

*Col·legi  
Abat Oliba*  

---

*Loreto*

# PID y Arduino

---

**Autor: Guillermo Adell Vega**

**Tutor: Carlo Gobatto**

**15/01/2020**

## **RESUMEN**

Investigación, comprensión y adaptación de un controlador PID en un proyecto práctico programado con Arduino.

## **RESUM**

*Investigació, comprensió i adaptació d'un controlador PID en un projecte pràctic programat amb Arduino.*

## **ABSTRACT**

*Research, understanding and adaptation of a PID controller in a practical project programmed with Arduino.*

## **PALABRAS CLAVES / KEYWORDS**

PID – Arduino – Electrónica – Colegio Abat Oliba Loreto - Tecnología
--

# ÍNDICE

---

INTRODUCCIÓN .....	3
MARCO TERÓRICO .....	5
1 Controlador PID .....	5
1.1 Qué es .....	5
1.2 Historia y aplicaciones.....	6
1.3 Clasificación.....	8
1.4 Funcionamiento.....	10
1.4.1 Proporcional .....	11
1.4.2 Integral .....	12
1.4.3 Derivativo .....	13
1.4.4 Constantes .....	13
1.5 Ajuste de parámetros .....	15
1.5.1 Ajuste empírico manual .....	15
1.5.2 Método de Ziegler-Nichols.....	16
1.6 Limitaciones .....	16
2 Arduino .....	17
2.1 Qué es .....	17
2.2 Historia y aplicaciones.....	19
2.3 Funcionamiento.....	20
2.3.1 Hardware.....	20
2.3.2 Software .....	21
MARCO PRÁCTICO .....	22
3 Maquetación .....	23
4 Esquemas eléctricos.....	25
5 Código del programa.....	27

5.1	Declaración de librerías, pines y variables .....	27
5.2	Setup .....	28
5.3	Loop.....	29
6	Puesta en marcha.....	31
CONCLUSIÓN .....		33
BIBLIOGRAFÍA .....		34
ANEXO I .....		36
ANEXO II .....		42
ANEXO III .....		46

# INTRODUCCIÓN

---

El presente trabajo de investigación, pretende comprender, explicar razonadamente cada uno de los términos mencionados en la tesis del proyecto y, por ende, adaptarlos entre ellos para terminar construyendo con Arduino un prototipo de controlador PID.

La motivación del proyecto viene dada por la afición al mundo de la electrónica, computación y tecnologías desde que tengo uso de razón. La idea principal a investigar, a diferencia, ha variado mucho desde hace dos años, aproximadamente. La más primitiva se centraba en el estudio y la realización de una mano robótica. Su particularidad era el modo de controlarla, que consistía en la imitación del movimiento de otra mano en un guante fabricado por mí mismo. La peculiaridad de este guante residía en adaptar unas placas de aluminio que actuaran como resistencia y cambiara la señal que se recibía en un microcontrolador a medida que se doblaban los dedos. A día de hoy el proyecto me sigue pareciendo una buena idea, el problema de entonces era que prescindía del conocimiento suficiente para realizarlo. Este proyecto evolucionó después a un brazo articulado robótico, cuya peculiaridad era implementar un sistema de control basado en el estudio de ondas cerebrales con el sensor *EMOTIV* visto por primera vez en el Mobile World Congress de Barcelona el año 2018. Como se puede imaginar, el presupuesto era demasiado elevado para llevarlo a cabo. La idea que se ha estudiado finalmente provino de una visita a las puertas abiertas de la Universidad Politécnica de Catalunya, concretamente para la carrera universitaria “Electrónica de telecomunicaciones” donde se me mostraron varios modelos prácticos de este grado. Teniendo en cuenta esto último, doy las gracias a la UPC por haberme inspirado en la elección del tema principal del proyecto y presentarme un nuevo concepto llamado controlador PID sobre el cual he decidido, finalmente, centrar mi estudio.

El trabajo se estructura en tres grandes bloques: marco teórico, marco práctico y conclusión. En el primero de ellos hay una subdivisión que determinan los dos principales conceptos del trabajo (los controladores PID y Arduino) y en ambos se realiza un estudio concreto de cada uno, analizando aquello que se ha creído relevante. En el segundo bloque, se explica de manera detallada y razonada como ha sido el proceso de la elaboración del prototipo PID en Arduino haciendo continua referencia a

todo el proceso de investigación que se ha realizado anteriormente. En el marco práctico se diferencian las partes de maqueta, esquemas eléctricos y programación con la finalidad de hacer entender de manera más clara el procedimiento al lector. El trabajo se cierra con una conclusión, que además de hacer una retrospectiva a aquello estudiado busca ver y comprender los problemas que hayan surgido durante el proceso de la parte teórica y, más bien, práctica y proponer ciertas soluciones para tratar de mediarlos.

# MARCO TERÓRICO

---

## 1 Controlador PID

### 1.1 Qué es

El controlador PID (Proporcional, Integral y Derivativo) es un mecanismo de control simultáneo por retroalimentación o *feedback* cuyo objetivo es eliminar o reducir las descompensaciones en estado estacionario<sup>1</sup> entre la señal de un valor medido y un valor deseado. Además, el controlador PID es capaz, incluso, de adquirir un efecto predictivo sobre la salida del proceso a través de la acción derivativa. (Moreno, 2001)

La idea de *feedback* es engañosamente simple y, sin embargo, extremadamente poderosa. En el caso de los controladores PID, su función es esencial ya que permite la substitución de los valores iniciales, que varían durante el proceso debido a perturbaciones, por su corrección. Se trata de un circuito cerrado, como se puede apreciar en la Imagen 1, que muestra la esquematización de la retroalimentación en un controlador PID. De este modo se consigue reducir los efectos de las perturbaciones, y hacer que un sistema o algoritmos sea insensible a las variaciones del proceso y que siga los comandos fielmente. La realimentación también ha tenido una profunda influencia en la tecnología. La aplicación del principio de realimentación ha generado importantes avances en el control autorregulado, la comunicación y la instrumentación.

---

<sup>1</sup> El error en estado estacionario de un sistema se corresponde con el valor de la señal a la salida del comparador (entre la señal de referencia y la medida por los sensores), cuando ésta se ha estabilizado a un valor constante. (García, González Sarabia, Fernández Pérez, Torre Ferrero, & Robla Gómez, 2013)

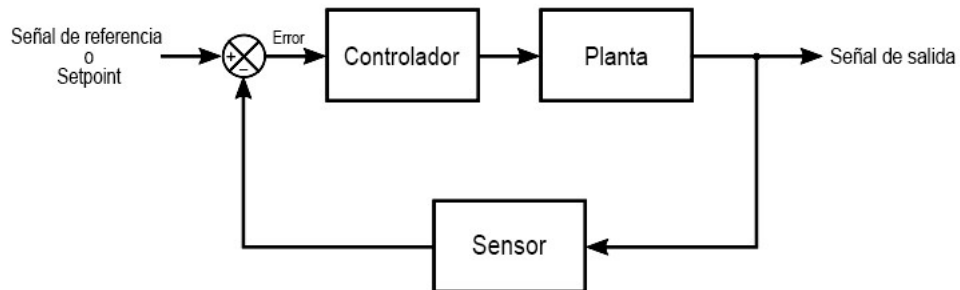


Imagen 1 “Diagrama de bloques de un sistema de feedback simple” (Aström & Hägglund, 1995)

Los controladores PID, o incluso los PI que se explicarán posteriormente, son suficientes para muchos problemas de control, particularmente cuando la dinámica del proceso es benigna y los requisitos del rendimiento son modestos. Es por ello por lo que estos sistemas de control se encuentran en grandes cantidades en todas las industrias. Aun así, muchas de las características básicas como el *anti-windup*<sup>2</sup> no se han desvelado debido a su consideración como secretos de los fabricantes. El uso de estos controladores es tal que el 95% de los lazos de control que existen en la industria son del tipo PID. (Aström & Hägglund, 1995)

El controlador PID a menudo se combina con lógica, máquinas secuenciales, selectores, y bloques de funciones simples para construir los sistemas complejos de automatización usados para la producción de energía, transporte y la fabricación. Es usado en el nivel más bajo; en un controlador multivariable por desacoplo con controladores PID, los puntos de ajuste del controlador PID vendrían dados por el controlador más complejo. Es por todo ello que los controladores PID, PI y sus derivados son esenciales en los procesos de control y en los procesos de autorregulación.

## 1.2 Historia y aplicaciones

A lo largo de la historia, los controladores PID han sobrevivido a muchos cambios en la tecnología. Desde los antiguos reguladores de Watt, de la época de la revolución

<sup>2</sup> Referido a la situación en un controlador retroalimentado PID donde un ocurre un amplio cambio en el *setpoint* o punto de ajuste y los términos integrales acumulan un error significativo durante la corrección y provocando un fallo en el control del sistema. (Jurado, 2012)



industrial, pasando por los controladores neumáticos, los controladores analógicos eléctricos y electrónicos (primero con válvulas y más tarde implementados con circuitos integrados) hasta los modernos controladores y microcontroladores. Elementos como los microprocesadores han tenido una gran influencia sobre su desarrollo, permitiendo ofrecer nuevas oportunidades para implementar funciones como el ajuste automático de parámetros y los cambios de modo de control.

Aunque G. R. de Prony, H.N. Throop y James C. Maxwell mejoraran la primera versión de un controlador muy primitivo (regulador centrífugo de James Watt de 1788) hasta conseguir un controlador con dos componentes, proporcional y derivativa, no fue hasta 1911 cuando el emprendedor estadounidense Elmer Sperry logró implementar el conjunto de J. C. Maxwell de 1868 para usarlo como un corrector automático de dirección en los barcos de la Armada de los Estados Unidos de América. El modelo del controlador neumático de Sperry fue el primero en ser desarrollado.(O'Dwyer, 2005)

El primer análisis teórico de un controlador PID, a diferencia, fue realizado en 1922 cuando Nicolás Minorsky, por error, descubre que las correcciones de la estabilidad direccional de embarcaciones dirigidas automáticamente de la Armada de los Estados Unidos no utilizan solamente las lecturas en el momento, sino que se apoyan también en lecturas anteriores. De esa manera, fue capaz de formular esta observación de manera matemática para poder incluirla en el sistema proporcional y derivativo. (Minorsky, 1922)

El análisis surgió del trabajo en la instalación y preparación para la prueba del mecanismo de dirección automático en el acorazado New Mwico, las pruebas del cual se llevaron a cabo en 1923. Hubo cierto interés en los sistemas de dirección totalmente automáticos casi desde la primera introducción de los motores de dirección servo-controlados en 1864. Aun así, se hicieron pocos avances hasta que las principales potencias navales comenzaron a revisar sus técnicas de control de fuego al principio de ese siglo. Esta revisión se hizo necesaria con el incremento de armas navales. (Bennett, 1984)

Hasta aproximadamente el 1940 las industrias habrían podido controlar todas las variables de un proceso mediante controladores todo-nada o manualmente. Debido a la tardía aparición de controladores PI o PID en el sector industrial, se vendieron más de 75.000 unidades de controladores todo-nada en diez años (1925 – 1935). Estos sistemas se popularizaron por su reducido coste y su sencillez, si es comparado con otros controladores dotados de componentes derivativas, integrales y/o proporcionales.

En adición, cuando los primeros controladores PI fueron patentados (1920), las patentes todo-nada llevaban 13 años en el sector industrial, lo cual dejaba hasta aproximadamente 1950 a los sistemas PID en desventaja.

A partir de entonces las industrias no fueron capaces de mantener todas las variables de un proceso controladas. A raíz de esto, se vio obligada a realizarse la implementación de controladores automatizados PID en los procesos industriales.

El avance de mayor magnitud respecto estos controladores fue la publicación del informe publicado por el inglés Albert Callander en 1934 llamado *Preliminary notes in automatic control* se mostraban las tablas que permiten la modificación de los valores de los controladores PD para procesos con demoras. En dos informes subsecuentes de 1935/6 y 1937, Callander y coautores propusieron las tablas que permiten la variación de los valores de los controladores PI y PID para un rango de procesos con demora. (Callender, Hartree, & Porter, 1935/6)

Desde 1950 se han realizado multitud de variaciones y mejoras las cuales normalmente han ido acompañadas de un desarrollo tecnológico a nivel general. Un ejemplo de ello es la mejora en los cálculos derivativos, integrales y proporcionales se han ido modificando con el objetivo de convertir el PID en un controlador más preciso, sencillo y barato.

Por otro lado, a nivel material, la aplicación de microcontroladores y la primera implementación de un controlador PID en uno de ellos en 1975, ha permitido a este conjunto adaptarse mejor a cualquier lugar donde se requiera su uso.

Hoy en día, los posibles usos del controlador PID son prácticamente infinitos. La razón de ello es la siguiente: cualquier proceso que necesite de una lectura de ciertos valores (por ejemplo, la temperatura, nivel de agua, cantidad de cualquier material en una cámara, estado de cámaras, compuertas, válvulas, motores...), el cálculo continuo e incluso de la anticipación del futuro y una señal de salida, requiere de un controlador de estas características.

### **1.3 Clasificación**

Desde la aparición del primer controlador más primitivo hasta los PID de hoy en día ha habido una fuerte evolución en ellos que ha dejado un legado inmenso de distintos tipos de controladores. Estos se pueden agrupar dependiendo de su procedencia:

- Fabricados por el hombre
- Naturales, sistemas biológicos

- Naturales y hechos por el hombre

Los controladores PID forman parte de aquellos producidos por el hombre.

Otro modo de clasificar de manera general los controladores es según el tipo de energía que utilizan para operar:

- Neumáticos
- Hidráulicos
- Electrónicos (Naboa, 2012)

Un ejemplo claro de controlador neumático es el regulador centrífugo de James Watt de 1788. Los PID actuales operan con fuente eléctrica, debido a su alto rendimiento y capacidad para cubrir infinitos propósitos.

Por otro lado, estos controladores se pueden subclasificar en el número de entradas y salidas:

- Una entrada y una salida o SISO (*single input, single output*)
- Una entrada y múltiples salidas o SIMO (*single input, multiple output*)
- Múltiples entradas y una salida o MISO (*multiple input, single output*)
- Múltiples entradas y múltiples salidas o MIMO (*multiple input, multiple output*)

El uso de una entrada o salida, o múltiples va a depender del uso último del controlador, de las variables que el proceso tenga y la complejidad. Como norma general, los PID pertenecerán al grupo de controladores SISO. (Veluvolu, 2011)

Las variables son función del tiempo, por ello, otra clasificación posible es la siguiente:

- De tiempo continuo
- De tiempo discreto
- De eventos discretos

En este caso los controladores PID son sistemas de tiempo discreto ya que las mediciones de las señales de entrada se realizan en instantes de tiempo con un período de margen constante entre cada una. En cambio, los sistemas de tiempo continuo, cuyas variables son denominadas analógicas, hacen mediciones continuas de las señales de entrada y por lo tanto se considera que el tiempo es infinitamente divisible. En este sentido, los controladores de tiempo continuo pueden ser una ventaja a la hora de realizar cálculos de matemáticas avanzadas ya que las variables no son digitales o discretas. Por otro lado, los sistemas de eventos discretos no son compatibles con los controladores PID debido a que las variables harán evolucionar el sistema solamente en condiciones o eventos determinados.

Otra clasificación más completa se basa en la acción de control. Según la ley de control existen las siguientes posibilidades:

- De dos posiciones u *on-off*
- Controlador Proporcional
- Controlador Integral
- Controlador Proporcional Integral (PI)
- Controlador Derivativo
- Controlador Proporcional Derivativo (PD)
- Controlador Proporcional Integral Derivativo (PID) (Naboa, 2012)

Por último, la característica que diferencia los PID de otros controladores es, como se ha comentado en su definición, la capacidad de realimentar su sistema gracias al control en lazo cerrado y, de alguna manera, tomar decisiones que se le ha programado previamente dependiendo de la señal de salida<sup>3</sup>. A diferencia de estos, los controladores de lazo abierto no son capaces de ajustar la acción de control y solamente actúa el proceso sobre la señal de entrada.<sup>4</sup> De este modo se requiere de una supervisión para ajustar la acción de control a aquello que se busca.

## 1.4 Funcionamiento

A diferencia de un controlador *on-off* donde para determinados valores hay una respuesta concreta y definida previamente en una expresión matemática sencilla, los controladores PID requieren de una cierta complejidad.

Las siglas PID, cómo se ha expresado anteriormente, hacen referencia a las acciones de control que forman el controlador, en este caso Proporcional Integral Derivativo. Su nombre proviene de la expresión matemática que permite al controlador desempeñar su función. Esta se puede expresar como la suma de las acciones de control que intervienen. Por consecuencia, la expresión matemática de los controladores formados por más de una acción mencionados en la clasificación según la ley de control del apartado 3 será la suma de sus integrantes. Por ejemplo, el controlador PI será la suma de la acción proporcional e integral.

---

<sup>3</sup> La señal de salida, representada comúnmente como  $y$ , hace referencia al valor de la variable que se desea controlar. (i.e. posición, velocidad, presión, temperatura, etc.)

<sup>4</sup> i.e. Un tanque con una manguera de jardín. Mientras que la llave siga abierta, el agua fluirá. La altura del agua en el tanque no puede hacer que la llave se cierre.

La expresión que define el controlador PID es la siguiente:

$$u(t) = K_p \cdot e(t) + K_i + \int_0^t e(t)dt + K_d \frac{de}{dt}$$

Donde  $u(t)$  es la señal de control o de salida en función del tiempo y las variables a tener en cuenta son  $K_p$  (Constante Proporcional),  $K_i$  (Constante Integral) y  $K_d$  (Constante Derivativa). (Aström & Hägglund, 1995)

Gracias al uso de las acciones integral y derivativa, se puede certificar que el controlador PID es un compensador de atraso-adelanto.

Esta expresión es una deducción de cálculos complejos donde se estudian las perturbaciones y distintas ecuaciones obtenidas de diagramas de bloques de controladores únicamente proporcionales. En adición, el uso de funciones de transferencia<sup>5</sup> permite encontrar las expresiones de distintos controladores. *“Si se desconoce la función de transferencia de un sistema, puede establecerse experimentalmente introduciendo entradas conocidas y estudiando la salida del sistema. Una vez establecida una función de transferencia, proporciona una descripción completa de las características dinámicas del sistema, a diferencia de su descripción física.”* (Naboa, 2012)

#### 1.4.1 Proporcional

La acción proporcional puede ser la principal del sistema debido a su efecto directo de corrección del error. Ésta fue la razón de la mejora del rendimiento de los controladores que la incluían respecto al sistema *on-off*. Los controladores todo-nada tienden a incrementar las oscilaciones de las variables respecto a la referencia debido a los cambios de las variables dependientes a través de todo su recorrido<sup>6</sup> producidos por errores mínimos. En cambio, en los controladores proporcionales este aspecto es corregido. La corrección que se aplique a las variables en el *output* va a ser siempre una ganancia múltiplo del error. Esta característica se muestra en la expresión matemática de la acción:

$$u(t) = K_p \cdot e(t) = K_p \cdot (y_{sp} - y)$$

---

<sup>5</sup> “La función de transferencia de un sistema es un modelo matemático porque es un método operacional para expresar la ecuación diferencial que relaciona la variable de salida con la variable de entrada.” (Naboa, 2012)

<sup>6</sup> El recorrido de un controlador todo-nada hace referencia a la amplitud (diferencia entre el menor y mayor de los valores) de corrección establecidos por el sistema.

Donde  $u(t)$  es la señal de control, es decir a aquel valor al que queremos que llegue la variable.  $K_p$  es, en este caso, la constante de proporcionalidad.  $K_p$  determina la banda proporcional, es decir el porcentaje de ganancia respecto al porcentaje de error obtenido con la medición. En la Imagen 2 se puede apreciar una representación gráfica del concepto. El error proviene de la diferencia entre el *setpoint* y el valor obtenido por la medición en un instante de tiempo concreto. (Mazzone, 2002)

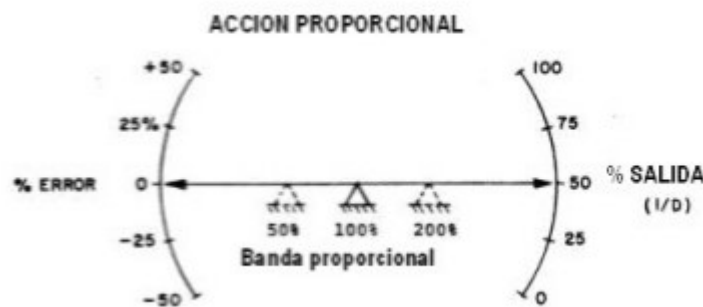


Imagen 2 “Representación gráfica de la acción proporcional” (Amadori, 2010)

#### 1.4.2 Integral

La componente derivativa tiene una desventaja, comúnmente ocurre que aquello que queremos controlar<sup>7</sup> se desvía del *setpoint*. La acción integral permite proporcionar una señal de salida proporcional al error acumulado en un tiempo determinado. A diferencia de la componente proporcional, la integral es capaz de disminuir y eliminar el error provocado por perturbaciones. La expresión matemática es la siguiente:

$$u(t) = K_i \cdot \int_0^t e(t) d(t)$$

En el caso de la acción integral, el error depende del tiempo y es por ello por lo que  $u$  (el error) se expresa en función del tiempo.  $K_i$  es la ganancia integral.

Esta componente tiene una propiedad asombrosa. En un caso hipotético donde hubiese un error en estado estacionario, es decir que el error  $e_0$  fuese constante y por consiguiente  $u_0$  también lo fuera, la acción proporcional mantendría un error constante. A diferencia, el controlador integral es capaz de eliminar el *off-set*<sup>8</sup> debido a la integración del error acumulado.

<sup>7</sup> Puede referirse también como *process variable* o variable controlada.

<sup>8</sup> El *off-set* hace referencia al estado de un controlador donde la variable controlada se mantiene a un valor distinto a la señal de referencia o *setpoint*.

### 1.4.3 Derivativo

El propósito principal de la acción derivativa es mejorar la estabilidad del circuito cerrado. Debido a la dinámica de los procesos, donde siempre habrá pequeñas perturbaciones presentes, tomará un cierto tiempo para que el cambio en la variable de control se refleje en la señal de control. Sin la acción derivativa, el error sería demasiado grande cuando se efectuase la corrección. (Aström & Hägglund, 1995)

La acción derivativa proporciona al sistema una predicción del efecto de la acción proporcional para estabilizar la variable controlada y corregir las posibles perturbaciones más rápidamente. Por consiguiente, no será posible un controlador únicamente derivativo ya que no proporciona ninguna corrección. Las combinaciones más frecuentes para controladores con la componente derivativa son PD y PID.

La relación *input-output* de un controlador con componentes proporcional y derivativa es la siguiente:

$$u(t) = K_p \cdot e(t) + K_d \cdot \frac{de}{dt}$$

En la expresión matemática se presenta la ecuación de un controlador PD, ya que no existe un controlador únicamente derivativo. Se puede apreciar cómo se realiza la predicción a través de extrapolación lineal, es decir, creando una línea tangente de la curva del error y extendiéndola más allá de donde se conoce. Además, ciertas variaciones de la expresión matemática expuesta pueden precisar más los cálculos. Un ejemplo de ello es a través de una expansión de la serie de Taylor. (Amadori, 2010)

### 1.4.4 Constantes

En un controlador PID existen tres constantes cuyos valores se le asignarán manualmente para tratar de conseguir un rendimiento óptimo del sistema.

$K_p$ ,  $K_i$  y  $K_d$  son las tres ganancias de las acciones proporcional, integral y derivativa, respectivamente.

Debido al distinto comportamiento entre las tres acciones, cada constante va a afectar de manera diferente al conjunto. Por consiguiente, el aumento o disminución de estos valores va a tener unas consecuencias positivas y, por contra, otras negativas. Por ello es necesario establecer el valor correcto de las ganancias para no impedir el correcto funcionamiento. En la imagen 3 se muestra una representación gráfica de los efectos de las tres componentes en el sistema.

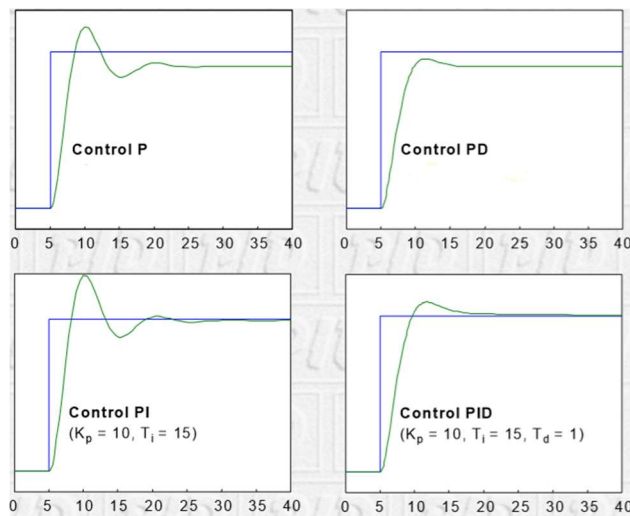


Imagen 3 “Gráficos de controladores P, PD, PI y PID” (García F. M., 2007)

En primer lugar,  $K_p$  afecta a la ganancia del controlador proporcional. La señal de control en el controlador proporcional, como se ha mostrado en el apartado 4.1, procede del producto de  $K_p$  por el error. Esto significa que la señal de control va a ser directamente proporcional al error y la relación  $\frac{u(t)}{e(t)}$  va a depender únicamente de la banda de proporcionalidad o  $K_p$ , mostrado gráficamente en la Imagen 2 como un fulcro entre la relación nombrada anteriormente. (García F. M., 2007)

En segundo lugar,  $K_i$  afecta a la ganancia del controlador integral. En la función del controlador integral se puede observar como  $K_i$  multiplica al error acumulado durante un periodo de tiempo. Entonces, se puede concluir que el valor  $K_i$  va a afectar a la velocidad de reacción del controlador ante un error de manera inversamente proporcional, es decir que, a menor  $K_i$ , mayor facilidad para reaccionar a cambios. Por contra, si el valor de la constante integral es demasiado pequeño puede provocar oscilaciones al sistema debido a que la velocidad es demasiado elevada. Por lo tanto, el valor  $K_i$  óptimo va a ser aquél que permita una reacción rápida del controlador, que elimine el *off-set* que se produce en el controlador proporcional y que provoque las mínimas oscilaciones del sistema posibles.

Por último,  $K_d$  es la constante que implica al controlador derivativo.  $K_d$  es una simplificación del producto de la constante proporcional  $K_p$  por el tiempo derivativo  $T_d$ . Por consecuente, la constante derivativa afecta de manera directamente proporcional a



la contribución de la acción derivativa y a la velocidad del sistema. En conclusión, cuando el valor de  $K_d$  sea grande, la estabilidad y velocidad del sistema aumenta.

A continuación, se puede observar distintas gráficas donde se implementan las distintas acciones del controlador en cada una. De este modo se puede ver claramente como afecta cada controlador con las constantes implícitas (el valor de las constantes se mantiene igual en todos los gráficos, con el fin de apreciar con más exactitud el efecto de estas en el sistema).

## 1.5 Ajuste de parámetros

Los controladores PID requieren de un ajuste preciso para garantizar su óptimo comportamiento, de lo contrario, los cálculos y correcciones no se efectuarían correctamente. Por otro lado, podría causar rotura mecánica o saturación del circuito. Si bien es cierto que no hay ninguna configuración perfecta para los parámetros de todos los controladores PID o sus variantes, sí que hay la posibilidad de conseguir los valores para el óptimo funcionamiento de éste, también denominado ajuste de un lazo de control. Los valores de los parámetros van a depender de la aplicación que se le dé al PID. Generalmente, se busca la estabilidad del sistema ante la corrección del controlador. Existen varios métodos para ajustar un lazo cerrado de un controlador PID. Entre ellos se destacan los siguientes:

- *Ajuste empírico manual*
- *Método de Ziegler-Nichols* (García F. M., 2007)

### 1.5.1 Ajuste empírico manual

Para este método, en primer lugar, se considera nulo el valor de las componentes integrales y derivativas del controlador. De este modo, el peso completo de la corrección recae sobre la acción proporcional. La finalidad de esta acción es establecer primero los valores óptimos de esta componente.

Para conseguirlo, en primer lugar, se incrementa el valor de la constante proporcional hasta que el valor de la señal de control oscile. Por último, se establece el valor de  $K_p$  a la mitad del valor asignado anteriormente.

Una vez se ha logrado esto, se procede a incrementar el valor de  $K_i$  para ajustar el proceso de corrección al tiempo deseado, puede que esta acción implique incrementar

la inestabilidad del sistema. Para corregir la inestabilidad que pueda haber surgido se incrementará el valor de  $K_d$  hasta obtener un resultado óptimo.

Con este método no se logra una precisión excelente. De todas maneras, es el proceso habitualmente usado debido a su sencilla realización.

### 1.5.2 Método de Ziegler-Nichols

En 1942 Ziegler y Nichols propusieron una fórmula empírica de sintonización. Es posible aplicar este método sin necesidad de conocer las funciones matemáticas de la planta. Gracias a este sistema podemos determinar el valor de las ganancias proporcional, integral y derivativa ya sea a partir de la respuesta de un controlador de lazo abierto o cerrado. Cada opción se ajusta mejor a un tipo de sistema y la finalidad.

Ziegler y Nichols propusieron parámetros de sintonización para controladores P, PI y PID aquellos con los cuales se obtienen resultados óptimos respecto a posibles oscilaciones del sistema.

Para ello se realiza una prueba del sistema para observar su comportamiento y extraer datos. Los datos se reemplazan en las tablas de parámetros Ziegler-Nichols para obtener los valores de las constantes.

Ziegler y Nichols ofrecieron también la posibilidad de sintonizar gran variedad de plantas a través de sus funciones de transferencia<sup>9</sup>. Los modelos no derivables no podrían ser, entonces, sintonizados por este método. Para ello se pueden realizar ciertos experimentos para aproximar los valores. (Pérez, Benítez Baltazar, Pacheco Ramírez, & Montaña Valle, 2014)

## 1.6 Limitaciones

Aunque los controladores PID puedan implementarse en la mayoría de los problemas de control, este algoritmo puede ser no ser el indicado en ciertas situaciones. Principalmente existen dos puntos débiles en este controlador; el ruido provocado por la acción derivativa y la oscilación sobre el valor del *setpoint*, es decir, el *off-set*. (Moreno, Apuntes de control PID, 2001)

---

<sup>9</sup> Una función de transferencia es un modelo matemático que a través de un cociente relaciona la respuesta de un sistema (señal de salida) con una señal de entrada. En la teoría de control, a menudo se usan las funciones de transferencia para caracterizar las relaciones de entrada y salida de componentes. (Iha(3057), 2013)

En el primer caso, un ajuste incorrecto de la constante derivativa puede provocar, si ésta adquiere un valor demasiado elevado para que la corrección sea más veloz, la aparición de ruido. El ruido hace de las correcciones de error una tarea imposible ya que desestabiliza el sistema por completo haciendo que las pequeñas variaciones que se producen en la planta sean mucho mayores en la señal de salida. Para salvar este error, el controlador PID debería de combinarse con un Filtro de Paso Bajo<sup>10</sup> para que eliminara las frecuencias altas del ruido. Por otro lado, el FPB y la acción derivativa podrían anularse entre ellos. Alternativamente, se puede retirar dicha acción sin que se produzcan grandes pérdidas de control en ciertas ocasiones. De ese modo se trataría de un controlador PI. (Verrastro, Alberino, & Folino , 2005)

En el segundo caso, cuando el controlador PID no se combina con un control de lazo abierto, en situaciones concretas la ganancia del lazo PID debe ser reducida para evitar el *off-set*. De ese modo, el rendimiento se vería gravemente afectado. Por ello, una solución es combinar el lazo cerrado PID con un control de lazo abierto. Conociendo las condiciones del sistema, se puede introducir un controlador *feed-forward*<sup>11</sup> para establecer los parámetros ideales del sistema sin necesidad de medir los parámetros del *setpoint*, evitando perturbaciones por ruido. El desempeño del sistema se verá mejorado solamente en el caso de implementar este lazo ya que no afecta al sistema retroalimentado PID que, a partir del ideal, corrige las oscilaciones que se generen. (Aström & Murray, 2009)

## 2 Arduino

### 2.1 Qué es

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs – light on a sensor, a finger on a button, or a Twitter message – and turn it into an output – activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino

---

<sup>10</sup> Circuito que transmite todas las frecuencias de una señal por debajo de una cierta frecuencia denominada de corte. (Real Academia de Ingeniería, 2010)

<sup>11</sup> El término *Feed-Forward* describe un tipo de sistema que responde a las alteraciones de manera predefinida, en contraste con los sistemas retroalimentados. (Aström & Murray, 2009)

programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. (Arduino, 2019)

Arduino es una compañía o, como ellos mismos se definen, una fuente de libre acceso para programar, tanto el *hardware* como el *software*, de un modo fácil.

Aunque *Arduino* se defina como una herramienta pensada para usuarios con poco conocimiento acerca de los microcontroladores y la programación, también disponen de módulos complejos que se adaptan a las necesidades de usuarios más experimentados. Hay muchos otros microcontroladores y plataformas disponibles para la programación física. Parallax Basic, Netmedia's BX-24, Stamp, MIT's Handyboard, Phidgets, y muchas otras ofrecen funcionalidades parecidas. Todas las herramientas mencionadas consiguen adaptar muchas de las características de la programación de microcontroladores y las ofrecen a un público poco experimentado. Arduino también facilita el proceso de trabajo con microcontroladores y, además, ofrece ventajas ante otros sistemas para profesores, estudiantes y principiantes interesados:

- Económico – El módulo más económico se puede montar a mano, e incluso los módulos Arduino montados por el fabricante no superan los 50€ de coste. En adición, el Software de Arduino es totalmente gratuito.
- Multiplataforma – El Software de Arduino (IDE) es compatible tanto en los sistemas operativos de Windows, como en Macintosh OSX o en Linux. Este hecho marca una diferencia ante otros sistemas, limitados a Windows, solamente.
- Fuente abierta, Software y Hardware extensibles – Tanto el hardware como el software se presentan como recursos de acceso gratuito y para cualquier sistema operativo. El Software de Arduino se puede ampliar a través de las librerías C++, con la posibilidad de programar en lenguaje AVR-C para entender detalles técnicos de la programación en C++. En cuanto al Hardware, Arduino publica los planes de sus módulos bajo la licencia "Creative Commons" para que diseñadores de circuitos experimentados puedan crear sus propios módulos, expandiéndolos y mejorándolos. Para aquellos usuarios más inexpertos, Arduino dedica, desde 2008, un apartado de su página web donde muestra paso a paso como montar tu propia placa a partir de pequeños microcontroladores. (Arduino, 2019)

## 2.2 Historia y aplicaciones

En 2003 Hernando Barragán empezó el proyecto para su tesis del máster en Interaction Design Institute Ivrea (IDII), en Italia. “Wiring” fue el nombre que le adjudicó.

El objetivo del proyecto era facilitar a artistas y diseñadores el trabajo con la electrónica, retirando los detalles complejos de la electrónica para permitir una herramienta más sencilla de utilizar.

Debido a ser la única persona que se graduó con una distinción en 2004, Barragán decidió continuar con su proyecto. En primavera de 2004, Wiring ya se usaba para enseñar computación física en IDII a través de un proyecto llamado “Strangely Familiar”. Este proyecto demostró el potencial de Wiring como una plataforma innovadora para el diseño de interacción.

En 2005, Massimo Banzi (supervisor de la tesis de Hernando Barragán) y David Mellis (estudiante de IDII) dieron apoyo para lograr un microcontrolador ATmega8 (microcontrolador que formaba la placa Wiring). Posteriormente copiaron el código fuente de Wiring i empezaron a trabajar en él, por separado, llamando al proyecto “Arduino”. (Barragán, 2016) (Fry, 2009)

A partir de 2012, se incorporaron nuevos microcontroladores a las nuevas placas. Los nuevos Cortex M3, ARM de 32 bits coexisten con los AVR de 8 bits, los microcontroladores que incorporaron las placas Arduino hasta 2012. (Banzi, Arduino Due is finally here, 2012)

En 2013 Adafruit Industries, la empresa dedicada a proveer y fabricar las placas estimó que 700.000 placas Arduino oficiales estaban siendo utilizadas por usuarios. (Cuartielles, 2013)

En julio de 2017, los fundadores originales de Arduino (Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino y David Mellis) recuperaron el control de la empresa por completo. La búsqueda del asociado perfecto se concluyó con el contrato con ARM Holdings. Por otro lado, ARM reconoció la independencia de Arduino y por ello, los fundadores decidieron apostar por permanecer con una investigación abierta a compartir componentes y estructuras de distintos proveedores, en tanto que esto aportara mejoras a su proyecto. (Banzi, 2017)

Las placas Arduino se han usado a lo largo del tiempo con muchos fines. Cualquier tarea que requiera de un conjunto electrónico para facilitarla puede ser programada con Arduino. Ejemplos prácticos que se han llevado a cabo con Arduino son: el software y hardware de aeronaves no tripuladas “Ardupilot”, impresoras 3D, el sistema de

retroiluminación en televisores Philips “Ambilight”, etc. (Hadi, Varianto, Riyanto, & Budiyo, 2014)

## 2.3 Funcionamiento

### 2.3.1 Hardware

El *hardware* es un componente básico del conjunto electrónico para que funcione. Según la Real Academia de Ingeniería, hardware es “*el conjunto interconectado de componentes que realizan funciones analógicas o digitales.*” Los componentes que se mencionan en la definición se pueden clasificar según su función dentro del conjunto. Siguiendo este método, encontramos las unidades de conexión, procesamiento, entrada y salida.

En el caso de tratar con un circuito electrónico caracterizado por contener una placa Arduino, la unidad de procesamiento es el microcontrolador que lleva la placa incorporado en ella. El microcontrolador, según la definición de la Real Academia de Ingeniería, es un “*circuito integrado programable de uso general que incorpora las funciones de una unidad de control de proceso de un computador.*”. Este componente es, además, el más importante del conjunto debido a que de él depende la ejecución del código de programación. (Real Academia de Ingeniería, 2010) (Arduino, 2019)

La placa Arduino contiene, además del microcontrolador, un número concreto de pines<sup>12</sup> de entrada y de salida, una conexión USB, un conector de alimentación y otras conexiones específicas. Cada modelo de placa tiene unas características que la hacen única respecto a las demás. Existen los siguientes modelos: Duemilanove, LilyPad, Nano, Fio, Mega, Pro, Bluetooth, Pro Mini y Mini. (Arduino, 2019)

A través de cables se conectan los componentes del circuito electrónico a los pines de la placa, por ejemplo, resistencias, diodos, potenciómetros, condensadores o transistores.

Todos los elementos anteriormente nombrados son componentes de salida ya que reciben una señal eléctrica de la placa y, entonces, se accionan de la manera en que se programa. Por otro lado, existen también componentes de entrada cuya función es medir una señal concreta y transmitirla al microprocesador. Ejemplos de estos elementos son los acelerómetros, teclados, giróscopos, termistores, etc. (Areny, 2008)

---

<sup>12</sup> Pequeñas clavijas terminales situadas en la placa para permitir conexiones entre ella y los periféricos. En Arduino los pines se pueden configurar como terminales de entrada o de salida. (Arduino, 2019)

### 2.3.2 Software

El software de Arduino consiste en la ejecución de un lenguaje de programación específico y un *firmware*<sup>13</sup> en una placa. El hardware de Arduino se programa con el lenguaje C++, simplificado con programación basada en el procesamiento IDE<sup>14</sup>.

La estructura del código de Arduino se basa en, al menos, dos partes. Estas son el *setup* y los *loops*. El *setup* se ejecuta al principio y solo una vez para establecer el modo de pin o la comunicación en serie y se declaran las variables. (D'Ausilio, 2011)

La segunda parte del código ejecuta un *loop* que permite que el *script*<sup>15</sup> cambie, responda a señales y controle la placa Arduino. El *loop*, como su nombre indica, se va a repetir de forma circular infinitamente, a menos de que en el código se diga lo contrario. La escriptura del código se limita a 59 funciones, 31 variables y 51 comandos de estructura preestablecidos por Arduino con la librería establecida por defecto. Un ejemplo es “digitalRead()” o “delay()”, que sirven para hacer una lectura del estado de un pin concreto y para provocar un retardo entre la anterior función y la siguiente, respectivamente. (Arduino, 2019)

Las funciones preestablecidas cubren 13 campos, los cuales son: I/O Digitales, I/O Analógicos, Familia de Due, Zero y MKR, I/O Avanzados, Tiempo, Matemáticas, Trigonometría, Caracteres, Números aleatorios, Bits y Bytes, Interrupciones Externas, Interrupciones, Comunicación y USB.

Los comandos de variables cubren tres campos: Constantes, Conversiones y Tipo de Datos.

Los comandos de estructura cubren nueve aspectos, estos son: Sketch, Control de estructura, Sintaxis, Operaciones Aritméticas, Operadores de comparación, Operadores de Boole, Operadores de acceso de puntero, Operadores Bitwise, Operadores compuestos.

Para abarcar otros campos como los cálculos de un controlador PID existen librerías que ofrecen funciones capaces de cubrir esas necesidades. Las librerías permiten extender las capacidades de Arduino aportando funcionalidades adicionales para uso

---

<sup>13</sup> “Referido a informática, término inglés que hace referencia a un elemento intermedio entre el hardware y el software. Se refiere al conjunto de microprogramas necesarios para ejecutar cada una de las instrucciones de máquina de un computador con unidad de control microprogramada.” (Real Academia de Ingeniería, 2010)

<sup>14</sup> Medio ambiente (IDE). El entorno de procesamiento incluye un editor de texto, un compilador y una ventana de visualización. Permite la creación de software dentro de un conjunto de restricciones cuidadosamente diseñado. (Fry & Reas, 2019)

<sup>15</sup> En sistemas operativos, colección de mandatos u órdenes que se agrupan en un fichero y que sirven de entrada a un sistema operativo o sistema software. (Real Academia de Ingeniería, 2010)

en Sketches<sup>16</sup>. Un número de librerías están instaladas con IDE, aunque también se pueden descargar o crear una propia. (Arduino, 2019)

Por otro lado, además del código, se necesita la interfaz de escritura del código. Ésta tiene la forma que se muestra en la Imagen 4. En primer lugar, al abrir el programa, se puede ver el espacio de escritura, de fondo blanco y con los tres bloques del código (definición de variables, Void setup y Void loop) espaciados. Sobre ello hay una barra con cinco iconos los cuales sirven para validar el código, subirlo a la placa, crear un nuevo documento, abrir un documento y guardar el actual (de izquierda a derecha respectivamente).

Todos los comandos que se utilicen en el trabajo práctico se explicaran más concretamente en el apartado correspondiente.

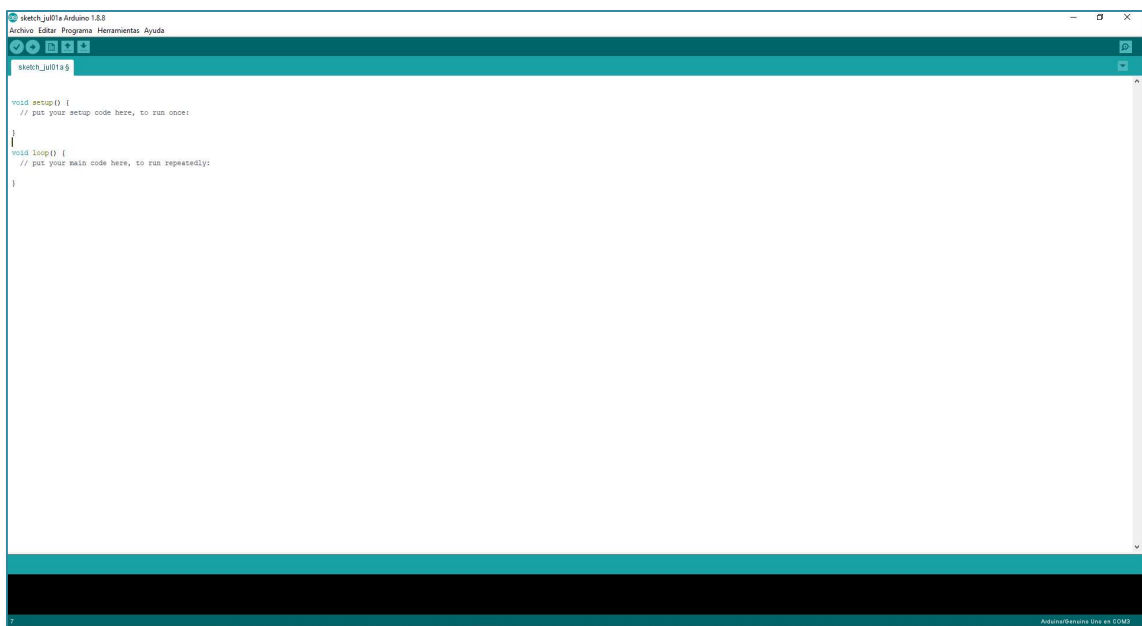


Imagen 4 “Interfaz software Arduino versión 1.8.8”

## MARCO PRÁCTICO

---

---

<sup>16</sup> Estructura básica del código. E.g. void Setup () o void Loop ().



La parte práctica de esta investigación, trata sobre la implementación de un controlador PID (con la posibilidad de anular las constantes de cada parte del controlador con el fin de diferenciar los comportamientos en sistemas P, PI y PID) en un sistema de estabilización mecánico que gracias a Arduino puede programarse para corregir el error que deseamos. La calidad del resultado es proporcional a la profundidad de la indagación que se ha realizado sobre los conceptos teóricos.

En este caso el controlador PID trata de corregir el error marcado por la posición de una pelota de tenis de mesa que se desliza sobre un raíl respecto a la posición cero (*setpoint* en el centro del raíl) y encender un LED para confirmarlo. Esto es posible gracias a un sensor de distancia colocado a un extremo y un servomotor que corrige la posición a una velocidad y tiempo determinados que vienen marcados por los cálculos que realiza una placa Arduino programada. Además, el sistema cuenta con tres botones que sirven para anular cada una de las constantes con la finalidad de ver el efecto que causa cada término (PID) en la corrección del error.

Para la estructuración de la explicación se han tenido en cuenta tres partes esenciales del proceso: la maqueta del prototipo, los esquemas eléctricos y el ensamblaje de estos en el prototipo y la programación del conjunto.

### **3 Maquetación**

Aunque el diseño del prototipo y su construcción pueda parecer lo más sencillo del conjunto, se han tenido que valorar y tener en cuenta muchos aspectos para que el rendimiento de éste fuera bueno. Las consideraciones han sido tales como el peso de cada una de las piezas, tamaño, disposición de los elementos electrónicos (servomotor, el sensor, botones o LEDs) y la unión de las piezas entre sí.

La primera parte a comentar es la base. Se trata de un tablón de madera de 30cm x 15cm x 2cm ya que debía ser una base robusta pero fácil de transportar al mismo tiempo. A continuación, collada a la base se encuentra un ortoedro vertical de madera cuyas dimensiones son 30cm x 6cm x 5cm y que tiene la función de sostener tanto el eje de la barra móvil. El raíl sobre el cual se desplaza la pelota se forma por una barra de 30cm x 2cm x 2cm de madera y sobre ella, se adhiere una canaleta de metal que permite guiar la pelota horizontalmente. Sobre la canaleta se ha pegado una tira de

LEDs. El sensor analógico<sup>17</sup> de distancia *Sharp GP2Y0A21*<sup>18</sup>, cuya medición mínima es a partir de los 10cm, permanece colocado a un extremo de la barra que tiene el movimiento. El soporte está hecho con tubos y un codo, ambos de plástico, que están encajados a presión (se ha añadido cinta para aumentar el grueso y que el rozamiento sea mayor para que no haya juego). El conjunto del movimiento del servomotor colocado en la base, el cual es rotatorio, se realiza a través de un sistema biela manivela. El servomotor hace girar un disco de madera de 5cm de radio. Una vara de madera de 8mm de grosor y 11cm de largo conectada a través de codos de jardinería, por una parte, al disco y por otra a la barra móvil permite que el movimiento rotatorio del eje del servomotor se convierta en uno alternativo en el eje vertical.

Se ha hecho uso de piezas tales como arandelas, topes, cola y cinta aislante para poder unificar todas las piezas entre sí.

Para terminar la construcción del modelo se ha situado la placa arduino y una fuente de alimentación del servomotor a un lado de la torre principal y otra fuente de alimentación al lado opuesto de la torre. Los botones, los LEDs y el interruptor del sensor y servomotor se han colocado encima de un pequeño tablón de madera situado en frente de la placa Arduino.

La maquetación ha sufrido cambios respecto a la idea más primitiva del modelo. Por un lado, el raíl se hizo, en un comienzo, de madera (enteramente) y debido al exagerado peso de esta pieza se vio forzada la reconsideración de un cambio. La solución fue hacer un soporte liviano de madera para adherir el raíl de aluminio encima. Por otro lado, el eje ha sido la mayor de las complicaciones ya que el modelo de servomotor ha variado durante el proceso por distintas razones (explicadas en el siguiente apartado) y con ello, el grosor del eje. Al principio se perforó la madera por donde tenía que pasar el eje con una broca de 8mm de diámetro (debido al eje que era necesario usar) que, posteriormente, he tenido que reducirlo aproximadamente a 4mm con la ayuda de un tubo de goma de 8mm, un tubo de plástico de 6mm y uno de 4mm. Por otro lado, el servomotor iba a colocarse a la altura del eje y collado con tornillos a la torre que sostiene la barra móvil. Debido al incremento del momento (fuerza multiplicado por distancia hasta el punto de rotación) cuando el motor se aleja del punto de giro de la

---

<sup>17</sup> La señal analógica es “aquella señal continua variable que se utiliza para representar un valor, tal como el valor de proceso o valor del punto de ajuste. Las configuraciones de hardware más típicas son 0 a 20mA, 4 a 20mA o 0 a 5Vcc.” (ElectricFor, 2010) A diferencia, la señal digital solo detecta o el valor máximo o el mínimo y no puede devolver un valor intermedio.

<sup>18</sup>Véase la ficha técnica en el Anexo I.

barra, se decidió oportuno crear un sistema de transmisión biela manivela y permitir al servomotor ejercer más fuerza a la barra con menos esfuerzo.

En las imágenes Anexo II se puede ver de manera más clara la evolución del montaje y el resultado final).

## 4 Esquemas eléctricos

Todos los elementos eléctricos que se han utilizado se han tenido que conectar de un modo concreto para lograr que funcionen como es deseado. Para ello, previamente se han estudiado los componentes para decidir el ensamblaje acorde a la situación.

En primer lugar, la placa Arduino UNO se alimenta de 9v por la entrada *Jack* ya existente. A continuación, en la imagen 5 se pueden ver tanto las conexiones del servomotor como las del sensor de distancia.

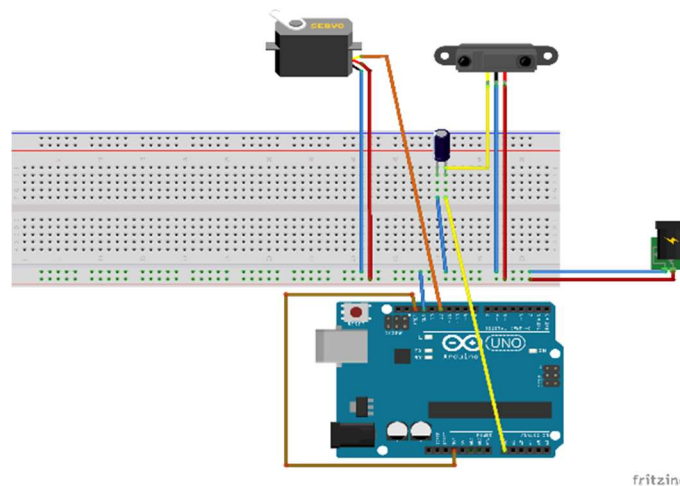


Imagen 5 “Esquema eléctrico: servomotor y sensor” (Espeso, 2015)

Por un lado, el servomotor tiene tres tomas: entrada de voltaje conectada al polo positivo de una fuente de alimentación externa de 6V<sup>21</sup>, la toma de tierra conectada al polo negativo de la fuente de alimentación y la entrada de señal digital conectada al pin 11. Por otro lado, el sensor de distancia se conecta a otras tres tomas: entrada de voltaje al polo positivo de la misma fuente de alimentación externa que el servo, toma de tierra al polo negativo de la fuente de alimentación y la entrada de señal analógica conectada al pin de entrada analógico de la placa Arduino UNO a través de un condensador

---

<sup>21</sup> El voltaje que se ha decidido suministrar al servomotor depende del valor de voltaje recomendado por el fabricante en la hoja de datos (véase en el subapartado “Datasheet TowerPro MG995” del Anexo I.

electrolítico de  $10\mu\text{F}$  (recomendado por el fabricante en la hoja de datos del sensor, véase el subapartado “Datasheet Sharp GP2Y0A21YK0F” del Anexo I). Existe la posibilidad de conectar el sensor sin el uso de un condensador, pero sin él la señal que se recibe es más irregular. En la imagen 6 se puede apreciar el efecto que causa un filtro de paso bajo (condensador de filtrado).



Imagen 6 “Efecto del condensador de filtrado” (Espeso, 2015)

En segundo lugar, todos los LEDs presentes en el modelo y los botones están conectados de la siguiente manera: alimentados por los 5V que ofrece la salida de voltaje de la placa de Arduino UNO en los pines digitales, se encuentran cuatro conexiones en paralelo. Una de ellas cuenta con una tira flexible de 9 LEDs RGB<sup>22</sup> de 3.4V según la hoja de datos que ofrece el fabricante conectados a través de una resistencia de  $320\ \Omega$  en serie al puerto digital 13 de la placa Arduino. Las otras tres conexiones constan de tres LED rojo de 1.9V con sus respectivas resistencias de  $220\ \Omega$  en serie, que se conectan en paralelo entre ellos. Tanto los botones como los LEDs se conectan a los pines digitales de la placa Arduino, los botones como *input\_pullup* (señal de entrada con una resistencia de seguridad en la placa, detecta si está siendo presionado) y los LEDs como *output* (manda el voltaje si el código se lo ordena). Las tomas a tierra se realizan en el pin GND de la placa. El esquema realizado en el programa “CrocodileClips” para hacer los cálculos y comprobaciones del funcionamiento del circuito se puede ver en la imagen 7, a continuación.

---

<sup>22</sup> Toda la información de este componente se muestra en el subapartado “Datasheet LED RGB WS2812B” del Anexo I.

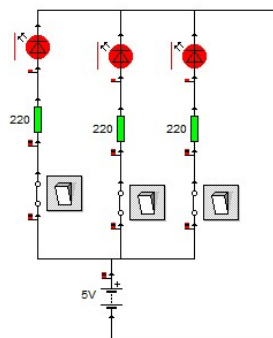


Imagen 7 “Esquema eléctrico LEDs y botones”

## 5 Código del programa

El código que se ha utilizado para controlar las acciones que envía y recibe la placa Arduino se muestra en el subapartado del Anexo III, “Código del programa”.

La estructura externa del código se ve claramente separada por los títulos entre barras inclinadas (“/”) para que no afecte al código. El programa que se ejecuta, tiene tres apartados: declaración de librerías, pines y variables, *setup* y *loop*.

### 5.1 Declaración de librerías, pines y variables

Cuando se abre el código, lo primero que se ve es la declaración de todas las librerías que se usan. Vienen referidas de la siguiente manera: “`#include<nombre de la librería>`”. En este caso se ha utilizado la librería “`Servo.h`” para controlar el servomotor, la “`PID_v1.h`” para hacer los cálculos del controlador PID, la “`SharpIR.h`” para hacer las lecturas del sensor a distancia y la “`Adafruit_NeoPixel.h`” para controlar la tira de LEDs RGB.

A continuación, se muestran los pines que ocupan cada componente electrónico respecto a la placa Arduino. Se han definido los LED para activar las componentes del controlador (PID) en los pines digitales 5,6 y 12. Por otro lado, se han definido los botones para activar los LED referidos anteriormente en los pines digitales 2,3 y 4.

Por otro lado, se ha definido el servomotor en el pin número 11, el sensor de distancia en el pin analógico A0 (ya que debe retornar valores entre 0 y 1024, solamente posible con pines analógicos) y la tira de LEDs flexible RGB en el pin 13.

Por último, en este apartado, se ha definido el número de LEDs que se han usado en la tira RGB, en este caso 9.

Para la definición de variables se pueden utilizar distintos tipos de datos. En primer lugar, se han definido variables que almacenan información numérica. En este bloque encontramos cinco variables “float” (floating-point numbers) que almacenan datos con decimales (más precisos que los enteros “int”). Éstas son la lectura anterior “last\_reading”, la distancia medida “dist” y las tres constantes. En este proyecto práctico, por sus características, ha resultado oportuno que las constantes adopten los siguientes valores para que el sistema funcione de manera correcta (no óptima ya que se han ajustado empíricamente):  $K_p = 0.65$ ,  $K_i = 0.2$  y  $K_d = 0.34$ . Además de las variables “float” se encuentran definidas siete variables “double”, las cuales doblan la precisión de las variables float ocupando, no obstante, más memoria RAM que éstas. Como variables dobles se hallan el *Setpoint*, el *Input*, el *Output*, el *ServoOutput* y otras tres constantes que se van a usar en función del estado de los botones para activar y desactivar las componentes del controlador.

Por otra banda, se tienen que definir también las funciones de las librerías, por ello encontramos las siguientes líneas de código: “PID myPID(&Input, &Output, &Setpoint, KpA, KiA, KdA, DIRECT);” donde establecemos que el nombre de la función (a la que nos referiremos más adelante) es “myPID”, la variable que determinará el *Input*, el *Output*, el *Setpoint*, las tres constantes y la manera de actuar “DIRECT” (puede actuar de manera directa o inversa). “SharpIR sensor( SharpIR::GP2Y0A21YK0F, sensorPin );” donde identificamos el modelo de sensor que corresponde y el pin que le hemos asignado en la placa Arduino. “Servo myServo;” donde definimos el nombre de la función del servo, “myServo” en mi caso. “Adafruit\_NeoPixel pixels = Adafruit\_NeoPixel(numled, LEDs, NEO\_GRB + NEO\_KHZ800);” donde asignaremos el número de LEDs usados en la tira, el pin donde los hemos conectado y el modelo marcado por el fabricante (Adafruit).

## 5.2 Setup

En este apartado del código definimos e iniciamos los procesos de cada función. En primer lugar, iniciamos el monitor serial a 9600 baudios (bits por segundo, es decir la cantidad de datos que se muestran en pantalla por segundo). En segundo lugar, establecemos que los botones serán “INPUT\_PULLUP” ya que las entradas que usan, permiten utilizar una resistencia interna de la placa como método de seguridad y los

LEDs serán “OUTPUT”. En tercer lugar, le asignaremos a los LEDs un valor “LOW”, es decir que se encuentren apagados al iniciar el programa. En cuarto lugar, le asignamos a la función “myServo” el pin donde se ha conectado el motor “ServoPin”. En quinto lugar, encontramos el *Setup* de la función PID, donde se le asigna al *input* la función “readPosition();” para que cuando el programa pida el valor *input*, se procese esa función (explicada posteriormente). También se asigna un modo automático de cálculo y se determinan los límites del servomotor, es decir 80 y -80 grados (prácticamente el máximo recorrido que puede realizar). Por último, se inicializa la función de la tira de LEDs RGB “pixels.begin(); pixels.show();”.

### 5.3 Loop

Esta parte del código es la que se va a ejecutar una y otra vez tras iniciar el programa. En este caso, se ha establecido un orden de funciones a las que se va a llamar para realizar las acciones que se le pidan en un orden concreto mostrado a continuación:

1. BotonKp();
2. BotonKi();
3. BotonKd();
4. Setpoint = 23;
5. Input = readPosition();
6. ControlPID();
7. ConsignaOK();

En primer lugar, se inicia la función “BotonKp” que se encuentra varias líneas más abajo con el siguiente comando “void BotonKp()”, cuyas acciones a realizar se encuentran delimitadas por “{“ y ”}”. Esta función realiza los siguientes pasos:

1. Crea unas variables que se usan como “contadores” únicamente en esta función (a través de una variable “static”), donde pueden adquirir dos estados “TRUE” o “FALSE”, o bien 1 o 0 (a través de una variable “boolean”):  
“static boolean s = 0; static boolean ON = 0; static unsigned long mils = 0;”
2. Empiezan dos funciones con carácter condicional: “if (condición){ ... }”, donde ocurren ciertas acciones dependiendo de que condición se cumpla. Si el botón Kp se pulsa, entonces la variable ON será 1 (es decir !ON cambia el estado de ON a su contrario, como se ha establecido que ON = 0, ahora es 1). Posteriormente se multiplica Kp por ON para mantener la constante que se usará

en el controlador (KpA como se ha mostrado en el *Setup* y no Kp). Por último, se enciende el LED correspondiente si se encuentra en este estado.

Por lo contrario, si el botón no se pulsa, KpA será 0 (ya que así se ha definido en las variables) y el LED permanecerá apagado.

Este ciclo se repite para las constantes Ki y Kd.

A continuación, se asigna al *Setpoint* el valor 23 ya que esa es la medición que hemos obtenido al colocar la pelota en el centro de la barra móvil y se llama a la función "ReadPosition" que asignará el valor momentáneo del *Input*. Esta función determina la lectura del sensor de distancia. En primer lugar, le aplicamos un retraso de 50ms para no sobreponer los datos de las lecturas (recomendado por el fabricante). En segundo lugar, iniciaremos la función de la librería de este componente "getDistance" la cual retorna directamente la distancia en centímetros en vez de tener que calcular la función que relaciona el voltaje que lee el sensor respecto a la distancia (ya que no es una función lineal ni hiperbólica, sino exponencial). Por otro lado, asignaremos los límites de lectura a 55 cm ya que es el extremo opuesto de la barra móvil. En último lugar, se imprimen la distancia obtenida en el monitor serial a través de la función "Serial.println (dist)".

Seguidamente se inicia la función "ControlPID". Ésta ejecuta los cálculos con el comando "Compute". Para ello, necesita el *Input*, que vendrá dado por la función "last\_reading" explicada más adelante. Por otro lado, establece que el ángulo de corrección que reciba el servomotor ("ServoOutput") será la suma de la corrección que haga el controlador más la posición cero del servomotor (también llamada Offset) en mi caso, sobre una superficie plana, 120.5 grados.

A continuación, definimos que ha de ejecutar la función que determina la lectura del sensor de distancia. En primer lugar, le aplicamos un retraso de 50ms para no sobreponer los datos de las lecturas (recomendado por el fabricante). En segundo lugar, iniciaremos la función de la librería de este componente "getDistance" la cual retorna directamente la distancia en centímetros en vez de tener que calcular la función que relaciona el voltaje que lee el sensor respecto a la distancia (ya que no es una función lineal ni hiperbólica, sino exponencial). Por otro lado, asignaremos los límites de lectura a 55 cm ya que es el extremo opuesto de la barra móvil. En último lugar, se imprimen la distancia obtenida en el monitor serial a través de la función "Serial.println (dist)".

Por último, se ha asignado un rango de aproximadamente tres centímetros entorno al centro de la barra para que un LED verde se encienda si la pelota se encuentra en esa



situación, indicando que ha logrado corregir el error correctamente. Esto se hace a través del comando `"pixels.setPixelColor (4, 0, 255, 0); pixels.show();"` ya que encendemos un LED de la tira RGB y no un diodo.

Todas las funciones explicadas que permanecen dentro del *Loop* se repiten constantemente hasta que manualmente lo cesamos abriendo el circuito que alimenta tanto el servomotor y el sensor como la placa a través de dos interruptores.

## 6 Puesta en marcha

Una vez el código se carga en la memoria de la placa Arduino y ésta lo ejecuta, el controlador se pone en marcha.

En primer lugar, la placa toma unos segundos para inicializar el programa lo cual puede ser detectado con el encendido de los LED integrados en la placa. Una vez este tiempo ha terminado, el servomotor se colocará en su posición cero ( $120,5^\circ$ ) para dejar la barra móvil en una posición de equilibrio, es decir horizontal. Si en este momento se coloca la pelota de tenis de mesa en cualquier punto de la barra, el controlador no actuará. Eso se debe a que el programa se inicia con las tres constantes anuladas (constantes = 0). Si primeramente pulsamos el botón asignado a la constante de proporcionalidad (con la letra "P"), es posible ver como la barra se va a inclinar de acorde a la corrección proporcional. Esto va a resultar en que la barra se va a inclinar hacia el lado opuesto según la distancia que le reste la pelota al centro de la barra multiplicado por la constante "Kp" (0.65 en este caso). El resultado va a ser que, visualmente, la pelota va a ir de un lado a otro sin apenas corregir y, matemáticamente, que el error va a tender a 0 ya que se multiplica por un número menor a 1.

En segundo lugar, si solamente pulsamos el botón asignado a la constante integral (Ki, estando Kp y Kd anuladas), será posible ver que el servomotor solamente va a corregir el error si la pelota se mantiene en el tiempo en una posición que no sea el centro. Visualmente, si se atara la pelota en una posición alejada a la deseada la barra se inclinará cada vez más y, matemáticamente, el error iría acumulando debido a la integral y la corrección sería cada vez mayor.

En último lugar, si solamente se encuentra pulsado el botón asignado a la constante derivativa (Kd), se puede apreciar que solamente corregirá la velocidad de la pelota que sea distinta a cero y no afectará a la posición. Matemáticamente, debido a que se deriva

la ecuación de la posición, se obtiene la ecuación de la velocidad. Entonces, lo que se hará es intentar mantener la ecuación derivada anulada (es decir, cero) para frenar la pelota.

Cuando se juntan varias componentes del controlador se pueden ver correcciones distintas del error dependiendo de cuales se activen.

Si el ruido, es decir las interferencias en el sistema, no es demasiado elevado, será óptimo utilizar un controlador PID (activando las tres componentes) para lograr una corrección rápida, eficaz y precisa.

## CONCLUSIÓN

---

Después de los últimos seis intensos meses de trabajo se puede concluir que sí que es posible adaptar un controlador PID en un microcontrolador de Arduino teniendo en cuenta sus limitaciones. Aun considerando que se ha logrado aquello que se buscaba, se ha de ser consciente de los múltiples obstáculos que este proyecto ha presentado, tanto en el marco teórico como en el práctico. Algunos de estos son tales como la difícil comprensión de cálculos que se realizan debido a la carencia de ciertos conocimientos matemáticos, las diversas veces que se ha tenido que repetir o replantear el proyecto práctico debido a aspectos que no se habían considerado importantes y que a la hora de realizarlo han sido un impedimento o la limitación que presenta el microcontrolador de 8 bits de Arduino y por lo tanto la limitada memoria RAM (donde se almacena aquello que programas) de 32Kb y dinámica (la que viene preinstalada en el microcontrolador para ejecutar aquello que se le programa) de 2Kb de este componente lo cual ha impedido ampliar el código para mejorarlo e incluso se ha visto afectado el rendimiento del controlador.

Como posibles mejoras del proyecto se presentan las siguientes propuestas:

- Cambio de Arduino UNO a Arduino Mega debido a la limitación de procesamiento y almacenaje de el primero respecto al segundo o bien la optimización del código a través del uso de variables menos precisas ("byte" o "int") que ocupen menos memoria, utilizar ciertas funciones que permitan almacenarse en la memoria dinámica y no en la RAM, etc.
- Como mejora del propio proyecto se propone implementar un potenciómetro y un nivel para poder variar el *Setpoint* si el terreno no es regular.
- Se puede considerar la opción de mejorar el sensor adquiriendo uno de mayor calidad que retorne las mediciones con menos ruido e interferencias que afectan al rendimiento del controlador.
- Por último, sería oportuno revisar los materiales usados para el ensamblaje de esta maqueta. Se propone el uso de una impresora 3D para mejorar la precisión de las piezas y de su diseño.

## BIBLIOGRAFÍA

---

- Amadori, A. (2010). *Algoritmo de control PID*. Madrid: El ABC de la automatización.
- Arduino. (2019). *Arduino*. Obtenido de [www.arduino.cc](http://www.arduino.cc)
- Areny, R. P. (2008). *Sensores y sensado*. Castelldefels: Escuela Politécnica Superior de Castelldefels, UPC.
- Aström, K. J., & Hägglund, T. (1995). *PID Controllers, 2nd Edition*. Carolina del Norte.
- Aström, K. J., & Murray, R. M. (2009). *An Introduction for Scientists and Engineers*. New Jersey: Princeton University Press.
- Banzi, M. (2012). *Arduino Due is finally here*. Arduino.
- Banzi, M. (6 de Octubre de 2017). Arduino reborn partners with ARM. (R. Wilson, Entrevistador)
- Barragán, H. (2016). *The Untold History of Arduino*. MIT.
- Bennett, S. (1984). Steering of Ships. *Control Systems Magazine*, 10-11.
- Callender, A., Hartree, D., & Porter, A. (1935/6). Time lag in a control system. *Philosophical Trans Royal Society of London*, 235, 415-444.
- Cantarero, T. Á. (s.f.). *Diseño del Controlador PID*. Sevilla: Departamento de Ingeniería de Sistemas y Automática, Universidad de Sevilla.
- Cuartielles, D. (5 de Abril de 2013). Arduino FAQ - with David Cuartielles. (Madea, Entrevistador)
- D'Ausilio, A. (2011). *Arduino: A low-cost multipurpose lab equipment*.
- ElectricFor. (2010). *Diccionario-Electricfor*. Obtenido de <http://www.electricfor.es/es/18072/diccionario/Salida-analogica.htm>
- Espeso, Á. (18 de Agosto de 2015). *Estudio Roble*.
- Fry, B., & Reas, C. (2019). *Processing*. Obtenido de [processing.org](http://processing.org)
- Fry, C. R. (23 de Septiembre de 2009). Interview with Casey Reas and Ben Fry. (D. Shiffman, Entrevistador)
- García, F. M. (2007). *Controladores PID, Ajuste en frecuencia*. Madrid: ETSI de Informática, UNED.
- García, F. M. (2007). *El controlador PID*. Madrid: ETSI de Informática, UNED.

- García, J. R., González Sarabia, E., Fernández Pérez, D., Torre Ferrero, C., & Robla Gómez, M. S. (2013). *Automática, Capítulo 3*. Cantabria: Universidad de Cantabria.
- Hadi, G. S., Varianto, R., Riyanto, B. T., & Budiyo, A. (2014). *Autonomous UAV System Development for Payload Dropping Mission*. Indonesia: The Journal of Instrumentation, Automation and Systems.
- Honeywell, D. (2000). *PID control, chapter 8*.
- Jimblom. (6 de Agosto de 2013). *Sparkfun. Arduino Shields*. Obtenido de [www.sparkfun.com](http://www.sparkfun.com)
- Jurado, J. G. (2012). *Diseño de sistemas de control multivariable por desacoplo con controladores PID*. Madrid: Escuela Técnica Superior de Ingeniería Informática.
- Iha(3057). (22 de 03 de 2013). *planetmath.org*. Obtenido de [planetmath.org](https://planetmath.org): <https://planetmath.org>
- Mazzone, V. (2002). *Controladores PID, Control Automático 1*. Quilmes: Automatización y Control Industrial Universidad Nacional de Quilmes.
- Minorsky, N. (1922). Directional stability of automatically steered bodies. *Journal of the American Society of Naval Engineering, Vol. 34*, 284.
- Moreno, M. A. (2001). *Apuntes de control PID*. La Paz: Universidad Mayor de San Andrés.
- Moreno, M. A. (2001). *Apuntes de Control PID*. Bolivia: Universidad Mayor de San Andrés la Paz .
- Naboa, M. G. (2012). *Determinación de los parámetros de un controlador PID para una planta con función de transferencia conocida*. Xalapa, Veracruz: Universidad Veracruzana .
- O'Dwyer, A. (2005). *PID control: the early years*. Cork: Cork Institute of Technology.
- Pérez, M. E., Benítez Baltazar, V. H., Pacheco Ramírez, J. H., & Montaña Valle, F. (2014). *Diseño de controladores P, PI y PID para el comportamiento dinámico de un servo-sistema hidráulico, basado en desarrollo experimental*. Epistemus. Real Academia de Ingeniería. (2010). *Diccionario Español de Ingeniería*. Obtenido de <http://www.diccionario.raing.es>
- Veluvolu, D. K. (2011). *Multivariable Control for MIMO processes*. KNU.
- Verrastro, C., Alberino, S., & Folino, P. (2005). *Control PID con esquema adaptivo de filtrado de ruido*. Buenos Aires: UTN-FRBA.



### Absolute Maximum Ratings

(T<sub>a</sub>=25°C, V<sub>CC</sub>=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V <sub>CC</sub>	-0.3 to +7	V
Output terminal voltage	V <sub>OL</sub>	-0.3 to V <sub>CC</sub> +0.3	V
Operating temperature	T <sub>amb</sub>	-10 to +60	°C
Storage temperature	T <sub>stg</sub>	-40 to +70	°C

### Electro-optical Characteristics

(T<sub>a</sub>=25°C, V<sub>CC</sub>=5V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Average supply current	I <sub>CC</sub>	L=80cm (Note 1)	—	30	40	mA
Distance measuring	ΔL	(Note 1)	10	—	30	cm
Output voltage	V <sub>OL</sub>	L=80cm (Note 1)	0.25	0.4	0.55	V
Output voltage differential	ΔV <sub>OL</sub>	Output voltage difference between L=10cm and L=80cm (Note 1)	1.65	1.9	2.15	V

\* L: Distance to reflective object

Note 1: Using reflective object: White paper (Made by Kodak Co., Ltd. gray cards R-27: white face, reflectance; 90%)

### Recommended operating conditions

Parameter	Symbol	Rating	Unit
Supply voltage	V <sub>CC</sub>	4.5 to 5.5	V

Fig. 1 Timing chart

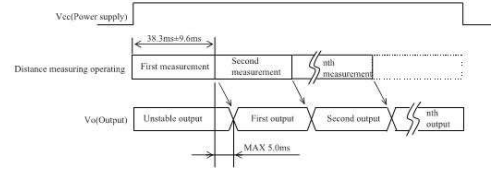
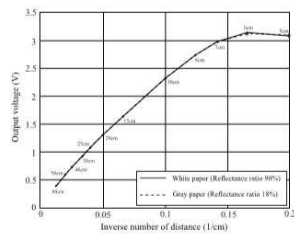
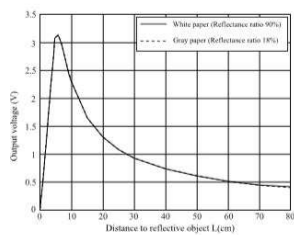


Fig. 2 Example of distance measuring characteristics(output)



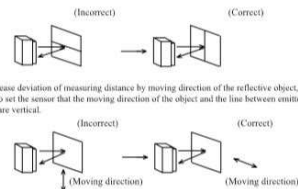
### Notes

#### Advice for the optics

- The lens of this device needs to be kept clean. There are cases that dust, water or oil and so on deteriorate the characteristics of this device. Please consider in actual application.
- Please don't do washing. Washing may deteriorate the characteristics of optical system and so on. Please confirm resistance to chemicals under the actual usage since this product has not been designed against washing.

#### Advice for the characteristics

- In case that an optical filter is set in front of the emitter and detector portion, the optical filter which has the most efficient transmittance at the emitting wavelength range of LED for this product (λ = 870 ± 70nm), shall be recommended to use. Both faces of the filter should be mirror polishing. Also, as there are cases that the characteristics may not be satisfied according to the distance between the protection cover and this product or the thickness of the protection cover, please use this product after confirming the operation sufficiently in actual application.
- In case that there is an object near to emitter side of the sensor between sensor and a detecting object, please use this device after confirming sufficiently that the characteristics of this sensor do not change by the object.
- When the detector is exposed to the direct light from the sun, tungsten lamp and so on, there are cases that it can not measure the distance exactly. Please consider the design that the detector is not exposed to the direct light from such light source.
- Distance to a mirror reflector can not be sometimes measured exactly.
- In case of changing the mounting angle of this product, it may measure the distance exactly.
- In case that reflective object has boundary line which material or color etc. are excessively different, in order to decrease deviation of measuring distance, it shall be recommended to set the sensor that the direction of boundary line and the line between emitter center and detector center are in parallel.



#### Advice for the power supply

- In order to stabilize power supply line, we recommend to insert a by-pass capacitor of 10μF or more between Vcc and GND near this product.

#### Notes on handling

- There are some possibilities that the internal components in the sensor may be exposed to the excessive mechanical stress. Please be careful not to cause any excessive pressure on the sensor package and also on the PCB while assembling this product.

### ● Presence of ODC etc.

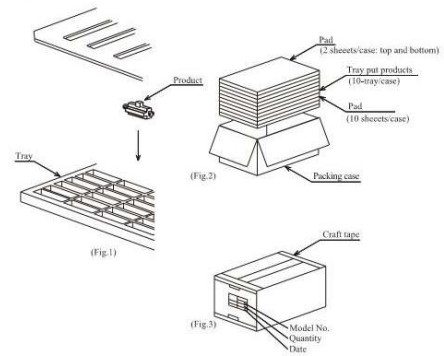
This product shall not contain the following materials.  
And they are not used in the production process for this product.  
Regulation substances : CFCs, Halon, Carbon tetrachloride, 1,1,1-Trichloroethane (Methylchloroform)

Specific brominated flame retardants such as the PBB and PBDE are not used in this product at all.

This product shall not contain the following materials banned in the RoHS Directive (2011/65/EU).  
• Lead, Mercury, Cadmium, Hexavalent chromium, Polybrominated biphenyls (PBB), Polybrominated diphenyl ethers (PBDE).

### ■ Package specification

#### Package composition



#### Packaging method

1. Put products of 100pcs. in tray. packing method is showed in the above fig.(Fig.1)
2. Put them (10-tray) in the packing box. Put pads on their top and bottom.  
And put pads on each trays (Total 10 sheets) (Fig.2).
3. Seal the packing box with craft tape.  
Print the model No., quantity, inspection date (1000 pcs./a packing box) (Fig.3).

### ■ Important Notices

The circuit application examples in this publication are provided to explain representative applications of SHARP devices and are not intended to guarantee any circuit design or license any intellectual property rights. SHARP takes no responsibility for any problems related to any intellectual property right of a third party resulting from the use of SHARP's devices.

Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device. SHARP reserves the right to make changes in the specifications, characteristics, data, materials, structure, and other contents described herein at any time without notice in order to improve design or reliability. Manufacturing locations are also subject to change without notice.

Observe the following points when using any devices in this publication. SHARP takes no responsibility for damage caused by improper use of the devices which does not meet the conditions and absolute maximum ratings to be used specified in the relevant specification sheet nor meet the following conditions:

- (i) The devices in this publication are designed for use in general electronic equipment designs such as:
  - Personal computers
  - Office automation equipment
  - Telecommunication equipment [terminal]
  - Test and measurement equipment
  - Industrial control
  - Audio-visual equipment
  - Consumer electronics
- (ii) Measures such as fail-safe function and redundant design should be taken to ensure reliability and safety when SHARP devices are used for or in connection

with equipment that requires higher reliability such as:

- Transportation control and safety equipment (i.e., aircraft, trains, automobiles, etc.)
- Traffic signals
- Gas leakage sensor breakers
- Alarm equipment
- Various safety devices, etc.
- (iii) SHARP devices shall not be used for or in connection with equipment that requires an extremely high level of reliability and safety such as:
  - Space applications
  - Telecommunication equipment [trunk lines]
  - Nuclear power control equipment
  - Medical and other life support equipment (e.g., scuba).

If the SHARP devices listed in this publication fall within the scope of strategic products described in the Foreign Exchange and Foreign Trade Law of Japan, it is necessary to obtain approval to export such SHARP devices.

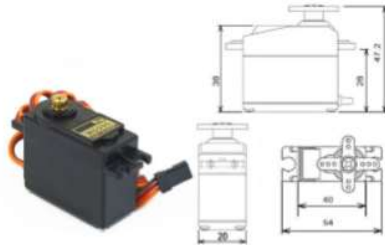
This publication is the proprietary product of SHARP and is copyrighted, with all rights reserved. Under the copyright laws, no part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, in whole or in part, without the express written permission of SHARP. Express written permission is also required before any use of this publication may be made by a third party.

Contact and consult with a SHARP representative if there are any questions about the contents of this publication.



## Datasheet Servomotor TowerPro Mg995

### MG995 High Speed Metal Gear Dual Ball Bearing Servo



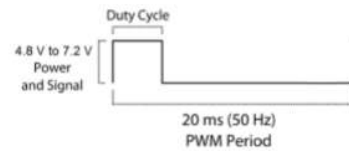
The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

This high-speed standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG995 Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!


#### Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 8.3 kgf·cm (4.8 V), 10 kgf·cm (6 V)
- Operating speed: 0.2 s/60° (4.8 V), 0.16 s/60° (6 V)
- Operating voltage: 4.8 V to 7.2 V
- Dead band width: 5 μs
- Stable and shock proof double ball bearing design
- Temperature range: 0 °C – 55 °C

PWM=Orange (⏏)  
Vcc = Red (+)  
Ground=Brown (-)



# Datasheet LED RGB WS2812B



## WS2812B

Intelligent control LED  
integrated light source

**Features and Benefits**

- Intelligent reverse connect protection, the power supply reverse connection does not damage the IC.
- The control circuit and the LED share the only power source.
- Control circuit and RGB chip are integrated in a package of 5050 components, form a complete control of pixel point.
- Built-in signal reshaping circuit, after wave reshaping to the next driver, ensure wave-form distortion not accumulate.
- Built-in electric reset circuit and power lost reset circuit.
- Each pixel of the three primary color can achieve 256 brightness display, completed 16777216 color full color display, and scan frequency not less than 400Hz/s.
- Cascading port transmission signal by single line.
- Any two point the distance more than 5m transmission signal without any increase circuit.
- When the refresh rate is 30fps, cascade number are not less than 1024 points.
- Send data at speeds of 800Kbps.
- The color of the light were highly consistent, cost-effective.

**Applications**


- Full-color module, Full color soft lights a lamp strip.
- LED decorative lighting, Indoor/outdoor LED video irregular screen.

**General description**

WS2812B is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant current control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

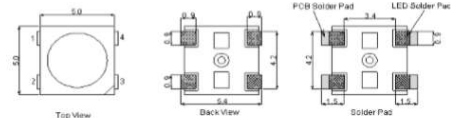
LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angle is large, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.



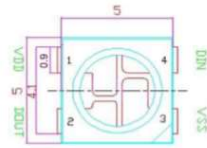
## WS2812B

Intelligent control LED  
integrated light source

**Mechanical Dimensions**



**PIN configuration**




**PIN function**

NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

**Absolute Maximum Ratings**

Parameter	Symbol	Ratings	Unit
Power supply voltage	$V_{DD}$	+3.5~+5.3	V
Input voltage	$V_i$	-0.5~+VDD+0.5	V
Operation junction temperature	$T_{opt}$	-25~+80	°C
Storage temperature range	$T_{stg}$	-40~+105	°C

**Electrical Characteristics** ( $T_A=-20\sim+70^{\circ}\text{C}$ ,  $V_{DD}=4.5\sim5.5\text{V}$ ,  $V_{SS}=0\text{V}$ , unless otherwise specified)



## WS2812B

Intelligent control LED  
integrated light source

Parameter	Symbol	conditions	Min	Typ	Max	Unit
Input current	$I_i$	$V_i=V_{DD}/V_{SS}$	—	—	$\pm 1$	$\mu\text{A}$
Input voltage level	$V_{IH}$	$D_{OUT}$ SET	0.7 $V_{DD}$	—	—	V
	$V_{IL}$	$D_{OUT}$ SET	—	—	0.3 $V_{DD}$	V
Hysteresis voltage	$V_H$	$D_{OUT}$ SET	—	0.35	—	V

**Switching characteristics** ( $T_A=-20\sim+70^{\circ}\text{C}$ ,  $V_{DD}=4.5\sim5.5\text{V}$ ,  $V_{SS}=0\text{V}$ , unless otherwise specified)

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Transmission delay time	$t_{PLZ}$	$CL=15\text{pF}$ , $DIN\rightarrow DOUT$ , $R_L=10\text{k}\Omega$	—	—	300	ns
Fall time	$t_{FALL}$	$CL=300\text{pF}$ , $DOUT\rightarrow TG\rightarrow OUTB$	—	—	120	$\mu\text{s}$
Input capacity	$C_i$	—	—	—	15	pF


**RGB IC characteristic parameter**

Emitting color	Model	Wavelength(nm)	Luminous intensity(mcd)	Voltage(V)
Red	13CBAUP	620-625	390-420	2.0-2.2
Green	13CGAUP	522-525	660-720	3.0-3.4
Blue	10RIMUX	465-467	180-200	3.0-3.4

**Data transfer time(TH+TL=1.25 $\mu\text{s}$ ±600ns)**

		0 code, high voltage time	0.4us	±150ns
TOH	0 code, high voltage time			
TIH	1 code, high voltage time	0.8us		±150ns
TOL	0 code, low voltage time	0.85us		±150ns
TIL	1 code, low voltage time	0.45us		±150ns
RES	low voltage time	Above 50us		

**Sequence chart:**



## WS2812B


Intelligent control LED  
integrated light source

0 code  $\overleftarrow{\text{TOH}} \overrightarrow{\text{TOL}}$

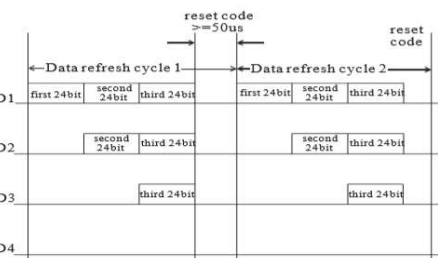
1 code  $\overleftarrow{\text{TIH}} \overrightarrow{\text{TIL}}$

RET code  $\overleftarrow{\text{Treset}}$

**Cascade method:**



**Data transmission method:**





## WS2812B

Intelligent control LED  
integrated light source

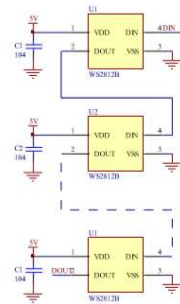
Note: The data of D1 is send by MCU, and D2, D3, D4 through pixel internal reshaping amplification to transmit.

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Note: Follow the order of GRB to send data and the high bit sent at first.

Typical application circuit:



## ANEXO II

---

### Imágenes de la evolución del modelo práctico

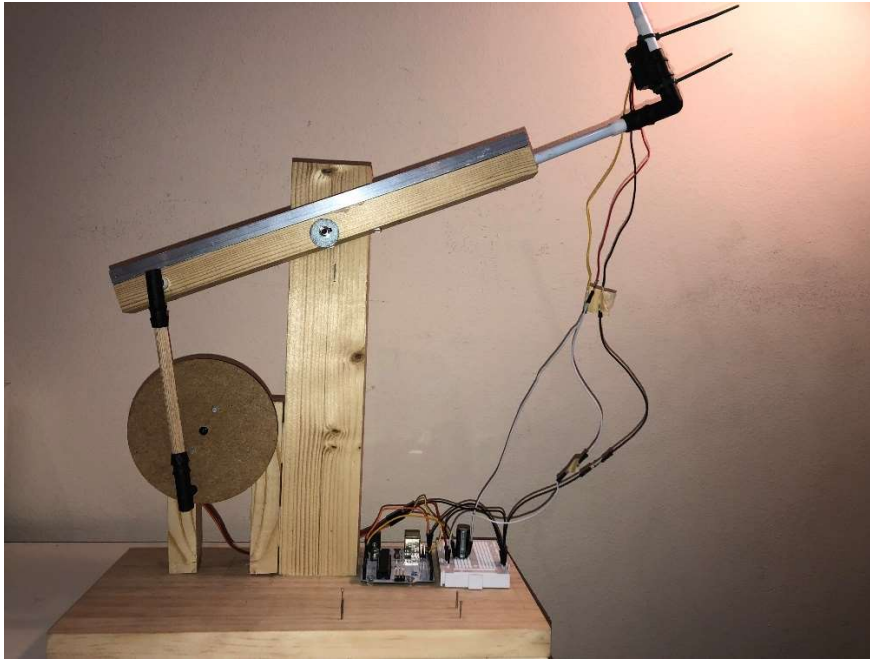
Primera maqueta sin cableado: vista frontal



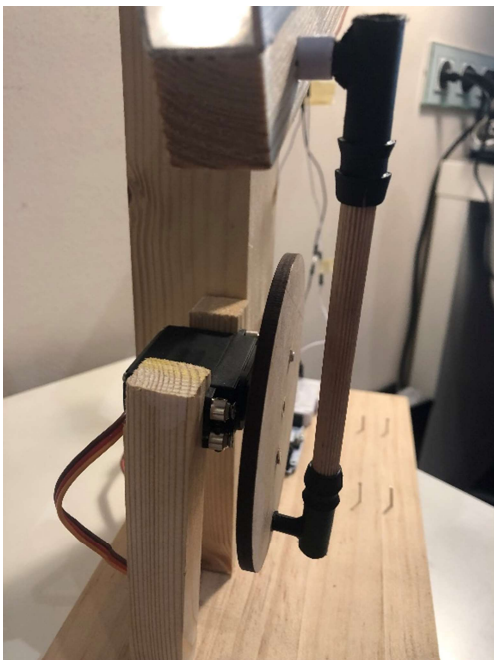
Primera maqueta sin cableado: vista de planta



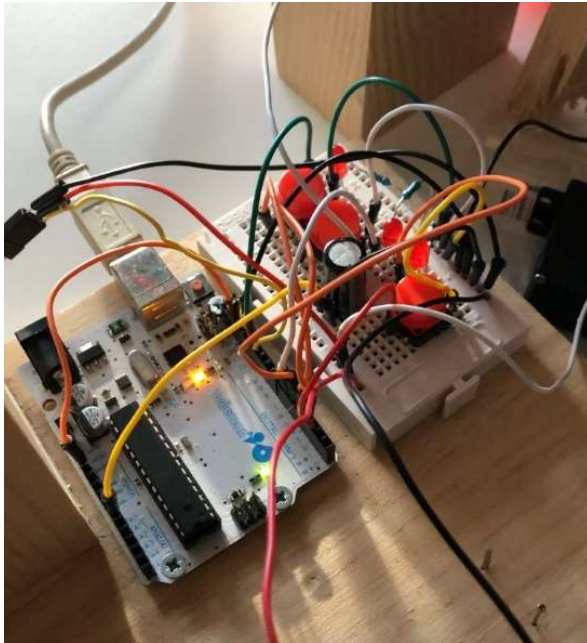
Segunda maqueta con cableado: vista frontal



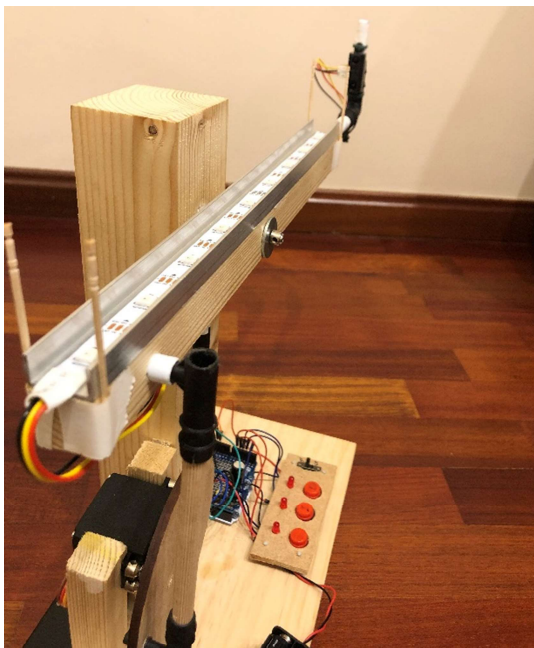
Segunda maqueta con cableado: vista lateral



Cableado en placa de prototipado

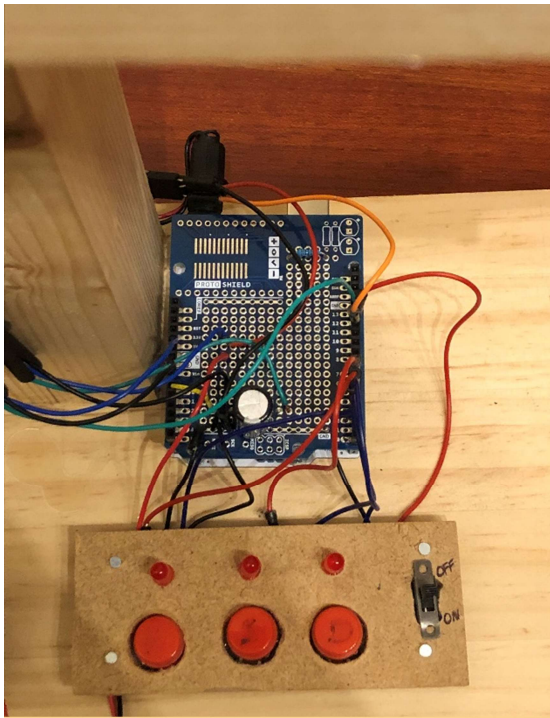


Maqueta final: vista oblicua





Cableado final sobre placa *Shield*



## ANEXO III

---

### Código del programa

```
//// LIBRERIAS ////
#include<Servo.h>
#include<PID_v1.h>
#include <SharpIR.h>
#include <Adafruit_NeoPixel.h>

//// PINES ////
#define pinLedKp 5
#define pinLedKi 6
#define pinLedKd 12
#define pinKp 2
#define pinKi 3
#define pinKd 4
#define servoPin 11
#define sensorPin A0
#define LEDs 13
#define numled 9

//// VARIABLES ////
float last_reading = 0;
float Kp = 0.65;
float Ki = 0.2;
float Kd = 0.34;
double Setpoint, Input, Output, ServoOutput;
double KpA = 0, KiA = 0, KdA = 0;
float dist;

PID myPID(&Input, &Output, &Setpoint, KpA, KiA, KdA, DIRECT);

SharpIR sensor( SharpIR::GP2Y0A21YK0F, sensorPin );
Servo myServo;

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(numled, LEDs, NEO_GRB +
NEO_KHZ800);

////////////////////
//// SETUP ////
```



```

////////////////////

void setup() {

    Serial.begin(9600);

    //// PIN MODE ////
    pinMode(pinKp, INPUT_PULLUP);
    pinMode(pinKi, INPUT_PULLUP);
    pinMode(pinKd, INPUT_PULLUP);
    pinMode(pinLedKp, OUTPUT);
    pinMode(pinLedKi, OUTPUT);
    pinMode(pinLedKd, OUTPUT);

    digitalWrite(pinLedKp, LOW);
    digitalWrite(pinLedKi, LOW);
    digitalWrite(pinLedKd, LOW);

    myServo.attach(servoPin);

    //// PID ////
    Input = readPosition();
    myPID.SetMode(AUTOMATIC);
    myPID.SetOutputLimits(-80, 80);

    //// LEDS ////
    pixels.begin();
    pixels.show();

}

////////////////////
//// LOOP ////
////////////////////

void loop() {

    BotonKp();
    BotonKi();
    BotonKd();
    Setpoint = 23;
    Input = readPosition();
    ControlPID();
}

```

```

ConsignaOK();
}

//// BOTON KP ////
void BotonKp() {
    static boolean s = 0;
    static boolean ON = 0;
    static unsigned long mils = 0;

    if (!s) {                                     //Si no está pulsado
        if (digitalRead(pinKp) == 0) {
            s = 1;
            mils = millis();
            ON = !ON;
            KpA = Kp * ON;
            myPID.SetTunings(KpA, KiA, KdA);
            digitalWrite(pinLedKp, ON);
        }
    }

    if (s) {                                     //Si está pulsado
        if (digitalRead(pinKp) == 1 && millis() > mils + 100) {
            s = 0;
        }
    }
}

//// BOTON KI ////
void BotonKi() {
    static boolean s = 0;
    static boolean ON = 0;
    static unsigned long mils = 0;

    if (!s) {
        if (digitalRead(pinKi) == 0) {
            s = 1;
            mils = millis();
            ON = !ON;
            KiA = Ki * ON;
            myPID.SetTunings(KpA, KiA, KdA);
            digitalWrite(pinLedKi, ON);
        }
    }
}

```

```

    }
}

if (s) {
    if (digitalRead(pinKi) == 1 && millis() > mils + 100) {
        s = 0;
    }
}
}

//// BOTON KD ////
void BotonKd() {
    static boolean s = 0;
    static boolean ON = 0;
    static unsigned long mils = 0;

    if (!s) {
        if (digitalRead(pinKd) == 0) {
            s = 1;
        }
    }

    mils = millis();
    ON = !ON;
    KdA = Kd * ON;
    myPID.SetTunings(KdA, KiA, KdA);
    digitalWrite(pinLedKd, ON);
}
}

if (s) {
    if (digitalRead(pinKi) == 1 && millis() > mils + 100) {
        s = 0;
    }
}
}

//// CONTROL PID ////

void ControlPID() {

    myPID.Compute();
}

```

```

last_reading = Input;
ServoOutput = 120.5 + Output;

myServo.write(ServoOutput);
}

//// PLOTTER POSICIÓN ////
float readPosition() {
    delay(50);

    dist = sensor.getDistance();
    if (dist > 55)
    {
        dist = 55;
    }

    Serial.println(dist);
    return dist;
}

////OK ////

void ConsignaOK() {
    if ( abs(dist) >= 20 && (abs(dist) <= 25)) {
        pixels.setPixelColor (4, 0, 255, 0); pixels.show();
    }
    else {
        pixels.setPixelColor (4, 0, 0, 0);
        pixels.show();
    }
}
}

```