Pero hay también mucho trabajo menos visible sobre programación y comunicación, ámbitos en los que se nos da una base en la carrera, pero he tenido que ampliar por mi cuenta para lograr los objetivos.

En cuanto a la programación, fue un reto utilizar un lenguaje como Python del que no había visto nada hasta entonces, pero ha sido muy útil para mi aprendizaje debido a la gran cantidad de aplicaciones que tiene y lo extendido que está en el mundo laboral.

Sobre la comunicación necesaria para conseguir que los alumnos pudieran conectarse al proyecto remotamente, ha sido el aspecto sobre el que más he necesitado aprender, puesto que la asignatura de comunicación que cursamos está más enfocada a aspectos teóricos que prácticos.

Por último, se ha tocado también el mundo de las páginas web, tema sobre el que no tenía ningún conocimiento y ahora tengo cierta base, lo cual es positivo para mí.

Por tanto, este trabajo me ha permitido poner en práctica conocimientos aprendidos a lo largo de estos cuatro años, y ha supuesto un aprendizaje muy útil, tanto por las cosas que se ven reflejadas en el trabajo como por aquellas que hice pero finalmente no se utilizaron. Además, hay que destacar la adquisición de capacidades para resolver los problemas que iban surgiendo, lo cual valoro muy positivamente como enseñanza sobre cómo afrontar dificultades.

10.3 Líneas futuras.

Para acabar con las conclusiones, voy a hablar acerca de las limitaciones que tiene el proyecto, y que pueden ser una línea futura de trabajo para que otro alumno continúe mejorándolo.

Como ya había comentado en las conclusiones técnicas, el proyecto podría mejorar su funcionamiento cambiando ciertas cosas de la maqueta. Por un lado, el cambio de motor por uno con mayor estabilidad, y por otro, el problema del escaso rango de medida de los sensores se podría solucionar probablemente eliminando la malla exterior que rodea a la viga, de forma que no interfiriera en las medidas del sensor.

En cuanto al desarrollo de las aplicaciones, podría plantearse llevar la introducción de los valores de control, la graficación de los datos y la visualización del stream al sitio web, de forma que se simplificara aún más el trabajo del alumno con la práctica. No obstante, como ya he comentado esto requiere de un conocimiento de desarrollo web que no es un objetivo en nuestro grado universitario.

11. Bibliografía.

[1] - Rodríguez de Luis, E. (2018, 18 septiembre). *De cero a maker: todo lo necesario para empezar con Raspberry Pi*. Xataka. Recuperado 25 de diciembre de 2021, de https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi

[2] - *Instalación Sistema Operativo de la Raspberry Pi*. (s. f.). Raspberry Pi. https://www.raspberrypi.com/software/

[3] - *Raspberry Pi 3 Pins, Tutorial para conocerlos a todos*. (2020, 6 mayo). Descubrearduino.com. Recuperado 25 de diciembre de 2021, de https://descubrearduino.com/raspberry-pi-3-pins/

[4] - Llamas, L. (2018, 9 enero). *Medir distancia con precisión con Arduino y sensor láser VL53L0X y VL6180X*. Luis Llamas. Recuperado 25 de diciembre de 2021, de https://www.luisllamas.es/arduino-sensor-distancia-vl53l0x/

[5] - *Datasheet VL53L0X*. (s. f.). https://www.st.com/. https://www.st.com/resource/en/datasheet/vl53l0x.pdf

[6] - *Sensor Ultrasonido HC-SR04*. (s. f.). Naylamp Mechatronics. Recuperado 25 de diciembre de 2021, de https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

[7] - *Qué es un Servomotor y para qué sirve*. (2021, 4 octubre). aula21 | Formación para la Industria. Recuperado 25 de diciembre de 2021, de https://www.cursosaula21.com/que-es-un-servomotor/

[8] – *ServoMotor Funcionamiento y Tipos de Servomotores*. (s. f.). Área Tecnológica. Recuperado 25 de diciembre de 2021, de https://www.areatecnologia.com/electricidad/servomotor.html

[9] - Fernandez, R. (s. f.). *Raspberry Pi Camera Módulo tutorial*. Unipython. Recuperado 25 de diciembre de 2021, de https://unipython.com/raspberry-pi-camera-modulo-tutorial/

[10] - *Controlador PID*. (2021, noviembre). Picuino. Recuperado 26 de diciembre de 2021, de https://www.picuino.com/es/control-pid.html

[11] – J. Pérez Turiel. Apuntes del campus virtual de la Universidad de Valladolid, asignatura de Control y Comunicaciones.

[12] - Castillo, J. A. (2020, 7 septiembre). *Puerto serie – Qué es, para qué sirve y tipos*. Profesional Review. Recuperado 26 de diciembre de 2021, de https://www.profesionalreview.com/2020/03/07/puerto-serie-que-es-para-que-sirve-y-tipos/

[13] - Gómez, E. (2017, 31 octubre). *Cómo funciona el Puerto Serie y la UART*. Rincón Ingenieril. Recuperado 26 de diciembre de 2021, de https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/

[14] - *El código ASCII*. (s. f.). código ASCII. Recuperado 26 de diciembre de 2021, de https://elcodigoascii.com.ar/

[15] - EcuRed. (s. f.). *Socket*. Recuperado 26 de diciembre de 2021, de https://www.ecured.cu/Socket#Principio_de_funcionamiento

[16] – A. D. Martín Santana y L. I. Martínez Santana. *Apuntes asignatura de Diseño Sistemas Operativos* (curso 2007/08). Universidad de Las Palmas de Gran Canaria. Recuperado 26 de diciembre de 2021, de http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/sockets.pdf

[17] - *Descripción y funcionamiento del Bus I2C*. (2019, 24 junio). Robots Didácticos. Recuperado 26 de diciembre de 2021, de http://robots-argentina.com.ar/didactica/descripcion-y-funcionamiento-del-bus-i2c/

[18] - Robledano, Á. (2019, 23 septiembre). *Qué es Python: Características, evolución y futuro*. OpenWebinars. Recuperado 26 de diciembre de 2021, de https://openwebinars.net/blog/que-es-python/

[19] - *Python: qué es, para qué sirve y cómo se programa*. (2021, 6 octubre). aula21 | Formación para la Industria. Recuperado 26 de diciembre de 2021, de https://www.cursosaula21.com/que-es-python/

[20] - *Qué es Python – Definición, características y ventajas*. (s. f.). Pythones.net. Recuperado 26 de diciembre de 2021, de https://pythones.net/que-es-python-y-sus-caracteristicas/#%C2%BFQue_es_Python

[21] - Martín, G. (2020, 30 junio). *Interfaces gráficas en Python con Tkinter*. Adictos al trabajo. Recuperado 26 de diciembre de 2021, de https://www.adictosaltrabajo.com/2020/06/30/interfaces-graficas-en-python-con-tkinter/

[22] - *hashlib - Cifrar con los algoritmos MD5 y SHA*. (2019, 28 marzo). Recursos Python. Recuperado 26 de diciembre de 2021, de https://recursos-python.com/guias-y-manuales/hashlib-md5-sha/

[23] - Moore, J. B. (2017). *Python interface to the VL53L0X ST Library*. GitHub. Recuperado 26 de diciembre de 2021, de https://github.com/pimoroni/VL53L0X-python

[24] - Jones, D. *Python interface to Raspberry Pi camera*. GitHub. Recuperado 26 de diciembre de 2021, de https://github.com/waveform80/picamera/blob/master/docs/examples/web_streaming.py

[25] - *PyInstaller Quickstart — PyInstaller bundles Python applications*. (s. f.). PyInstaller. Recuperado 26 de diciembre de 2021, de https://www.pyinstaller.org/

[26] - *Pyqtgraph 0.12.3 documentation*. (s. f.). Pyqtgraph. Recuperado 26 de diciembre de 2021, de https://pyqtgraph.readthedocs.io/en/latest/introduction.html

ANEXO I.

ÍNDICE DE FIGURAS