Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

CONTROL DE POSICIÓN DE UNA BOLA MEDIANTE ARDUINO PARA EL ANÁLISIS DE CONTROLADORES PID

***DOCUMENTO UNITARIO** - MEMORIA*

Alumno/Alumna: Sedano, Jimenez, Asier

Director/Directora: Sainz de Murieta, Mangado, Joseba Andoni

Curso: 2018-2019

Fecha: Bilbao, 22, 7, 2019

RESUMEN

Este proyecto expone el control de posición de una bola a través de la programación en Arduino, para el análisis de diferentes configuraciones del controlador PID. El caso práctico desarrollado consiste en un sistema barra y bola, el cual es un sistema no lineal e inestable. Se han implementado diferentes configuraciones del controlador PID para observar su funcionamiento y obtener conclusiones acerca de ellos.

El objetivo principal de este proyecto es el de utilizar un microcontrolador para el control de una maqueta de control de posición de una bola, con el objetivo de poder acercar a los alumnos que no conocen la teoría de control o que se adentran por primera vez en este mundillo de la regulación como se puede controlar un sistema de manera real. En este caso, se va a utilizar la maqueta como método de docencia. Para ello, se ha realizado todo el control del sistema en la programación de Arduino consiguiendo un control aceptable de la maqueta, puesto que el error de posición de la bola es mínimo.

Con lo cual, para que un alumno que no conozca la teoría de control pueda entender cómo funciona un controlador PID, se ha implementado una interfaz de usuario en el sistema, la cual utiliza un panel de control diseñado para conseguir una comunicación interactiva entre usuario y control de la maqueta, para así poder modificar cada una de las acciones de los controladores PID implementados y ver qué sucede en el control.

Hay que decir que para la consecución del proyecto, uno de los objetivos era obtener los distintos parámetros de los controladores implementados en la programación, a través de la realización de una simulación del sistema por medio de Matlab, en la cual no se tuvo éxito, debido a las aproximaciones y linealizaciones realizadas a la hora de obtener el modelo de la planta. Por ello, las constantes utilizadas en los distintos controladores implementados en el sistema se obtuvieron mediante el método de ajuste por ensayo-error logrando buenos resultados.

En general se puede decir que se han cumplido todos los objetivos impuestos al inicio, y que se han adquirido diferentes conocimientos acerca de diferentes materias que antes se desconocían. De todo el proceso de estudio y diseño, se presentan varias conclusiones y se proponen algunas líneas de mejora, como pueda ser realizar una

interfaz de usuario gráfica que proporcione el comportamiento del sistema en tiempo real.

Palabras clave: controlador PID, placa PCB, interfaz de usuario.

LABURPENA

Proiektu honek bola bateko posizioaren kontrola azaltzen du, Arduinoaren programazioaren bidez kontrolatzaile desberdin analizatzeko. Garatutako kasu praktikoa barra eta pilota sistema bat da, sistema hau sistema ez lineal eta ezegonkorra da. PID-a kontrolatzailearen konfigurazio desberdinak inplementatu dira bere funtzionamendua ikusteko eta ondorioak haiei buruz lortu ahal izateko.

Proiektu honen helburu nagusia, mikrokontrolatzaile bat ondo erabiltzea da maketa baten kontrola kontrolatzeko, honekin kontrolaren teoria ezagutzen ez duten edo lehen aldiz sartzen diren ikasleei hurbiltzeko. Kasu honetan, maketa irakaskuntza metodo gisa erabiliko da. Horretarako, sistemaren kontrol guztia Arduino programazioan burutu da, maketaren kontrol onargarria lortuz, zeren eta, pilotaren posizio errorea oso txikia baita.

Honen bidez, kontrol-teoria ezagutzen ez duen ikasle batek PID kontroladorea nola funtzionatzen duen ulertu ahal izango du, horretarako erabiltzaile-interfazea inplementatu da sisteman, erabiltzailearen eta maketa kontrolaren arteko komunikazio interaktiboa lortzeko aginte-panel bat diseinatu da, horrekin kontrolatzaileen akzioetako bakoitza aldatu ahal izateko esta zer gertatzen den kontrolean ikusteko.

Proiektuaren lorpenerako esan behar da, helburuetako bat programan ezarritako kontrolagailuen parametroak lortzea izan zen, Matlab-en bidez sistemaren simulazio bat eginez, arrakasta gabe, sistema fisikoaren modeloa lortu ahal izateko egindako hurbilketak eta linealtasunak direla eta. Hori dela eta, sisteman ezarritako kontrolagailu desberdinetan erabiltzen diren konstanteak akats-doikuntza metodoaren bidez lortu ziren, emaitza onak lortuz.

Oro har, hasieran ezarritako helburu guztiak bete direla esan daiteke, eta lehenago ezezagunak ziren hainbat gairi buruzko ezagutzak eskuratu direla esan daiteke. Azterketa eta diseinu prozesu osoari esker, hainbat ondorio aurkezten dira eta hobekuntza bide batzuk proposatzen dira, esate baterako, erabiltzaile-interfaze grafiko bat sisteman inplementatzea, sistemaren portaera denbora errealean ikusi ahal izateko.

Gako-hitzak: PID kontrolatzailea, PCB plaka, erabiltzaile-interfazea.

ABSTRACT

This project exposes the control of position of a ball through the programming in Arduino, for the analysis of different configurations of the PID controller. The practical case developed consists of a bar and ball system, which is a non-linear and unstable system. Different configurations of the PID controller have been implemented to observe its operation and draw conclusions about them.

The main objective of this project is to use a microcontroller to control a model of ball position control, with the aim of bringing students who do not know control theory understand how to control a system in a way real. In this case, the model will be used as a teaching method. For this, all the control of the system has been carried out in the Arduino programming, obtaining an acceptable control of the model, since the position error of the ball is minimal.

To achieve the aforementioned objective, a user interface has been implemented in the system, which uses a control panel designed to achieve an interactive communication between user and control of the model, in order to modify each of the actions of the controllers PID implemented and see what happens in the control, and so the user can draw conclusions from each of the parameters of the different implemented PIDs.

It must be said that for the achievement of the project, one of the objectives was to obtain the different parameters of the controllers implemented in the programming, through the realization of a simulation of the system by means of Matlab, in which there

was no success, due to the approximations and linearizations made when obtaining the model of the plant. For this reason, the constants used in the different controllers implemented in the system were obtained by the trial-error adjustment method, achieving good results.

In general, it can be said that all the objectives imposed at the beginning have been met, and that different knowledge about different subjects that were previously unknown was acquired. Due to the study and design process, several conclusions are presented and some lines of improvement are proposed, such as implementing a graphical user interface that provides the behavior of the system in real time.

Keywords: PID controller, PCB board, user interface or Human-Machine Interface (HMI).

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

MEMORIA

1 INTRODUCCIÓN

En este apartado se detallan, por un lado, los objetivos de este trabajo de fin de grado (TFG), y por otro, su alcance y justificación, es decir, hasta donde se ha llegado con este trabajo de fin de grado y las razones por las qué se ha realizado.

El trabajo fin de grado (TFG) consiste en el análisis, modelado, programación y control de una maqueta de control de posición orientada a la docencia.

Sobre el sistema de control de posición de la bola, se implementarán diferentes tipos de reguladores y se relacionarán los resultados obtenidos con los estudios teóricos derivados del modelado de la planta. Además, se sacarán las conclusiones oportunas de cómo afectan cada una de las acciones de los controladores implementados en la respuesta del sistema, a través de la implementación de una interfaz de usuario, para que mantenga informado al usuario en todo momento acerca del control del sistema.

Así mismo, se documentará todo el proyecto de forma que los materiales elaborados puedan servir para facilitar la comprensión por parte del alumnado de varios conceptos relativos a la regulación y el control de sistemas.

Todo el TFG se realizará siguiendo la filosofía Open Source: hardware, software y conocimiento libre.

1.1 Objetivos

Dentro de este apartado se habla de los objetivos que se han perseguido para la consecución de este trabajo fin de grado.

El objetivo principal de este trabajo de fin de grado es el de utilizar un microcontrolador para el control de una maqueta de control de posición de una bola, con el objetivo de poder acercar a la gente que no conoce la teoría de control o que se adentra por primera vez en este mundillo de la regulación como se puede controlar un sistema de manera real. En este caso, se va a utilizar la maqueta como método de docencia, es decir, para facilitar la comprensión por parte del alumnado que se adentra por primera vez en la regulación, de varios conceptos relativos a la regulación automática o control de sistemas, como pueda ser, ver visualmente como afectan cada una de las acciones del controlador a la respuesta del sistema al modificarlas, y poder visualizar de manera real la estabilidad, rapidez y error del sistema de control, es decir, como se comporta el sistema tanto en régimen transitorio como en régimen permanente.

Los diferentes objetivos que se han tratado de perseguir para la realización del proyecto son los siguientes:

- Realización de la programación software correcta mediante Arduino para obtener un control del sistema óptimo con un error de posición mínimo, para conseguir que la bola se establezca en la posición deseada de la barra ante perturbaciones externas.
- Obtención de un modelo matemático del sistema real constituido por la maqueta, sus distintos componentes y la bola.
- Realización de simulaciones del modelo matemático obtenido mediante el software Matlab Simulink, para obtener las constantes del controlador PID idóneas que realicen un control óptimo.
- Desarrollo e implementación de distintos tipos de controladores PID sobre la plataforma Arduino.
- Obtener conclusiones de cómo afectan cada una de las acciones del controlador PID en un sistema real.

- Diseño, desarrollo e implementación de una interfaz de usuario que permita al usuario interactuar con la maqueta y con los diferentes parámetros de los controladores implementados.
- Elaboración de un material claro y comprensible para facilitar el estudio de distintos aspectos referentes a la regulación automática y control de sistemas con el uso de un Arduino.

Como objetivos personales, cabe destacar el poder aprender a utilizar y a programar un microcontrolador con el objetivo de controlar un sistema y sus distintos componentes, aprender a implementar un controlador PID en la programación del sistema, además de poder aprender a diseñar desde cero una placa PCB de manera real para implementarla en la práctica.

1.2 Alcance

Teniendo en cuenta los objetivos expuestos en el anterior apartado, se puede decir que hay varias tareas que realizar para la consecución de todos los objetivos.

Como se sabe, todo sistema está formado por un actuador y un sensor, con lo cual, se han estudiado las diferentes posibilidades para seleccionar el actuador y el sensor que mejor se adapta al sistema que se estudia.

Además, se ha estudiado la programación software del microcontrolador Arduino utilizado para poder controlar de manera adecuada los diferentes componentes que conforman el sistema barra y bola.

Se ha obtenido el modelo matemático del sistema barra y bola, y se ha simulado en el software Matlab Simulink para conocer el comportamiento del sistema y ver si es necesaria la utilización de un controlador, con ella se puede ver si el sistema necesita de un lazo cerrado. También se ha realizado la simulación del sistema para intentar conseguir de manera eficaz unas constantes del controlador PID que realicen el control del sistema de manera óptima.

Se ha creado e implementado una interfaz de usuario para poder interactuar con el sistema y con los diferentes parámetros de los controladores implementados, por medio de una placa PCB diseñada utilizando un entorno software de diseño de circuitos eléctricos y electrónicos llamada KiCad.

Por último, gracias a la placa PCB diseñada, se han ido modificando los diferentes parámetros del controlador utilizado, y se han obtenido las conclusiones de cómo afectan cada una de las acciones del controlador utilizado a un sistema real.

1.3 Justificación

Este proyecto se realiza para poder exponer de manera real los conocimientos adquiridos de la regulación automática y del control de sistemas. De esta forma se decide llevar a cabo el control del conocido sistema barra y bola, el cual es un mecanismo simple diseñado específicamente para el estudio teórico y práctico de los principios básicos y avanzados de la ingeniería de control.

Como el primer contacto de una persona con un controlador suele ser de forma teórica, por medio de un libro de texto o de la explicación de un profesor, puede llegar a no entenderse de manera adecuada el concepto de controlador, por lo tanto, se comprende mucho mejor cuando se toma contacto de manera práctica. Por ello se ha realizado este TFG, el cual precisa de un sistema sencillo en el que se pueden implementar diferentes configuraciones del controlador y se pueden variar las acciones correspondientes a cada controlador mediante una interfaz, además de poder observar los resultados de manera sencilla. Por todo esto, se utiliza como sistema físico un sistema barra y bola, para el estudio, modelado, programación y control del sistema.

En definitiva, este trabajo de fin de grado se ha realizado para que los materiales elaborados puedan servir para facilitar la comprensión por parte del alumnado o de cualquier persona que se esté iniciando en el mundo del control, de varios conceptos relativos a la regulación y el control de sistemas. Normalmente en el grado se estudia la regulación y el control de sistemas de manera teórica, con lo cual, esta es una manera para que una persona que no comprenda muy bien cómo funciona esto de la regulación y control de sistemas acuda a la maqueta y vea visualmente como afectan cada una de

las acciones del controlador PID (modificando las acciones del controlador a través de la interfaz de usuario diseñada) en la respuesta del sistema, para así lograr entender mejor la regulación automática y la teoría de control de sistemas. También puede ayudar a alumnos que ya hayan cursado las materias relativas al control de sistemas, y no comprendan la teoría de control en su totalidad, para hacerse una idea de cómo funciona un controlador de manera práctica en un sistema sencillo que es como mejor se ve.

En cuanto a la motivación personal a realizar este proyecto, es la de poder desarrollar un control del sistema de manera real utilizando un Arduino, y aprendiendo a programar y controlar mediante el Arduino los distintos componentes que conforman el sistema utilizado en el TFG, además de poder aprender a implementar distintos controladores PID en un microcontrolador, para la consecución de un control optimo del sistema.

2 ANTECEDENTES

2.1 Estado del arte

En este apartado se va realizar un repaso de las técnicas existentes relacionadas con el proyecto.

Hoy en día la mecatrónica se está masificando, dado que, cada vez es más común ver a la gente interesada en los dispositivos electrónicos que ruedan por el suelo o vuelan, como es el caso del boom de los drones. Esto propicia que se despierte el interés de las personas por la robótica, la automatización y la regulación de sistemas, ya que cada vez hay más posibilidades y facilidades para el ciudadano de a pie. En estas posibilidades y facilidades para el ciudadano de a pie entran en juego los software de filosofía open source, los cuales facilitan la programación y control de los diferentes sistemas a cualquier persona, tenga o no conocimientos sobre programación o control; más adelante se hablara sobre la filosofía open source.

La mecatrónica se define como la ciencia que trata de aunar las disciplinas de la mecánica, la electrónica, las técnicas de control y la informática para ofrecer la posibilidad de crear sistemas autónomos que faciliten las labores del ser humano o que le sirvan como ocio. Este caso, se centra en la mecatrónica de un ámbito más divulgativo, la cual no incluye maquinaria pesada de las grandes fábricas, sino más bien pequeños mecanismos que realicen funciones básicas.

Entre los sistemas que tienen interés para la aplicación en cuestión destaca el sistema barra y bola, este sistema se ha posicionado desde hace tiempo como un referente en el estudio de sistemas dinámicos no lineales a nivel docente e investigador. Por este motivo existen bastantes modelos desarrollados de distintas dimensiones y características. Básicamente, las diferencias principales entre un modelo y otro, si únicamente atendemos a los elementos integrantes, radican en la forma de medir la posición de la bola y en la forma de cómo actuar sobre el sistema, lo que conlleva también un cambio de las características geométricas del mecanismo.

En cuanto a la filosofía open source como se ha dicho anteriormente, el uso y la disponibilidad de poder acceder a software con dicha filosofía, lleva a que cada vez más gente se adentre en este mundo, puesto que con este tipo de software las posibilidades y las facilidades para programar y controlar sistemas complejos y menos complejos son bastante importantes, tanto para gente que tiene conocimientos como para la que no.

El open source o software libre de código abierto son aquellos programas de software que pueden ser utilizados con total libertad por parte de los usuarios. Así, cualquier persona puede libremente usarlo, estudiarlo, redistribuirlo, comercializarlo, y, con los conocimientos informáticos adecuados, modificarlo para hacer mejoras y/o adaptaciones sobre el código fuente del programa. Es muy interesante esta filosofía porque tanto para el software como para el hardware, la gente desde su casa puede ir mejorando la plataforma y transmitiéndola a los demás usuarios, es una fuente de información y conocimiento muy grande.

El potencial del open source o código abierto es muy grande, puesto que, se centra en la premisa de que, al compartir el código, el programa resultante tiende a ser de calidad superior al software privado. El movimiento a favor del software libre va creciendo día a día en todo el mundo y cada vez son más los programas libres disponibles, y en ocasiones nos encontramos con programas de software libre que hacen las mismas cosas que el software privado.

3 MARCO TEÓRICO

3.1 Descripción general

En la figura que se muestra a continuación, pueden observarse las diferentes partes de las que constará el proyecto y de las cuales se derivarán las distintas actividades a realizar para la consecución de los objetivos planteados:

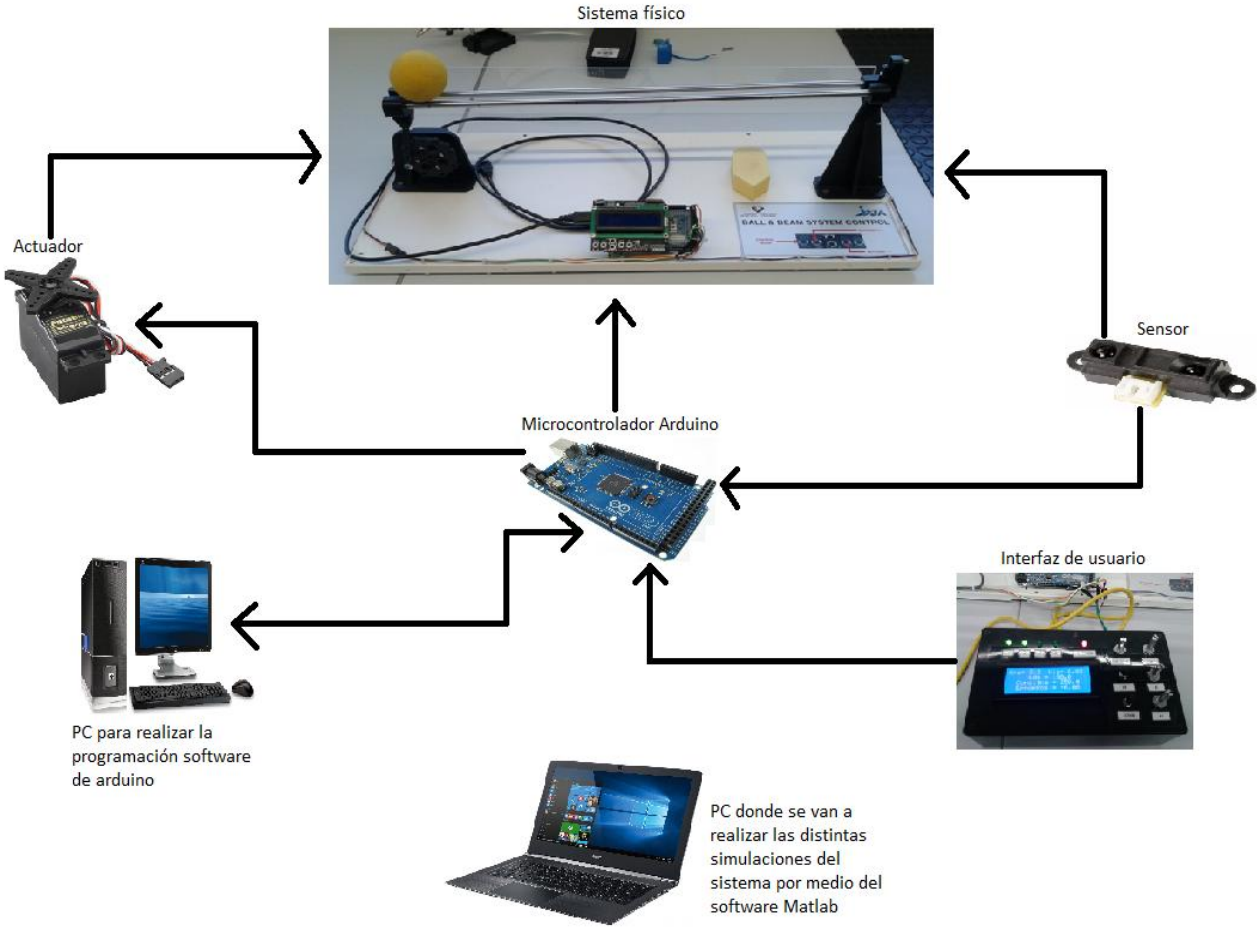


Figura 3.1 Diagrama de bloques del proyecto.

Breve descripción de cada parte del proyecto:

- Sistema físico: Se dispone de una maqueta en el laboratorio de tipo barra y bola, la cual se utiliza para el estudio de las diferentes configuraciones de los controladores y de sus diferentes parámetros. Es la planta del sistema.

- Microcontrolador: Es necesaria la utilización de un microcontrolador para poder controlar el sistema barra y bola. Va a ser el núcleo del proyecto, es decir, el dispositivo más importante del sistema, puesto que, los demás dispositivos van a estar conectados a él por medio de su plataforma hardware, y es el dispositivo que va a controlar cada una de las partes que conforman el proyecto, a través de su programación software en el PC; todo esto para conseguir que la bola se establezca en la posición deseada por el usuario ante perturbaciones externas.

- Actuador: Es el dispositivo que va a actuar sobre la maqueta cuando sea necesario, es decir, es el dispositivo que va a ser manipulado por la señal de control para reducir el error de la variable controlada.

En este caso, es el dispositivo necesario para modificar la inclinación de la barra, y así poder mover la bola de posición. Va a ser un motor que va a ser controlado en posición, y que por medio de su accionamiento va a modificar su ángulo de giro, modificando así, el ángulo de inclinación de la barra.

- Sensor: El sensor se utiliza para recibir la información necesaria del exterior para poder controlar una variable física del sistema, normalmente esta variable física medida la transforma en una variable eléctrica en forma analógica o digital a su salida, para poderla cuantificar y manipular a través de otro dispositivo como pueda ser un microcontrolador; en ocasiones es necesario un acondicionamiento de la señal de salida del sensor para adaptarla y que la pueda interpretar otro dispositivo.

En este sistema se va a utilizar un sensor para medir la posición de la bola en la barra, y así, poder conocer en todo momento en donde está situada la bola. Es un dispositivo imprescindible en el sistema, puesto que da la información necesaria para poder controlar la posición de la bola en el tiempo. En el control se utiliza para obtener la señal de error, la cual, es la posición deseada por el usuario menos la posición medida por el sensor. La señal de error es la variable más importante en los sistemas de control, porque, conociéndola se puede ir corrigiendo el error hasta obtener un error mínimo, es decir, hasta situar la bola en la posición deseada del usuario.

- Interfaz de usuario: Una interfaz de usuario, HMI (Human-Machine Interface), se define como una interfaz hombre-máquina, en la cual se utiliza un panel de control diseñado para conseguir una comunicación interactiva entre usuario y proceso/máquina, con la función de transmitir ordenes, visualizar gráficamente los resultados y obtener una situación del proceso/máquina en tiempo real.

En este sistema se va a implementar una interfaz de usuario que permita interactuar con el control de la maqueta, para así, a través de está, poder analizar y sacar conclusiones acerca de los diferentes parámetros de los controladores implementados.

- PC para realizar la programación software de Arduino: Este PC representa la realización del programa en el software de Arduino para el correcto funcionamiento del control de la maqueta. Con la implementación de la programación se van a controlar los distintos dispositivos.

- PC donde se van a realizar las distintas simulaciones: En el diagrama de bloques, el portátil representa la obtención del modelado de la planta, para su implementación en las distintas simulaciones a realizar, para la obtención de los distintos parámetros de los controladores, o por lo menos para la obtención de una primera aproximación. Estas simulaciones se van a llevar a cabo con el software Matlab Simulink. Para luego los resultados obtenidos, implementarlos y compararlos con el sistema real.

A continuación, se analizarán las distintas partes que se han desarrollado en el proyecto, con la mejor elección para cada una de las partes, para la mejor consecución del proyecto total.

3.2 Maquetas

Las maquetas tienen una gran importancia debido a que se utilizan para plasmar lo que se quiere realizar a gran escala en pequeña escala, es decir, para probarlo antes de pasar a un sistema más complejo.

Las maquetas se hacían y se siguen realizando a mano, mediante herramientas que, con el paso del tiempo, van mejorando y surgiendo nuevas tecnologías que ayudan a realizar mejores trazos, mejores acabados y mejores presentaciones, como pueda ser, las impresoras 3D.

Además de todo esto, las maquetas son importantes a la hora de probar las simulaciones realizadas (para realizar una simulación es necesario obtener el modelado de cada maqueta) en un sistema real, porque no es lo mismo realizar una simulación del sistema (las simulaciones suelen tener pequeñas desviaciones), dado que, las simulaciones no suelen ser exactas, a aplicarlo al sistema real con todos sus componentes en funcionamiento, por lo que es muy importante tener una maqueta del sistema en el que se pueda ver la respuesta real del sistema, sobre todo para el estudio de la regulación, ya que, los resultados obtenidos en la simulación como pueda ser, tener una primera aproximación de los parámetros de los controladores que se vayan a implementar, en muchas ocasiones, no son los adecuados a la hora de implementarlos en la maqueta.

Una de las partes más importantes de este proyecto es el estudio de la regulación automática, para ello se va a utilizar una maqueta, en este caso, se va a utilizar como método de docencia, para facilitar la comprensión por parte del alumnado de varios conceptos relativos a la regulación y control de sistemas, como pueda ser, como afectan cada una de las acciones del controlador PID a la respuesta del sistema al modificarlas.

A continuación, se van a exponer diferentes maquetas que existen en el mercado para el estudio de la regulación automática:

- Maqueta para el estudio de procesos de control de temperatura:

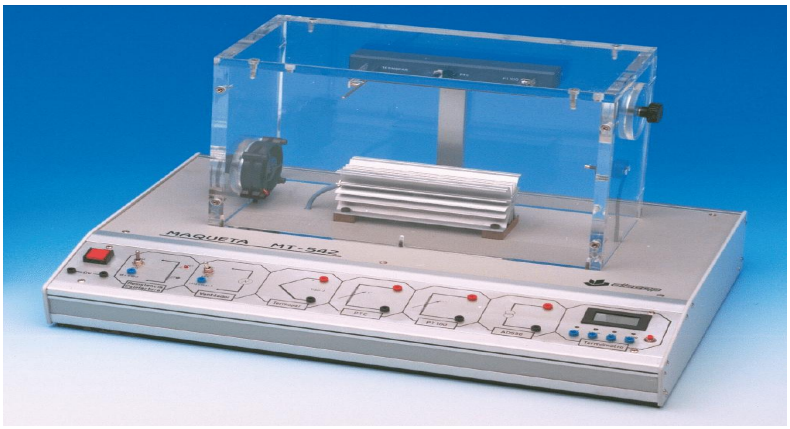


Figura 3.2 Maqueta para el estudio de procesos de control de temperatura.

- Maqueta para el estudio de procesos de control de velocidad y posición:

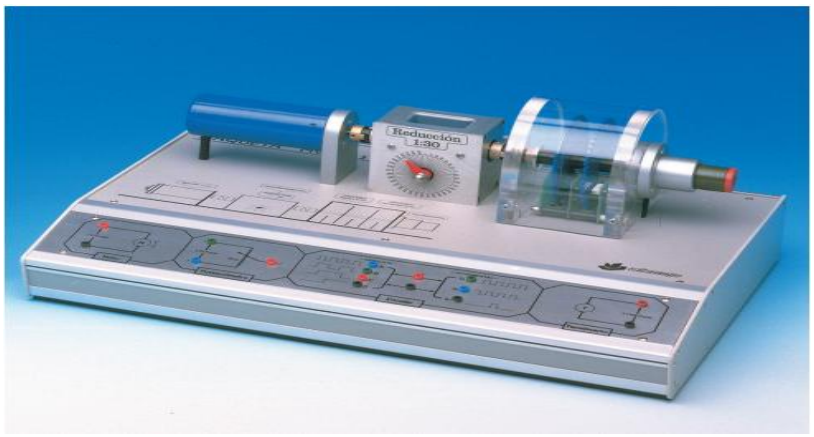


Figura 3.3 Maqueta para el estudio de procesos de control de velocidad y posición.

- Maqueta para el estudio de procesos de nivel y caudal:

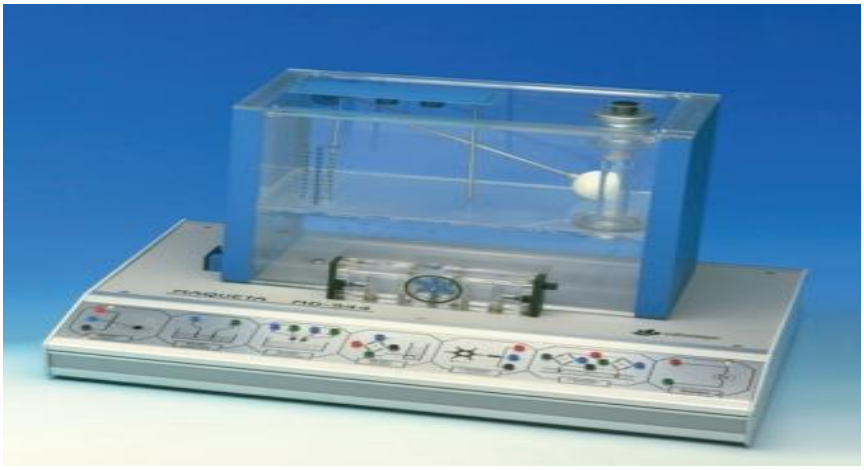


Figura 3.4 Maqueta para el estudio de procesos de nivel y caudal.

- Maqueta para el control de posición con plataforma:

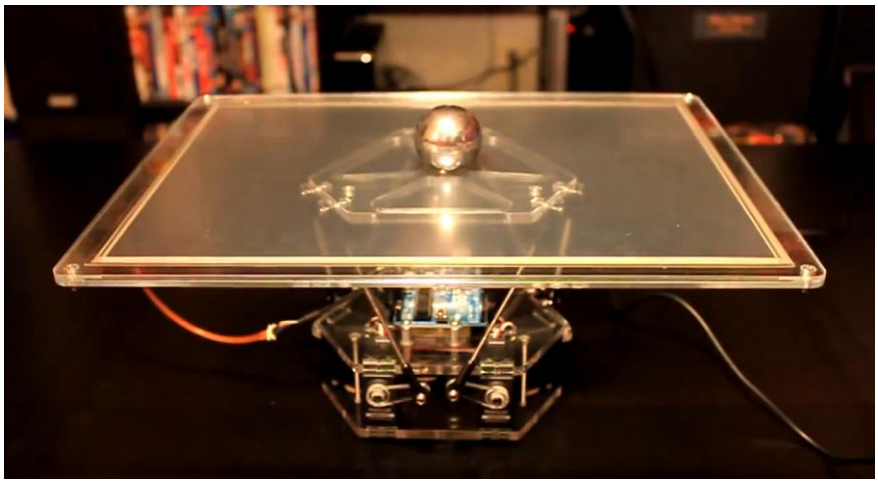


Figura 3.5 Maqueta para el control de posición con plataforma.

- Maqueta para el control de posición orientada a la docencia:

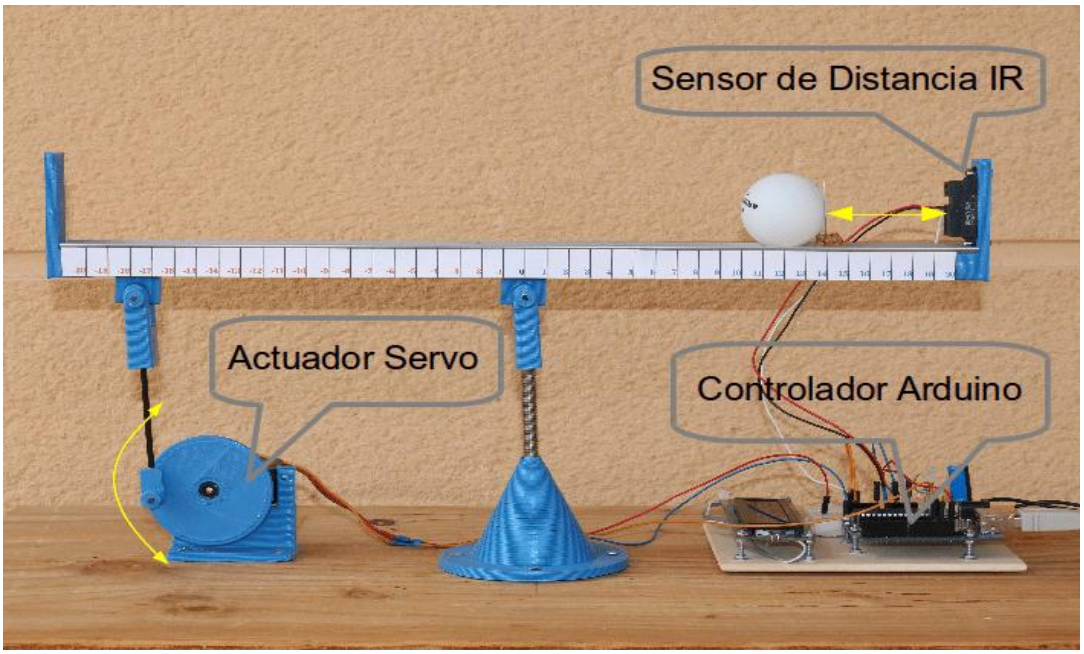


Figura 3.6 Maqueta para el control de posición orientada a la docencia.

En este caso, como en el laboratorio se dispone de un sistema barra y bola, dicha maqueta se ha utilizado para el estudio, modelado, programación y control del sistema.

3.2.1 Descripción de la maqueta utilizada

El sistema de barra y bola es un importante y clásico modelo de laboratorio para comprender la ingeniería de control y sistemas. Es muy popular porque es un sistema simple y fácil de entender que puede ser utilizado para estudiar muchos de los métodos clásicos y modernos de diseño en ingeniería de control.

Como se ha mencionado ya en el apartado 2.1 de Estado del arte, este sistema se ha posicionado desde hace tiempo como un referente en el estudio de sistemas dinámicos no lineales a nivel docente e investigador, ya que posee una propiedad muy interesante para el estudio de la regulación y control de sistemas como es, la inestabilidad en lazo abierto. Por este motivo existen bastantes modelos desarrollados de distintas dimensiones y características.

Básicamente, las diferencias principales entre un modelo y otro, si únicamente atendemos a los elementos integrantes, radican en la forma de medir la posición de la bola y en la forma de cómo actuar sobre el sistema, lo que conlleva también un cambio de las características geométricas del mecanismo.

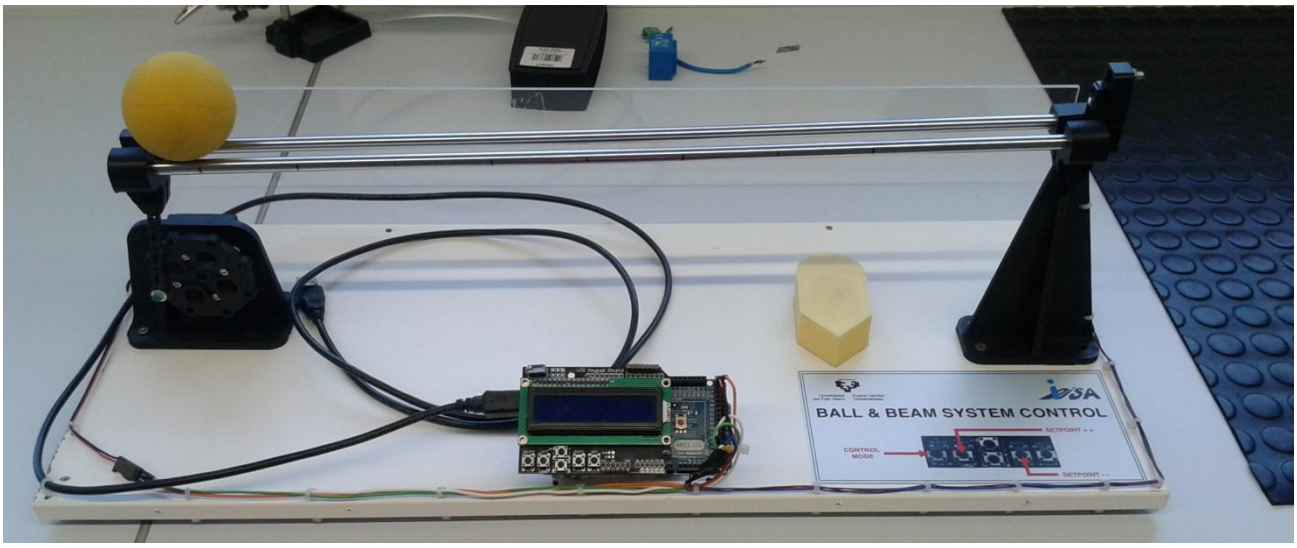


Figura 3.7 Maqueta barra y bola utilizada.

El sistema mostrado en la *figura 3.7*, es un sistema con un único grado de libertad en el que una bola rueda sin deslizar sobre una barra, gracias al giro que un motor imprime sobre la misma. Las piezas de dicha maqueta están fabricadas con una impresora 3D, la cual consta de:

- Una barra horizontal de 49,5cm de longitud que realiza la función de carril por donde puede rodar una bola.
- Dicha barra está introducida en ambos extremos en una pieza, para que la barra forme parte de la estructura.
- Un soporte vertical para el extremo derecho de la maqueta, en donde en su parte superior va situado el sensor, el cual va instalado en una cajera.
- Una bola de goma presurizada de frontenis de 55mm de diámetro que rueda a lo largo de la barra.
- Una placa en el extremo izquierdo para situar el motor, el cual va introducido en un disco o engranaje circular, en donde por acción del motor el disco gira.
- Una biela que une el disco en donde está introducido el motor, con la barra horizontal, esto se utiliza para transferir el movimiento rotatorio del disco

accionado por el motor, en un movimiento ascendente y descendente de la barra horizontal, y así poder mover la bola de posición.

- Además de lo dicho anteriormente, es imprescindible como ya se conoce el uso de un sensor para medir la posición de la bola en la barra; de un motor-actuador para poder modificar la inclinación de la barra y a su vez poder desplazar la bola por la barra; por último, el uso de un microcontrolador, el cual va a controlar dicho sistema.

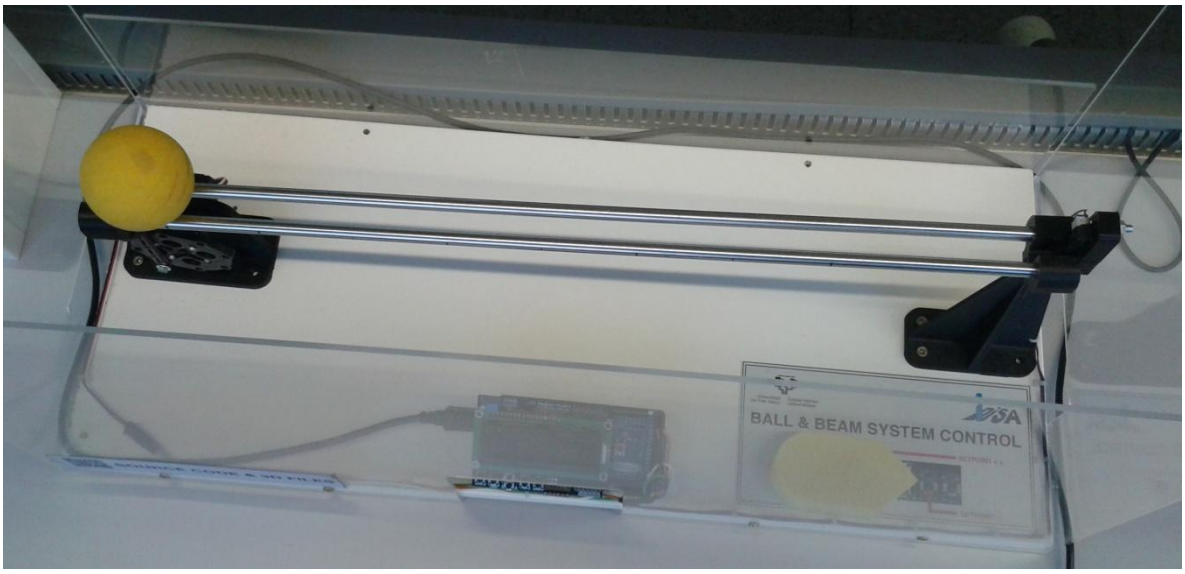


Figura 3.8 Maqueta barra y bola utilizada, vista superior.

El objetivo de control que se presenta consiste en ir modificando el ángulo de giro del actuador, para así, a su vez, modificar la inclinación de la barra y situar la bola con el menor error posible en la posición especificada de la barra de manera automática. Como se ha comentado anteriormente, es un sistema inestable en lazo abierto, dado que se puede observar que si no se utiliza un controlador, es decir, si no se realimenta la salida, al aplicar una tensión al motor la barra se inclinará y nunca volverá a su posición de equilibrio, debido a que al no aplicarse ningún tipo de controlador la señal de control no va ir corrigiéndose, como si lo hace con un sistema en lazo cerrado (con controlador). Entonces se puede decir, de que el sistema en lazo abierto tiene un comportamiento inestable ya que para una entrada acotada la salida tiende a infinito.

3.3 Actuador a utilizar

En cuanto a la elección del actuador del sistema, cabe destacar las diferentes posibilidades que existen atendiendo al tipo de motor utilizado:

a) Motor simple más encoder: Un motor de corriente continua de campo magnético permanente alimentado por una tensión variable en función de la posición de la bola y un encoder incremental que proporciona la inclinación de la barra.

Figura 3.9 Motor de corriente continúa con encoder.

b) Motor simple y sensores magnéticos: Un motor de corriente continua de campo magnético permanente alimentado por una tensión variable en función de la posición de la bola y un par de sensores de efecto Hall que proporcionan la inclinación de la barra. Dichos sensores se colocan en la base de la estructura enfrentados a sendos imanes colocados por debajo de la barra, de esta forma el sensor emite una señal eléctrica proporcional a la distancia del imán.

c) Motor paso a paso: Es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de girar una cantidad de grados dependiendo de sus entradas de control. El motor paso a paso se comporta de la misma manera que un conversor digital-analógico (D/A) y puede ser gobernado por impulsos procedentes de sistemas digitales. Este motor presenta las ventajas de tener precisión y repetitividad en cuanto al posicionamiento. Entre sus principales

aplicaciones destacan los robots, drones, radiocontrol, automatización, etc. El problema de estos motores paso a paso es que necesitan de un puente en H para ser controlados, es decir, para poder rotar en un sentido y en el otro.

Figura 3.10 Motor paso a paso.

d) Servomotor: Es un dispositivo parecido a un motor de corriente continua, con la capacidad de ubicarse en cualquier posición dentro de su rango de operación en respuesta a una señal de control, y mantenerse estable en dicha posición. Un servomotor es un motor eléctrico que puede ser controlado tanto en velocidad como en posición, capaz de controlar la inclinación de la barra por sí mismo gracias a una señal PWM.

Figura 3.11 Servomotor.

Como lo que se quiere por la naturaleza del sistema es controlar el ángulo de giro del actuador de manera precisa, para ello, la mejor opción es el servomotor, puesto que, es

el que controla el ángulo de giro de manera más idónea, precisa y sencilla. Entonces, como actuador del sistema se va utilizar un servomotor.

3.3.1 Introducción a los servomotores

Un servomotor es un dispositivo parecido a un motor de corriente continua, con la capacidad de ubicarse en cualquier posición dentro de su rango de operación en respuesta a una señal de control, y mantenerse estable en dicha posición.

Un servomotor es un motor eléctrico que puede ser controlado tanto en velocidad como en posición. Está conformado por un motor de CC, una reductora y un circuito de control.

Es un motor especial al que se ha añadido un sistema de control (tarjeta electrónica), un potenciómetro y un conjunto de engranajes. Con anterioridad los servomotores no permitían que el motor girara 360 grados, solo aproximadamente 180 grados; sin embargo, hoy en día existen servomotores en los que puede ser controlada su posición y velocidad en los 360 grados (los llamados servomotores de rotación continua). Los servomotores son comúnmente usados en robótica y en modelismo como aviones, barcos, helicópteros y trenes para controlar de manera eficaz los sistemas motores y los de dirección. Por ejemplo, en un timón de un avión o barco no se quiere un giro continuo, sino un motor al que se le pueda indicar el ángulo que deseamos en grados y que mantenga esa orientación hasta que le demos una orden contraria.

Los servomotores hacen uso de las señales PWM (modulación por ancho de pulsos) para controlar la dirección o posición de los motores de corriente continua, es decir, posiciona su eje en un ángulo preciso en función de la señal de control PWM. La mayoría de los servos funcionan con 5V y trabajan en la frecuencia de los 50Hz, así las señales de control PWM tendrán un periodo de 20ms. La electrónica dentro del servomotor responderá al ancho de la señal modulada.

3.4 Elección del sensor idóneo

3.4.1 Introducción

En este proyecto es necesaria la utilización de un sensor, puesto que es imprescindible conocer la posición de la bola en la barra para realizar el control de posición de la bola.

Un sensor es un dispositivo capaz de variar una propiedad ante magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas con un transductor en variables eléctricas. Las variables de instrumentación pueden ser, por ejemplo: intensidad lumínica, temperatura, distancia, aceleración, inclinación, presión, desplazamiento, fuerza, torsión, humedad, movimiento, etc.

Un sensor se diferencia de un transductor en que el sensor está siempre en contacto con la magnitud que la condiciona o variable de instrumentación con lo que puede decirse también que es un dispositivo que aprovecha una de sus propiedades con el fin de adaptar la señal que mide para que la pueda interpretar otro dispositivo como por ejemplo un microcontrolador. Esta función la realizan los acondicionadores de señal que pueden ser totalmente independientes del sensor o estar total o parcialmente incluidos. Un sensor no es más que un dispositivo diseñado para recibir información de una magnitud normalmente física del exterior y transformarla en otra magnitud, normalmente eléctrica codificada en forma analógica o digital, que seamos capaces de cuantificar y manipular.

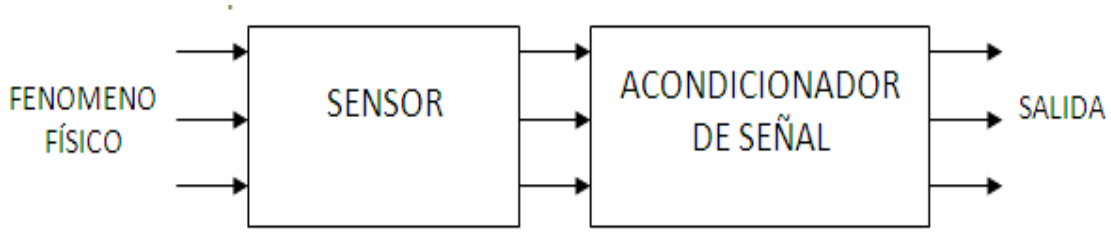


Figura 3.12 Diagrama de bloques de un sensor.

Los sensores pueden estar conectados a un ordenador para obtener ventajas como son el acceso a la toma de valores desde el sensor, una base de datos, etc.

Los sensores se suelen aplicar en: industria automotriz, robótica, industria aeroespacial, medicina, etc.

3.4.2 Tipos de sensores de medida

Ahora brevemente se van a mostrar los diferentes sensores que existen en el mercado:

- Según el principio de funcionamiento:

    1. Activos: Son aquellos que generan señales representativas de las magnitudes a medir en forma autónoma, sin requerir de fuente de alimentación. Ejemplo: sensores piezoeléctricos, fotovoltaicos, termoeléctricos, electroquímicos, magnetoeléctricos.
    2. Pasivos: Son aquellos que generan señales representativas de las magnitudes a medir por medio de una fuente auxiliar. Ejemplo: sensores de parámetros variables (de resistencia variable, de capacidad variable, de inductancia variable).

- Según el tipo de señal eléctrica que generan:

    1. Digitales: Son aquellos que frente a un estímulo pueden cambiar de estado ya sea de cero a uno o de uno a cero, en este caso no existen estados intermedios y los valores de tensión que se obtienen son únicamente dos, 5V y 0V. Devuelven una señal codificada en forma de pulsos.
    2. Analógicos: Son aquellos que, como salida, emiten una señal comprendida por un rango de valores instantáneos (0-10V o 4-20mA) que varían en el tiempo, y son proporcionales a los efectos que se están midiendo.
    3. Temporales: Son aquellos que entregan una señal variable en el tiempo, la cual, puede ser una onda sinusoidal, triangular o cuadrada.

- Según el rango de valores de salida:

    1. Todo o nada (on-off): Devuelven una señal binaria 0-1, solo poseen estos dos estados. Se consideran dentro de los digitales.

2. Sensores de medida: En estos se obtiene una salida proporcional a la señal de entrada.

- Según el nivel de integración:

    1. Discretos: Sensor en el que el circuito de acondicionamiento se realiza mediante componentes electrónicos separados e interconectados entre sí.
    2. Sensores integrados: Elemento sensor y circuito acondicionador construidos en un único circuito integrado, monolítico o hibrido.
    3. Sensores inteligentes: Suelen realizar al menos una de las siguientes funciones: cálculos numéricos, comunicación en red, autocalibración y autodiagnostico, o múltiples medidas con identificación del sensor.

- Según el tipo de variable física a detectar:

    1. Sensores de temperatura.
    2. De presión.
    3. De flujo.
    4. De desplazamiento.
    5. De nivel.
    6. De velocidad.
    7. De fuerza y par.
    8. De presencia o proximidad.
    9. De aceleración.
    10. De humedad.

Una vez visto los diferentes tipos de sensores que existen en el mercado, se puede decir que para este proyecto la utilización de un sensor de presencia o proximidad es imprescindible, debido a que es necesario un sensor para medir la distancia que hay hasta la bola, para que en todo momento se conozca la posición de la bola en la barra.

3.4.3 Sensores de presencia o proximidad

Ahora se van a exponer los diferentes sensores de presencia o proximidad que existen, para seleccionar el sensor más adecuado para el propósito que nos ocupa:

- Sensores de proximidad inductivos:

  Sirven para detectar la presencia de piezas metálicas en un rango de distancias del orden de milímetros.

  El sensor funciona con un campo magnético, es decir, en el momento que un objeto metálico entra en el campo magnético, empiezan a circular corrientes de Eddy dentro del objeto metálico, esto altera la reluctancia del circuito magnético, y atenúa el circuito oscilante haciendo variar la amplitud de oscilación. La detección de dicha amplitud permite obtener una señal de salida todo o nada (on-off).

  La principal utilización de los sensores inductivos de proximidad es como interruptores final de carrera con algunas ventajas con respecto a los electromecánicos, tales como: ausencia de contacto con el objeto a detectar, robustez mecánica, resistencia a ambientes agresivos y altas temperaturas y bajo precio. Se suelen utilizar para: detectar la ruptura de brocas, para detectar la posición de las válvulas para saber si están abiertas o cerradas, para detectar la ruptura de puntas de fresadora, etc. En definitiva, son unos sensores que se utilizan para detectar cosas muy pequeñas.

- Sensores de proximidad capacitivos:

  Son similares a los inductivos, con la principal diferencia de que los capacitivos producen un campo electrostático en lugar de un campo magnético. Esto hace que los sensores capacitivos puedan detectar objetos metálicos y no metálicos, tales como papel, vidrio, líquidos, plástico, etc.

  El funcionamiento de un sensor de proximidad capacitivo es el siguiente, cuando un objeto se aproxima a la superficie de sensado y éste entra al campo electrostático de

los electrodos, cambia la capacitancia y hace que el circuito oscilador al que está conectado comience a oscilar. Entonces hay un circuito que detecta la amplitud del oscilador y cuando llega a un punto la salida del sensor cambia. Cuando el objeto se aleja la amplitud desciende y el sensor pasa a su estado normal.

Hay que decir que cuanto mayor es la constante dieléctrica del objeto a detectar, mayor será la distancia a la que puede ser detectado el objeto y viceversa.

Una de las principales aplicaciones de estos sensores es la detección a través de recipientes. Por ejemplo, el agua tiene una constante dieléctrica mucho más alta que el plástico, esto le da la posibilidad al sensor de detectar objetos a través del recipiente. De esta forma se puede detectar el nivel de líquidos.

- Sensores de proximidad ópticos:

  Los sensores de proximidad ópticos emplean fotocélulas como elementos de detección. Estos sensores pueden detectar tanto objetos metálicos como objetos no metálicos, eso sí, solo pueden detectar material sólido, es decir, el material líquido no lo pueden detectar. Dentro de los sensores de proximidad ópticos hay varios tipos, los más destacados son:

  - Los que disponen de un cabezal que incorpora un emisor de luz y una fotocélula de detección, actuando por reflexión y detección del haz de luz, reflejado sobre el objeto que se pretende detectar.

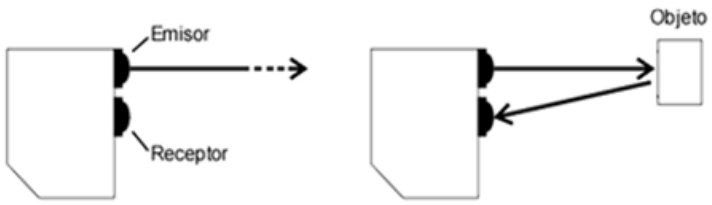


Figura 3.13 Modo de trabajo de un sensor óptico o infrarrojo.

El modo de trabajo de estos sensores es el siguiente, el emisor emite un haz de luz y cuando un objeto interfiere en su recorrido, la luz por reflexión se refleja en el objeto que va parcialmente hacia el receptor (fotocélula de detección) lo

que hace que cambie su estado. Dentro de este modo de trabajo están los denominados sensores infrarrojos.

- Los denominados como modo barrera trabajan con un emisor y un receptor separados, están diseñados para detectar mayores distancias.

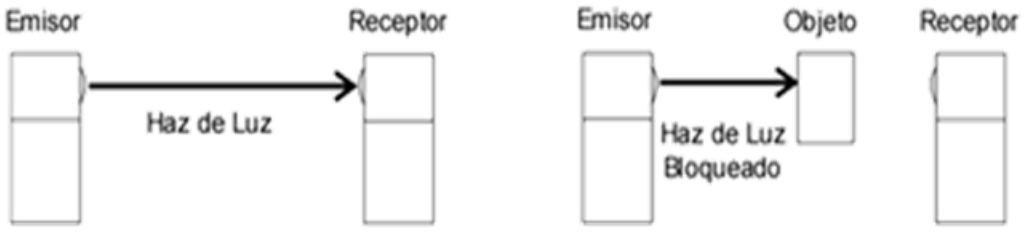


Figura 3.14 Modo de trabajo de los sensores ópticos llamados como modo barrera.

Su funcionamiento es el siguiente, el emisor produce un haz de luz que en condiciones normales llega al receptor produciéndose una especie de barrera de luz. Cuando un objeto interfiere en el haz de luz, el receptor deja de recibirlo, modificando su salida.

Ambos tipos suelen trabajar con frecuencias luminosas en la gama de los infrarrojos. Además de esto, hay que decir que los sensores de proximidad ópticos se suelen utilizar en varias aplicaciones.

- Sensores de proximidad por ultrasonidos (sensores ultrasónicos):

Los sensores por ultrasonidos o sensores ultrasónicos son detectores de proximidad que trabajan libres de roces mecánicos y que detectan objetos basándose en la emisión-recepción de señales de sonido de alta frecuencia.

Su funcionamiento es el siguiente, el sensor tiene un disco piezoeléctrico montado en su superficie, el cual produce ondas de sonido, las cuales, cuando alcanzan a un objeto reflector de sonido, producen un eco, el sensor recibe el eco producido y lo convierte en señales eléctricas. Para detectar un objeto, los sensores trabajan según el tiempo de transcurso del eco, es decir, se valora la distancia temporal entre el

impulso de emisión y el impulso del eco. En definitiva, el sensor emite un sonido y mide el tiempo que la señal tarda en regresar. Cuando el objeto entra dentro del rango de operación preestablecido la salida del interruptor cambia de estado. Cuando el objeto se sale del rango preestablecido la salida regresa a su estado original.

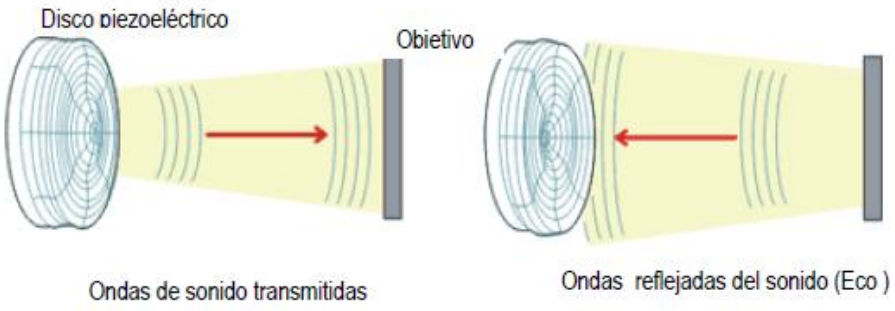


Figura 3.15 Funcionamiento del sensor ultrasónico.

Los sensores por ultrasonidos detectan objetos a distancias que van desde centímetros hasta 8m. Estos sensores trabajan solamente donde tenemos presencia de aire (no pueden trabajar en el vacío, necesitan medio de propagación), y pueden detectar objetos con diferentes formas, diferentes colores, superficies y de diferentes materiales. Los objetos a detectar pueden ser sólidos, líquidos o polvorientos, sin embargo, han de ser deflectores de sonido. Pueden detectar objetos transparentes cosa que los sensores ópticos no pueden.

Estos sensores suelen aplicarse para realizar el control de calidad de piezas, para el sensado de altura de piezas, para el sensado de fisuras, para medir el nivel de recipientes de agua, etc.

- Sensores de presencia o proximidad electromecánicos:

  Los sensores de presencia o proximidad electromecánicos más conocidos como finales de carrera, consisten en una especie de interruptor que se activa o se desactiva cuando un elemento mecánico lo acciona.

  Las principales desventajas de estos sensores son las siguientes: que tienen una respuesta lenta, que son ruidosos y voluminosos, que requieren de mantenimiento y que tienen una vida limitada.

Estos sensores se suelen utilizar en general en todas las máquinas que tengan un movimiento rectilíneo de ida y de vuelta, o que sigan una trayectoria fija, es decir, aquellas que realicen un recorrido fijo, como pueda ser en ascensores, robots, en puertas mecánicas, etc.

A continuación se muestra un pequeño resumen de lo visto hasta ahora. La siguiente tabla ayuda a seleccionar el sensor de presencia o proximidad más adecuado para el propósito que se quiera:

MATERIAL		DISTANCIA DETECCIÓN	TIPO SENSOR
SÓLIDO	METÁLICO	<50mm	*Inductivo*
		>50mm	*Ultrasónico u óptico*
	NO METÁLICO	<50mm	*Capacitivo*
		>50mm	*Ultrasónico u óptico*
LÍQUIDO	TRANSPARENTE	<50mm	*Capacitivo*
		>50mm	*Ultrasónico*
	OPACO	<50mm	*Capacitivo*
		>50mm	*Óptico*
POLVO O GRANULADO	METÁLICO	<50mm	*Inductivo*
		>50mm	*Ultrasónico*
	NO METÁLICO	<50mm	*Capacitivo*
		>50mm	*Ultrasónico*

Figura 3.16 Tabla de los sensores de presencia o proximidad.

En este caso, como lo que se va a medir es la posición de una bola, el sensor a utilizar tiene que poder detectar objetos sólidos no metálicos. Además, como se sabe que en el rango de medida que nos vamos a mover va estar en torno a 0 - 500mm tiene que poder medir distancias mayores a 50mm. Con lo cual, viendo la tabla anteriormente expuesta se puede decir que los sensores que mejor se adaptan a este sistema barra y bola son los sensores infrarrojos y los sensores ultrasónicos.

3.4.4 Sensor infrarrojo vs Sensor ultrasónico

Según se ha visto en el anterior apartado, los sensores de presencia o proximidad más adecuados para medir la distancia de una bola en el sistema barra y bola son: los sensores infrarrojos y los sensores ultrasónicos.

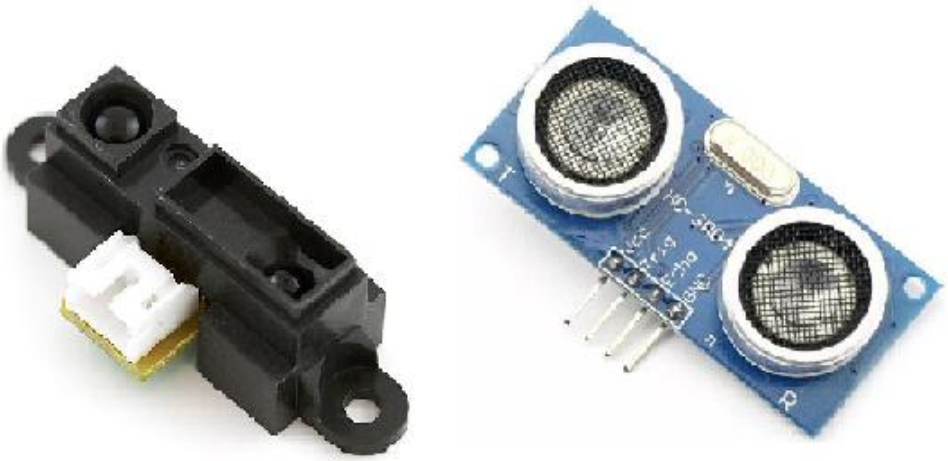


Figura 3.17 Sensor infrarrojo vs sensor ultrasónico.

Se barajaron estas dos ideas, pero viendo las características de uno y de otro, se eligió utilizar un sensor infrarrojo, debido a varios motivos que se explican a continuación, además de los vistos en el apartado anterior.

El sensor infrarrojo tiene una desventaja respecto al ultrasónico, que es que no puede detectar objetos transparentes, pero como en el sistema se va a medir la distancia de una bola amarilla, eso no afecta al sistema a controlar. También se ha elegido porque tiene una mayor precisión, una velocidad de respuesta más rápida, y por el rango de medición en el que nos vamos a mover en la barra horizontal (donde va estar la bola), que es más adecuado a un sensor infrarrojo (rango de medida de 10 a 80cm aproximadamente), puesto que, el sensor ultrasónico se suele utilizar en rangos de medida de hasta 8m.

Como el sensor tiene que detectar una bola que va por el carril de una barra horizontal se puede utilizar el sensor infrarrojo, puesto que, requieren de una línea de visión directa entre el sensor y el objeto a medir y en este caso se da, sin embargo, los sensores ultrasónicos cubren todo el espacio y no necesitan una línea de visión directa.

Como resultado de ello, pueden detectar personas detrás de obstáculos, y objetos que estén fuera del carril de la barra horizontal, con lo que puede llevar a errores de medición. También son más sensibles a los movimientos de menor importancia, tales como movimientos de la mano.

Los sensores infrarrojos son muy adecuados para espacios cerrados. Los sensores ultrasónicos, por su parte, son adecuados para espacios en los que una línea de visión no es posible, como espacios compartimentados, y en los espacios que requieren un mayor nivel de sensibilidad.

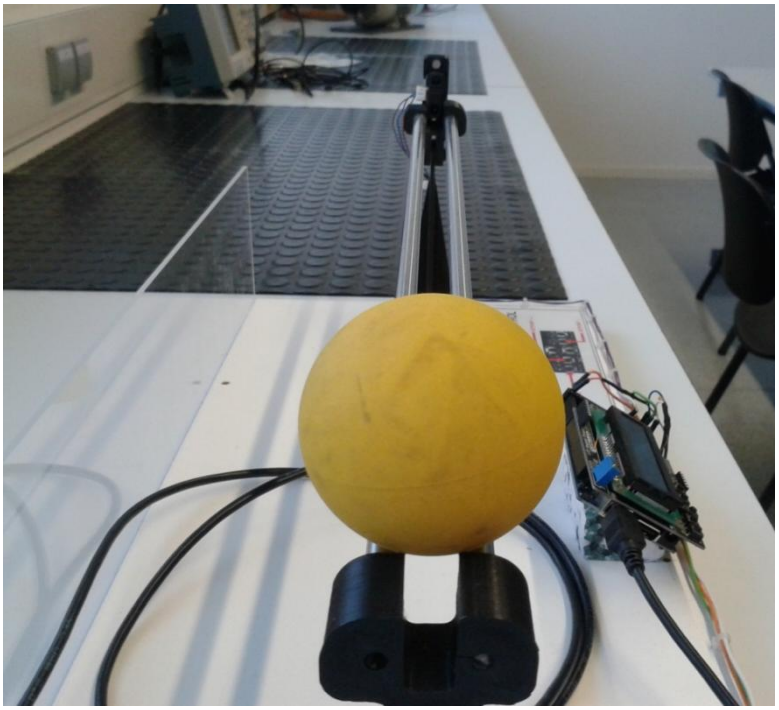


Figura 3.18 Visión directa entre el sensor infrarrojo y la bola.

En definitiva, el sensor infrarrojo es el sensor seleccionado para realizar la tarea de medir la distancia que hay hasta la bola, para que en todo momento se conozca la posición de la bola en la barra horizontal, esto es necesario, para controlar la posición de la bola en todo momento. Hay que decir como se ha dicho anteriormente, que la bola se va a mover por el carril de una barra horizontal, con lo cual, habrá una línea de visión directa entre el sensor infrarrojo y la bola, por lo que, el sensor infrarrojo es muy adecuado.

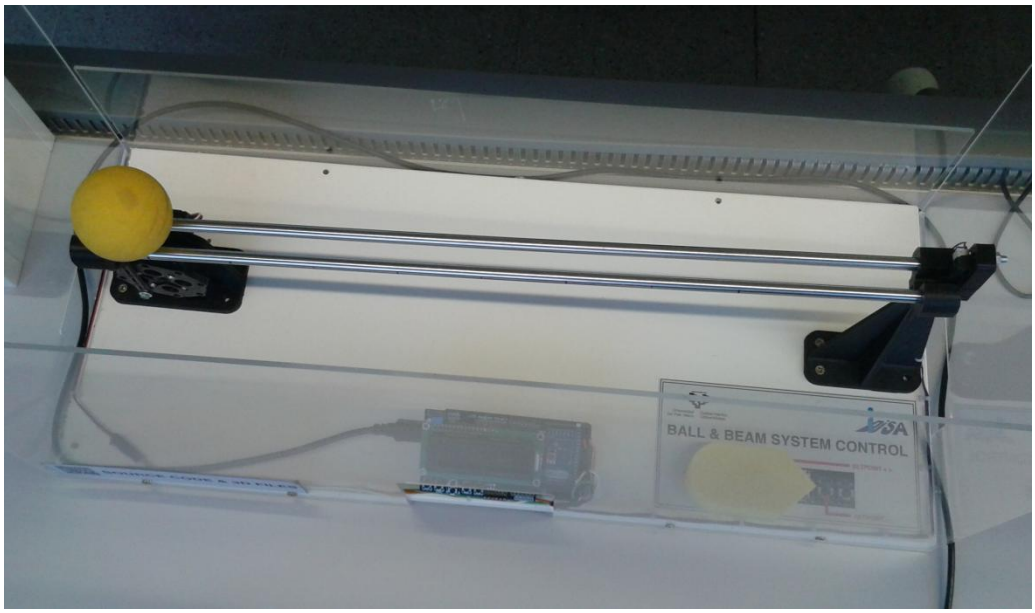


Figura 3.19 Maqueta sistema barra y bola.

3.5 Microcontrolador

En este proyecto es necesaria la utilización de un microcontrolador para poder controlar el sistema barra y bola (maqueta); para ello hay infinidades de posibilidades entre las cuales se ha elegido la plataforma Arduino.

Uno de los principales motivos por los que se ha seleccionado la plataforma Arduino como microcontrolador es su filosofía de código abierto (open source) basada en hardware y software flexibles y fáciles de usar. Además de esto, también se ha elegido porque es una de las plataformas microcontroladoras más baratas que existen en el mercado. Este tipo de microcontroladores hoy en día se están utilizando en infinidades de aplicaciones debido a que son fáciles de usar.

Dentro de la plataforma Arduino hay diferentes placas Arduino con diferentes características, cada una preparada para ser utilizada en diferentes condiciones, como pueda ser: la placa Arduino UNO, Arduino MEGA, Arduino NANO, etc.

En este proyecto se barajó la idea de utilizar la placa Arduino UNO, pero como para realizar el control del sistema era necesario que el micro tuviese 2 timers internos

de 16 bits, y el Arduino UNO solo consta de 3 timers uno de los cuales solo es de 16 bits, por eso la plataforma Arduino elegida para realizar el control del sistema ha sido el Arduino MEGA ADK, dado que además de tener 6 timers, 4 de ellos de 16 bits, tiene más pines de entrada y salida que el Arduino UNO, y pensando en la mejor consecución del proyecto y en posibles ampliaciones futuras, se cree que es la mejor opción. El Arduino MEGA ADK con respecto al UNO es prácticamente igual con las diferencias anteriormente mencionadas, es decir, mayor número de timers internos, con lo cual más pines de salida PWM disponibles y más pines tanto digitales como analógicos.

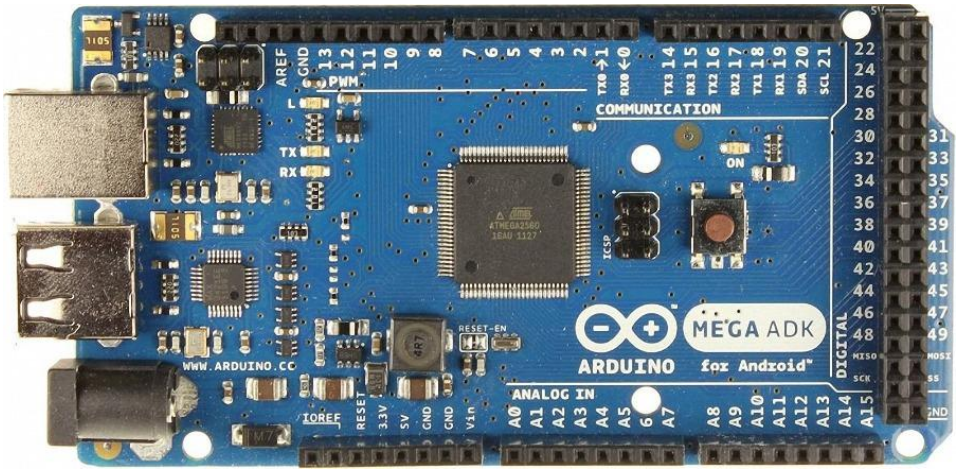


Figura 3.20 Placa Arduino MEGA ADK utilizada.

Placa Arduino Mega ADK

En este apartado se van a mostrar las características fundamentales de la placa utilizada para el control del sistema.

El Arduino MEGA ADK es una placa basada en el microcontrolador ATmega2560. Tiene una interfaz de host USB para conectarse con teléfonos basados en Android, basados en el MAX3421e IC.

Cuenta con 54 pines de entrada/salida digitales (de los cuales 15 se pueden usar como salidas PWM), 16 entradas analógicas, 4 UART (puertos serie de hardware), un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, un cabezal ICSP, y un botón de reinicio. Está placa está diseñada para proyectos más

complejos, debido a su tamaño más alargado, es la placa recomendada para proyectos basados en impresoras 3D y para proyectos de robótica.

Las especificaciones técnicas de la placa Arduino MEGA ADK son las siguientes:

Tabla 3.1 Especificaciones técnicas del Arduino MEGA ADK.

Microcontrolador	**ATmega2560**
Tensión de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Pines de E / S digitales	54 (de los cuales 15 proporcionan salida PWM)
Pines de entrada analógica	16
Corriente DC por Pin de E / S	40 mA
Corriente DC para 3.3V Pin	50 mA
Memoria flash	256 KB de los cuales 8 KB utilizados por el gestor de arranque
SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16 MHz
Chip de host USB	MAX3421E
Longitud	101.52 mm
Anchura	53.3 mm
Peso	36 g

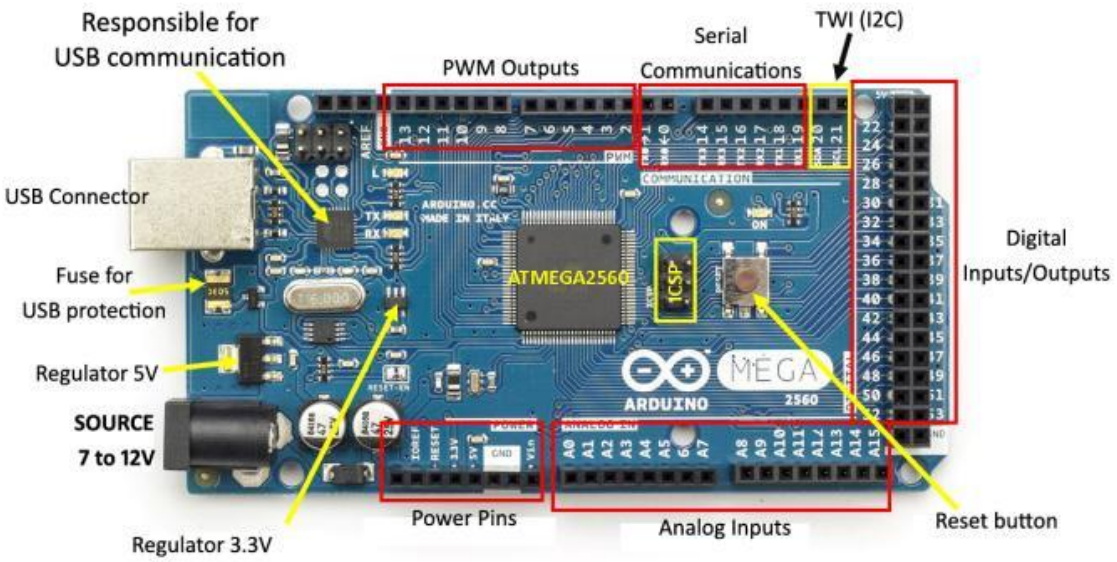


Figura 3.21 Diagrama de las diferentes características hardware del Arduino MEGA ADK.

El Arduino recibe los 5V necesarios para su funcionamiento a través de su conexión USB con el PC, además de esto utiliza la conexión USB con el PC para cargar en el microcontrolador Arduino la programación realizada.

En cuanto a la programación, Arduino proporciona un entorno de programación sencillo y potente para programar, además incluye las herramientas necesarias para compilar el programa y cargar el programa compilado, por medio del cable USB anteriormente mencionado en la memoria flash del microcontrolador. Además, el IDE de Arduino ofrece un sistema de gestión de librerías y placas muy práctico.

El lenguaje de programación de Arduino es C++, aunque no es un C++ puro, sino que es una adaptación proveniente de avr-libc que provee de una librería de C de alta calidad para usar con compiladores de C y C++ en los microcontroladores AVR de Atmel.

Una diferencia del lenguaje de programación de Arduino frente al lenguaje C++ standard es la estructura del programa.

Un programa Arduino se denomina sketch o proyecto y tiene la extensión .ino. La estructura básica de un sketch de Arduino es bastante simple y se compone de al menos tres partes, las cuales son: comentarios introductorios que describen el programa,

llamadas a librerías y declaración de variables; la parte de *void setup()*; y por último la parte del *void loop()*.

Figura 3.22 Estructura básica de un programa de Arduino.

La parte de *void setup()* es la parte encargada de recoger la configuración, además en esta parte se declara si un pin digital es de entrada o salida y es donde se inicializan las interfaces de la comunicación serie y de la pantalla LCD entre otras, también es donde se establece la configuración de los registros de los timers. En cuanto a la parte de *void loop()* es la que contiene el programa que se ejecuta de manera cíclica. Estas dos funciones son imprescindibles para que el programa trabaje de manera adecuada.

Dicho de otra manera, programar una placa Arduino consiste en traducir a líneas de código las tareas automatizadas que se quieren hacer leyendo los sensores y en función de las condiciones del entorno programar la interacción con el mundo exterior mediante actuadores.

Respecto a la programación cabe destacar lo siguiente:

- Las entradas analógicas disponen de 1024 puntos (0-1023 puntos), puesto que el convertidor analógico digital de Arduino proporciona 10 bits.

- Arduino no es capaz de proporcionar una autentica salida analógica, con lo cual, para salvar esta limitación y simular una salida analógica se emplea un truco, que consiste en activar una salida digital durante un tiempo y mantenerla apagada durante el resto. El promedio de la tensión de salida, a lo largo del tiempo, será igual al valor analógico deseado. Para realizar esto se utiliza la modulación de ancho de pulso denominada PWM. La proporción de tiempo que está activada la señal, respecto al periodo total, se denomina duty cycle, y generalmente se expresa en tanto por ciento.

  Una señal PWM es suficiente para emular una señal analógica en muchas aplicaciones. Las señales PWM son comúnmente usadas para el control de velocidad de motores de corriente continua (si decrementas el duty cycle sobre la señal de control del circuito de potencia que actúa sobre el motor, quiere decir que se transmite menos tensión con lo cual, el motor se mueve más lentamente, reduce la velocidad; y si se incrementa el duty cycle lo contrario, aumenta la velocidad), para ajustar la intensidad de brillo de un LED, etc.

  En la programación de Arduino para las señales PWM solo se utilizan 8 bits, con lo que solo se dispone de 256 puntos (0-255 puntos). Hay que decir que las señales PWM en la programación no se suelen declarar en el *void setup()*, ni como entrada ni como salida, son señales digitales, pero a la hora de la programación como simulan una salida analógica se programan como si fuesen analógicas, con *analogWrite(pin, 255)*.

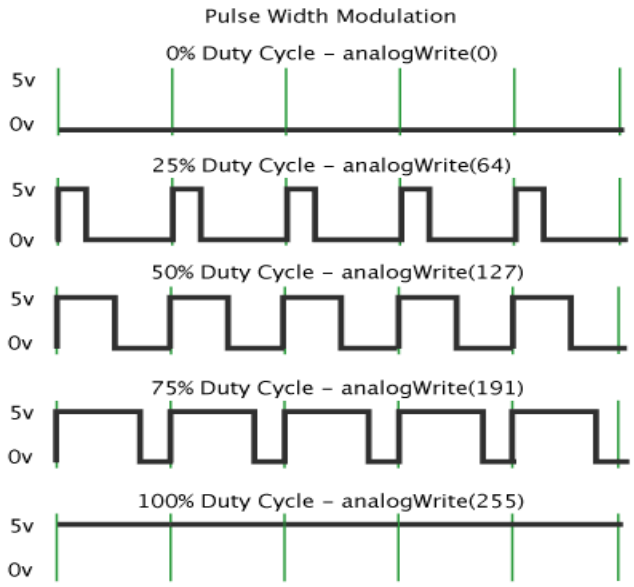


Figura 3.23 Señales PWM en Arduino.

Arduino implementa por hardware salidas PWM en varios de sus pines, que aparecen identificados en la placa con el símbolo "~". En el Arduino MEGA ADK se dispone de 15 salidas PWM de 8 bits en los pines de 2 a 13 y 44 a 46.

- La placa Arduino MEGA ADK dispone de 6 timers, de los cuales, 4 son de 16 bits: el timer 1, 3, 4 y 5 son de 16 bits, y el timer 0 y 2 son de 8 bits.

En conclusión, la placa Arduino MEGA ADK va a ser el núcleo del proyecto, es decir, el componente más importante del sistema, puesto que, los demás dispositivos van a estar conectados a ella por medio de su plataforma hardware, y es la que va a controlarlos, a través de su programación software en el PC; todo esto para conseguir que la bola se establezca en la posición deseada de la barra ante perturbaciones externas.

3.5.1 Como se va a controlar el servomotor

Como se ha visto en el apartado *3.3 Actuador a utilizar*, el actuador que se va a utilizar para poder controlar el ángulo de giro del actuador, y a su vez, poder controlar la inclinación de la barra para situar la bola en la posición deseada, va a ser un servomotor.

Hay varias formas para programar un servo en la plataforma Arduino, como ya se sabe, para controlar el ángulo de giro de un servo, es necesario controlarlo con señales PWM variando el duty cycle de la señal, es decir, variando la tensión que se le aplica al servomotor, por medio de la señal de control. Las 3 formas que hay para programar un servomotor en la plataforma Arduino son las siguientes:

- Con la librería especifica del servomotor *(#include <Servo.h>)*:

  "Servo" es una librería estándar de Arduino, que viene incluida en su IDE. Se utilizan las funciones especificas de la librería "servo" para programar y controlar el servomotor, para ello, se utiliza la función *servo.write(angulo),* la cual, permite ajustar la posición del servo con un valor entre 0 y 180 que sería el rango de movilidad del servo de 0º a 180º. Ésta instrucción que es la utilizada habitualmente en la librería "servo" solo proporciona 180 puntos distintos, es decir, una resolución muy pequeña, con lo cual, la precisión del ángulo de giro del servo será muy mala.

  En cambio, si se utiliza la función *servo.writeMicroseconds(µS)* se obtiene la máxima precisión en el giro del servo con esta librería, siendo µS los microsegundos de duración del pulso a nivel alto en un periodo de trabajo del servo de 20ms establecido por fábrica. En este caso, se dispondrá de varios puntos dependiendo del rango de trabajo del servo, en este caso de 1800 puntos distintos, puesto que el rango de trabajo del servo está entre 500µS (0º) y 2300µS (180º), con lo cual, se obtiene una resolución aceptable.

  En definitiva, con la instrucción *servo.writeMicroseconds(µS)* se obtienen 1800 puntos distintos en lugar de los 180 puntos que se obtienen utilizando la instrucción más básica *servo.write(ángulo).*

- Con la función *analogWrite()*:

  Es la función por defecto para las señales PWM.

  Las señales PWM utilizan solo 8 bits, con lo que el valor máximo, es decir, lo que sería un duty cycle del 100% es de 2^8-1 = 255, con lo cual, un valor entre 0 y 255. Hay que decir que las señales PWM no es necesario declararlas, es decir, que no es necesario ni declararlas como entrada ni como salida, son señales digitales, pero a la hora de la programación se programan como una señal analógica *analogWrite(pin, 255).*

Este caso es la peor opción de todas debido a que utiliza 8 bits, es decir, dispondrá de solo 255 puntos distintos muy inferior a los 1800 puntos que se obtienen con la librería del servo, con lo cual, con este caso la resolución es muy pequeña, debido a esto, la precisión es bastante mala y por ello es la opción de programación del servo menos recomendada.

- Por último, configurando los registros de uno de los timers del microcontrolador de Arduino; en este caso, se utiliza un Arduino MEGA ADK que está basado en el microcontrolador ATmega2560. Se va a configurar los registros de uno de los timers que forman el microcontrolador ATmega2560 para generar una señal PWM que controle el ángulo de giro del servomotor. Con ella, se va a ir modificando el duty cycle de la señal dependiendo de las necesidades de control, es decir, si se necesita un ángulo de giro superior el duty cycle de la señal aumentará, y si necesita un ángulo de giro inferior el duty cycle de la señal disminuirá.

  Esta es la forma más adecuada para controlar el ángulo de giro del servomotor, puesto que es la que dispone de una mayor resolución (junto con la de la librería, utilizando la función *servo.writeMicroseconds(µS)*), con lo cual, es la que tiene una mayor precisión, debido a esto hay menos posibilidades de error en el control. Por todo esto, es la forma que se ha utilizado para controlar el ángulo de giro del servomotor, como se verá en el apartado *4.3.1 Control del servomotor*.

Hay que decir que las 2 primeras formas son con la configuración por defecto, es decir, sin tocar ningún registro, y la última forma que es la que se va a utilizar en la programación del proyecto, se realiza configurando los registros de uno de los timers del microcontrolador Arduino.

3.5.2 Como se va a controlar el sensor y métodos de linealización

La programación en el software Arduino del sensor infrarrojo para realizar la lectura y visualizarla, se realiza exactamente igual que para otras entradas analógicas. Cabe destacar, la aplicación de un filtro paso bajo por software en la programación para despreciar los valores incoherentes de medida del sensor, realizando la media de unos

cuantos valores de medida del sensor, con esto, aparte de despreciar los valores incoherentes de medida, también se consigue mejorar la precisión de medida del sensor.

Además de esto, se programa para que la relación entre la lectura del sensor y la distancia medida sea lineal, es decir, se va a realizar una curva de linealización del sensor en la programación con diferentes valores de lectura y distancia, midiendo la lectura del sensor a diferentes distancias de la bola, con ello, se van a ensayar diferentes técnicas de linealización del sensor, las cuales van a ser: linealización del sensor tramo a tramo utilizando la función *map* de la programación, y la linealización del sensor por el método de regresión lineal. Una vez se ensayen ambas técnicas, se elegirá para la programación del sistema, la que mejor se aproxime a los valores reales de medida; esto se verá en el apartado *4.4.2 Control y diferentes métodos de linealización del sensor*.

3.5.3 Controlador PID

Un controlador PID es un mecanismo de control por realimentación que calcula la desviación o error existente entre el valor medido y el que se quiere obtener (valor deseado, consigna), para realizar una corrección que ajuste el proceso.

El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error acumulado, esto asegura que aplicando un esfuerzo de control suficiente, el error en estado estacionario del régimen permanente se reduce a cero, con lo cual, se consigue un sistema preciso. El Derivativo considera la tendencia del error y permite una repercusión rápida de la variable después de presentarse una perturbación en el proceso. La suma de estas tres acciones se usa para ajustar el proceso. Ajustando estas tres variables en el algoritmo de control del PID, el controlador puede proveer un control diseñado para las especificaciones técnicas que requiera el proceso a realizar. El uso del PID para control no garantiza un control óptimo del sistema o la estabilidad del mismo.

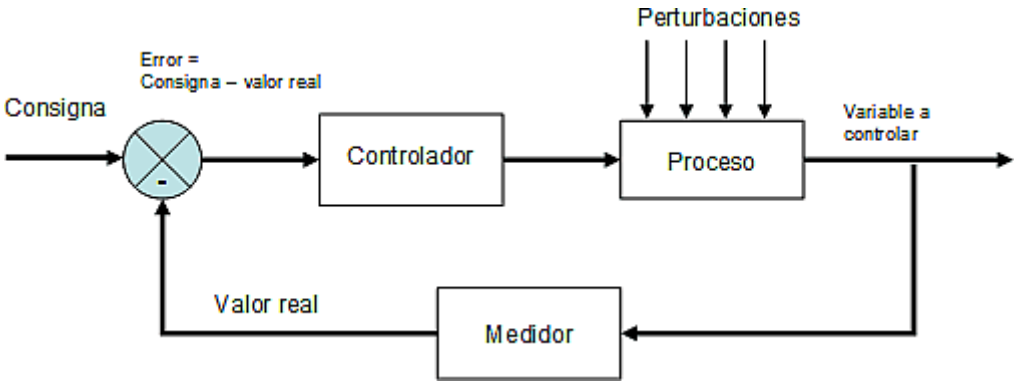


Figura 3.24 Diagrama de bloques típico de un sistema de control.

Dicho de otra manera, teóricamente cuanta más acción proporcional se ponga, más mejora la rapidez del sistema, pero más disminuye la estabilidad. La acción integral elimina el error en estado estacionario, es decir, es una acción de control que actúa muy bien en el régimen permanente, pero cuanta más acción integral se le ponga al sistema, más aumenta la precisión del sistema, pero más disminuye la estabilidad del mismo, por el efecto del polo en el origen. Cuanta más acción derivativa se ponga más mejora el rebose del sistema, es decir, aumenta el amortiguamiento del sistema, con lo cual, mejora la estabilidad del sistema, pero lo hace más lento; esta acción va actuar bien en el régimen transitorio.

Dicho esto, la base para obtener un control óptimo y estable es jugar con las diferentes acciones del controlador (K_P, K_I y K_D), y ver que combinación de las 3 proporcionan el control óptimo y la estabilidad correcta para lo que se quiere, cumpliendo unas especificaciones técnicas establecidas en el sistema anteriormente.

Algunas aplicaciones pueden solo requerir de uno o dos modos de los que provee este sistema de control. Dependiendo del proceso se ajustará mejor un regulador que tenga las tres acciones (PID), solo dos (PI o PD) o incluso solo una (P). Un controlador PID puede ser llamado también P, PI o PD en la ausencia de las acciones de control respectivas.

Si se analizan detenidamente las características principales de cada uno de los tipos de control existentes se puede llegar a la conclusión de que, dependiendo del tipo

de control que se desee realizar es conveniente el empleo de un tipo concreto de controlador.

Control proporcional, P:

En este tipo de control la salida generalmente sube hasta el valor final, produciendo un rebose y un error en estado estacionario.

- Una señal de error grande produce una señal de control grande.
- No corrige una señal de error no nula en el permanente.
- Para valores grandes pueden provocar salidas oscilatorias, que pueden llegar a inestabilizar el sistema.
- Puede controlar cualquier planta estable pero posee un desempeño limitado.
- Disminuye el amortiguamiento del sistema, por lo que aumentará su rapidez.

Control proporcional derivativo, PD:

Este tipo de control se utiliza para eliminar el rebose y mejorar la estabilidad del sistema, para ello es necesario un mecanismo que prediga que va a producirse ese rebose de forma que pueda atajar ese rebose a tiempo. Esto se consigue añadiendo a la señal de control un término proporcional a la derivada del error.

- La acción derivativa no es capaz de corregir el error en estado estacionario, disminuye el error en el permanente, pero no consigue eliminarlo completamente.
- La acción derivativa es un tipo de control de anticipación, actúa en función de la derivada del error, es decir, de si el error está subiendo o bajando.
- Mejora el amortiguamiento del sistema por lo que reduce el rebose, es decir, mejora la estabilidad del sistema.
- Amplifica las señales de alta frecuencia, como el ruido, por lo que no será adecuado en sistemas que presenten mucho ruido. Es uno de los motivos por el que no se suele utilizar el término derivativo en varios sistemas.
- En general, valores elevados de la acción derivativa hacen el sistema más lento.

Control proporcional integral, PI:

Este tipo de control se utiliza para eliminar el error en estado estacionario, es decir, para realizar un control preciso.

- Consigue eliminar completamente el error en estado estacionario del permanente, gracias a la acción integral. Mientras se mantenga el error la señal de control se modifica en el sentido que lo disminuya.
- En general el término integral disminuye la estabilidad del sistema. Los polos en s=0 de la función de transferencia en lazo abierto hacen que el sistema en lazo cerrado sea menos estable.
- En los instantes iniciales predomina la acción proporcional, y a medida que transcurre el tiempo y persiste el error, la acción integral va aumentando.
- Filtra el ruido de alta frecuencia debido al término integral, por lo que será adecuado para sistemas que presentan mucho ruido.

Control PID

El control proporcional integral derivativo, en general va a eliminar el error en estado estacionario, al contener un término integral, proporcionando una respuesta más estable debido a la acción derivativa.

- La presencia del integrador puro en este tipo de controladores hace que el sistema sea más inestable. Inestabilidad que se puede compensar en parte con la acción derivativa.
- Consiguen eliminar el error en régimen permanente, gracias a la acción integral.
- Aumento considerable del amortiguamiento del sistema por lo que limita ampliamente el rebose, gracias a la acción derivativa.
- En líneas generales, disminuye el tiempo de subida y el tiempo de establecimiento del sistema.
- Filtran con bastante efectividad el ruido de alta frecuencia.

Una vez visto esto, se puede decir que para realizar el control de posición de una bola es necesaria la utilización de todas las acciones, es decir, se va a utilizar un controlador PID por las siguientes razones:

- Es necesaria la utilización del término proporcional para controlar la posición de la bola, teniendo en cuenta la señal de error, puesto que es la diferencia entre el valor de consigna y la posición medida de la bola mediante el sensor.
- Es necesaria la utilización del término derivativo para controlar la velocidad de la bola, es decir, para reaccionar lo antes posible ante cambios del error de posición, esto es, para que el sistema sea lo más estable posible.
- Por último, es necesaria la utilización del término integral para eliminar el error en estado estacionario, puesto que, una de las características fundamentales que tiene que cumplir este sistema es la precisión, es decir, si el valor de la consigna es de 250mm tiene que posicionar la bola a 250mm, el error tiene que ser mínimo. El término integral considera la posición de la bola y cuanto tiempo lleva allí, puesto que actúa como un acumulativo del error y cuando el error acumulado es notable la acción integral actúa, con lo cual, cuanto más tiempo pase alejado del valor de la referencia más irá incrementándose el error integral hasta que reduzca completamente el error.

En cuanto a la implementación de un controlador PID en Arduino, cabe destacar que existen varias formas para ello. Se puede implementar utilizando librerías como pueda ser la librería "PID" de Arduino, o como se ha implementado en este proyecto aplicando la fórmula clásica de los controladores PID en ecuaciones en diferencias (tiempo discreto (z)), puesto que, los procesadores digitales implementan los algoritmos de control de esta forma, más adelante se detallará en profundidad.

Teniendo en cuenta la gestión de los tiempos de muestreo, la mejor forma para implementar un controlador PID en Arduino es mediante la utilización de interrupciones, debido a que con las interrupciones puedes generar una interrupción cada "x" microsegundos en la programación, para realizar la función de un periodo de muestreo.

Una desventaja de utilizar las interrupciones para gestionar los tiempos de muestreo, es que solo se puede aplicar un controlador PID, es decir, para casos en los que haya que utilizar más de un controlador PID utilizar interrupciones para gestionar los tiempos de muestreo no es adecuado. En este caso, solo se va a utilizar un controlador PID, con lo cual, se van a generar interrupciones cada "x" microsegundos

para realizar la función de un periodo de muestreo y poder calcular tanto la salida del PID, como la posición de la bola cada cierto periodo de tiempo.

3.5.4 Timers en Arduino

3.5.4.1 Introducción.

Un timer o timer/counter es una pieza hardware integrada en el controlador Arduino. Es como un reloj y se puede utilizar para medir diferentes eventos de tiempo. Los timers pueden ser programados por algunos registros especiales. Con estos registros se puede programar el prescaler del reloj del micro (normalmente el reloj del micro de Arduino es de 16MHz, entonces con el prescaler seria 16MHz/prescaler), o puede configurarse el modo de generación de forma de onda, el modo de comparación de salida y muchas otras cosas.

Los timers están asociados a los pines PWM de cada Arduino. Las funciones PWM por hardware, emplean los timer para generar la onda de salida. Cada timer da servicio a 2 o 3 salidas/pines PWM. Para ello dispone de un registro de comparación por cada salida, también llamado canal de comparación. Cuando el tiempo (registro contador TCNTx) alcanza el valor del registro de comparación de salida, es decir, cuando TCNTx = OCRxy, la salida modifica su valor dependiendo del modo de comparación de salida configurado.

Cada salida conectada a un mismo timer comparte la misma frecuencia, aunque pueden tener distintos duty cycles, dependiendo del valor de su registro de comparación de salida OCRxy.

Dependiendo del microcontrolador utilizado el Arduino tiene más o menos timers disponibles:

- El microcontrolador ATmega328P o el ATmega328 dispone de 3 timers, solo uno de ellos es de 16 bits (Timer 1), todos los demás son de 8 bits:

  Es el caso de Arduino Uno, Mini y Nano:

- El Timer 0 controla las salidas/pines PWM 5 y 6. (8 bits)
- El Timer 1 controla las salidas/pines PWM 9 y 10. (16 bits)
- El Timer 2 controla las salidas/pines PWM 3 y 11. (8 bits)

La diferencia más importante entre el timer de 8 y 16 bits es la resolución del timer, es decir, 8 bits significan 256 valores donde 16 bits significa 65536 valores para una resolución más alta o un conteo más largo. En este proyecto se utilizan diferentes timers de 16 bits, puesto que es lo mejor, porque de la otra manera te puedes quedar corto y la resolución es más pequeña lo que lleva a una mala precisión.

- El microcontrolador ATmega2560, el cual utiliza el Arduino MEGA dispone de 6 timers, los tres primeros son exactamente iguales a los del Arduino UNO, y los 3 siguientes son todos de 16 bits, es decir, tiene 4 timers de 16 bits (hay que decir que todos los timers tienen 3 pines asociados, excepto el timer 0 y el timer 2 que tienen solo 2 pines, además de ser los únicos que son de 8 bits):
  - El Timer 0 controla las salidas/pines PWM 4 y 13. (8 bits)
  - El Timer 1 controla las salidas/pines PWM 11, 12 y 13. (16 bits)
  - El Timer 2 controla las salidas/pines PWM 9 y 10. (8 bits)
  - El Timer 3 controla las salidas/pines PWM 2, 3 y 5. (16 bits)
  - El Timer 4 controla las salidas/pines PWM 6, 7 y 8. (16 bits)
  - El Timer 5 controla las salidas/pines PWM 44, 45 y 46. (16 bits)

En este proyecto se utiliza un Arduino MEGA ADK, el cual utiliza un microcontrolador ATmega2560, se muestra en esta figura su esquema de pines, se puede ver como los pines de comparación de salida de los timers OCxy (por ejemplo, OC1A), están asociados a las salidas/pines anteriormente mencionadas:

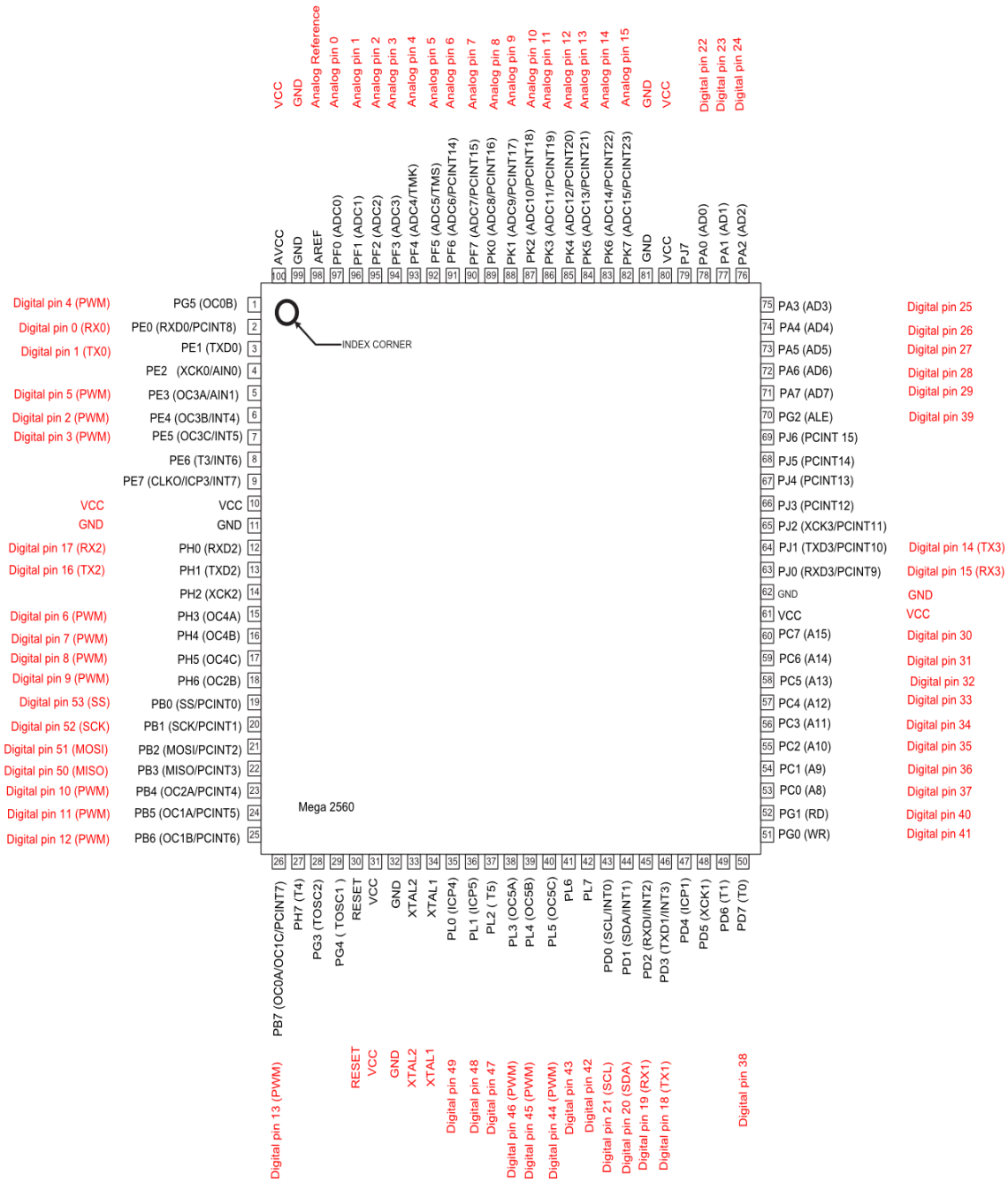


Fig.1. Esquema de pines del microcontrolador ATmega2560 del Arduino MEGA.

3.5.4.2 Explicación de los registros de los timers.

Como se ha dicho en el apartado anterior, el hardware del timer se puede configurar con algunos registros especiales, los cuales se van a explicar a continuación.

Para configurar los diferentes registros de los timers se va al datasheet del microcontrolador del Arduino utilizado, en este caso del Arduino MEGA ADK que es el ATmega2560, y ahí se puede ver cómo se deben configurar los diferentes registros de los timers para lo que se quiera generar.

Antes de pasar a explicar los registros especiales de los timers, hay que decir que cuando los registros se escriben de manera general hay que saber que: la "x" minúscula significa el número del timer, y la "y" minúscula significa el canal de la comparación de salida (A, B o C).

Ahora se van a exponer y a explicar los registros de configuración del timer más importantes:

- TCCRxy: Registro de control del timer/counter. En estos registros se puede configurar el modo de comparación de salida mediante los bits COMxy1:0 ubicados en el registro de control A del timer (TCCRxA); se puede configurar el modo de generación de forma de onda mediante los bits WGMx3:0 ubicados en los registros de control A y B del timer (TCCRxA y TCCRxB); y por último se puede configurar el prescaler del reloj del micro mediante los bits CSx2:0 ubicados en el registro de control del timer B (TCCRxB).

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3.25 Registros de control de los timers/counters, el TCCRxA y el TCCRxB respectivamente.

- TCNTx: Registro timer/counter. Es el registro contador del timer, donde se almacena el valor actual del timer, el timer actualiza automáticamente este registro.

- OCRxy: Registro de comparación de salida. Este registro se compara todo el tiempo con el registro contador del timer TCNTx, y cuando se igualan dependiendo del modo de operación establecido por los bits que configuran el

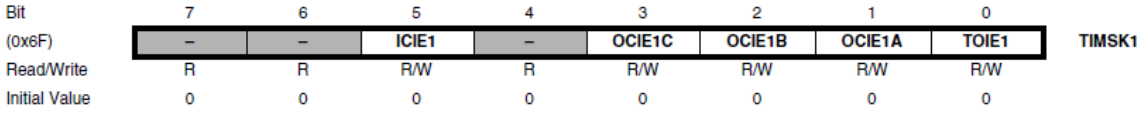
modo de generación de forma de onda WGMx3:0 y por los bits que configuran el modo de comparación de salida COMxy1:0, en la señal de salida pasa una cosa u otra, es decir, el resultado de igualdad puede ser utilizado por la configuración del generador de forma de onda para generar una señal PWM o una salida de frecuencia variable en el pin de comparación de salida (OCxy).

- ICRx: Registro de captura de entrada (solo para timers de 16 bits). El registro de captura de entrada ICRx solo se puede escribir cuando se utiliza un modo de generación de forma de onda que lo utiliza para establecer un valor fijo TOP del contador (valor máximo que puede alcanzar el timer, es decir, valor máximo que puede alcanzar el registro contador del timer TCNTx).

Los registros TCNTx, OCRxy y ICRx son registros de 16 bits por que el timer utilizado es de 16 bits, es decir, el valor máximo que se le puede asignar a estos registros es el de 2^{16}-1, es decir 65535. En cambio, si el timer utilizado es de 8 bits, los registros TCNTx, OCRxy y ICRx son de 8 bits.

- TOP: Valor máximo que puede alcanzar el timer (Según el modo de generación de forma de onda seleccionado). El registro contador TCNTx alcanza el valor más alto en la secuencia de conteo cuando se iguala al valor definido en TOP, se puede decir que el valor establecido en TOP es la asignación del periodo de la señal de salida. El valor TOP o máximo valor que puede alcanzar el timer/counter, puede definirse en algunos modos de operación por el registro de comparación de salida OCRxA, por el registro de captura de entrada ICRx o por un conjunto de valores fijos. La asignación depende del modo de generación de forma de onda seleccionado, con lo cual, la secuencia de conteo es determinada por la configuración del modo de generación de forma de onda seleccionado, mediante los bits WGMx3:0 ubicados en los registros de control A y B del timer (TCCRxA y TCCRxB).

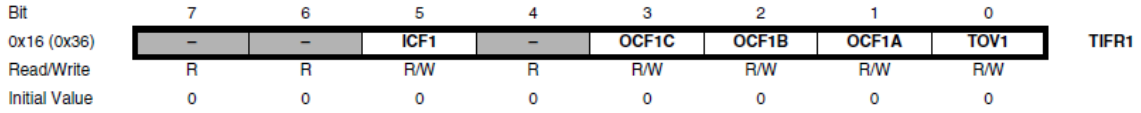
- TIMSKx: Registro de máscara de interrupción del timer/counter. Registro que hay que configurar para poder generar interrupciones. Sirve para habilitar/deshabilitar los diferentes tipos de interrupciones del timer.



Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3.26 Registro de máscara de interrupción del timer/counter, TIMSKx.

- TIFRx: Registro de flag de interrupción del timer/counter. Indica una interrupción del timer pendiente.



Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3.27 Registro de flag de interrupción del timer/counter, TIFRx.

- TOVx: Es el indicador (flag) de desbordamiento del timer/counter, se establece de acuerdo con el modo de generación de forma de onda seleccionado por los bits WGMx3:0. TOVx se puede usar para generar una interrupción de CPU.

Ahora se van a explicar los bits más representativos de los registros anteriores de manera más extensa, con estos bits se configura el timer de un modo u otro, esto dependiendo de qué modo de comparación de salida (COMxy1:0), de qué modo de generación de forma de onda (WGMx3:0) y de que prescaler del reloj del micro (CSx2:0) se seleccione o se configure:

Bits de selección de modo de generación de forma de onda WGMx3:0:

Con estos bits se selecciona: el modo de operación (lo más importante); como se va definir el valor TOP, es decir, si se va a definir con el registro OCRxy o con el registro ICRx o con valores predefinidos; en qué momento se va a actualizar el registro de comparación de salida OCRxy; además de cuándo se va a establecer a 1 el flag de desbordamiento del timer TOVx.

Los diferentes modos de operación se pueden dividir en dos bloques, los que generan una señal PWM y los que no generan una señal PWM:

Table 17-2. Waveform Generation Mode Bit Description[1]

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Figura 3.28 Modos de generación de forma de onda.

Dentro de los modos de operación que no generan una señal PWM están: el modo normal y el modo CTC (modo que borra el timer cuando hay una comparación igualada).

- Modo normal:

  Es el modo de operación más simple (WGMx3: 0 = 0). En este modo, el registro contador TCNTx siempre realiza un conteo ascendente, y no se realiza el borrado del contador, es decir, el contador cuenta hasta el límite de conteo de un timer de 16 bits que es 65535. En el momento que el contador sobrepasa su valor máximo (MAX = 0xFFFF (65535)), el contador se reinicia (BOTTOM = 0x0000) iniciando de nuevo la secuencia de conteo.

- Modo CTC (modo que borra el timer cuando hay una comparación igualada):

En "Clear Timer on Compare Match mode" o en el modo CTC, el registro de comparación de salida OCRxA o el registro de captura de entrada ICRx son utilizados para manipular la resolución del contador, es decir, para definir el valor TOP, el valor máximo que puede alcanzar el registro contador TCNTx.

En el modo CTC, el registro contador del timer TCNTx aumenta hasta que coincide con el registro OCRxA (WGMx3:0 = 4) o con el registro ICRx (WGMx3:0 = 12) momento en el cual salta a la rutina de interrupción y el contador se borra y se establece a cero e inicia de nuevo la secuencia de conteo, con esto, se puede generar una interrupción cada "x" microsegundos.

Este modo permite un mayor control de la frecuencia de salida de la comparación igualada.

El modo CTC es el modo de operación más adecuado para generar interrupciones, cuando se utiliza hay que configurar el registro de máscara de interrupción TIMSKx, para habilitar el tipo de interrupción que se vaya a utilizar.

Por otro lado, los modos de operación que generan una señal PWM son: el modo "Fast PWM", el modo "PWM phase correct" y el modo "PWM phase and frequency correct".

Dentro de estos modos de operación hay dos tipos:

- El de operación de pendiente única:
  - Modo Fast PWM:

    El modo fast PWM (WGMx3:0 = 5, 6, 7, 14 o 15) proporciona una opción de generación de forma de onda PWM de alta frecuencia. El fast PWM difiere de las otras opciones PWM por su operación de pendiente única.

    Este modo funciona de la siguiente forma, el contador cuenta desde BOTTOM (valor 0 del contador) hasta TOP y luego se reinicia empezando de nuevo la secuencia de conteo. En el modo de comparación de salida no inversora (COMxy1:0 = 2), el pin de comparación de salida (OCxy), esto es, la señal de salida del pin se pone a nivel bajo cuando el registro contador TCNTx iguala al registro OCRxy, y se pone a nivel alto cuando el contador

está en 0 (en BOTTOM). Al invertir el modo de comparación salida, se realiza justo al revés como se puede ver en el diagrama de tiempos.

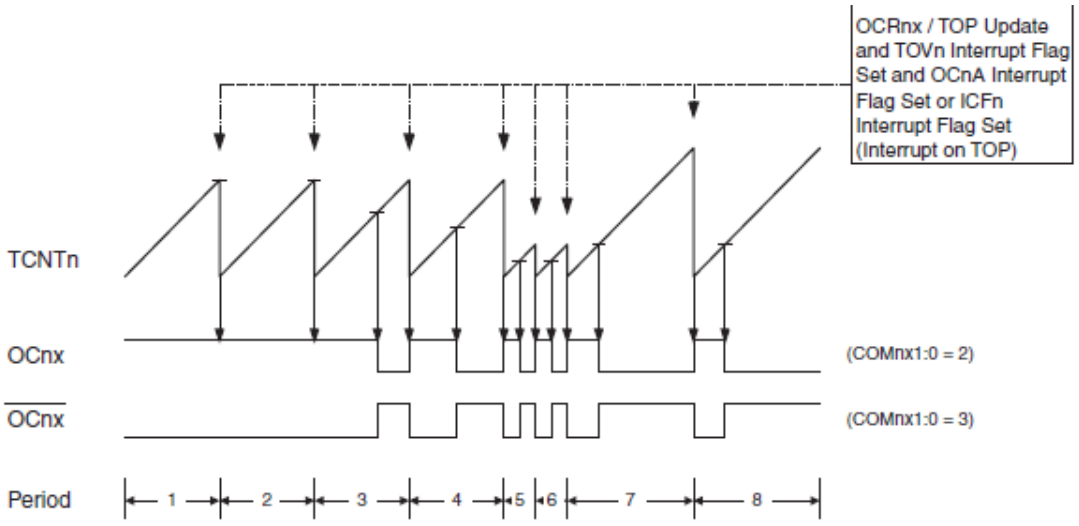


Figura 3.29 Diagrama de tiempos del modo fast PWM.

La figura muestra el funcionamiento del modo fast PWM cuando se utiliza OCRxA o ICRx para definir TOP. El valor del registro TCNTx está en el diagrama de tiempos mostrado como un histograma para ilustrar la operación de pendiente única. Las pequeñas marcas de línea horizontal en las pendientes TCNTx representan la comparación de igualdad entre OCRxy y TCNTx.

En este diagrama de tiempos se puede ver tanto su operación de pendiente única, como el funcionamiento de los diferentes modos de comparación de salida.

La diferencia de este modo (Fast PWM) respecto a los demás modos PWM es que, por su operación de pendiente única su frecuencia puede ser dos veces más alta que los demás modos PWM que utilizan la operación de doble pendiente. Debido a que el modo fast PWM es de alta frecuencia, es adecuado para aplicaciones de regulación de potencia, rectificación y DAC. La alta frecuencia permite componentes externos de pequeño tamaño (bobinas, condensadores), por lo tanto, reduce el coste total del sistema.

- Los de operación de doble pendiente:

- Modo PWM Phase Correct:

  El modo PWM phase correct (WGMx3:0 = 1, 2, 3, 10 u 11) proporciona una opción de generación de forma de onda de alta resolución. La resolución de la PWM a generar para el modo PWM phase correct se puede fijar en 8 bit, 9 bit o 10bit, o puede estar definida mediante los registros OCRxA o ICRx.

  El modo PWM phase correct es al igual que el modo PWM phase and frequency correct, basado en una operación de doble pendiente. Esta operación de doble pendiente funciona de la siguiente manera: el contador cuenta repetidamente desde BOTTOM (0x0000) hasta TOP y luego desde TOP hasta BOTTOM. En el modo de comparación de salida no invertido (COMxy1:0 = 2), el pin de comparación de salida o de señal de salida OCxy se borra, seestablece a nivel bajo cuando hay una igualdad entre TCNTx y OCRxy mientras el contador realiza el conteo ascendente, y se establece a 1 cuando la igualdad ocurre mientras el contador realiza el conteo descendente. En el modo de comparación de salida invertido (COMxy1:0 = 3), la operación se invierte como se puede ver en el diagrama de tiempos.

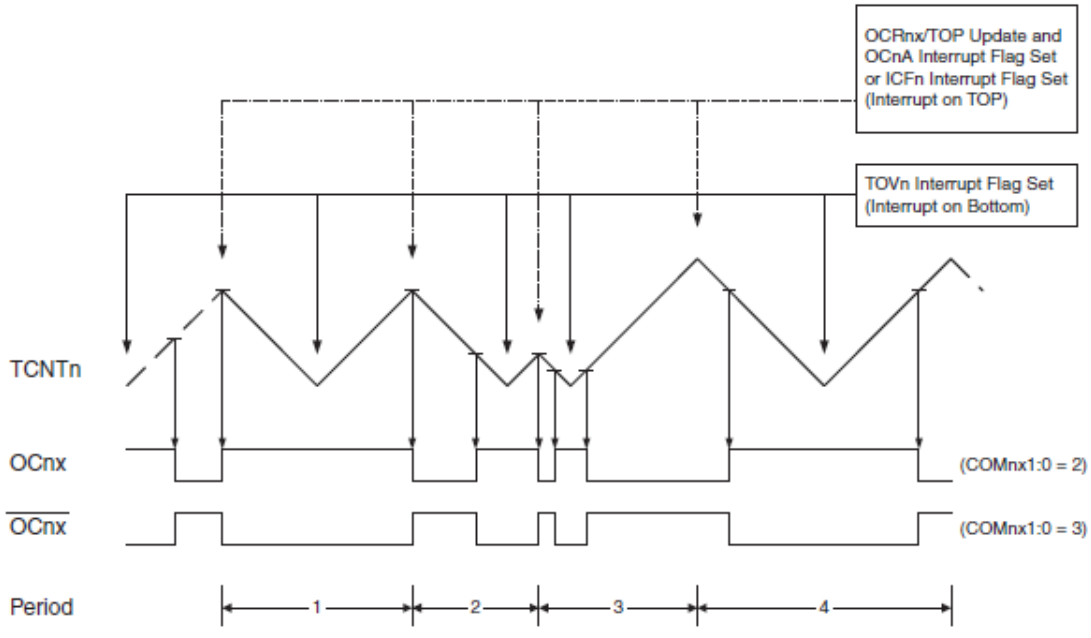


Figura 3.30 Diagrama de tiempos del modo PWM phase correct.

La figura muestra el funcionamiento del modo PWM phase correct cuando se utiliza OCRxA o ICRx para definir TOP. El valor TCNTx está en el

diagrama de tiempos mostrado como un histograma para ilustrar la operación de doble pendiente. Las pequeñas marcas de línea horizontal en las pendientes TCNTx representan la igualdad entre OCRxy y TCNTx.

En este diagrama de tiempos se puede ver la operación de doble pendiente, además de ver cómo funcionan los diferentes modos de comparación de salida. También como se ve en el diagrama, la actualización del registro OCRxy y del valor de TOP se produce en el valor máximo de conteo TOP, con lo cual, el periodo de la señal PWM comienza y finaliza en TOP. Debido a esto y a su naturaleza de funcionamiento, en este modo de operación se pueden dar salidas asimétricas, como se puede ver en el diagrama de tiempos.

La operación de doble pendiente proporciona una frecuencia de operación máxima más baja en comparación con la operación de pendiente única. Sin embargo, debido a la característica simétrica de los modos PWM de doble pendiente, estos modos son preferidos para aplicaciones de control de motores. Dicho de otra forma, este modo de operación PWM phase correct se suele utilizar en aplicaciones de control de motores, que es el caso del proyecto debido a que se necesita seleccionar un modo de operación para controlar el ángulo de giro del servomotor.

Una desventaja de este modo con respecto al modo PWM phase and frequency correct, es que se recomienda utilizar el modo PWM phase and frequency correct en lugar del modo PWM phase correct cuando se cambia el valor TOP mientras el timer está funcionando, esto es debido a que por su naturaleza de funcionamiento en este modo se pueden dar salidas asimétricas, en contraste con el modo de operación PWM phase and frequency correct, en el cual siempre se dan salidas simétricas. Cuando se utiliza un valor fijo de TOP prácticamente no hay diferencias entre los dos modos de operación.

- Modo PWM Phase and Frequency Correct:

El modo PWM phase and frequency correct (WGMx3:0 = 8 o 9) proporciona una opción de generación de forma de onda PWM de fase y frecuencia correctas de alta resolución. El modo PWM phase and frequency correct es, al igual que el modo PWM phase correct, basado en una operación de doble pendiente.

Este modo de operación es prácticamente igual al modo PWM phase correct con las siguientes diferencias:

La diferencia principal entre el modo PWM phase correct y el modo phase and frequency correct es el momento en que se actualiza el registro de comparación de salida OCRxy y el valor TOP. En el modo PWM phase and frequency correct el registro OCRxy se actualiza en BOTTOM, es decir, cuando el contador está a 0, con lo cual, el periodo de la señal PWM comienza y finaliza en BOTTOM; y en el modo PWM phase correct se actualiza en TOP, con lo cual, el periodo de la señal PWM comienza y finaliza en TOP, como se puede ver en los distintos diagramas de tiempo. Debido a esto, como se ha dicho antes, se recomienda utilizar el modo PWM phase and frequency correct en lugar del modo PWM phase correct cuando se cambia el valor TOP mientras el timer está funcionando. Esto es debido a que en el modo phase correct se pueden dar salidas asimétricas y en el modo phase and frequency correct siempre se dan salidas simétricas.

Cuando se utiliza un valor fijo de TOP prácticamente no hay diferencias entre los dos modos de operación.

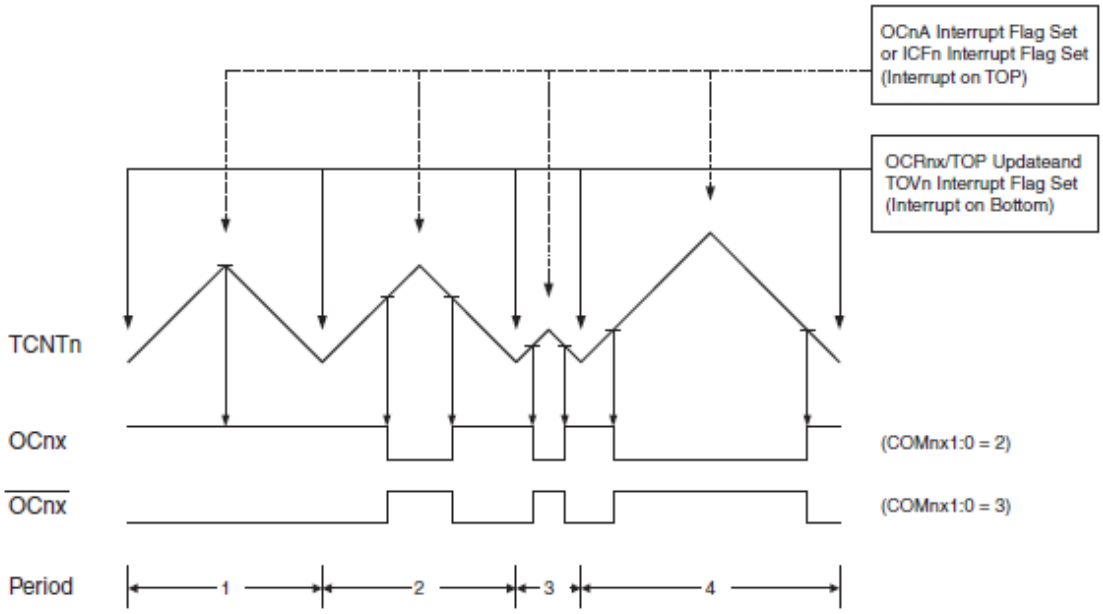


Figura 3.31 Diagrama de tiempos del modo PWM phase and frequency correct.

La figura muestra el funcionamiento del modo PWM phase and frequency correct cuando se utiliza OCRxA o ICRx para definir el valor TOP. El valor TCNTx está en el diagrama de tiempos mostrado como un histograma para ilustrar la operación de doble pendiente. Las pequeñas marcas de línea horizontal en las pendientes TCNTx representan la comparación igualada entre OCRxy y TCNTx.

Como se ve en el diagrama de tiempos, la salida generada es, en contraste con el modo PWM phase correct, simétrica en todos los periodos. Como los registros de comparación de salida OCRxy y el valor de TOP se actualizan en BOTTOM, es decir, cuando el valor del contador es 0, la longitud de las pendientes ascendente y descendente siempre será igual. Esto da pulsos de salida simétricos y, por lo tanto, es de frecuencia correcta.

También se puede ver como el periodo de la señal PWM comienza y finaliza en BOTTOM, debido a que actualiza el registro de comparación de salida OCRxy y el valor de TOP en BOTTOM; además en este diagrama de tiempos también se puede ver el comportamiento de los diferentes modos de comparación de salida, tanto el de la salida no invertida como el de la salida invertida.

Como se ha dicho en el apartado anterior, la operación de doble pendiente proporciona una frecuencia de operación máxima más baja en comparación con la operación de pendiente única. Sin embargo, debido a la característica simétrica de los modos PWM de doble pendiente, estos modos son preferidos para aplicaciones de control de motores.

Bits de selección de modo de comparación de salida COMxy1:0:

Estos bits que configuran el modo de comparación de salida (COMxy1:0) se utilizan para establecer la polaridad de la señal de salida en los pines de comparación de salida OCxy, es decir, dependiendo de qué modo de comparación de salida se seleccione la señal de salida empezará a nivel alto y luego cuando se produzca la igualdad (TCNTx = OCRxy) bajará a nivel bajo o viceversa, o realizará un toggle (toggle es cambiar el valor de salida, es decir, si está a 0 ponerlo a 1 y si está a 1 ponerlo a 0).

El generador de forma de onda utiliza los bits que configuran el modo de comparación de salida COMxy1:0 de manera diferente en los modos normal, CTC y PWM.

Para comparar las acciones de salida en los modos “non-PWM”, “Fast PWM”, y “Phase Correct and Phase and Frequency Correct PWM” mirar la figura que viene a continuación, es decir, son las que se utilizan para configurar el modo de comparación de salida mediante los bits COMxy1:0 según el modo de operación utilizado.

Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Table 17-4 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode.

Table 17-4. Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. See "Fast PWM Mode" on page 146. for more details.

Table 17-5 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct and frequency correct PWM mode.

Table 17-5. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 =9 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting Set OCnA/OCnB/OCnC on compare match when downcounting
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting Clear OCnA/OCnB/OCnC on compare match when downcounting

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1//COMnC1 is set. See "Phase Correct PWM Mode" on page 148. for more details.

Figura 3.32 Modos de comparación de salida, para los diferentes modos de operación.

En los modos PWM estos bits controlan básicamente si la salida PWM generada debe invertirse o no (inverted or non-inverted PWM), es decir, si cuando se produzca la igualdad (TCNTx = OCRxy) la señal de salida se pone a nivel bajo o viceversa. Para modos que no son PWM, los bits COMxy1:0 controlan si la salida debe establecerse a 1 (nivel alto), a 0 (nivel bajo) o realizar un "toggle" (toggle es cambiar el valor de salida, es decir, si está a 0 ponerlo a 1 y si está a 1 ponerlo a 0) cuando se produzca una igualdad TCNTx = OCRxy.

Para todos los modos, cuando se establecen los bits COMxy1:0 = 0, no se realizará ninguna acción en el pin de comparación de salida OCxy en la próxima igualdad.

El modo de operación, es decir, el comportamiento del timer/counter y de los pines de comparación de salida OCxy, es definido mediante la combinación del modo de generación de forma de onda configurado mediante los bits WGMx3:0 y el modo de comparación de salida configurado mediante los bits COMxy1:0. Los bits del modo de comparación de salida no afectan a la secuencia de conteo, mientras que los bits del modo de generación de forma de onda si, debido a que en ellos se establece el valor TOP.

Bits de selección de reloj del micro:

La fuente de reloj del Arduino es establecida por los bits de selección de reloj CSx2:0 ubicados en el registro de control B del timer (TCCRxB), con ellos se establece el valor del prescaler del Arduino, y con lo cual, se obtiene la fuente de reloj del Arduino = 16MHz / prescaler. Se sabe por el apartado del microcontrolador que la fuente de reloj del Arduino por defecto, es decir, si no se le aplica ningún prescaler es de 16MHz.

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figura 3.33 Bits de selección de reloj CSx2:0.

3.5.5 Interrupciones en Arduino

Como se ha visto en el apartado *3.5.3 Controlador PID* se van a utilizar interrupciones, debido a que es la mejor opción para gestionar los tiempos de muestreo tanto de la medida del sensor como de la señal de control del controlador PID, es decir, es la mejor opción para que cada cierto periodo tiempo se ejecute la señal de control del controlador y se mida la posición de la bola mediante el sensor, para realizar la labor de un periodo de muestreo.

Una interrupción es una señal que interrumpe la actividad normal del microcontrolador y salta a atender la rutina de servicio de interrupción (ISR), una vez finaliza el algoritmo de la interrupción, el procesador vuelve al punto en donde detuvo la ejecución del programa y continúa con su ejecución normal.

Hay 2 clases de interrupciones en Arduino: las interrupciones hardware o externas, que son por ejemplo las que pulsando un pulsador se va a la rutina de servicio de interrupción, y las otras son las interrupciones programadas con los timers, en esta segunda clase de interrupciones, los timers, hacen lo mismo que las interrupciones hardware, pero en lugar de dispararse cuando se cumple un cierto proceso hardware en uno de los pines, se dispara cuando ha transcurrido un tiempo preciso, previamente programado, es decir, que cada cierto periodo de tiempo va a saltar a la rutina de servicio de interrupción (ISR).

En este proyecto se va a utilizar las interrupciones programadas mediante los timers para gestionar el tiempo de muestreo tanto de la medida del sensor, como del controlador PID de la mejor manera posible.

Hay varias posibilidades para realizar esta clase de interrupciones, se pueden realizar mediante el uso de una librería llamada “TimerOne”, pero la forma más adecuada para realizar esta clase de interrupciones es programando directamente los registros internos del microcontrolador del Arduino MEGA ADK, el ATmega2560, para ello, no hace falta el uso de ninguna librería para programar un timer en Arduino.

Como se ha dicho anteriormente una interrupción es un evento externo que interrumpe el programa en ejecución, es decir, deja la ejecución del programa detenida cuando salta

a la rutina de interrupción, y ejecuta una rutina especial de servicio de interrupción llamada ISR. Después de que se haya terminado la rutina de servicio de interrupción ISR, el programa en ejecución continúa por donde se había interrumpido anteriormente, con la siguiente instrucción del programa.

Para que se pueda llamar a una rutina de servicio de interrupción ISR, se deben cumplir las siguientes condiciones:

- Las interrupciones deben estar habilitadas.
- La máscara de interrupción correspondiente debe estar habilitada.

Por defecto en Arduino, las interrupciones de firmware están habilitadas. Las máscaras de interrupción se activan / desactivan configurando los bits en el registro de máscara de interrupción (TIMSKx).

Cuando se produce una interrupción, se establece un "flag" en el registro indicador (flag) de interrupciones (TIFRx). Esta interrupción se borrará automáticamente al ingresar al ISR o al borrar manualmente el bit en el registro de indicador (flag) de interrupción TIFRx.

Dentro de las interrupciones programadas con los timers hay diferentes tipos de interrupciones, esto es debido a que un timer puede generar diferentes tipos de interrupciones, las cuales se van a ver a continuación:

- Interrupción por desbordamiento (overflow) del timer:

  El desbordamiento del timer significa que el timer ha alcanzado el valor límite de conteo, es decir, el valor TOP configurado en los registros. Cuando se produce una interrupción por desbordamiento del timer, el bit que indica el desbordamiento del timer TOVx se pondrá a 1 en el registro de flag (indicador) de interrupción TIFRx. Cuando se activa el bit TOIEx que habilita la interrupción por desbordamiento del timer en el registro de máscara de interrupción TIMSKx, se llamará a la rutina de servicio de interrupción ISR por desbordamiento del timer de la siguiente manera en la programación Arduino: *ISR(TIMERx_OVF_vect)*.

- Interrupción por comparación igualada de salida:

  Una interrupción por comparación igualada de salida ocurre cuando el registro contador TCNTx alcanza el valor del registro de comparación de salida OCRxy establecido. Cuando ocurre esto el indicador (flag) de igualdad OCFxy se pone a 1 en el registro de indicador (flag) de interrupción TIFRx. Para poder ejecutar este tipo de interrupción se debe habilitar la interrupción de comparación igualada de salida en los registros internos del microcontrolador, por medio del bit OCIExy ubicado en el registro de máscara de interrupción TIMSKx, y se debe llamar a la rutina de servicio de interrupción ISR de comparación igualada de salida, por medio de la siguiente instrucción en la programación Arduino: *ISR(TIMERx_COMPy_vect)*.

- Interrupción por captura de entrada:

  Una interrupción de captura de entrada ocurre cuando el registro contador TCNTx alcanza el valor del registro de captura de entrada ICRx. Cuando ocurre esto el indicador (flag) de captura de entrada ICFx se pone a 1 en el registro de flag (indicador) de interrupción TIFRx. Para poder ejecutar este tipo de interrupción se debe habilitar la interrupción de captura de entrada en los registros internos del microcontrolador, por medio del bit ICIEx ubicado en el registro de máscara de interrupción TIMSKx, y se debe llamar a la rutina de servicio de interrupción ISR de captura de entrada, por medio de la siguiente instrucción en la programación Arduino: *ISR(TIMERx_CAPT_vect)*.

Para el cometido de este proyecto, la mejor opción para que cada cierto periodo de tiempo salte a la rutina de servicio de interrupción (ISR), es utilizar el tipo de interrupción por comparación igualada, es decir, se va a generar la interrupción cuando TCNTx sea igual al valor de OCRxy establecido.

3.6 Interfaz de usuario a realizar

Uno de los objetivos de este trabajo fin de grado es el de diseñar, desarrollar e implementar una interfaz de usuario que permita interactuar con el control de la maqueta y con los diferentes parámetros de los controladores implementados.

Una interfaz de usuario, HMI (Human-Machine Interface) se define como una interfaz hombre-máquina, en la cual se utiliza un panel de control diseñado para conseguir una comunicación interactiva entre operador y proceso/máquina, con la función de transmitir ordenes, visualizar gráficamente los resultados y obtener una situación del proceso/máquina en tiempo real. Es la principal herramienta utilizada por operarios y supervisores de línea para coordinar y controlar procesos industriales y de fabricación. El HMI traduce variables de procesos complejos en información útil y procesable.

Para la realización de la interfaz, se barajó la idea de realizar una interfaz de usuario gráfica mediante el software llamado processing de filosofía open source. El software processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, el cual se utiliza para desarrollar gráficamente un modelo de programación

Está idea se desechó, y finalmente se decidió diseñar, desarrollar e implementar una placa PCB para la realización de la interfaz de usuario; para el diseño y desarrollo de la misma, se va a utilizar el software llamado KiCad de filosofía open source.

El programa KiCad es un paquete software libre con filosofía open source para la automatización del diseño electrónico, cuenta con un entorno para el desarrollo de esquemas electrónicos y otro para el diseño de los circuitos impresos (PCB) partiendo del esquema desarrollado, es decir, realiza la conversión del esquema a la placa de circuito impreso.

Entonces utilizando el software KiCad para el diseño de la placa PCB, se quiere realizar una interfaz de usuario, en la cual, el usuario pueda interactuar con la maqueta eligiendo diferentes modos de funcionamiento, en los cuales se van a implementar diferentes tipos

de reguladores que se van a utilizar para poder analizar y comprender las diferencias que hay entre ellos; el usuario va a poder elegir el modo de funcionamiento que desee. Además de todo esto, se va dar la opción al usuario en uno de los modos de funcionamiento de poder variar las constantes del controlador PID (K_p, K_i, K_d), además de poder establecer el valor a la consigna, para que pueda visualizar de manera clara como afectan cada una de las acciones del PID en la respuesta del sistema al modificarlas. Durante todo el proceso se va a ir visualizando en una pantalla todo lo explicado anteriormente, van a ir apareciendo mensajes en una pantalla para ir explicando en cada momento que debe introducir o que debe realizar el usuario.

Para realizar todo esto, los componentes más importantes que van a conformar la interfaz de usuario aparte de la placa PCB van a ser los encoders rotativos, puesto que es necesario un dispositivo para poder variar cada una de las constantes del controlador PID y el valor de la consigna; y la pantalla LCD, la cual, es muy importante, puesto que es en donde se van a visualizar todas las variables: las tres constantes del PID, la consigna, el error de posición de la bola y los diferentes mensajes explicativos que van aparecer durante la navegación.

A continuación se va a explicar de una manera más extensa cómo funcionan, como se conectan y como se programan los 2 componentes más importantes que conforman la interfaz de usuario (aparte de la placa PCB).

3.6.1 Pantalla LCD

La pantalla LCD basada en el controlador HD44780 de Hitachi es un periférico muy común, que se utiliza ampliamente en proyectos con arduino y microcontroladores en general, sin embargo, es bien sabido que este tipo de pantalla requiere muchos pines del microcontrolador para ser controlada, debido principalmente a que utiliza un bus paralelo para comunicarse. Afortunadamente existe una solución muy fácil y económica para este problema: un adaptador basado en el PCF8574 llamado módulo I2C que permite conectar la pantalla al Arduino usando solamente dos líneas digitales a través del bus I2C. Dichos pines, pueden además ser compartidos por otros periféricos.

Figura 3.34 Pantalla LCD de 20x4.

Como se va a ver a continuación utilizar un módulo I2C para conectar una pantalla LCD al Arduino es una ventaja enorme, puesto que, simplifica las conexiones de la pantalla LCD con el Arduino, de manera que se pueda aprovechar mejor los pines de I/O.

Las pantallas LCDs se suelen utilizar habitualmente para mostrar información o ciertos datos del sistema, como pueda ser la temperatura, humedad, presión o voltaje. La pantalla LCD a utilizar, va a ser una pantalla de 20x4, es decir, de 20 columnas y 4 filas.

En este caso como se ha mencionado anteriormente, la pantalla LCD de 20x4 se utiliza para visualizar todas las variables: las tres constantes del controlador PID (KP, KI, KD) en cada momento, que se utilizan para controlar el sistema barra y bola, el valor de consigna establecido, el error de posición de la bola, además de mostrar el modo de funcionamiento seleccionado (MODO 1, MODO 2...) y los diferentes mensajes explicativos que van aparecer durante la navegación.

La pantalla LCD de 20x4 como se ha comentado anteriormente se puede conexionar de 2 formas diferentes:

- Conexión directa: Utiliza 16 o 12 pines para su conexión, lo cual, es una pérdida de recursos. Necesita la librería Liquid Crystal para su programación, además de esto, es necesario un potenciómetro para ajustar el contraste de la pantalla LCD.

- Conexión I2C: Solo se emplean 4 pines: masa (GND), alimentación (5V), SDA y SCL. Las librerías necesarias para programar la pantalla LCD conectada al Arduino por medio del módulo I2C son: la librería Wire (Wire.h) y la librería LiquidCrystal_I2C (LiquidCrystal_I2C.h). El módulo I2C se conecta con la pantalla LCD mediante la soldadura de ambos pines. Es un método mucho más efectivo, puesto que se simplifican de una manera bastante importante las conexiones de la pantalla LCD con el Arduino, de 12 pines de la conexión directa a conectar solamente 4 pines con el uso del módulo I2C.

Pantalla LCD con conexión directa

La mayoría de las pantallas LCD que se están haciendo ahora, vienen con una fila de dieciséis pines. Los primeros catorce pines se utilizan para controlar la visualización y los dos últimos son para la iluminación de fondo.

En la siguiente tabla se describen los pines de la pantalla LCD:

Tabla 3.2 Descripción de los pines de la pantalla LCD.

Pin	Descripción
VSS (1)	GND
VDD (2)	5V
V0 (3)	Ajusta el contraste de la pantalla LCD (se conecta al potenciómetro)
RS (4): Register Selecter	Selector entre comandos y datos. Controla las zonas de la memoria de la pantalla LCD donde escribimos los datos
R/W (5): Read/Write	Selecciona el modo lectura o escritura

Pin	Descripción
E (6)	Activa los registros de la pantalla LCD para poder leer y escribir en ellos (Sincronización de lectura de datos)
D0 - D7 (7-14)	Pines de datos de 8 bit. Los estados de estos pines (high o low) son los bits que estás escribiendo en modo escritura o los bits que estás leyendo en modo lectura
A (15)	Ánodo de los LEDs de la iluminación de fondo (Alimentación luz de fondo 5V)
K (16)	Cátodo de los LEDs de la iluminación de fondo (GND luz de fondo)

LCD	VSS	VDD	RS	R/W	E	D4	D5	D6	D7	A	K
Arduino	GND	5V	12	GND	11	5	4	3	2	5V	GND

En la tabla anterior se pueden ver las conexiones entre la pantalla LCD y los pines de Arduino. Como se puede ver, solo se van a utilizar 4 bits de datos y van a ser los 4 últimos bits de la pantalla LCD.

En la siguiente imagen se muestra la correspondencia con los pines físicos de la pantalla LCD.

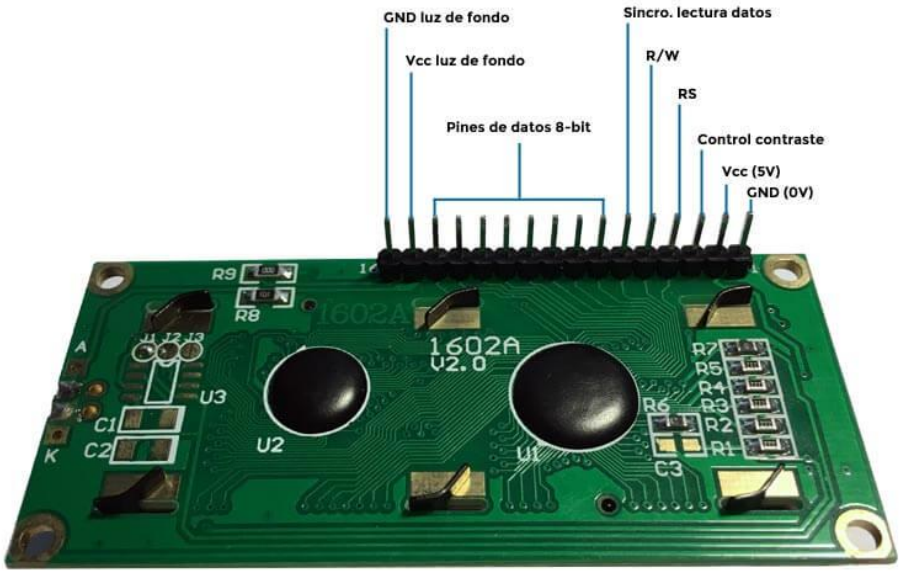


Figura 3.35 Descripción de los pines de la pantalla LCD.

Las conexiones entre la pantalla LCD y el Arduino serían de la siguiente manera, como se puede ver en esta imagen más grafica:

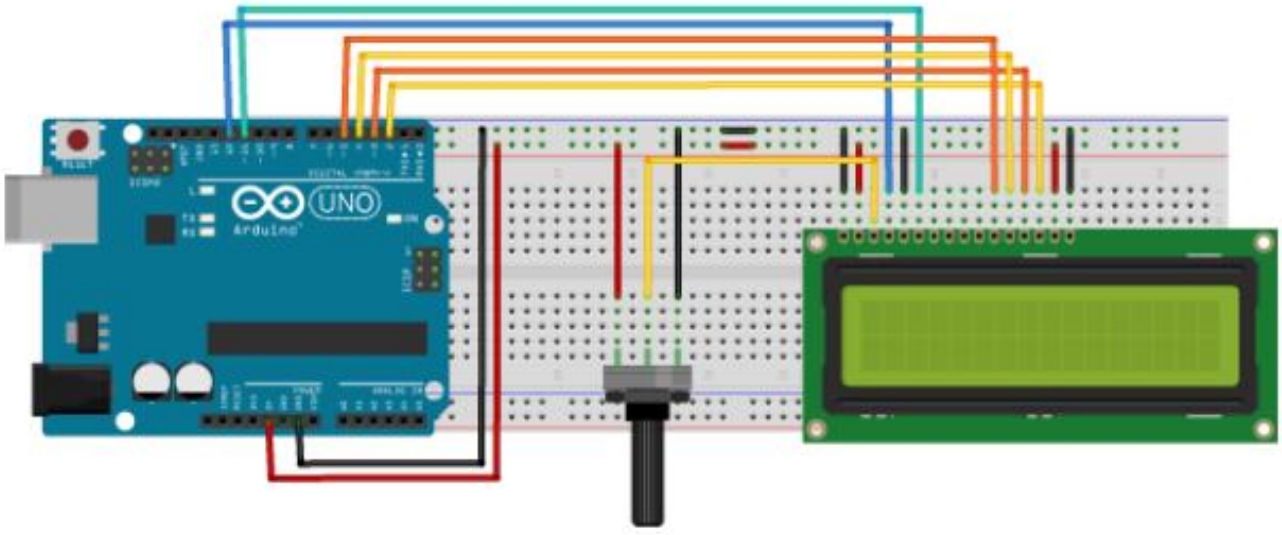


Figura 3.36 Conexiones entre pantalla LCD y Arduino.

Las pantallas LCDs en Arduino se programan mediante la librería Lyquid Crystal, para añadir la librería en la programación Arduino, se declara de la siguiente manera en el código: #include <LiquidCrystal.h>.

Esta librería permite que una placa Arduino controle pantallas LCDs basadas en el chipset Hitachi HD44780 (o una compatible), que se encuentra en la mayoría de las pantallas LCD basadas en texto. La librería funciona en modo de 4 u 8 bits, es decir, usa 4 u 8 líneas de datos además de las líneas RS, Enable (E) y, opcionalmente, las líneas de control R/W. Además de incluir la librería en la programación, se utilizan las funciones específicas de la librería para programar la pantalla LCD como se quiera.

Pantalla LCD conectada a un módulo I2C

Como se ha explicado anteriormente, este tipo de conexión con respecto a la conexión directa tiene múltiples ventajas, ya que, mientras con la conexión directa se utilizan 12 pines, con el uso del módulo I2C se reducen solamente al uso de 4 pines de Arduino, por lo que este método de conexión es mucho más efectivo, puesto que, tanto para el diseño de la placa PCB como para el montaje de la interfaz de usuario, este método reduce notablemente la complejidad de ambos. Por todo esto, es el método de conexión utilizado para conectar la pantalla LCD al Arduino.

El bus I2C fue desarrollado por Philips, y consiste en una conexión maestro-esclavo. Se basa en un protocolo de comunicación que emplea dos líneas de control, una para transmitir los datos, SDA, y otra, el reloj asíncrono que indica cuando leer los datos, SCL, más masa (GND) y alimentación (5V). Cada dispositivo conectado al bus I2C tiene su dirección exclusiva, de 7 bits, (así que, en teoría, podemos conectar $2^7 = 128$ dispositivos), uno de estos dispositivos, debe actuar como maestro (master), el cual es el que controla la señal de reloj. Es multi maestro, con lo cual, varios dispositivos pueden ser el maestro, por lo que el dispositivo con función de maestro puede cambiar, pero solo uno puede estar activo a la vez, y proporciona un protocolo de arbitraje y detección de colisiones.

A este bus se le puede llamar de diferentes formas I2C, TWI (interfaz de 2 hilos), etc, pero siempre es lo mismo.

La idea es que todos los componentes se conecten en paralelo a las dos líneas del bus, SDA (señal de datos) y SCL (señal de reloj). Como se ha mencionado antes, en cada momento solo puede haber un maestro, en este caso el Arduino, y los demás se configuran como esclavos, es decir, la pantalla LCD se configura como esclavo.

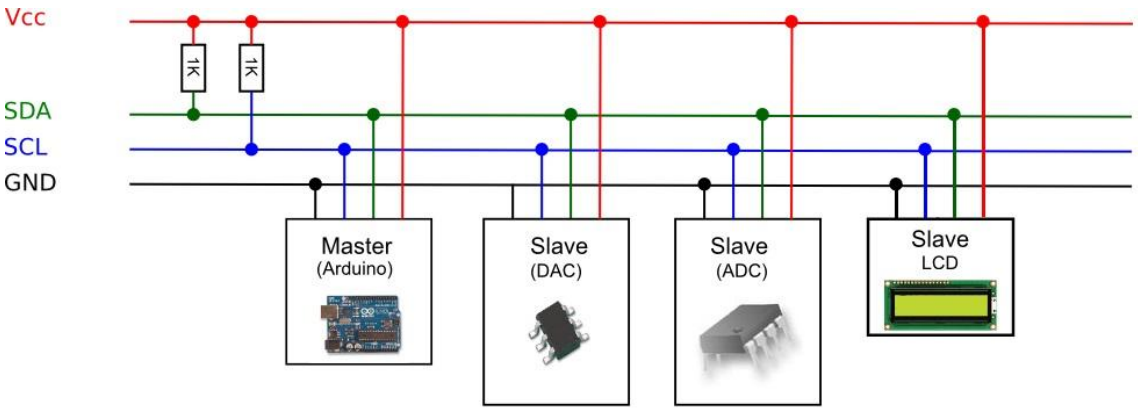


Figura 3.37 Conexiones maestro-esclavo con el bus I2C.

Como se puede ver en la figura hay unas resistencias pull-up conectadas a SDA y SCL, en este caso, las resistencias vienen incorporadas en los pines SDA y SCL de la placa Arduino MEGA que se utiliza.

Para aplicar el bus I2C de comunicación se utiliza el módulo I2C entre la pantalla LCD (esclavo) y el Arduino MEGA ADK (maestro):

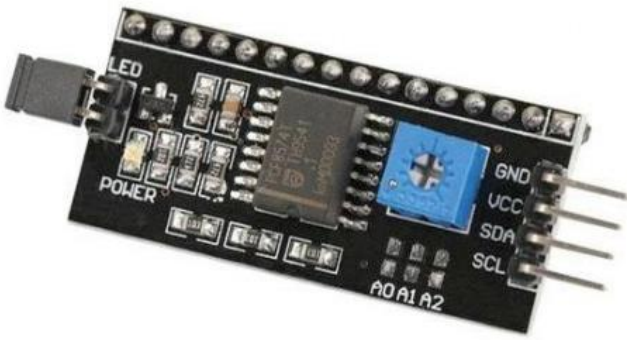


Figura 3.38 Módulo I2C a utilizar.

Para conectar la pantalla LCD de 20×4 por medio del bus I2C a la placa Arduino, el primer paso es soldar el adaptador I2C en la parte trasera de la pantalla. Al finalizar la soldadura, el adaptador I2C debe verse de la siguiente manera:

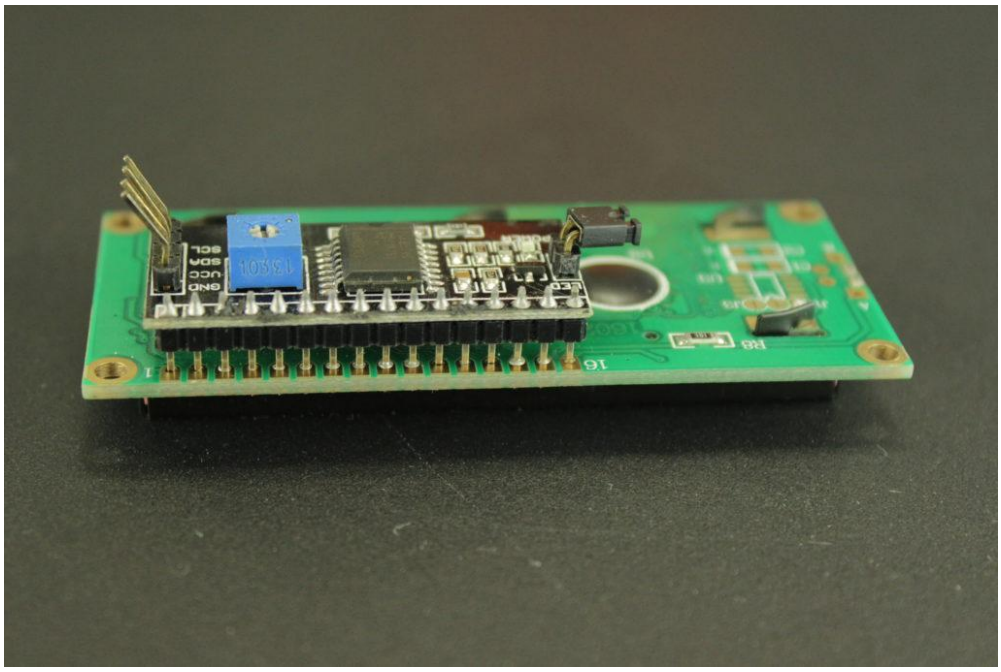


Figura 3.39 Módulo I2C soldado a la pantalla LCD.

Como se aprecia el utilizar un adaptador I2C para la pantalla LCD permite ahorrar bastante tiempo de conexionado y reducir la complejidad entre la conexión hardware de la placa Arduino con la pantalla LCD, pues ya incluye el potenciómetro para regular el contraste de la pantalla. También incluye todo lo necesario para el funcionamiento del backlight (iluminación de fondo de la pantalla), pudiendo incluso controlar esta función a través del software Arduino.

Una vez que se tiene soldado el módulo I2C hay que conectar el módulo I2C al Arduino, en este caso al Arduino MEGA ADK, para ello solo se emplean 4 pines: masa (GND), alimentación (5V), SDA (señal de datos) y SCL (señal de reloj). Hay que decir que en el Arduino MEGA ADK, los pines del bus I2C corresponden a los pines 20 (SDA) y 21 (SCL) de la placa Arduino.

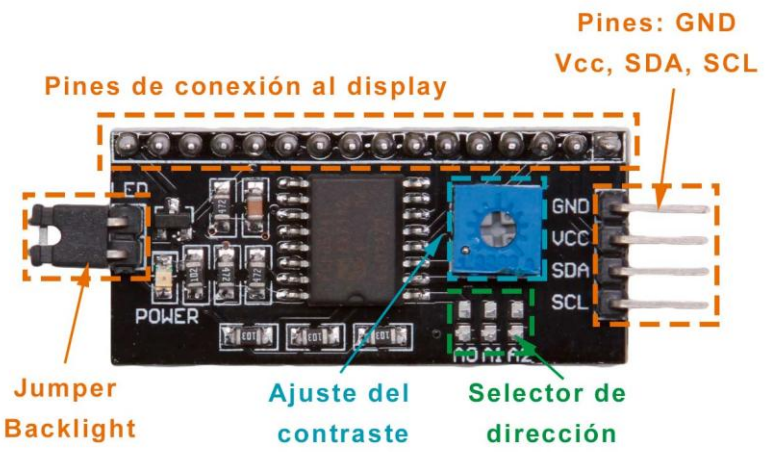


Figura 3.40 Explicación hardware del módulo I2C.

Una vez expuestas las conexiones hardware tanto de la pantalla LCD con el módulo I2C, como la conexión entre el módulo I2C y la placa Arduino, a continuación se va a explicar de manera breve, como se debe realizar la programación software de una pantalla LCD conectada a un módulo I2C con Arduino de manera correcta.

Para programar la pantalla LCD mediante la utilización de un módulo I2C con Arduino, es necesario utilizar 2 librerías para que pueda comunicarse con el chip PCF8574: una la librería Wire (Wire.h) que forma parte del IDE de Arduino, y la librería LiquidCrystal_I2C (LiquidCrystal_I2C.h), la cual, es una librería que hay que agregar al IDE de Arduino. Esta última librería se ha descargado de la siguiente página web: [1]

La librería Wire gestiona el protocolo de comunicaciones completo, es decir, es la librería que permite la comunicación con dispositivos I2C/TWI. La librería LiquidCrystal_I2C se utiliza para poder programar la pantalla LCD mediante el uso del módulo I2C, dispone de funciones similares (algunas idénticas) a las de la librería oficial (LiquidCrystal). Con estas 2 librerías se debe configurar en la programación para su correcto funcionamiento, la dirección del módulo I2C y las constantes que indican las columnas y filas que dispone la pantalla LCD a utilizar. Como se puede ver en la siguiente instrucción, así se debe configurar la pantalla LCD con el módulo I2C:

LiquidCrystal_I2C lcd(0x3F, 20, 4);

Para conocer la dirección del módulo I2C conectado se utiliza un programa llamado escáner (I2C_scan), el cuál proporciona la dirección del módulo I2C. El programa I2C_scan se ha descargado de la siguiente página web, el link está situado en el apartado del "scanner de I2C": [2].

Una vez aplicado el programa llamado escáner, desde la comunicación serie se observa la respuesta del sketch, en la cual, aparece la dirección del módulo I2C que se necesita para poder comunicarse con la pantalla LCD. En este caso la dirección del módulo I2C es la siguiente:

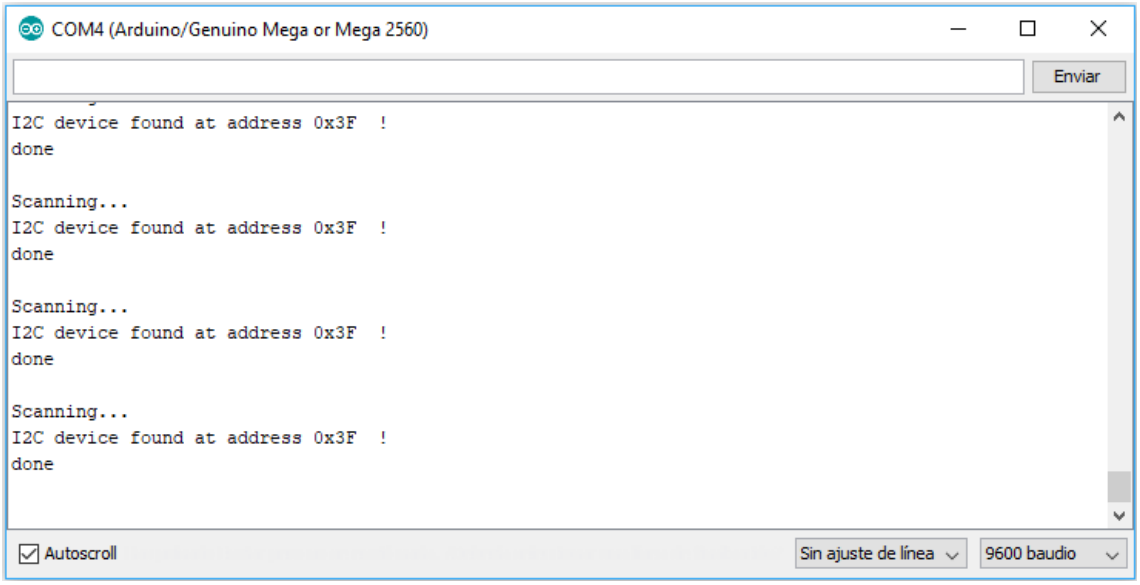


Figura 3.41 Dirección obtenida del módulo I2C.

Como se puede ver en la figura la dirección del módulo I2C conectado es 0x3F.

Las funciones fundamentales de la librería LiquidCrystal_I2C que se deben utilizar en la programación son las siguientes: en el *void setup*, es decir, en la configuración de la programación se debe iniciar la librería con la función *lcd.init()*, además se debe encender la iluminación de la pantalla LCD por medio de la función *lcd.backlight()*.

Además de estas funciones también se pueden utilizar las funciones típicas de la librería oficial (LiquidCrystal), además de las que se van a exponer hay otras, pero se exponen las más típicas:

- Para borrar toda la pantalla por si hay algo escrito se utiliza la función *lcd.clear()*.
- Para situar el cursor en las coordenadas donde se quiere escribir se utiliza la siguiente función *lcd.setCursor(C, F)*. Se pone el número de columna y el número de fila en donde se quiere situar el cursor.
- Para escribir en la pantalla LCD se utiliza la siguiente función *lcd.print("Texto a mostrar")*, obviamente también se pueden mostrar los valores de las diferentes variables.

3.6.2 Encoder rotativo con pulsador

Uno de los componentes más importantes de la interfaz de usuario van a ser los encoders rotativos con pulsador, los cuales se van a utilizar para establecer el valor de las constantes del controlador PID (K_P, K_I, K_P), además de asignar el valor de posición de la bola que se desea (consigna).

Se barajo la posibilidad de utilizar potenciómetros en vez de los encoders rotativos, pero se vio que los encoders rotativos tienen bastantes ventajas con respecto a los potenciómetros, puesto que a la hora de realizar la lectura de la señal de un potenciómetro con Arduino, las dificultades son mayores porque son señales analógicas, es decir, es tan limitadas por el convertidor analógico digital de Arduino de 10 bits (1023 puntos), por lo que al intentar establecer un valor exacto a la consigna o a las constantes del controlador PID la lectura del potenciómetro no es precisa puesto que solamente cuenta con 1023 puntos para ello. El encoder rotativo en comparación con el potenciómetro, además de ser preciso gracias a su naturaleza digital, puesto que actúa como un contador digital contando los pulsos que genera, no está limitado en giro, es decir, se puede establecer el valor que se quiera.

Por todo esto, se han utilizado encoders rotativos en vez de potenciómetros.

Existen múltiples tipos de encoders rotativos, pueden ser de varios tipos como absolutos o incrementales. En este caso se ha utilizado el típico encoder rotativo digital incremental con pulsador. Un encoder rotativo es un dispositivo electro mecánico que

convierte el movimiento de giro de un eje en una señal proporcional al giro de dicho eje. Se parece mucho a un potenciómetro, pero a diferencia de este, puede girar indefinidamente tanto en sentido horario como en sentido antihorario.

Figura 3.42 Encoder rotativo incremental con pulsador utilizado.

Estos encoders se suelen utilizar como mando o dispositivo de control en sustitución de potenciómetros, para la navegación en pantallas LCD y para la medición de velocidades de giro en un eje. En este caso, se han utilizado como se ha mencionado anteriormente para establecer los valores tanto de las constantes del controlador PID, como de la consigna.

Funcionamiento, conexiones hardware y programación mediante Arduino.

En cuanto al funcionamiento hay que decir que un encoder rotativo incremental, a diferencia de un potenciómetro, no tiene límite de giro sino que continua generando pulsos a medida que giramos el eje tanto en un sentido como en el otro, el encoder utilizado produce 30 pulsos aproximadamente por cada vuelta completa. A la hora de girar el eje provoca con unos pequeños microinterruptores una señal digital (un pulso) por cada muesca que gira.

En cuanto a las conexiones hardware del encoder rotativo con pulsador, cabe destacar que dispone de 5 pines: además de necesitar de alimentación y GND dispone de 3 salidas, una para el tren de pulsos A (output A), otra para el tren de pulsos B (output B),

y por ultimo para el pulsador. Las conexiones del encoder son las que se muestran a continuación:

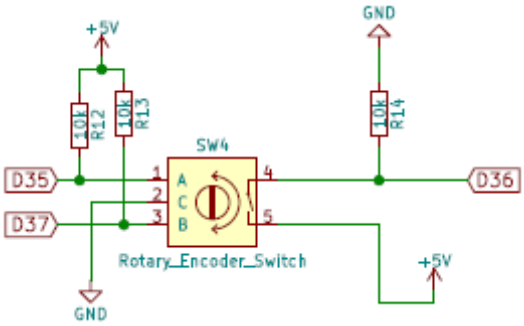


Figura 3.43 Conexiones hardware de los encoders rotativos con pulsador.

Los pines utilizados para conectar el encoder con el Arduino se declaran como pines de entrada en la programación.

El encoder dispone de dos salidas A y B para poder conocer el sentido de giro del eje además del numero de pulsos que se ha girado, es decir, al girar el eje del encoder se generan dos trenes de pulsos diferentes (tren de pulsos A y tren de pulsos B), esto es, dependiendo de cómo sean ambos trenes de pulsos se puede determinar si se ha girado el eje en sentido horario o en sentido antihorario.

El funcionamiento para conocer si se ha girado el eje en sentido horario o en sentido antihorario es el siguiente: al girar el eje del encoder se generan 2 señales cuadradas de salida desfasadas 90º entre sí, ya que primero se pasa por un micro interruptor y luego por el otro:

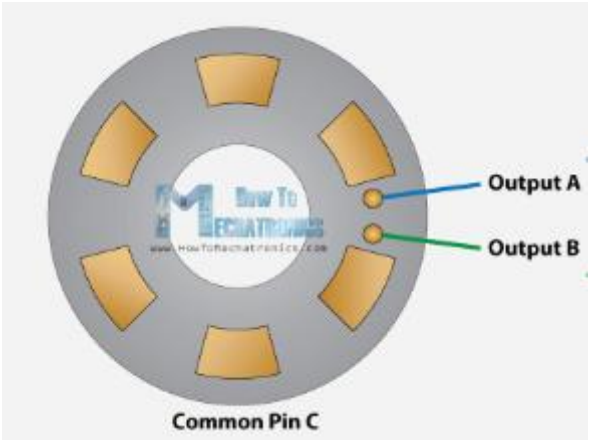


Figura 3.44 Microinterruptores entre las muescas del encoder.

Entonces, la clave para conocer en qué sentido se ha girado el encoder, es realizar la comparación de los dos trenes de pulsos siempre con respecto al flanco de bajada de la señal A, es decir, cuando la señal A de salida este en el flanco de bajada, en ese momento el sentido de rotación queda indicado por el valor de la señal B, como se muestra en la siguiente figura:

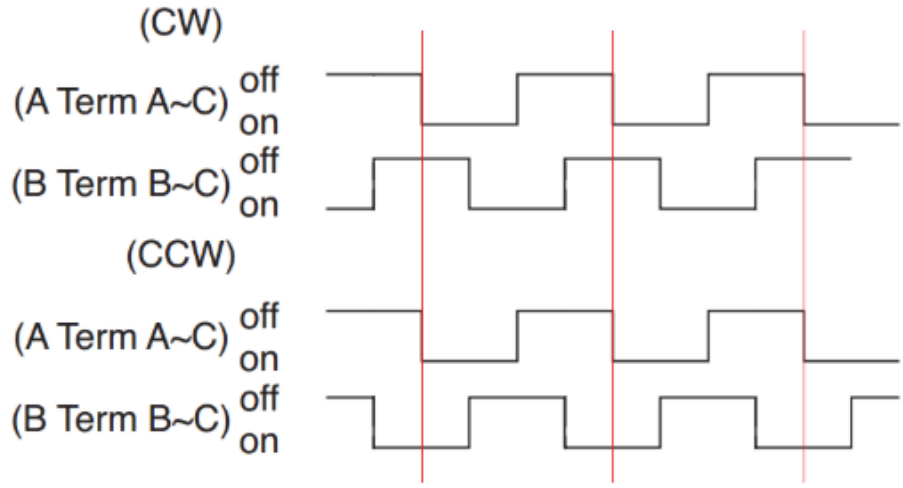


Figura 3.45 Diagrama de pulsos de las salidas A y B.

Como se indica en la imagen, cuando el eje se gira en sentido horario como se pasa primero por el micro interruptor de la señal A y luego por el B, la señal B estará desfasada 90º con respecto a la señal A, con lo cual, en el instante del flanco de bajada de la señal A, la señal B estará en un valor alto, es decir, cuando la señal A y la señal B tengan estados diferentes el encoder se ha girado en sentido horario. Mientras que con un giro antihorario sucede lo contrario a lo explicado anteriormente, la señal B estará en un estado bajo, es decir, cuando ambas señales tengan el mismo estado el encoder se ha girado en sentido antihorario.

Conociendo todo esto, la programación software de Arduino para el encoder rotativo es la siguiente:

```
int outputA = 35;

int outputB = 37;
```

```
float counter = 0;
int aState;
int aLastState;
void setup() {
  pinMode (outputA,INPUT);
  pinMode (outputB,INPUT);
  Serial.begin (9600);
  // Se lee el valor inicial del estado de la salida A.
  aLastState = digitalRead(outputA);
 }
void loop()
{
  aState = digitalRead(outputA); // Lee el estado "actual" de la salida A
  // Si la lectura actual y la anterior de la salida A es diferente, eso significa que ha llegado un pulso, se ha girado el encoder.
  if (aState != aLastState){
    // Si el estado de la salida B es diferente al estado actual de la salida A, eso significa que el encoder se ha girado en sentido horario.
    if (digitalRead(outputB) ¡= aState) {
      counter ++;
    }
    // Si no, el encoder se ha girado en sentido antihorario.
    else {
      counter --;
    }
    Serial.print("Position: ");
    Serial.println(counter);
```

```
  }

  aLastState = aState; // Se actualiza la variable del "estado anterior" de la salida A con el del "estado actual".

}
```

4 DISEÑO

4.1 Análisis del control del sistema

Para realizar el control de este sistema es necesario realizar un sistema en lazo cerrado, es decir, va a ser un sistema de control realimentado negativamente, debido a que como se ha dicho en apartados anteriores es un sistema inestable en lazo abierto, además, como se desea que ante posibles perturbaciones del sistema el control vaya corrigiendo la posición de la bola para establecerla en la posición especificada, para ello, es imprescindible realizar un sistema en lazo cerrado. Este sistema va a ser un sistema de control realimentado, para poder corregir las posibles desviaciones que genere la perturbación.

Los sistemas de control en lazo cerrado, alimentan al controlador la señal de error de actuación, que es la diferencia entre la señal de entrada (referencia) y la señal de realimentación (valor real medido por un sensor), a fin de reducir el error y llevar la salida del sistema a un valor deseado.

Análisis del control de posición de una bola:

- Posición deseada de la bola, R(s) (referencia): Se quiere que la bola se posicione en una posición determinada de la barra, y que se mantenga en esa posición ante perturbaciones externas.
- Variable a controlar: En este sistema barra y bola se realiza el control de posición de una bola, con lo cual, en este sistema se va a controlar la posición de de la bola, X(s) en mm.
- Variable manipulada: Es la variable que manipula el controlador para poder modificar la variable controlada, en este caso la posición de la bola, y así establecer la bola en la posición deseada de la barra. En este caso, la variable que va a manipular el controlador es el ángulo de giro del servomotor (en rad), Θ(s) [rad]. Para manipular la señal del ángulo de giro del servomotor en la programación software de Arduino, se ha generado una señal PWM para poder controlar el servo, con lo cual, modificando el duty cycle de esa señal PWM se

consigue manipular el ángulo de giro del servomotor, puesto que se modifica la tensión aplicada al servomotor.

- Controlador: Como se ha dicho en apartados anteriores el controlador que se va a utilizar para controlar el sistema es un controlador PID, puesto que, es necesaria la participación de todas las acciones del PID para controlar de manera correcta el sistema.
- Planta: Sistema real constituido por la maqueta barra y bola, y sus distintos componentes.
- Sensor: En este caso, será el sensor infrarrojo utilizado para medir la posición de la bola en la barra en mm. El bloque del sensor habitualmente suele estar en la realimentación del sistema constituido por una constante, que suele realizar la conversión de unidades para poder obtener el error de actuación de manera correcta.

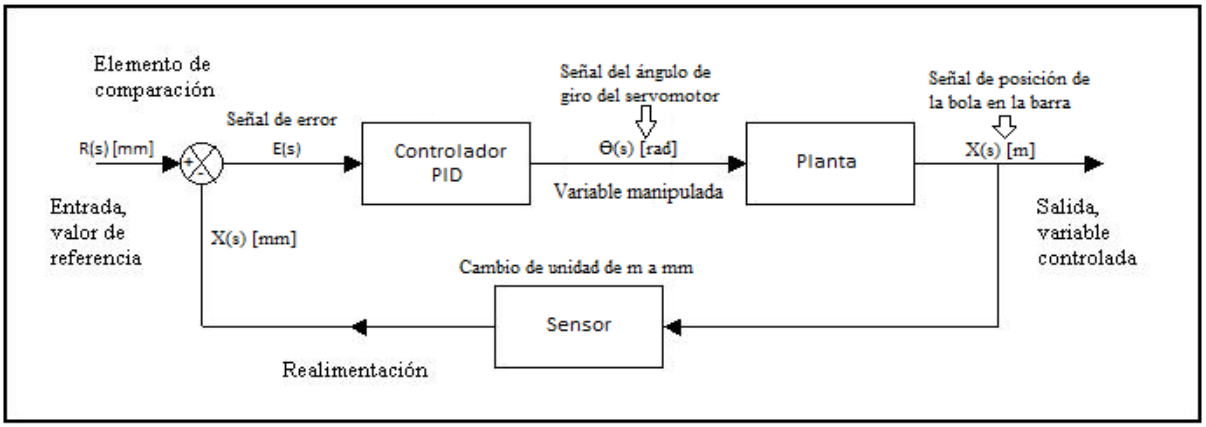


Figura 4.1 Sistema de control en lazo cerrado para el sistema barra y bola.

En este caso, el control del sistema se debe realizar en tiempo discreto (z), puesto que, se utiliza un dispositivo digital (Arduino) para controlar el sistema.

El desarrollo de procesadores digitales especializados ha desplazado los sistemas de control analógico en la mayoría de las aplicaciones industriales. Entre las ventajas que ofrece el control digital por ordenador está la flexibilidad de modificación de los algoritmos, o la implementación de algoritmos complejos, es decir, permite leyes de control que alcanzan grados de exactitud muy elevada, además los componentes

digitales son más fiables que los analógicos, son más robustos y menos influenciables por envejecimiento.

En cuanto a los inconvenientes cabe destacar que el control se realiza en momentos determinados de tiempo, por medio del denominado periodo de muestreo del algoritmo. El control analógico trabaja en tiempo real. La limitación de resolución es otro de los inconvenientes del control digital, puesto que el procesador va a estar limitado por un número de bits que es capaz de procesar como una palabra finita (en el caso de Arduino el convertidor analógico digital dispone de 10 bits, y para las señales PWM dispone de 8 bits). El control analógico tiene una resolución infinita.

En los sistemas en los que el control está basado en un procesador digital habrá una parte del sistema que opera en tiempo continuo y otra parte en tiempo discreto; como es sabido la parte que opera siempre en tiempo continuo es la planta, puesto que siempre es analógica, sin embargo, como el control del sistema está basado en un procesador digital los controladores operan en tiempo discreto.

4.2 Servomotor utilizado

El servomotor utilizado en el proyecto es un *Futaba S3003*, que no es de rotación continua, su rango de movilidad es de 180º; tiene una frecuencia de trabajo de 50Hz, es decir, un periodo de trabajo de 20ms y funciona alimentándolo a 5V (alimentación de 4,8V a 6V). Es un buen servo de características intermedias e ideal para pequeños proyectos; tiene un par elevado y buenas prestaciones. Conociendo estas especificaciones del servomotor lo que se quiere, en este proyecto, es controlar la posición (ángulo de giro) del servo mediante señales PWM; más adelante se indicará como se programa y como se genera la señal PWM para controlar la posición del servo.

Figura 4.2 Servomotor Futaba S3003 utilizado.

Las especificaciones técnicas del servomotor *Futaba S3003*, que es el servomotor utilizado en el proyecto, son las siguientes:

Detailed Specifications			
Control System:	+Pulse Width Control 1520usec Neutral	Current Drain (4.8V):	7.2mA/idle
Required Pulse:	3-5 Volt Peak to Peak Square Wave	Current Drain (6.0V):	8mA/idle
Operating Voltage:	4.8-6.0 Volts	Direction:	Counter Clockwise/Pulse Traveling 1520-1900usec
Operating Temperature Range:	-20 to +60 Degree C	Motor Type:	3 Pole Ferrite
Operating Speed (4.8V):	0.23sec/60 degrees at no load	Potentiometer Drive:	Indirect Drive
Operating Speed (6.0V):	0.19sec/60 degrees at no load	Bearing Type:	Plastic Bearing
Stall Torque (4.8V):	44 oz/in. (3.2kg.cm)	Gear Type:	All Nylon Gears
Stall Torque (6.0V):	56.8 oz/in. (4.1kg.cm)	Connector Wire Length:	12"
Operating Angle:	45 Deg. one side pulse traveling 400usec	Dimensions:	1.6" x 0.8"x 1.4" (41 x 20 x 36mm)
360 Modifiable:	Yes	Weight:	1.3oz. (37.2g)

Figura 4.3 Especificaciones técnicas del servomotor Futaba S3003.

En cuanto a las conexiones del servomotor con el Arduino, el servomotor consta de 3 cables, y la función de cada uno es la que se muestra a continuación:

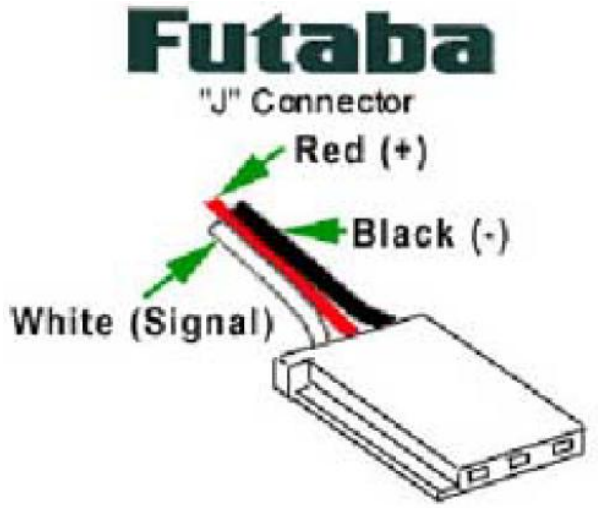


Figura 4.4 Conexiones del servomotor.

Como se puede ver en la *figura 4.4 conexiones del servomotor*, se conecta al Arduino mediante tres cables: cable blanco, va a ser por el que se transmita la señal PWM de control del servo, se va a conectar a un pin de salida digital del Arduino, en este caso, al pin digital 46; el cable rojo se va a conectar a la alimentación (5V) para el funcionamiento del servomotor; y por último el cable negro se va a conectar a masa (GND).

El servomotor va a ser el actuador del sistema, es decir, es el componente que va a inclinar la barra en un sentido y en el otro para poder mover la bola de posición, por medio de una biela que une el disco en donde está introducido el servo, con la barra horizontal, como se puede ver en la siguiente figura:

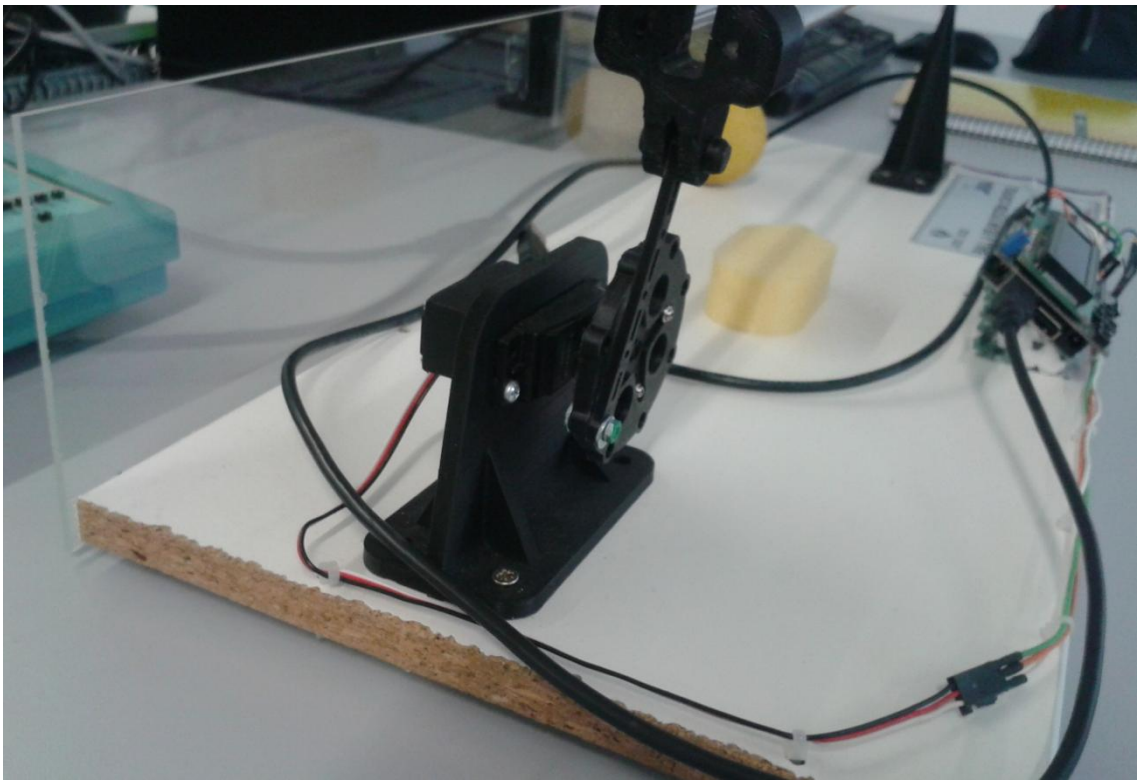


Figura 4.5 El servomotor introducido en el disco y unido a la barra mediante una biela.

El servomotor es el que va actuar (mediante una biela) para que la barra se incline cuando la bola no esté en la posición deseada (posición asignada en la consigna), cuando se lo transmita y como se lo transmita en términos de posición del servo el microcontrolador, por medio de la señal de control PWM programada y calculada mediante un controlador PID. Básicamente la señal de control del sistema va a calcular el ángulo de giro al que debería situarse el servomotor, para colocar y estabilizar la bola en la posición deseada (consigna), con esta información, el servomotor actúa e inclina la barra con el ángulo de giro calculado en la señal de control (controlador PID). Al actuar el servomotor el disco en el cual está introducido el servo se mueve, lo que conlleva que la biela se desplace, y por medio del desplazamiento de la biela la barra se inclina en un sentido o en el otro, puesto que la biela es el elemento que transfiere el movimiento del servomotor introducido en el disco a la barra horizontal, es decir, transforma el movimiento rotatorio del servomotor en un movimiento ascendente y descendente de la barra horizontal. Este proceso se repite hasta que la bola queda situada en el valor de consigna especificado.

4.3 Sensor infrarrojo utilizado

Antes de nada, cabe mencionar, que la respuesta real de los sensores tiene un rango de medida limitado, nunca es del todo lineal por ello hay que realizar la linealización del sensor, y suele estar afectada por perturbaciones del entorno exterior como pueda ser ruido, por lo que requieren de un acondicionamiento de señal. También se va aplicar un filtro por software para despreciar los valores incoherentes que se puedan medir, realizando una media de unos cuantos valores.

Entre los sensores infrarrojos hay una amplia gama de posibilidades, pero en este proyecto se ha utilizado el sensor infrarrojo Sharp GP2Y0A21YK.

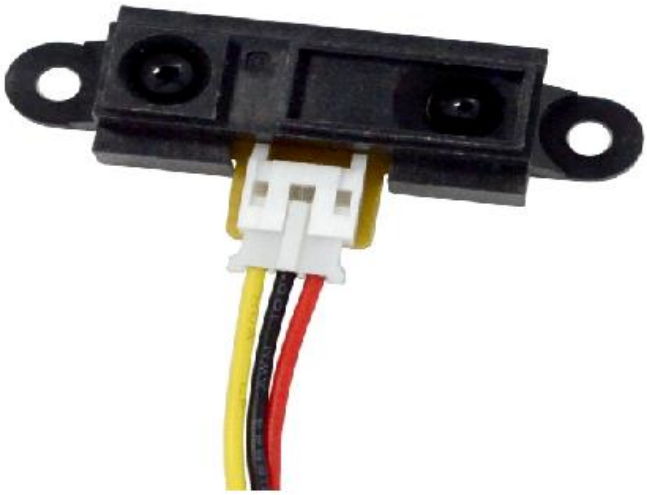


Figura 4.6 Sensor infrarrojo Sharp GP2Y0A21YK.

El sensor infrarrojo Sharp GP2Y0A21YK tiene las siguientes características:

1. No detecta objetos transparentes.
2. Tipo de distancia de salida (voltaje analógico), es decir, la señal de salida que va a transmitir el sensor al Arduino va a ser un voltaje analógico, de 3,1V como máximo, como se puede ver en la curva característica del sensor; luego por medio del convertidor analógico digital del Arduino de 10 bits, ese voltaje analógico se transforma en un valor de 0 a 1023, que es el que sale a la hora de hacer la lectura del sensor en la programación.

3. Tiene un rango de medida de 10 a 80cm. Si la bola se acerca a más de 10cm o está más lejos de 80cm, la medida del sensor puede ser errónea.
4. El circuito de control externo es innecesario.
5. Bajo coste.

Hay que decir que el sensor infrarrojo funciona con 5V, y como la tensión de salida es una tensión relacionada con la distancia medida, como se puede ver en la curva característica del sensor:

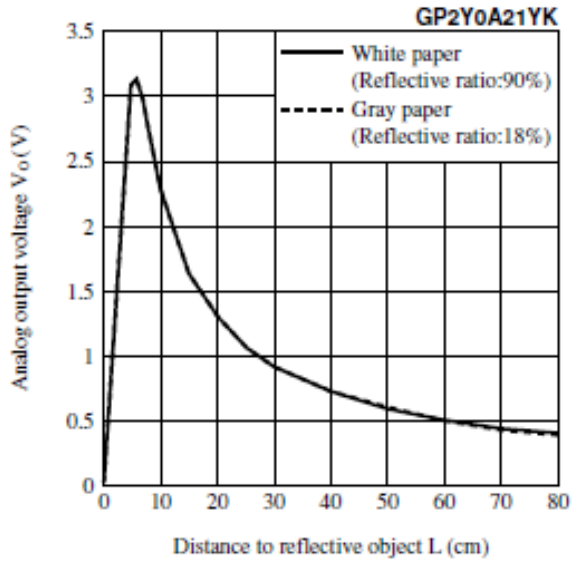


Figura 4.7 Curva característica del sensor infrarrojo GP2Y0A21YK.

En la curva característica se puede ver como la tensión de salida máxima del sensor que se va a medir es de 3,1V, con lo cual, para conseguir una mayor resolución, se configurará la referencia de tensión del Arduino a 3,3V mediante la instrucción "*analogReference(EXTERNAL);*" y se conectará el pin AREF con la salida de 3,3V del Arduino.

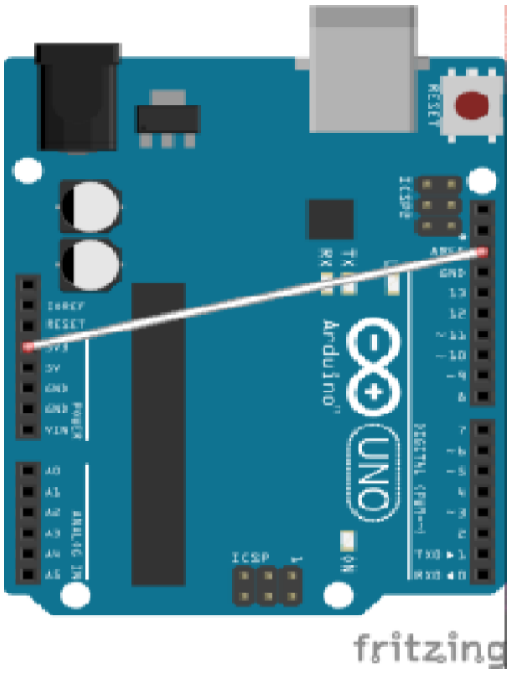


Figura 4.8 Fig. 11. Conexión de la referencia analógica a la salida de 3,3V.

De esta forma, los 1024 puntos (de 0 a 1023) que proporciona el convertidor analógico digital de 10 bits del Arduino tendrán un fondo de escala de 3,3V en lugar de los 5V por defecto.

Resolución con fondo de escala de 5V (por defecto) = $\frac{5V}{1023\ lectura\ max.}$ = 4,888 mV/bit

Resolución con fondo de escala de 3,3V = $\frac{3{,}3V}{1023\ lectura\ max.}$ = 3,226 mV/bit

Gracias a esto como se puede ver, se obtiene una mayor resolución, es decir, una mayor precisión en las medidas del sensor (porque se reducen las posibilidades de error en las medidas), puesto que el tramo de tensión que aumenta entre cada lectura es más pequeño.

A la hora de realizar la conexión hardware del sensor infrarrojo con el Arduino, cabe destacar que el sensor infrarrojo tiene 3 cables que se conectan a: uno a GND (cable negro), otro a alimentación, 5V (cable rojo), y por último a un pin analógico del Arduino (cable amarillo), por el cual se transmitirá la señal de salida del sensor

infrarrojo (voltaje analógico), en este caso, el sensor está conectado al pin analógico 15 de Arduino.

Para tener una señal de salida más precisa y repetitiva, se acondiciona la señal del sensor mediante un filtro paso bajo, para filtrar los posibles ruidos que pueda tener la señal de salida del sensor. Para llevar a cabo el filtrado se conectará un condensador electrolítico de 10µF entre la salida del sensor y masa.

En definitiva, el objetivo del sensor infrarrojo en el sistema, es el de medir la posición de la bola en todo momento, esto es necesario, para controlar la posición de la bola, es decir, si la bola está muy alejada del centro (tomando como el centro el valor de la consigna) el controlador PID programado manipulará el duty cycle de la señal PWM que controla el ángulo de giro del servomotor aumentándolo (en el caso de que el valor de la consigna fuese superior al valor medido por el sensor), con lo cual, al aumentar el duty cycle de la señal PWM el servomotor incrementará su ángulo de giro, mediante está acción, la barra horizontal se inclinará por medio de la biela que las une, y así de manera repetitiva, hasta posicionar de nuevo la bola en el centro de la barra o lo que es lo mismo hasta que el error sea nulo.

4.4 Programación software mediante Arduino para el control del sistema

A continuación, se va a mostrar cómo se han programado y controlado cada una de las partes que conforman el proyecto, a través de la programación software de Arduino.

4.4.1 Control del servomotor

Como se ha visto en el apartado *3.5.1 Como se va a controlar el servomotor*, se va a configurar los registros de uno de los timers del Arduino para controlar el ángulo de

giro del servomotor, para ello se va a generar una señal PWM en uno de los pines del Arduino.

Como se ha mencionado en el apartado *3.5.4 Timers en Arduino* el Arduino MEGA ADK cuyo microcontrolador es el ATmega2560 consta de 6 timers de los cuales 4 son de 16 bits, con lo cual, se va a utilizar uno de ellos para este cometido.

Se va a utilizar el timer 5 del Arduino MEGA ADK para generar la señal PWM que va a controlar el ángulo de giro del servomotor, debido a esto el servomotor se debe conectar a uno de los pines PWM que controla el timer 5. Hay que decir, que el timer 5 es un timer de 16 bits, lo cual es muy importante, puesto que tiene una mayor resolución que los timers de 8 bits, con lo cual, un conteo y una precisión más elevada. Como se puede ver en el pin mapping del Atmega2560, el timer 5 controla los siguientes pines PWM del Arduino MEGA ADK:

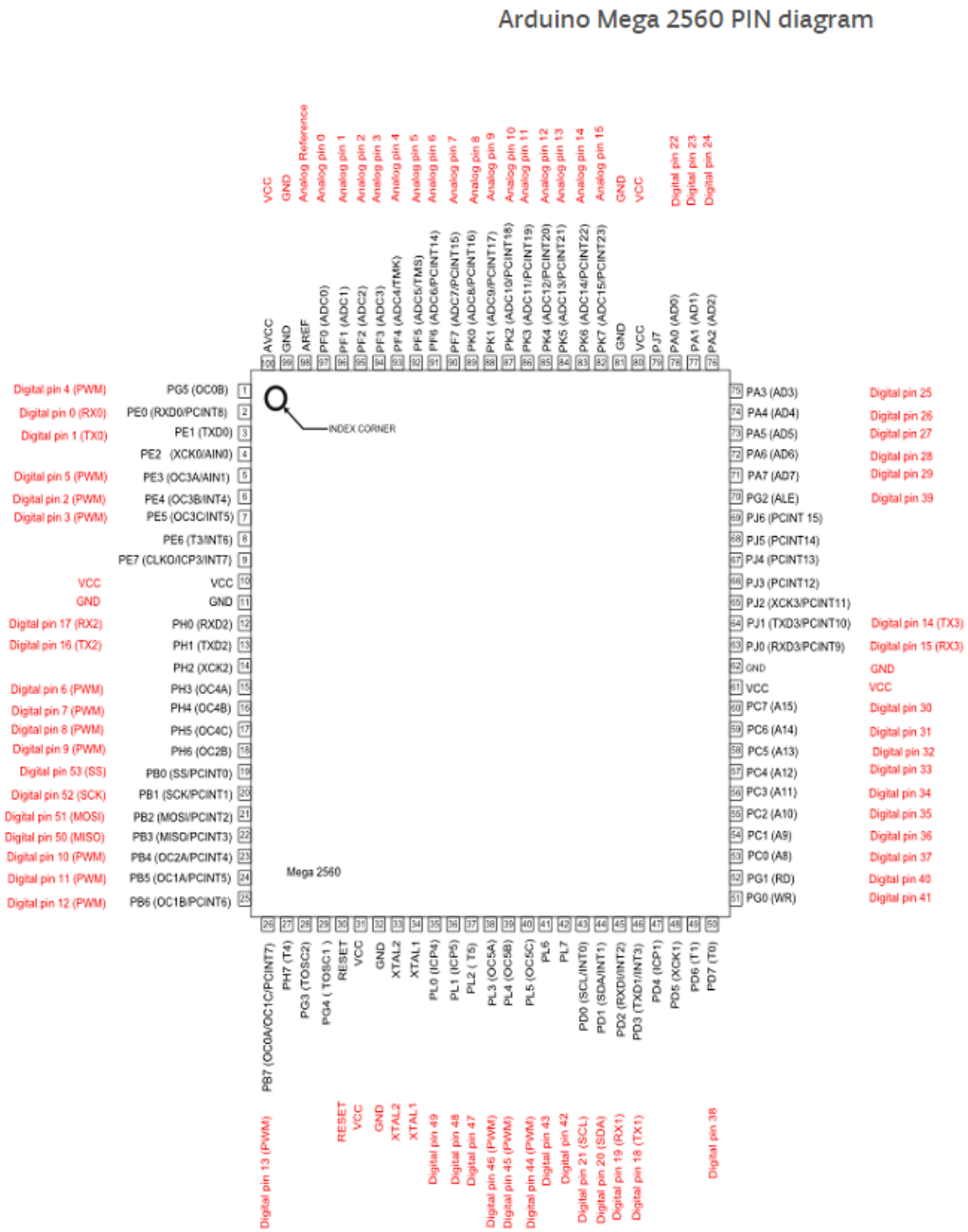


Figura 4.9 Diagrama de pines del microcontrolador ATmega2560 del Arduino MEGA ADK.

- El pin PWM 44 (OC5C)
- El pin PWM 45 (OC5B)
- El pin PWM 46 (OC5A)

Entonces como se va a utilizar el timer 5 para generar la señal PWM que va a controlar el ángulo de giro del servomotor, el servo va estar conectado al pin digital de salida 46 del Arduino Mega, que se declara mediante esta instrucción en la programación software de Arduino:

pinMode(46, OUTPUT); // Se configura el pin 46 (servo) como un pin de salida digital.

El registro del timer 5 OC5A es el que está asociado al pin PWM 46 del Arduino MEGA ADK.

4.4.1.1 Configuración de los registros del timer 5 (pin 46 – OC5A) para generar la señal PWM que controla el ángulo de giro del servo.

Teniendo en cuenta lo explicado en el apartado *3.5.4 Timers en Arduino*, se van a configurar los registros del timer 5 (pin 46 – OC5A) para generar la señal PWM que controla el ángulo de giro del servo.

Lo primero es seleccionar el modo de generación de forma de onda más adecuado, para lo cual, se sabe que es necesario una señal PWM, en la cual, se pueda variar su duty cycle, para poder modificar el ángulo de giro del servomotor. Entonces partiendo de esa base es necesario establecer un valor fijo de TOP, para lo cual, es necesario utilizar el registro de captura de entrada ICR5 para definir el valor TOP, y así dejar libre el registro OCR5A para poder utilizarlo como registro que establece el duty cycle de la señal PWM. En el registro ICR5 que define el valor TOP, se establece el periodo de la señal PWM, que en este caso va a ser de 20ms debido a que el servomotor tiene un periodo de 20ms por fábrica.

Sabiendo esto, ya se sabe que el modo de operación a seleccionar tiene que ser un modo PWM y que defina el valor TOP mediante el registro ICR5.

Como los modos de operación “PWM phase correct” y “PWM phase and frequency correct” están basados en una operación de doble pendiente, la cual proporciona una frecuencia de operación máxima más baja en comparación con la operación de pendiente única, debido a esto y a la característica simétrica de los modos PWM de doble pendiente, estos modos se suelen utilizar en aplicaciones de control de

motores, que este es el caso, con lo cual uno de estos 2 modos de operación hay que seleccionar para controlar el ángulo de giro del servomotor.

El modo de operación seleccionado es el modo "PWM phase and frequency correct" debido a que el modo "PWM phase correct" tiene alguna desventaja respecto al "PWM phase and frequency correct" a pesar de ser prácticamente iguales.

Dicho todo esto, el modo de generación de forma de onda seleccionado es WGM53:0 = 8, es decir, es un modo "PWM phase and frequency correct" que utiliza el registro ICR5 para definir un valor fijo de TOP y que actualiza el registro de comparación de salida OCR5A en BOTTOM (cuando el contador está a 0), además establece a 1 el flag de desbordamiento del timer TOV5 en BOTTOM también:

Table 17-2. Waveform Generation Mode Bit Description(1)

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Figura 4.10 Modo de generación de forma de onda seleccionado para generar la señal PWM que controla el ángulo de giro del servomotor.

Una vez conocido el modo de generación de forma de onda seleccionado, se aplica la fórmula que establece el valor TOP para la frecuencia o periodo de la señal PWM que se quiere generar, en este caso, el periodo de la señal PWM va a ser de 20ms debido a que el servomotor tiene un periodo de trabajo de 20ms por fábrica:

Nota: Para obtener el valor TOP hay que probar con diferentes valores del prescaler, lo más adecuado es que cuanto mayor sea el TOP mejor, puesto que se obtiene una mayor resolución, con lo cual se va a empezar probando con el valor N=1 del prescaler para obtener el valor TOP que genere una señal PWM con un periodo de 20ms.

Esta es la fórmula a aplicar:

$$f_{OC5APFCPWM} = \frac{f_{clk\_I/O}}{2*N*TOP}$$

Donde:

- $f_{clk\_I/O}$: Es la frecuencia de reloj del micro de Arduino → 16MHz
- N: Representa el factor del prescaler del micro (1, 8, 64, 256 o 1024).
- $f_{OC5APFCPWM}$: Es la frecuencia de la señal PWM que se quiere generar utilizando el modo de operación "PWM phase and frequency correct", en este caso, como se ha dicho antes, la señal PWM que se va a generar va a tener un periodo de 20ms, es decir, una frecuencia de 50Hz.

- Probando con N = 1:

$$f_{OC5APFCPWM} = 50Hz = \frac{16*10^6 Hz}{2*1*TOP} \rightarrow TOP = 160000$$

Este valor no coge, como son registros de 16 bits, el valor máximo posible es de 2^{16}-1 que es igual a 65535.

- Probando con N = 8:

$$f_{OC5APFCPWM} = 50Hz = \frac{16*10^6 Hz}{2*8*TOP} \rightarrow TOP = 20000$$

Este valor si coge, con lo cual, el prescaler del micro se debe configurar a 8.

Entonces, por el modo de generación de forma de onda seleccionado, el valor TOP se establece mediante el registro de captura de entrada ICR5, con lo cual, ICR5 = 20000 = TOP con este valor se establece un periodo de 20ms a la señal PWM generada.

Se puede decir que el valor de 20000 es 20000µs, puesto que da la casualidad que el periodo de trabajo del servo es de 20ms (20ms = 20000µs).

Por lo tanto, como se puede ver se debe configurar el prescaler del reloj del micro a 8 mediante los bits CS52:0 ubicados en el registro de control B del timer (TCCR5B):

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figura 4.11 Bits de selección de reloj, para que el prescaler del reloj del micro este configurado a 8.

Los 3 bits de selección de reloj (CS52:0) seleccionan la fuente de reloj que utilizará el timer 5. En este caso como se ha visto en la fórmula anterior, el prescaler del reloj del micro se configura a 8, es decir, la fuente de reloj del micro entre 8: 16MHz/8.

A continuación se va a obtener la resolución utilizada en bits para generar el periodo de la señal PWM, dicho de otra manera, son los bits que va utilizar el timer 5 para alcanzar el valor máximo TOP configurado (20000), se calcula empleando la siguiente ecuación:

$$R_{PFCPWM} = \frac{\log(TOP+1)}{\log(2)}$$

Sustituyendo el valor calculado en la anterior fórmula, TOP = 20000, la resolución utilizada es la siguiente:

$$R_{PFCPWM} = \frac{\log(20000+1)}{\log(2)} \rightarrow R_{PFCPWM} = 14{,}288 \approx 15 \text{ bits}$$

La resolución es de 15 bits, el timer 5 va a utilizar 15 bits para alcanzar el valor máximo de conteo (TOP = 20000 = ICR5, que son 20ms de periodo de la señal), es decir, 32767 puntos, muchos más que si se habría programado el servomotor utilizando la función por defecto de Arduino *analogWrite()* que utiliza solamente 8 bits, 255 puntos; también mejor que utilizando la librería especifica del servomotor.

Se sabe que cuanta mayor resolución mejor, ya que el control del servomotor se realizará con mayor precisión y exactitud.

Ahora se va a configurar el modo de comparación de salida mediante los bits COM5A1:0 ubicados en el registro de control A del timer, TCCR5A. Como el modo de operación seleccionado es el modo "PWM phase and frequency correct" se debe ir a la tabla de configuración del modo de comparación de salida del "PWM phase and frequency correct":

Table 17-5 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct and frequency correct PWM mode.

Table 17-5. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 =9 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting Set OCnA/OCnB/OCnC on compare match when downcounting
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting Clear OCnA/OCnB/OCnC on compare match when downcounting

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1//COMnC1 is set. See "Phase Correct PWM Mode" on page 148. for more details.

Figura 4.12 **Modo de comparación de salida seleccionado.**

Se ha seleccionado el modo de comparación de salida no invertido. Gracias a este modo de comparación de salida seleccionado, COM5A1:0 = 2 (el no invertido), y debido a que el modo de operación configurado ("PWM phase and frequency correct") está basado en una operación de doble pendiente, el pin de comparación de salida OC5A (pin digital 46 en Arduino) se pone a nivel bajo cuando el registro contador TCNT5 alcanza el valor del registro de comparación de salida OCR5A establecido mientras el contador realiza el conteo ascendente, y se pone a nivel alto cuando el registro contador TCNT5 alcanza el valor establecido en OCR5A mientras el contador realiza el conteo descendente. Dicho de otra manera, cuando el registro contador TCNT5 realiza el conteo ascendente y alcanza e iguala el valor establecido en el registro OCR5A, la señal de salida del pin OC5A (pin 46 de Arduino) se sitúa a nivel bajo, en cambio, cuando el registro contador TCNT5 lleva a cabo el conteo descendente y alcanza e iguala el valor de OCR5A, la señal de salida del pin OC5A se sitúa a nivel alto.

Ejemplo de cómo se genera la señal PWM con los bits configurados de la forma que se ha explicado anteriormente:

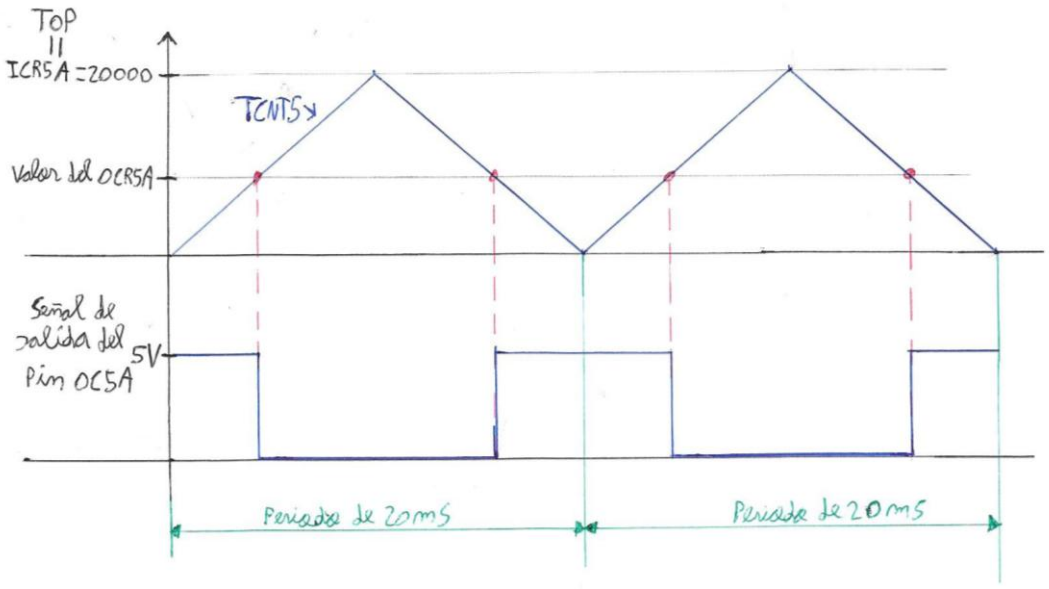


Figura 4.13 Diagrama de tiempos, con los bits configurados para generar una señal PWM de 20ms de periodo.

El periodo de la señal PWM comienza y finaliza cuando el valor del registro contador se pone a 0 (TCNT5 = 0, es decir en BOTTOM), como se puede ver en el diagrama de tiempos mostrado arriba. Cabe destacar que el valor máximo que puede alcanzar el registro contador del timer TCNT5 es el definido en el registro de captura de entrada ICR5 = 20000, puesto que es el registro que establece el valor TOP. Como se puede ver en el diagrama, una vez que el registro contador TCNT5 alcanza el valor del registro de captura de entrada ICR5, inicia la secuencia de conteo descendente hasta llegar a 0. También se puede ver como se modifica la señal de salida del pin OC5A, cuando el registro contador TCNT5 iguala el valor establecido en el registro de comparación de salida OCR5A, teniendo en cuenta el modo de comparación de salida configurado.

Una vez seleccionado el modo de generación de forma de onda mediante los bits WGM53:0, el prescaler del reloj del micro mediante los bits CS52:0 y el modo de comparación de salida mediante los bits COM5A1:0, ahora se implementan los registros en los cuales están ubicados los bits anteriormente mencionados:

TCCR5A - Registro de control A del timer/counter 5

Bit	7	6	5	4	3	2	1	0
	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
	1	0	0	0	0	0	0	0

Como solo se utiliza el pin de comparación de salida OC5A (pin 46), los modos de comparación de salida de los pines de comparación de salida OC5B (pin 45) y OC5C (pin 44) se van a configurar para que funcionen de manera normal, es decir, se van a establecer a 0, para que cualquier componente se pueda conectar a ellos y funcionen de manera normal.

TCCR5B - Registro de control B del timer/counter 5

Bit	7	6	5	4	3	2	1	0
	ICNC5	ICES5	-	WGM53	WGM52	CS52	CS51	CS50
	0	0	0	1	0	0	1	0

Como el registro de captura de entrada ICR5 se utiliza para definir el valor TOP, el bit de selección de flanco de captura de entrada, ICES5 se deshabilita, es decir, se pone a 0.

Estos registros son los que se implementan en la programación software de Arduino de la siguiente manera, se implementan en la parte de la configuración *void setup()*:

//Se emplea el timer 5 para la generación de la PWM que va a controlar el ángulo de giro del servo (pin de salida 46 - OC5A).

pinMode(46, OUTPUT); // Se configura el pin PWM 46 (servo) como un pin de salida digital.

TCCR5A = B10000000; //Modo de comparación de salida no invertido en OC5A (pin 46) y modo de generación de forma de onda “PWM phase and frequency correct” estableciendo el valor TOP con el registro ICR5.

TCCR5B = B00010010; //Prescaler del reloj del micro a 8 y modo de generación de forma de onda "PWM phase and frequency correct" estableciendo el valor TOP con el registro ICR5.

ICR5 = 20000; // Establece el valor TOP de la señal PWM, es decir, mediante este registro se establece el periodo de la señal PWM, que va ser de 20ms (periodo de trabajo del servo).

OCR5A = reposo; // Para que empiece el control del sistema con la barra horizontal (ángulo de giro del servo a 90º, es decir, reposo = 1400). El registro OCR5A establece el valor del duty cycle de la señal PWM.

En definitiva, implementando estos registros en la programación software de Arduino, lo que se consigue es generar una señal PWM para poder controlar el ángulo de giro del servomotor, por medio de la modificación del duty cycle de la señal PWM, o lo que es lo mismo, por medio de la modificación de la tensión aplicada al servo; con esto se consigue posicionar el servomotor en cualquier posición angular dentro de su rango de movilidad de 180º.

4.4.1.2 Comprobación de la configuración de los registros del timer 5, rango de trabajo del servomotor y valor que sitúa la barra en horizontal, para el correcto funcionamiento del control del servo.

Se ha utilizado el siguiente programa Arduino para comprobar la configuración de los registros del timer 5 anteriormente expuestos, y para obtener tanto el rango de trabajo del servomotor, como el valor del registro de comparación de salida OCR5A que sitúa la barra horizontal (en reposo):

// Librería para poder programar pantallas LCDs:

#include <LiquidCrystal.h>

// Inicializar la biblioteca con los números de los pines de interfaz, es decir, los pines que se conectan al arduino.

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup() {

lcd.begin(16, 2); // Inicializa la interface de la LCD y configura el número de columnas y filas de la pantalla LCD.

lcd.print("LECTURA SERIE"); // Imprime un mensaje en la pantalla LCD.

Serial.begin(19200); // Se inicializa la interface de la comunicación serie.

pinMode(46, OUTPUT); // Se configura el pin 46 (servo) como un pin de salida digital.

// Se emplea el timer 5 para la generación de la señal PWM que va a controlar el ángulo de giro del servo (pin de salida 46).

TCCR5A = B10000000; // Modo de comparación de salida no invertido en OC5A (pin 46) y modo de generación de forma de onda "PWM phase and frequency correct" estableciendo el valor TOP con el registro ICR5.

TCCR5B = B00010010; // Prescaler del reloj del micro a 8 y modo de generación de forma de onda "PWM phase and frequency correct" estableciendo el valor TOP con el registro ICR5.

ICR5 = 20000; // Establece el valor TOP de la señal PWM, es decir, mediante este registro se establece el periodo de la señal PWM, que va ser de 20ms (periodo de trabajo del servo).

OCR5A = 800; //Registro del timer 5 que es el que va establecer el tiempo que va estar arriba la señal PWM del servo, dependiendo si es mayor o menor el servo girará más grados o menos. En este caso, valor inicial por defecto de 800 que sería 800microseg. = 0,8ms.

}

char cadena[24];

byte contador=0;

```
int valor=0;
```

// Este bucle se utiliza para poder escribir en la comunicación serie y almacenar el valor escrito en una variable, para poder asignar ese valor escrito al registro de comparación de salida OCR5A, que es el valor que establece el duty cycle de la señal PWM, es decir, cuanto mayor sea el valor del duty cycle más grados girará el servomotor.

// También se imprime en la pantalla LCD el valor escrito en la comunicación serie.

```
void loop(){
  if(Serial.available()){
    memset(cadena, 0, sizeof(cadena));
    while (Serial.available()>0){
      delay(5);
      cadena[contador]=Serial.read();
      contador++;
    }
    valor=atoi(cadena);
    lcd.setCursor(0, 1); // Establece el cursor en la columna 0, fila 1 (nota: la fila 1 es la segunda fila).
    lcd.print("          "); // Para que no se queden números fantasma en la pantalla.
    lcd.setCursor(0, 1); // Establece el cursor en la columna 0, fila 1 (nota: la fila 1 es la segunda fila).
    lcd.print(valor); // Imprime el "valor" escrito en el monitor serie.
    OCR5A = valor; // El registro OCR5A es igual al "valor" escrito en el monitor serie.
    contador = 0; // Se pone el contador a 0.
  }
}
```

Para realizar la comprobación de la configuración de los registros del timer 5, y para obtener tanto el rango de trabajo del servo, como el valor en el cual se sitúa la barra en reposo (horizontal), para ello se ha utilizado el servomotor, la comunicación serie de Arduino y la pantalla LCD, con la aplicación de esta programación.

Este programa se basa en controlar el ángulo de giro del servomotor mediante la comunicación serie, es decir, se ha programado para que se escriba un valor en el monitor serie, el cual va a ser introducido en el registro de comparación de salida OCR5A, y enviado mediante la señal PWM generada en el pin 46 al servo. Dependiendo del valor que se introduzca en el monitor serie, es decir, en el registro OCR5A, la señal PWM estará a nivel alto más tiempo o menos tiempo, o lo que es lo mismo el duty cycle será mayor o menor, con lo cual, dependiendo del tiempo que este la señal PWM a nivel alto se le aplicará más o menos tensión al servo, provocando que gire más grados o menos grados. La pantalla LCD solo se utiliza para mostrar los valores introducidos.

Como se puede ver en el programa se han configurado los registros del timer 5 tal y como se han expuesto en el apartado anterior *4.3.1.1*, para generar una señal PWM con un periodo de 20ms, la cual, es luego la que se utiliza para controlar el ángulo de giro del servomotor en la maqueta.

Se comprueba si la configuración de los registros del timer 5 es la idónea para generar una señal PWM en el pin PWM 46 de Arduino, utilizando un osciloscopio. Con este osciloscopio se muestra la señal que se ha generado en el pin PWM 46 del Arduino:

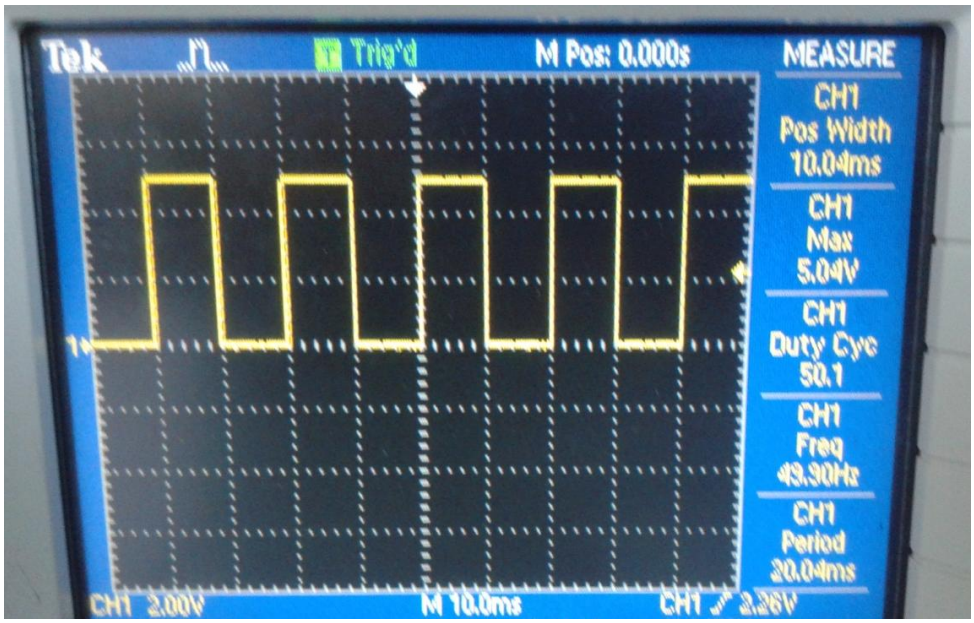


Figura 4.14 Señal PWM generada con la configuración de los registros del timer 5 anteriormente expuestos, en el pin PWM 46 del Arduino.

Como se puede ver mediante el osciloscopio, con esta configuración de los registros del timer 5 se ha generado una señal PWM con un periodo de 20ms en el pin de comparación de salida OC5A, es decir, en el pin PWM 46 de Arduino, justo lo necesario para poder controlar el ángulo de giro del servomotor, con lo cual, se puede decir que está configuración de los registros del timer 5 es la correcta. La señal PWM generada en la figura anterior tiene un duty cycle del 50%, es decir, el registro de comparación de salida OCR5A = 10000.

En las siguientes figuras se puede ver la misma señal PWM generada en el pin 46, pero con diferente duty cycle, es decir, con un valor del registro de comparación de salida OCR5A distinto:

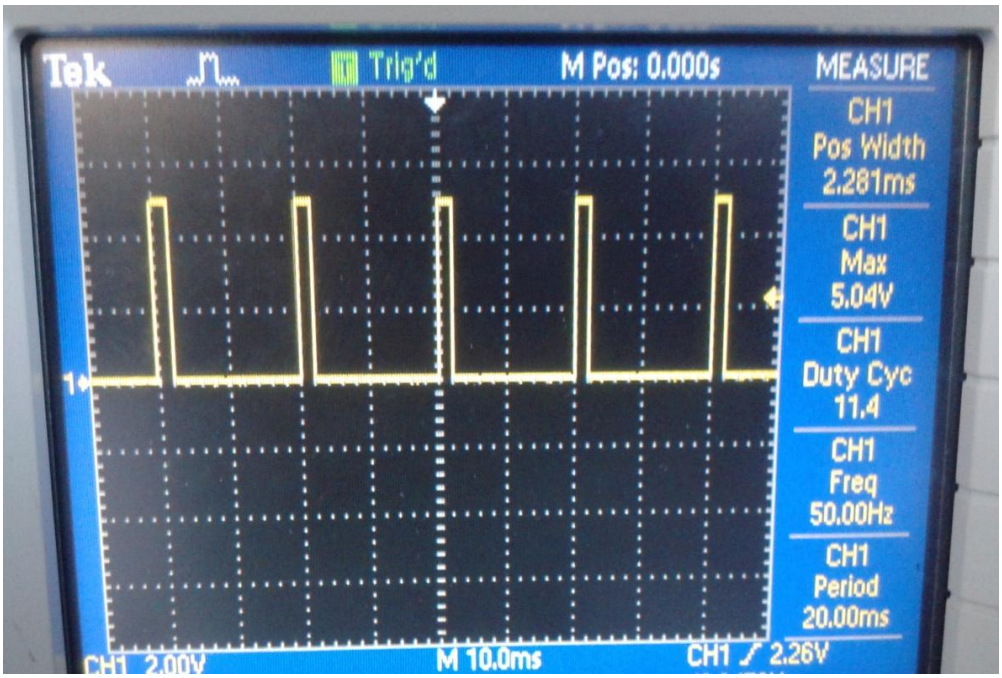


Figura 4.15 Señal PWM generada en el pin 46 de Arduino, con OCR5A = 2300.

Figura 4.16 Señal PWM generada en el pin 46 de Arduino, con OCR5A = 400.

Además de comprobar la configuración de los registros del timer 5, se ha obtenido el rango de trabajo del servomotor Futaba S3003, para ello se han realizado varios ensayos con diferentes valores del registro de comparación de salida OCR5A, hasta llegar a la

conclusión de que el rango de trabajo del servo utilizado es de 500 a 2300 (OCR5A), es decir, como se ha mostrado en la configuración de los registros del timer el valor de 20000 (TOP = ICR5 = 20000) se puede decir que es 20000µs, con lo cual el rango de trabajo del servo es de 500µs a 2300µs o lo que es lo mismo de 0,5ms a 2,3ms. Cabe destacar, que la señal PWM que se va a enviar al servo va a estar a nivel alto solo en el rango de 0,5ms a 2,3ms en un periodo de 20ms. Entonces cuando la señal PWM este a nivel alto entre 0,5ms y 2,3ms, en ese rango el servomotor se puede mover hacia cualquier posición angular dentro de su rango de movilidad, de 0 a 180 grados.

Se puede ver, que cuando se introduce un valor de 500 al registro de comparación de salida OCR5A (OCR5A = 500), el ángulo de giro del servo se sitúa a 0º (posición más alta de la biela), y cuando se establece un valor de 2300, el ángulo de giro del servo se sitúa a 180º (posición más baja de la biela), el cual es su límite de giro, puesto que el servomotor utilizado tiene un rango de movilidad de 180º.

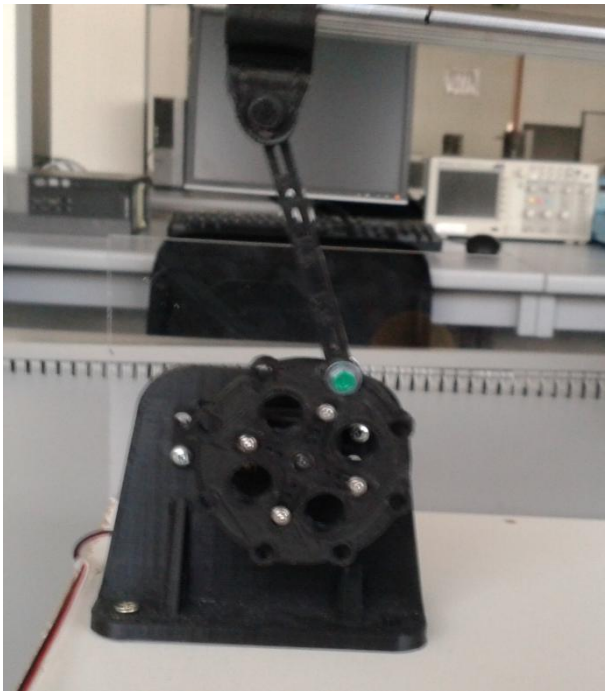


Figura 4.17 Ángulo de giro del servomotor a 0º, OCR5A = 500.

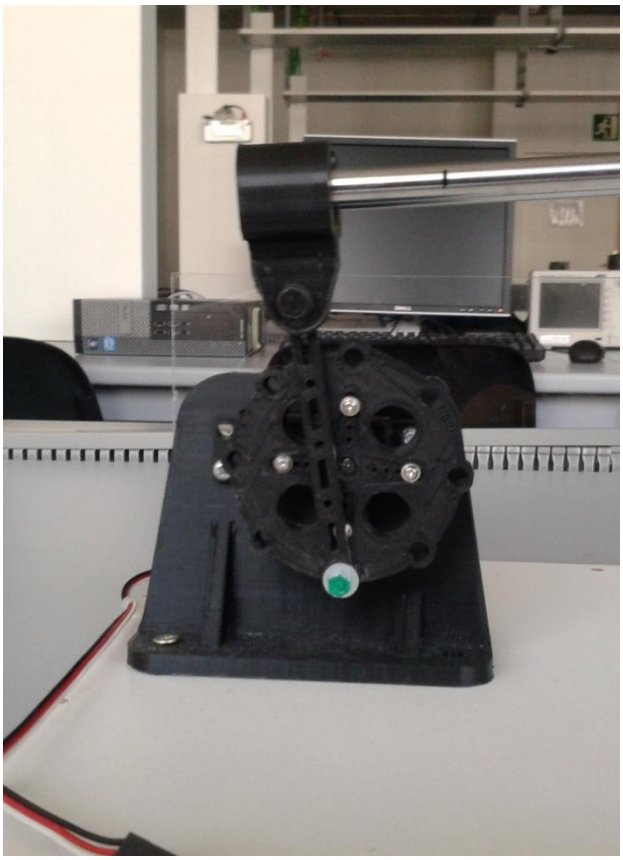


Figura 4.18 Ángulo de giro del servomotor a 180º, OCR5A = 2300.

El rango de trabajo del servomotor se utiliza en la programación software de Arduino, para que el duty cycle de la señal PWM este siempre dentro del rango de trabajo del servo, en otras palabras, este siempre dentro del rango de movilidad del servo entre 0 y 180º, esto se realiza para que el valor introducido al registro de comparación de salida OCR5A no esté fuera de los límites físicos del actuador:

DC = constrain(DC,500,2300); // Acota el duty cycle de la señal PWM que va controlar el ángulo de giro del servo, entre los valores en los cuales el servo trabaja, es decir, lo acota en el rango de trabajo del servo.

OCR5A = DC; // El registro de comparación de salida del timer 5 es igual al valor de duty cycle calculado.

Con este mismo programa también se ha obtenido el valor del registro de comparación de salida OCR5A que sitúa la barra en reposo (en horizontal), para ello, se han probado diferentes valores de OCR5A, se ha utilizado un nivel para poder visualizar cuando la

barra esta horizontal, y con las diferentes pruebas realizadas en la maqueta, se ha llegado a la conclusión que el valor de OCR5A que sitúa la barra en horizontal es OCR5A = 1400, con lo cual, cuando se aplica un valor de 1400 al registro de comparación de salida OCR5A el ángulo de giro del servo se sitúa a 90º, barra en reposo.

Figura 4.19 Ángulo de giro del servomotor a 90º, sitúa la barra en horizontal; OCR5A = 1400.

Este valor es necesario conocerlo para poder empezar el control de la posición de la bola con la barra en reposo a 90º, esto es, en horizontal, y así, si el error de posición es positivo incrementará el ángulo de giro del servo poniéndolo a un ángulo superior a 90º, y si el error de posición es negativo decrementará el ángulo de giro poniéndolo a un ángulo inferior a 90º, para que la barra se pueda inclinar a ambos lados, esto es necesario para poder realizar de manera correcta el control de posición de la bola.

En la programación software de Arduino se introduce de la siguiente manera el valor de OCR5A que sitúa la barra en horizontal:

float reposo = 1400; // Valor que mantiene la barra horizontal (en reposo).

OCR5A = reposo; // Para que empiece el control del sistema con la barra horizontal (ángulo de giro del servo a 90º). Esto es necesario para poder realizar el control de posición de una bola de manera eficaz.

En definitiva, la señal PWM que controla el ángulo de giro del servomotor trabaja de la siguiente manera, como se puede ver en la siguiente figura:

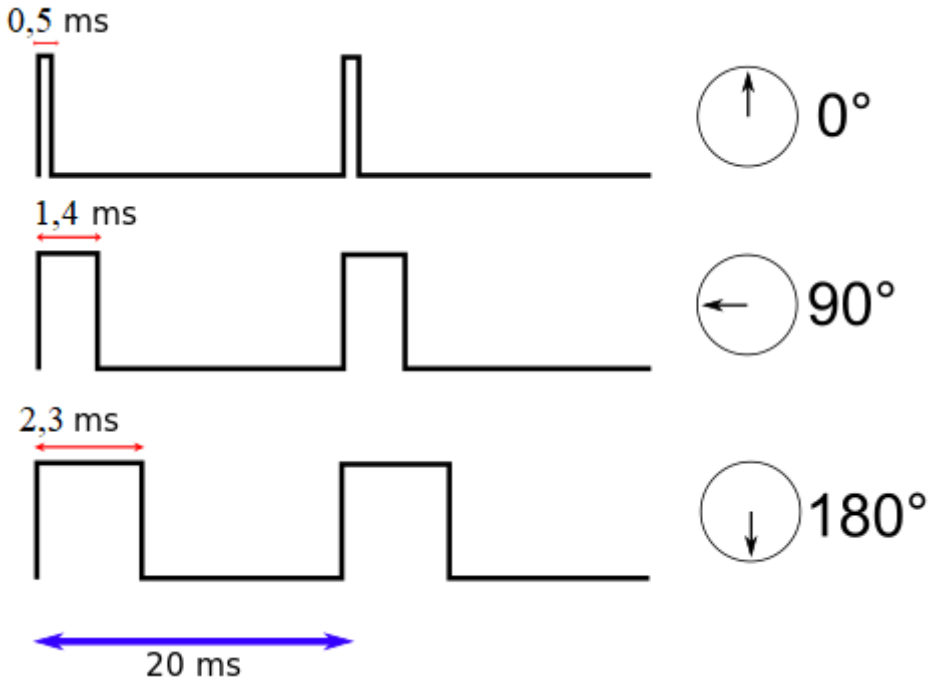


Figura 4.20 Señal PWM que controla el ángulo de giro del servomotor.

En este caso con este servomotor, cuando la señal está a nivel alto durante 0,5ms en el periodo de 20ms impuesto por el fabricante del servomotor, el ángulo de giro del servo se sitúa a 0º, cuando está a nivel alto durante 1,4ms el ángulo de giro del servo se sitúa a 90º (barra en reposo), y por ultimo cuando está a nivel alto durante 2,3ms el ángulo de giro del servo se sitúa a 180º que es el máximo que puede girar el servomotor utilizado.

4.4.2 Control y diferentes métodos de linealización del sensor.

Como se sabe y como se ha visto en la curva característica del sensor infrarrojo utilizado, la respuesta real de los sensores nunca es del todo lineal por ello hay que

realizar la linealización del sensor, para ello se han ensayado 2 técnicas diferentes de linealización del sensor y se ha seleccionado la que más se aproxima a los valores reales, las cuales se van a ver a continuación.

Antes de pasar a ver los métodos de linealización del sensor, lo primero que se realiza es recoger un numero x de medidas, en este caso 15 medidas relacionando la lectura del sensor con la distancia real medida. La programación para obtener estas medidas, es exactamente igual que para otras entradas analógicas, en este caso cabe destacar que el pin analógico de entrada utilizado para realizar la lectura del sensor infrarrojo es el pin 15 (A15).

Para relacionar la lectura del sensor con la distancia real en mm, se ha ido desplazando la bola a lo largo de la barra tomando nota de las diferentes medidas, es decir, se ha ido cogiendo las lecturas del sensor para diferentes distancias (en mm) de la bola; dicho de otra manera, se ha ido poniendo la bola a diferentes distancias, y cogiendo la lectura del sensor en cada distancia, esto se realiza para obtener las lecturas del sensor a diferentes distancias, para luego poder realizar la curva de linealización.

En el programa desarrollado para este propósito la lectura del sensor se visualiza tanto en la comunicación serie (monitor serie) como en la pantalla LCD. Como se ha dicho anteriormente en el apartado *4.3 Sensor infrarrojo utilizado*, cabe destacar la importancia de la instrucción "*analogReference(EXTERNAL);*" que se pone en la parte de configuración de la programación (void setup), para referenciar la tensión del Arduino a 3,3V en vez de a los 5V por defecto, esto es debido, a que se sabe que la tensión de salida máxima del sensor va a ser de 3,1V (visto en su curva característica), con lo cual, para obtener una mayor precisión en la medida se a referenciado a 3,3V, es decir, para obtener una mayor resolución.

El programa desarrollado para este propósito es el siguiente:

```
void setup()
{
 // Inicializa la interfaz de la LCD y configura el número de columnas y filas de la pantalla LCD:
 lcd.begin(16,2);
```

```
  Serial.begin(9600); // Inicializo la interface de la comunicación serie.
  analogReference(EXTERNAL); // AREF conectado a 3.3V.
  delay(100);
}

void loop()
{
  lcd.setCursor(0,0); //Posiciona el cursor en la primera fila y en la primera columna.
  // Imprime un mensaje en la pantalla LCD:
  lcd.print("Lectura sensor:");
  //Para guardar la lectura del sensor infrarrojo en la variable "lectura".
  lectura = analogRead(A15);
  // Establece el cursor en la columna 6, fila 1
  // (nota: la fila 1 es la segunda fila):
  lcd.setCursor(6, 1);
  lcd.print("    ");  //Para que no se queden números fantasma en la pantalla.
  lcd.setCursor(6, 1);  // Establece el cursor en la columna 6, fila 1 (nota: la fila 1 es la segunda fila).
  lcd.print(lectura);  // Imprime la lectura del sensor en la pantalla LCD.
  Serial.print("Lectura = ");  //Imprime la palabra "Lectura" en el monitor serie.
  Serial.println(lectura);  //Imprime la lectura del sensor en el monitor serie.
  delay(1000);  // Se ha puesto un delay de 1 segundo para que pase un segundo entre lecturas.
}
```

Se han obtenido 15 medidas a lo largo de la barra, para conseguir la curva de linealización del sensor infrarrojo, teniendo en cuenta el rango de medida del sensor

utilizado (10 a 80cm), se ha empezado a obtener las muestras a partir de los 10cm, puesto que con valores inferiores la lectura del sensor puede ser errónea:

Tabla 4.1 Medidas obtenidas para la realización de la linealización del sensor.

Nº medidas	Distancia [mm]	Lectura del sensor
1	100	768
2	125	624
3	150	538
4	175	457
5	200	403
6	225	355
7	250	312
8	275	277
9	300	247
10	325	229
11	350	209
12	375	191
13	400	178
14	425	166
15	450	154

Estas medidas se van a situar en un Excel y se va a realizar la gráfica relacionando la distancia real medida en mm (eje “x”) con la lectura del sensor (eje “y”).

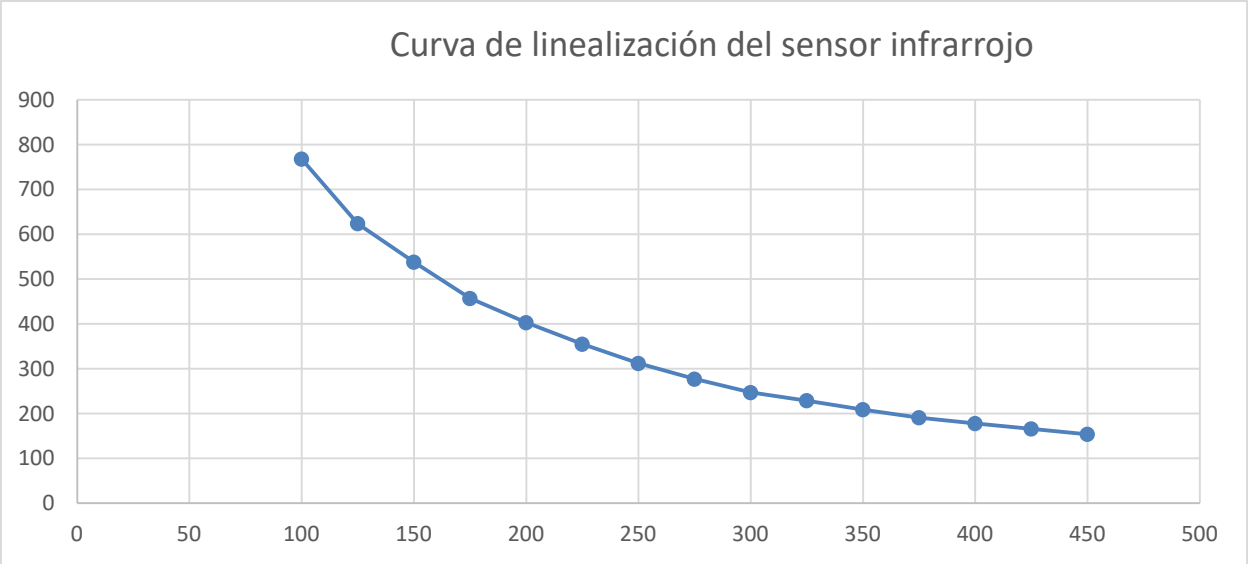


Figura 4.21 Curva de linealización del sensor infrarrojo.

Una vez se tiene la gráfica realizada, se puede ver como la gráfica es coherente con la curva característica del sensor, es decir, a mayor distancia medida menor lectura del sensor. Estas medidas en la programación Arduino quedan definidas como:

int dist_cal [] = {100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450}; // Las distancias medidas con sus correspondientes lecturas del sensor.

int lect_cal [] = {768, 624, 538, 457, 403, 355, 312, 277, 247, 229, 209, 191, 178, 166, 154}; // Estas son las medidas obtenidas para la linealización del sensor, Nº de muestras recogidas 15.

Todo esto se ha realizado para poder linealizar el sensor en la programación. Con estas medidas, ya se puede proceder a realizar los diferentes métodos de linealización del sensor infrarrojo.

4.4.2.1 Linealización del sensor tramo a tramo

Este método de linealización se ha realizado mediante la utilización de la función "map" de la programación. Como se ha visto anteriormente, las medidas obtenidas para realizar los diferentes métodos de linealización quedan definidas como 2 array, uno para las

lecturas del sensor y otro para las distancias reales medidas, ambas forman un array de 15 posiciones:

int dist_cal [] = {100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450}; // Las distancias medidas con sus correspondientes lecturas del sensor.

int lect_cal [] = {768, 624, 538, 457, 403, 355, 312, 277, 247, 229, 209, 191, 178, 166, 154};

Teniendo en cuenta esto, la programación de este método de linealización es el siguiente.

Lo primero que se realiza en la programación es obtener la lectura del sensor infrarrojo y almacenarla en una variable llamada "medida", ese valor se acota entre el mínimo y el máximo valor del array de las lecturas del sensor, para que este dentro de los valores adecuados para poder realizar la linealización:

medida = analogRead(sensorPin); // En la variable "medida" almacena el valor de la lectura del sensor infrarrojo.

medida = constrain(medida, lect_cal[14], lect_cal[0]); // Se acota la lectura medida por el sensor entre el mínimo y el máximo valor del array de "lectura calculada".

Una vez hecho esto, para poder transformar esa lectura del sensor obtenida, en una distancia en mm a almacenar en la variable "posicion" de manera lineal, se aplica la curva de linealización del sensor con el siguiente algoritmo:

for(int i =0; i<14; i++){ // Aplicamos curva de linealización de ADC a mm (esto se realiza para conocer en que tramo del array está situada la lectura del sensor).

if (medida <= lect_cal[i] && medida > lect_cal[i+1]){

posicion = map(medida,lect_cal[i],lect_cal[i+1],dist_cal[i],dist_cal[i+1]) ;

// Aquí se realiza la conversión de la lectura del sensor que está entre 2 medidas del array de lectura, a un valor que está entre las 2 mismas medidas, es decir, entre las 2 mismas muestras, pero del array de distancia. Así se obtiene el valor de posición de la bola.

posicion = constrain(posicion,dist_cal[0],dist_cal[14]); // Se acota la señal de posición de la bola entre el mínimo y el máximo valor del array de "distancia calculada", para que este dentro de los valores utilizados en la realización de la curva de linealización.

}

}

Como se ve en los comentarios de la programación, básicamente lo que sucede es que el sensor lee la posición de la bola, esa lectura se identifica entre que tramos o entre que posiciones del array de las lecturas está situada, y conociendo esto, se realiza la conversión de esa lectura a un valor de distancia en mm mediante la función “map”, es decir, convierte ese valor de lectura que está entre 2 posiciones o medidas del array de las lecturas, a un valor de distancia (posición de la bola en mm) que está entre las mismas posiciones del array pero del array de distancia. Por último, se acota la distancia en mm obtenida de manera lineal almacenada en la variable “posicion”, entre el mínimo y el máximo valor del array de las distancias para que este dentro de los valores utilizados en la realización de la curva de linealización.

Como se ha visto, mediante este método de linealización se obtiene la posición de la bola en mm por medio de la lectura del sensor aplicando la función “map” tramo a tramo.

Después de esto se aplica el mencionado filtro digital paso bajo en la programación, para evitar que haya medidas de posición incoherentes, para ello, se recogen en un array las ultimas 8 medidas de posición (se almacenan en un array llamado “filtro[]”) y se obtiene la media de posición con ellas, así se evitan tanto posibles medidas de posición incoherentes, como se mejora la precisión de la medida de posición de la bola, puesto que, una vez situadas las 8 medidas de posición en el array se saca el valor medio, y de esta forma las medidas muy pequeñas o muy grandes provocadas por el posible ruido de la señal de salida del sensor quedan fuera, se desprecian.

La programación de este filtro digital paso bajo se realiza con el siguiente algoritmo en cada ciclo de programa:

for (int i=0; i<n-1; i++){ // Se desplaza el array a la izquierda, eliminando la medida de posición más antigua.

filtro[i]=filtro[i+1]; // Esto es para que se vaya actualizando el array, se desplaza todo el array un lugar a la izquierda, dejando libre el ultimo hueco para poner la medidade posición nueva.

}

filtro[n-1] = posicion; // La medida de posición actual la guardo al final del array (filtro[7]).

posicion=0; // La variable de posición se establece a 0.

for (int i=0; i<n; i++){

posicion = posicion + filtro[i]; // Aquí se suman todas las medidas de posición que hay en el array que son 8.

}

posicion = posicion / n; // Se saca la media de las medidasde posición de la bola en mm.

Esto se realiza como se ha dicho, para hacer el efecto de un filtro paso bajo, para despreciar las posibles medidas de posición incoherentes debidas al ruido en la señal, y para mejorar la precisión de las medidas de posición.

Todo esto que se ha explicado, es el método de linealización del sensor tramo a tramo que se ha ensayado en la maqueta, el cual no es muy preciso, debido a que tiene un error entorno al centímetro en medidas de hasta 300mm, a partir de ahí, se incrementa el error al medir distancias mayores, pudiendo llegar a tener un error de 3 a 4cm.

En definitiva la medida de posición de la bola con este método de linealización, en medidas superiores a 300mm se aleja mucho del valor real.

4.4.2.2 Linealización del sensor por el método de regresión lineal

La regresión lineal es una fórmula que relaciona 2 variables, en la cual mediante la variable independiente (tensión de salida del sensor) se obtiene el valor de la variable dependiente (posición de la bola en mm).

Este método de linealización se realiza mediante una fórmula que se obtiene mediante la curva de linealización por medio de Excel, esta fórmula llamada regresión lineal relaciona la distancia real de un objeto en mm con la tensión de salida del sensor, entonces insertando esta regresión lineal en la programación se obtiene el valor de posición de la bola en mm en función de la tensión de salida del sensor.

Para obtener la regresión lineal de manera adecuada hay que saber que la variable dependiente que es la que se desea obtener, debe estar en el eje “y” de la gráfica y la variable independiente en el eje “x”. Con lo cual, en este caso, como lo que se desea obtener es la posición de la bola en mm en función de la tensión de salida conocida, lo que se hace es poner en el eje "x" las tensiones de salida del sensor y en el eje "y" las distancias de la bola en mm.

Antes de nada cabe destacar, que se intentó probar la regresión lineal obtenida por medio de la gráfica que relaciona la lectura del sensor con la distancia de la bola en mm, que es la curva de linealización utilizada en el anterior método de linealización, sin éxito, debido a que cuando se metía en la fórmula un valor de lectura del sensor conocido, no me daba el valor de distancia que me tenía que dar, es decir, no me daba los valores anteriormente medidos para la realización de la curva de linealización, con lo cual, se probó a realizar el método de linealización del sensor por regresión lineal relacionando la tensión de salida del sensor con la distancia en mm dándome en está ocasión valores correctos, con lo cual, es la que se ha utilizado en la programación.

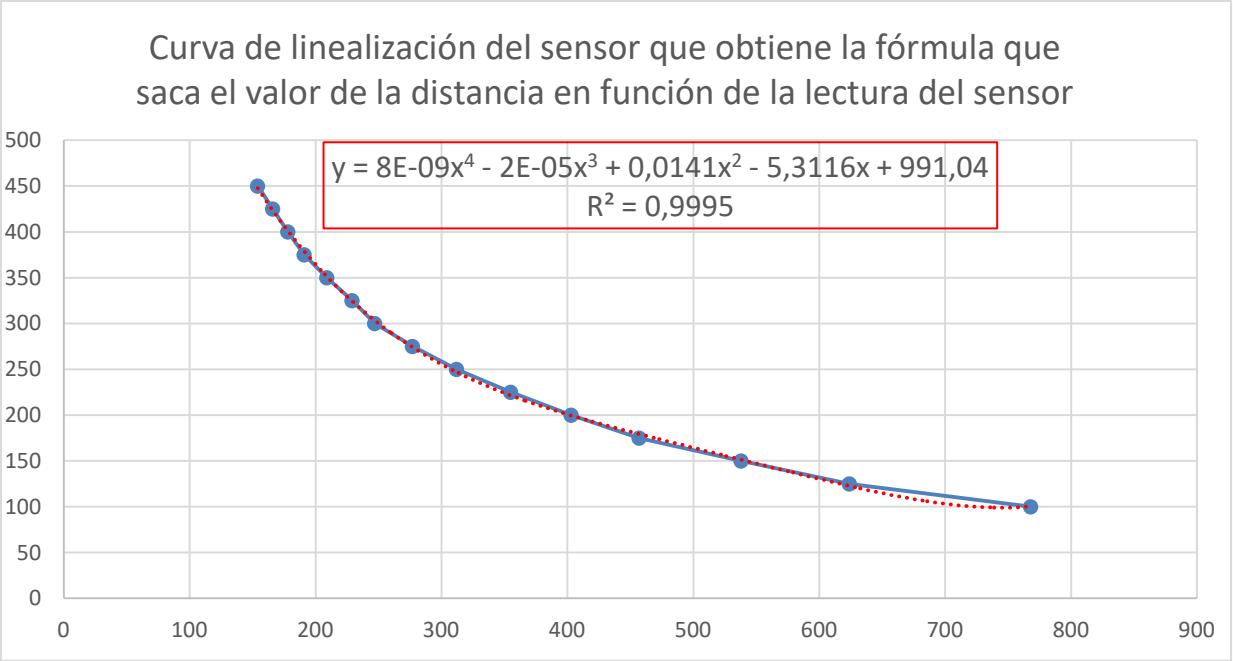


Figura 4.22 Curva de linealización del sensor que obtiene la regresión lineal que saca el valor de la distancia en función de la lectura del sensor sin éxito.

Dicho esto, para obtener la regresión lineal que relaciona la tensión de salida del sensor (que es la tensión que llega al Arduino y se transforma a un valor entre 0 y 1023, por medio del convertidor analógico digital de Arduino de 10bits) con la posición de la bola en mm, lo primero que se debe realizar es obtener la curva de linealización que relaciona ambas variables, para ello se parte de las medidas anteriormente expuestas y utilizadas en el anterior caso:

int dist_cal [] = {100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450}; // Las distancias medidas con sus correspondientes lecturas del sensor.

int lect_cal [] = {768, 624, 538, 457, 403, 355, 312, 277, 247, 229, 209, 191, 178, 166, 154};

Partiendo de estas medidas, se han ido calculando todas las tensiones de salida del sensor, por medio de las lecturas del sensor anteriormente medidas, para así relacionar la tensión de salida del sensor con la distancia en mm para poder realizar la curva de linealización en Excel.

Para pasar las lecturas a un valor de tensión de salida del sensor, lo que hay que saber es que, la tensión que llega procedente del sensor al Arduino se transforma a un valor entre 0 y 1023, por medio del convertidor analógico digital de Arduino de 10 bits, y como la tensión del Arduino está referenciada a 3,3V, por medio de la instrucción "*analogReference(EXTERNAL)*" de la programación, se sabe que cuando la tensión de salida del sensor sea de 3,3V, la lectura del sensor tendrá un valor de 1023, con lo cual haciendo una regla de tres se obtienen los valores de las tensiones de salida del sensor para cada una de las 15 medidas:

$$\text{Tensión de salida del sensor} = \text{lectura del sensor} * \frac{3{,}3V}{1023}$$

Aplicando está fórmula a todas las lecturas del sensor anteriormente medidas se obtiene la siguiente tabla:

Tabla 4.2 Medidas obtenidas relacionando la distancia con la tensión de salida del sensor.

Nº medidas	Distancia [en mm]	Voltaje [V]
1	100	2,477
2	125	2,013
3	150	1,735
4	175	1,474
5	200	1,3
6	225	1,145
7	250	1,006
8	275	0,893
9	300	0,797
10	325	0,739
11	350	0,674
12	375	0,616
13	400	0,574
14	425	0,535
15	450	0,497

Con esta tabla se ha realizado la curva de linealización del sensor que relaciona la distancia de la bola en mm (eje "y") con la tensión de salida del sensor (eje "x"):

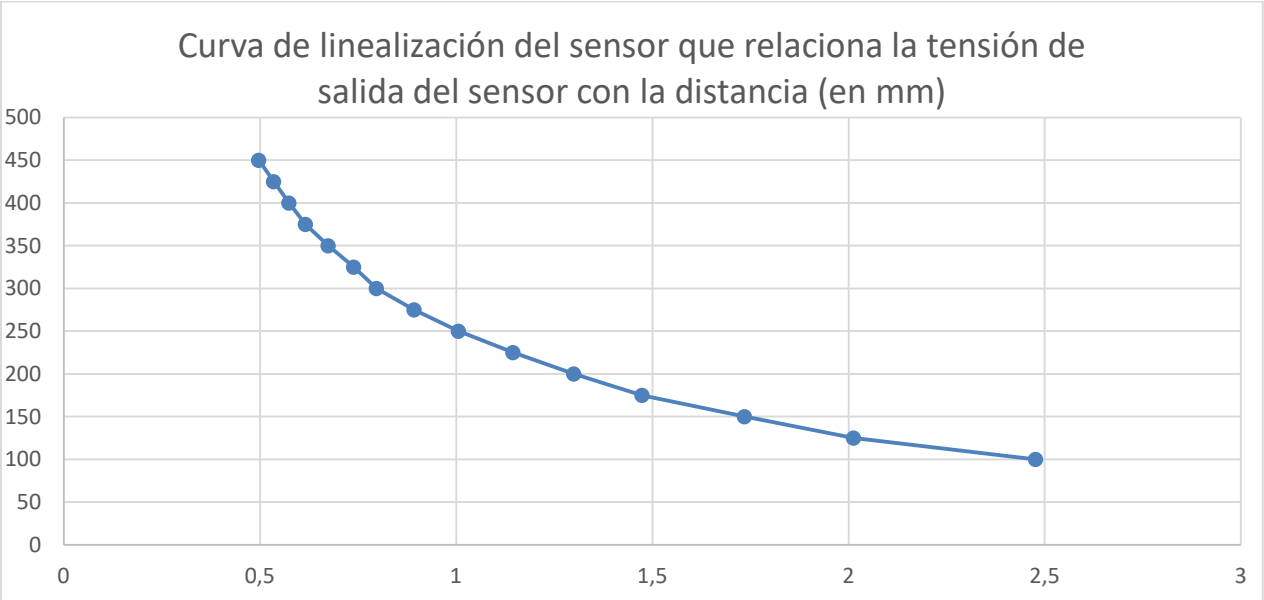


Figura 4.23 Curva de linealización del sensor que relaciona la tensión de salida del sensor con la distancia de la bola en mm.

Como se puede ver, la curva de linealización del sensor es coherente con la curva característica del sensor, es decir, a mayor distancia medida menor tensión de salida del sensor.

Una vez realizada la curva de linealización, se obtiene la regresión lineal, para ello, se ha agregado una línea de tendencia polinómica de orden 6 (de color rojo en la gráfica), para que la regresión lineal tenga la máxima precisión posible, puesto que, cuanto mayor orden tenga más precisa será la regresión lineal, es decir, para que el coeficiente de correlación R sea lo más cercano posible a 1, eso significará que los valores de distancia obtenidos mediante la regresión lineal serán unos valores muy próximos a los reales.

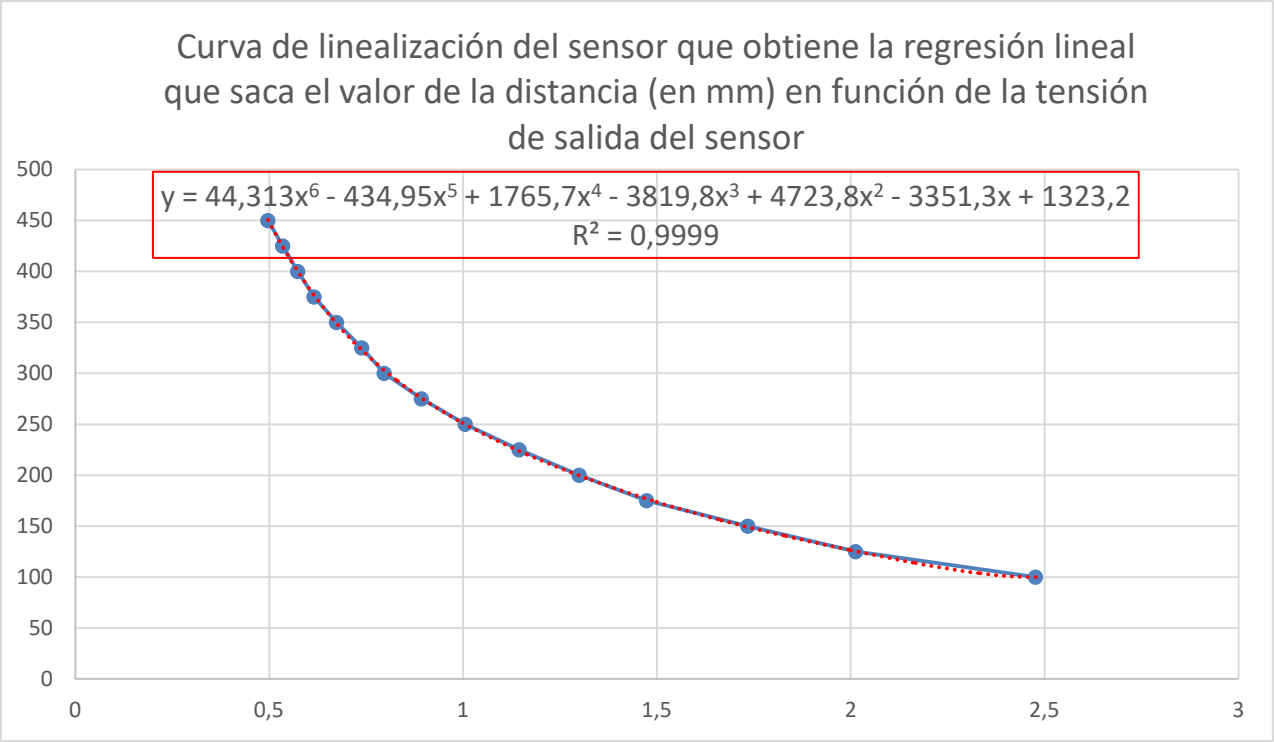


Figura 4.24 Curva de linealización del sensor que obtiene la regresión lineal que calcula el valor de la distancia en mm en función de la tensión de salida del sensor.

Una vez agregada la línea de tendencia (de color rojo en la gráfica) en las condiciones anteriormente mencionadas, se obtiene la regresión lineal que calcula el valor de la distancia (en mm) en función de la tensión de salida del sensor:

$$y = 44{,}313x^6 - 434{,}95x^5 + 1765{,}7x^4 - 3819{,}8x^3 + 4723{,}8x^2 - 3351{,}3x + 1323{,}2$$

$$R^2 = 0{,}9999$$

Está regresión lineal es la que se ha implementado en la programación software de Arduino, la cual relaciona la distancia real con la tensión de salida del sensor, es decir, con esta fórmula dependiendo del valor de la tensión de salida del sensor (obtenida en la programación al realizar una regla de tres con la lectura del sensor), va a calcular la posición de la bola real de manera lineal.

Una vez obtenida la regresión lineal, se realiza la programación de este método de linealización empezando por almacenar en una variable llamada "medida" la lectura del sensor:

medida = analogRead(sensorPin); // En la variable "medida" se almacena el valor de la lectura del sensor infrarrojo.

medida = constrain(medida, lect_cal[14], lect_cal[0]); //Se acota la lectura medida por el sensor entre el mínimo y el máximo valor del array de "lectura calculada".

Pero como lo que se necesita obtener es la tensión de salida del sensor para poder aplicar la regresión lineal, se emplea la regla de tres anteriormente mencionada, haciendo una regla de tres se obtiene el valor de la tensión de salida del sensor en función del valor de la lectura del sensor, y se almacena en una variable denominada "voltaje":

*voltaje = medida * (3.3 / 1023.0);* // Mediante la lectura del sensor (que va de 0 a 1023), se obtiene el valor de la tensión de salida del sensor (0 a 3.3 V).

Una vez hecho esto, se implementa la regresión lineal obtenida con la curva de linealización del sensor. Se introduce la regresión lineal en la programación, por medio de la siguiente instrucción:

*posicion = 44.313 * pow(voltaje, 6) - 434.95 * pow(voltaje,5) + 1765.7*pow(voltaje,4) - 3819.8 * pow(voltaje, 3) + 4723.8 * pow(voltaje, 2) - 3351.3 * voltaje + 1323.2;*

Entonces, con está regresión lineal se obtiene la posición de la bola en función del valor de la tensión de salida del sensor, y se guarda en la variable "posicion".

Por último, el valor obtenido de manera lineal de posición de la bola se acota entre el mínimo y el máximo valor del array de las distancias, para que este dentro de los valores utilizados en la realización de la curva de linealización, es decir, para que no allá ninguna medida de posición que este fuera de esos valores.

posicion = constrain(posicion,dist_cal[0],dist_cal[14]); // Se acota la señal de posición de la bola entre el mínimo y el máximo valor del array de "distancia calculada".

Una vez realizado todo esto, la programación es prácticamente igual que en el método de linealización anterior, puesto que se aplica la programación del filtro digital paso bajo para evitar que haya medidas de posición incoherentes debidas al posible ruido de la señal, y para mejorar la precisión de las medidas de posición.

La programación del filtro paso bajo digital es exactamente igual, que en el método de linealización del sensor tramo a tramo, visto en el apartado *4.4.2.1 Linealización del sensor tramo a tramo*.

Este método de linealización del sensor por regresión lineal se ha programado en Arduino como se ha visto, y se ha ensayado en la maqueta, en la cual se ha visto, que mide bastante bien la posición de la bola, tiene un error muy pequeño respecto al valor real, inferior a 1cm en medidas de hasta 30cm; a partir de ahí, sí que es verdad que cuando se miden distancias mayores a 30cm, el error aumenta pudiendo llegar a ser de 3cm (en el intervalo de 30 a 40cm). En medidas entre 40 a 45cm el error se reduce poniéndose en torno al centímetro.

En definitiva, una vez vistos y ensayados en la maqueta ambos métodos de linealización del sensor, se puede decir que la medida de la posición de la bola se aproxima más al valor real por medio de este método de linealización (linealización del sensor por regresión lineal), que por el método de tramo a tramo, sobre todo en medidas de distancia superiores a 30cm.

Con lo cual, debido a esto, el método de linealización del sensor utilizado en la programación software de Arduino para la medida de posición de una bola en el sistema que se estudia, va a ser, el método de linealización del sensor por regresión lineal, puesto que, como se ha dicho antes, es el método que más se aproxima a los valores reales, la diferencia entre el valor medido y el valor real es bastante pequeño.

4.4.3 Generación de la interrupción en Arduino

Como se ha explicado en el apartado *3.5.5 Interrupciones en Arduino*, se va a programar una interrupción por medio de los registros de los timers internos de Arduino para que cada cierto periodo de tiempo salte a la rutina de servicio de interrupción, para así poder medir y ejecutar la posición de la bola y la señal de control del controlador PID respectivamente, para ello, se va a emplear el tipo de interrupción de comparación igualada, la cual, es la mejor opción para gestionar los tiempos de muestreo.

4.4.3.1 Selección de periodo de muestreo y timer para generar una interrupción cada cierto periodo de tiempo.

El periodo de muestreo trata del tiempo que transcurre entre una muestra tomada de una señal y otra tomada de la misma señal. Este periodo de muestreo en principio se selecciona en base al tiempo que necesita el sensor para realizar la lectura de la señal.

Se han hecho diferentes pruebas con diferentes periodos de muestreo, pero finalmente el periodo de muestreo seleccionado para muestrear la señal de control del controlador PID en tiempo discreto y para medir la posición de la bola mediante el sensor es de 60ms, es decir, 60000 microsegundos en la programación Arduino:

int periodo = 60000; // Periodo de muestreo en microsegundos.

El periodo de muestreo seleccionado es de 60ms debido a que con tiempos de muestreo más pequeños no le daba tiempo al sensor a realizar la lectura, entonces se ha seleccionado un periodo de muestreo de 60ms para asegurarse que le da tiempo al sensor a realizar la lectura.

Como se ha mencionado anteriormente, se va a emplear el tipo de interrupción por comparación igualada de salida para generar una interrupción cada 60ms, es decir, cuando el registro contador TCNTx alcance el registro de comparación de salida OCRxy (registro que establece el periodo de muestreo) se va a saltar a la rutina de servicio de interrupción ISR, dejando la ejecución del programa detenida, además

cuando ambos registros se igualan aparte de saltar a la rutina de interrupción, el registro contador TCNTx se pone a 0 e inicia de nuevo la secuencia de conteo para volver a generar de nuevo la interrupción en el tiempo estipulado, en este caso, cada 60ms.

Para ello, se va a utilizar el timer 4 del Arduino MEGA ADK, esto es, configurando los registros del timer 4 lo que se pretende es generar una interrupción cada 60ms para poder ejecutar la señal de control del PID y obtener la medida de posición de la bola cada 60ms.

El timer 4 utilizado es un timer de 16 bits, lo cual es muy importante, puesto que tiene una mayor resolución que los timers de 8 bits, por consiguiente, un conteo y una precisión más elevada.

Como se puede ver en el pin mapping del microcontrolador del Arduino utilizado, el Atmega2560, el timer 4 controla los siguientes pines PWM del Arduino MEGA ADK:

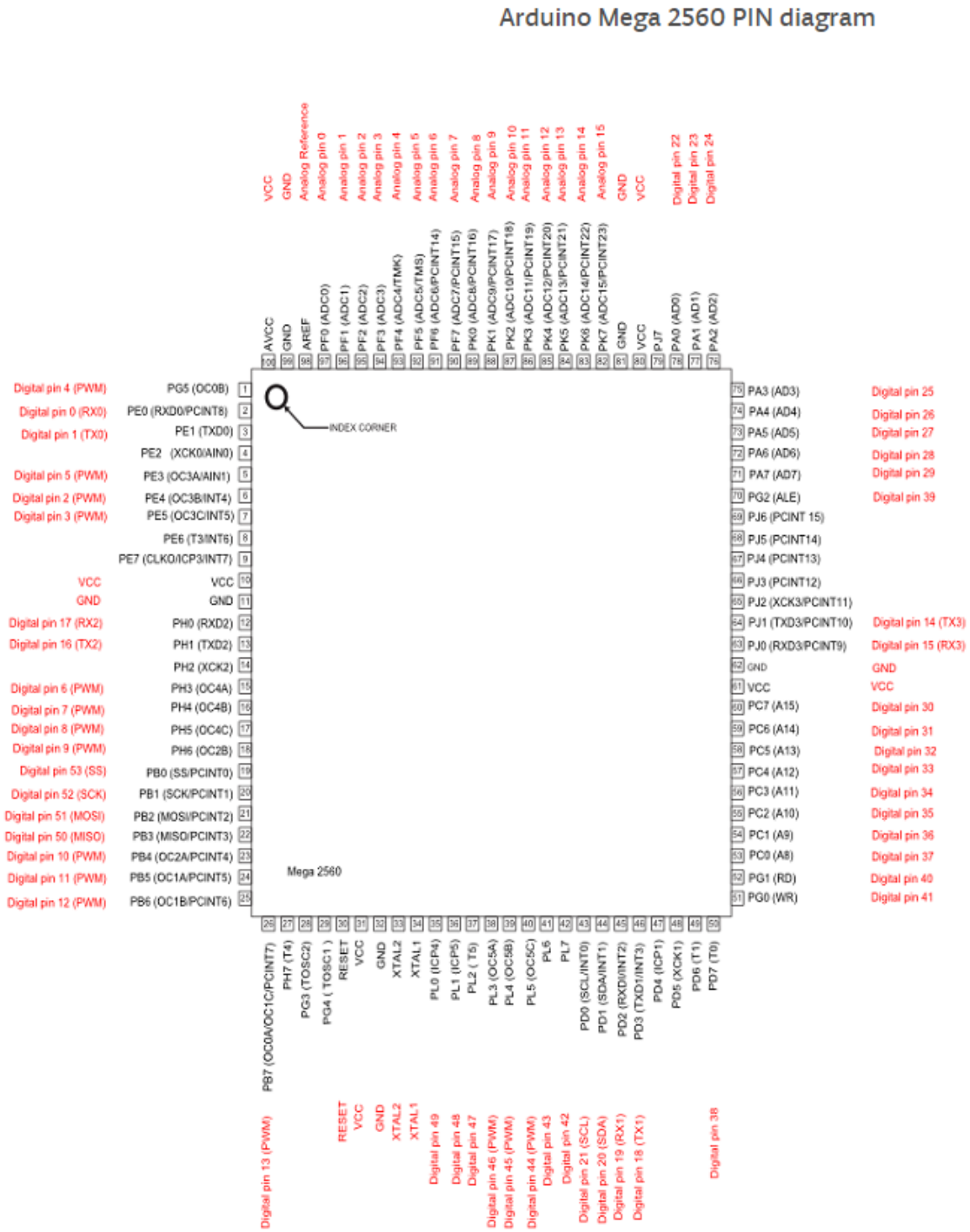


Figura 4.25 Diagrama de pines del microcontrolador ATmega2560 del Arduino MEGA ADK.

- El pin PWM 6 (OC4A)
- El pin PWM 7 (OC4B)
- El pin PWM 8 (OC4C)

Conociendo esto, el pin de comparación de salida utilizado para este cometido es el OC4A, es decir, el pin digital 6 del Arduino MEGA ADK.

4.4.3.2 Configuración de los registros del timer 4 (pin 6 – OC4A) para generar una interrupción cada 60ms que realice la función de un periodo de muestreo del sistema.

Teniendo en cuenta lo explicado en el apartado *3.5.4 Timers en Arduino*, se van a configurar los registros del timer 4 (pin 6 - OC4A) para generar una interrupción cada 60ms que realice la función de un periodo de muestreo del sistema.

Lo primero es seleccionar el modo de generación de forma de onda (WGM43:0) más adecuado, para lo cual, como se ha visto en el apartado *3.5.4 Timers en Arduino* el modo de operación más adecuado para generar interrupciones es el modo CTC (modo que borra el contador del timer cada vez que hay una comparación igualada), por eso, es el modo de operación que se va a utilizar para este cometido.

Dentro del modo de operación CTC hay 2 posibilidades: una definiendo el valor TOP (valor máximo que puede alcanzar el registro contador TCNT4) mediante el registro de comparación de salida OCR4A o definiendo el valor TOP mediante el registro de captura de entrada ICR4. En este caso, como lo que se va a emplear es una interrupción por comparación igualada, se ha seleccionado el modo CTC que define el valor TOP con el registro de comparación de salida OCR4A, con lo cual, el modo de generación de forma de onda seleccionado para generar una interrupción cada 60ms es WGM43:0 = 4.

Table 17-2. Waveform Generation Mode Bit Description[1]

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Figura 4.26 Modo de generación de forma de onda seleccionado para generar una interrupción cada 60ms que gestione los tiempos de muestreo.

Una vez seleccionado el modo de generación de forma de onda, se aplica la fórmula para establecer el valor del registro OCR4A para generar una interrupción cada 60 milisegundos.

Entonces como lo que se quiere generar es una interrupción cada 60ms, se va a poner en la formula los 60ms, y se va a obtener el valor del registro OCR4A para que cada 60ms salte el programa a la rutina de servicio de interrupción (ISR).

Hay que decir, que por el modo de generación de forma de onda seleccionado el registro OCR4A va a definir el valor TOP, es decir, OCR4A = TOP. El valor del registro OCR4A establece cada cuanto tiempo se quiere que se genere una interrupción, esto es, establece el periodo de muestreo en la programación.

NOTA: Para obtener el valor del registro OCR4A hay que probar con diferentes valores del prescaler, lo más adecuado es que cuanto mayor sea el valor del registro OCR4A mejor, puesto que se obtiene una mayor resolución, con lo cual, se va a empezar probando con el valor N=1 del prescaler para obtener el valor del registro OCR4A que genere una interrupción cada 60ms.

Esta es la fórmula a aplicar:

$$f_{OC4A} = \frac{f_{clk\_I/O}}{2*N*(1+OCR4A)}$$

Donde:

- $f_{clk\_I/O}$: Es la frecuencia de reloj del micro de Arduino → 16MHz
- N: Representa el factor del prescaler del micro (1, 8, 64, 256 o 1024).
- f_{OC4A}: Es la frecuencia de la señal que va a hacer que cada x microsegundos se genere una interrupción, en este caso, que cada 60ms se genere una interrupción, con lo cual, 60ms son 16,6667Hz de frecuencia.

- Probando con N = 1:

$$f_{OC4A} = 16{,}6667Hz = \frac{16*10^6 Hz}{2*1*(1+OCR4A)} \rightarrow (1 + OCR4A) = 480000$$

Este valor no coge, como son registros de 16 bits, el valor máximo posible es de 2^{16}-1 que es igual a 65535.

- Probando con N = 8:

$$f_{OC4A} = 16{,}6667Hz = \frac{16*10^6 Hz}{2*8*(1+OCR4A)} \rightarrow (1 + OCR4A) = 60000 \rightarrow OCR4A = 59999$$

Este valor si coge, con lo cual, el prescaler del micro se debe configurar a 8.

Entonces, por el modo de generación de forma de onda seleccionado, el valor TOP se establece mediante el registro de comparación de salida OCR4A, con lo cual, en la programación Arduino se va a poner OCR4A = 59999 = TOP, con este valor el registro de comparación de salida OCR4A va a generar una interrupción cada 60ms, como se sabe, el valor del registro OCR4A es el que establece el periodo de muestreo.

Se puede decir que el valor de 60000 son 60000µs, puesto que, da la casualidad que el periodo de muestreo seleccionado es de 60ms = 60000µs, con lo cual, dicho esto, en la programación software de Arduino al registro OCR4A se le asigna el valor de 59999 de la siguiente manera:

int periodo = 60000; // Periodo de muestreo en microsegundos (periodo de muestreo de 60ms).

OCR4A = periodo-1; // Como se puede ver en la fórmula OCR4A = 60000-1

Por lo tanto, como se ha visto al aplicar la fórmula, se debe configurar el prescaler del reloj del micro a 8 mediante los bits CS42:0 ubicados en el registro de control B del timer (TCCR4B).

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figura 4.27 Bits de selección de reloj, para que el prescaler del reloj del micro este configurado a 8 (para generar una interrupción cada 60ms).

Los 3 bits de selección de reloj seleccionan la fuente de reloj (el prescaler) que utilizará en este caso, el timer 4. Como se ha visto en la fórmula, el prescaler del reloj del micro se configura a 8, es decir, la fuente de reloj del micro va a ser de 16MHz/8.

A continuación, se va a configurar el modo de comparación de salida mediante los bits COM4A1:0 ubicados en el registro de control A del timer TCCR4A. Como el modo de operación seleccionado es el modo CTC se debe ir a la tabla de configuración del modo de comparación de salida de no PWM (“non-PWM”):

Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Figura 4.28 Modo de comparación de salida seleccionado, para generar la interrupción.

Se ha seleccionado el modo de comparación de salida que mantiene el funcionamiento del puerto normal y desconecta el pin de comparación de salida OC4A que es el pin digital 6 en el Arduino. Esto se hace, para que se pueda utilizar el pin digital 6, es decir, para que se pueda conectar un componente al pin digital 6 y funcione de manera normal, puesto que la interrupción se genera internamente en el pin digital 6.

Dicho esto, el funcionamiento para generar una interrupción cada 60ms va a ser el siguiente: la interrupción se genera cuando el registro contador TCNT4 alcanza el valor establecido en el registro de comparación de salida OCR4A (en este caso OCR4A = 59999), cuando ocurre esto, se salta a la rutina de servicio de interrupción (ISR) dejando la ejecución del programa normal detenida, además cuando ambos registros se igualan aparte de saltar a la rutina de interrupción, el registro contador TCNT4 se pone a 0 e inicia de nuevo la secuencia de conteo, para volver a realizar el proceso de nuevo.

Una vez seleccionado el modo de generación de forma de onda mediante los bits WGM43:0, el prescaler del reloj del micro mediante los bits CS42:0 y el modo de comparación de salida mediante los bits COM4A1:0, ahora se implementan los registros en los cuales están ubicados los bits anteriormente mencionados:

TCCR4A - Registro de control A del timer/counter 4

Bit	7	6	5	4	3	2	1	0
	COM4A1	COM4A0	COM4B1	COM4B0	COM4C1	COM4C0	WGM41	WGM40
	0	0	0	0	0	0	0	0

Como no se utilizan los pines de comparación de salida OC4B (pin 7) y OC4C (pin 8), se configura el modo de comparación de salida de ambos pines al igual que para el pin de comparación de salida OC4A (pin6), en funcionamiento normal, es decir, también se ponen a 0, para que cualquier componente se pueda conectar a ellos y funcionen de manera normal.

TCCR4B - Registro de control B del timer/counter 4

Bit	7	6	5	4	3	2	1	0
	ICNC4	ICES4	-	WGM43	WGM42	CS42	CS41	CS40
	0	0	0	0	1	0	1	0

Como el registro de captura de entrada ICR4 no se utiliza para nada, el bit de selección de flanco de captura de entrada, ICES4 se deshabilita, es decir, se pone a 0.

Cuando se quieren generar interrupciones como es el caso, se debe añadir el registro de máscara de interrupción del timer/counter TIMSK4 a los 2 anteriores, puesto que, es el registro que habilita/deshabilita los diferentes tipos de interrupciones del timer.

TIMSK4 - Registro de máscara de interrupción del timer/counter 4

Bit	7	6	5	4	3	2	1	0
	-	-	ICIE4	-	OCIE4C	OCIE4B	OCIE4A	TOIE4
	0	0	0	0	0	0	1	0

Como el tipo de interrupción que se va a emplear para generar una interrupción cada 60ms es la interrupción por comparación igualada de salida, es decir, que cuando el registro TCNT4 sea igual al registro OCR4A se va a generar la interrupción saltando a la rutina de interrupción ISR y dejando la ejecución del programa normal detenida (en espera), cuando finalice la ejecución del algoritmo de la rutina de la interrupción, el programa continua ejecutando el algoritmo normal en donde se había quedado hasta que vuelva a saltar otra vez a la rutina de interrupción, y así todo el rato.

Por lo tanto, para poder generar este tipo de interrupción primero hay que habilitarla en el registro de máscara de interrupción del timer 4 (TIMSK4), por medio del bit OCIE4A, que habilita la interrupción por comparación igualada en el pin de comparación de salida OC4A (pin6), que es lo que se ha hecho en la configuración del registro de máscara de interrupción, establecer a 1 el bit OCIE4A para habilitar la interrupción por comparación igualada de salida en el pin OC4A (pin6).

Por último, hay que decir que se debe llamar a la rutina de servicio de interrupción ISR de comparación igualada de salida, por medio de la siguiente instrucción en la programación Arduino: *ISR(TIMER4_COMPA_vect*).

Todos estos registros que se han visto, son los que se implementan en la programación software de Arduino para generar una interrupción cada 60ms, que gestione de la mejor manera posible los tiempos de muestreo, para poder ejecutar la señal de control del PID y poder obtener la posición de la bola por medio del sensor infrarrojo cada cierto periodo de tiempo.

De esta manera se implementan los registros de configuración del timer 4 en la programación software de Arduino, hay que decir que los registros se implementan en la parte de configuración del programa, es decir, en el *void setup()*:

//Se emplea el timer 4 para generar una interrupción cada 60ms, para realizar la función de un periodo de muestreo (pin 6 - OC4A).

TCCR4A = B00000000; // Modo de generación de forma de onda "CTC" con comparación con el registro OCR4A y modo de comparación de salida normal.

TCCR4B = B00001010; //Prescaler del reloj del micro a 8 y modo de generación de forma de onda "CTC" con comparación con el registro OCR4A.

OCR4A = periodo-1; // Establece cada cuanto tiempo se quiere que se genere una interrupción, es decir, mediante este registro se establece el periodo de muestreo, que va ser de 60ms.

TIMSK4 = B00000010; // Habilita la interrupción por comparación igualada de salida, poniendo a 1 el bit OCIE4A.

4.4.4 Implementación del controlador PID en Arduino.

Una vez seleccionado el método de linealización del sensor, y haber visto como se controla el servomotor y como se genera una interrupción cada 60ms con el uso de los timers de Arduino, se implementa el controlador PID en la programación.

Como se ha dicho en el apartado *3.5.3 Controlador PID*, se va a utilizar el controlador PID para controlar el sistema barra y bola. Lo que se pretende con este controlador es realizar el control de posición de la bola por realimentación, el cual, calcula el error de posición de la bola existente realizando la resta entre la posición deseada de la bola (consigna) y el valor de posición medido por el sensor, y en función del error obtenido, se obtiene una señal de control que modifica el ángulo de giro del servo, corrigiendo así, la señal de error, esto es, ajusta el proceso ante perturbaciones externas. Como el sistema, es un sistema en lazo cerrado, es decir, es un sistema de control realimentado, este proceso ocurre hasta que el error quede totalmente corregido en el régimen permanente.

La ecuación del controlador PID es la siguiente:

$$u(t) = K_P * e(t) + K_I * \int_0^t e(t)dt + K_D * \frac{de(t)}{dt}$$

Sabiendo que u(t) es la señal de control del PID (señal de salida del PID) y e(t), la señal del error de posición de la bola. K_P, K_I y K_D son las constantes proporcional, integral y derivativa respectivamente que son las que se deben obtener para conseguir un control óptimo.

Como se ha mencionado en el apartado *3.5.3 Controlador PID*, se va a utilizar un controlador PID por las siguientes razones:

- Es necesaria la utilización del término proporcional para controlar la posición de la bola, teniendo en cuenta la señal de error, puesto que es la diferencia entre el valor de consigna y la posición medida de la bola. Es decir, para que la barra se incline en el sentido adecuado.
- Es necesaria la utilización del término derivativo para controlar la velocidad de la bola, es decir, para reaccionar lo antes posible ante cambios del error de posición, esto es, para que el sistema sea lo más estable posible.
- Por último, es necesaria la utilización del término integral para eliminar el error en estado estacionario, puesto que, una de las características fundamentales que tiene que cumplir este sistema es la precisión, es decir, si el valor de la consigna es de 250mm tiene que posicionar la bola a 250mm, el error tiene que ser mínimo. El término integral considera la posición de la bola y cuanto tiempo lleva allí, puesto que actúa como un acumulativo del error, y cuando el error acumulado es notable la acción integral actúa, con lo cual, cuanto más tiempo

pase alejado del valor de la referencia más irá incrementándose el error integral hasta que reduzca completamente el error.

En este caso, u(t) va a ser la señal del ángulo de giro del servomotor en radianes (Θ(s)). Teniendo en cuenta esto, en la programación software de Arduino lo que se obtiene con el controlador PID es el valor del duty cycle de la señal PWM que controla el ángulo de giro del servo, como ya se sabe, dependiendo del valor del duty cycle se aplicará una tensión mayor o menor al servomotor, dependiendo de la tensión que se aplique al servomotor girará más o menos grados; dicho de otra manera, se obtiene el ángulo de giro del servomotor necesario para posicionar la bola en el valor de consigna especificado, o lo que es lo mismo, para eliminar o reducir el error de posición de la bola.

En este caso el control del sistema se debe implementar en tiempo discreto (z), puesto que, se utiliza un microcontrolador Arduino, es decir, un dispositivo digital para controlar el sistema.

Dicho todo esto, a la hora de implementar un controlador PID en la programación software de Arduino, se implementa aplicando la ecuación del PID, pero en vez de en ecuaciones diferenciales, en ecuaciones en diferencias (tiempo discreto), puesto que las ecuaciones en diferencias son las utilizadas en los procesadores digitales a la hora de implementar los algoritmos de control de los sistemas. Los sistemas discretos van a estar representados mediante ecuaciones en diferencias.

Sabiendo que las ecuaciones en diferencias se obtienen realizando aproximaciones numéricas de las derivadas y las integrales de una ecuación diferencial, en este caso de la ecuación diferencial del controlador PID, se obtiene lo siguiente, como se va a ver a continuación.

Hay varios métodos de aproximación numérica tanto de integración como de diferenciación para discretizar los controladores: método de Euler hacia adelante, método de Euler hacia atrás y método trapezoidal. En este caso, se va a realizar la transformación de ecuaciones diferenciales a ecuaciones en diferencias basándose en el método de Euler hacia atrás, que es el método que más se suele utilizar.

1. Aproximación numérica de la integración:

   Consiste en aproximar la integral $\frac{1}{S}$ mediante integración numérica empleando la formula de aproximación de Euler. La idea es aproximar el área de debajo de la curva mediante rectángulos.

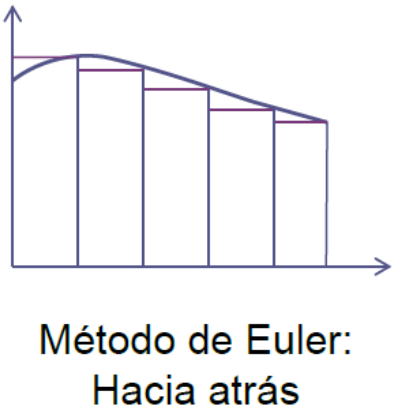


Figura 4.29 Aproximación numérica de la integración empleando el método de Euler hacia atrás.

   Partiendo del término integral de la ecuación del controlador PID se obtiene:

$$u(t) = \int_0^t e(t)dt \rightarrow u_{nT} = T\sum_{i=1}^{n} e_{iT} \rightarrow u_{nT} = T\sum_{i=1}^{n-1} e_{iT} + Te_{nT} \rightarrow$$

   Finalmente se obtiene la ecuación en diferencias de la integral basándose en el método de Euler hacia atrás:

$$u_{nT} = u_{(n-1)T} + Te_{nT}$$

   La señal del término integral es igual a la suma de la señal del término integral anterior más el error de posición actual. Con esto el término integral se va ir acumulando en el tiempo. Cuanto más tiempo pase, más grande se va a hacer el término integral.

2. Aproximación numérica de la diferenciación:

   Partiendo del término derivativo de la ecuación del controlador PID se obtiene:

$$u(t) = \frac{de(t)}{dt} \rightarrow u(t) = \frac{e(t) - e(t-T)}{T} \rightarrow$$

La ecuación en diferencias de la derivada basándose en el método de Euler hacia atrás es la siguiente:

$$u(t) = \frac{e_{nT} - e_{(n-1)T}}{T}$$

La señal del término derivativo es igual a la resta del error de posición actual menos el error de posición anterior.

3. El termino proporcional en ecuaciones en diferencias es el siguiente:

$$u_{nT} = e_{nT}$$

La señal del término proporcional es igual al error de posición actual, exactamente igual que en la ecuación diferencial.

Entonces, la ecuación del controlador PID queda de la siguiente manera en ecuaciones en diferencias:

$$u(t) = K_P * e_{nT} + u_{(n-1)T} + K_I * Te_{nT} + K_D * \frac{(e_{nT} - e_{(n-1)T})}{T}$$

Sabiendo todo esto, el controlador PID se ha implementado con el siguiente algoritmo en la programación software de Arduino:

//**********CONTROLADOR PID **********

// Se implementa un controlador PID para corregir la posición de la bola ante perturbaciones externas, es decir, se va utilizar un controlador PID para corregir las posibles desviaciones que ocasione la perturbación.

// Como se ha explicado antes, el controlador PID se implementa en Arduino poniendo su ecuación característica en ecuaciones en diferencias (tiempo discreto (z)), en este caso se ha transformado a ecuaciones en diferencias por el método de aproximación numérica llamado método de Euler hacia atrás.

error_pos = consigna - posicion; //El error entre la consigna y la posición es la entrada al PID.

//********** Cálculo de la derivada del error de posición

// Se va a aplicar un filtro digital paso bajo al cálculo de la derivada del error de posición.

for (int i=0; i<m-1; i++){ //Desplazo el array a la izquierda, eliminando la derivada del error más antigua.

filtroD[i] = filtroD[i+1]; // Esto es para que se vaya actualizando el array, se desplaza todo el array un lugar a la izquierda, dejando libre el ultimo hueco para poner el cálculo nuevo.

}

filtroD[m-1] = (error_pos - error_pos_ant); // La derivada del error actual la guardo al final del array (posición [4]).

// La derivada del error o el error diferencial se calcula realizando la resta del error de posición actual menos el error de posición anterior.

error_dif = 0; // Se pone la variable de la derivada del error a 0.

for (int i=0; i<m; i++){ // Se calcula la media.

error_dif = error_dif + filtroD[i]; // Aquí se suman todos los cálculos de la derivada del error almacenadas en el array (5).

}

error_dif = error_dif / m; // Se saca la media de las medidas de la derivada del error de posición.

// Con esto se mejora la precisión del cálculo de la derivada del error (acción derivativa). Aquí acaba el algoritmo del filtro paso bajo digital.

//********** Cálculo de la integral del error de posición

// Solo se aplica la acción integral, cuando los valores del error de posición estén dentro de los límites establecidos para la acción integral. Se puede decir que esto es un anti-wind up.

if(abs(error_pos)>lim_inf && abs(error_pos)<lim_sup){

// Se acota la acción integral en las proximidades del equilibrio, entre 5 y 40, es decir, si el error de posición está entre 5 y 40mm tanto positivos como negativos se aplica el cálculo de la integral del error de posición.

// Esto es para limitar la acción integral, porque si tiene demasiado error que corregir o si se acumula mucho error en la acción integral, el sistema se hace muy lento debido a los retrasos que se generan e inestable.

*error_int = error_int + Ki * error_pos;* // Aquí se calcula la integral del error, que es el cálculo de la integral del error anterior más la suma del error de posición actual multiplicado por el valor de K_I.

}

else {

error_int = 0; // Si el error de posición esta fuera de los límites de la acción integral, se realiza un controlador PD, para que el sistema no sufra retrasos y no se haga tan lento e inestable.

}

//********** Acción PID

*salida_PID = Kp * error_pos + Kd * error_dif + error_int;* // Fórmula para obtener la señal de salida del controlador PID, es decir, para obtener el ángulo de giro del servomotor necesario para establecer la bola en la posición deseada. Se aplica la ecuación característica del controlador PID en ecuaciones en diferencias, vista anteriormente.

DC = reposo + salida_PID; // El reposo es el valor en el cual la barra está horizontal, es decir, es el punto de equilibrio de la barra, ángulo de giro del servo a 90º.

//Con lo cual, lo que se hace es sumar o restar ese valor de reposo con la salida del PID, para que si la salida del PID es negativa la barra se incline hacia un lado, y si es positiva la barra se incline hacia el otro lado.

DC = constrain(DC,500,2300); // Se acota el duty cycle de la señal PWM que va controlar el ángulo de giro del servo, entre los valores en los cuales el servo trabaja, es decir, se acota en el rango de trabajo del servo (rango de movilidad del servo de 0º a 180º).

OCR5A = DC; // El registro de comparación de salida del timer 5 es igual al valor del duty cycle de la señal PWM del servo calculada, este valor es el que va hacer que el ángulo de giro del servo se posicione en la posición necesaria para que el error de posición desaparezca. Este registro es el que va a generar el duty cycle calculado en la señal PWM utilizada para el control del servo.

4.4.4.1 Explicación de las instrucciones fundamentales

Ahora se va a desgranar el algoritmo del controlador PID visto anteriormente y se van a explicar las instrucciones más importantes de él, además de explicar el porqué de algunas instrucciones.

Primero antes de nada hay que decir, que hay que asegurarse que la función que ejecuta el PID lo haga regularmente, es decir, con un periodo de muestreo determinado, para ello, como se ha visto en el apartado *4.4.3 Generación de la interrupción en Arduino* se utilizan las interrupciones, para que cada cierto periodo de tiempo se ejecute la señal de control, en este caso, como el periodo de muestreo seleccionado es de 60ms se va a generar una interrupción cada 60ms.

Una vez que se asegura que el PID se ejecuta a intervalos regulares, los cálculos correspondientes a la parte derivada e integral se simplifican, es decir, el término T del periodo de muestreo desaparece de la ecuación del PID en ecuaciones en diferencias (tiempo discreto), la cual se ha implementado en la programación como se ha visto antes:

$$u(t) = K_P * e_{nT} + u_{(n-1)T} + K_I * e_{nT} + K_D * (e_{nT} - e_{(n-1)T})$$

Cálculo de la derivada del error:

Como se ha visto en el algoritmo del PID implementado en la programación software de Arduino (visto en el anterior apartado), en el cálculo de la derivada del error, acción derivativa, se ha aplicado un filtro paso bajo digital, con esto, se ha conseguido un control más estable, puesto que sin esta aplicación el control del sistema era un poco inestable debido a que la bola no se terminaba de parar (realizaba pequeñas oscilaciones) y el servo estaba todo el rato haciendo movimientos cortos, con lo cual, de esta manera se han solucionado los problemas de inestabilidad que había en el control, y se ha conseguido un control más estable gracias a que ahora la bola se queda quieta y el servomotor no se mueve apenas. Por estas razones, además de la de mejorar la precisión del cálculo de la derivada del error, puesto que desprecia los valores incoherentes realizando una media, se ha aplicado un filtro paso bajo digital por software en el cual se utiliza la matriz "filtroD[]" para almacenar y tratar las ultimas 5 medidas del cálculo de la derivada del error.

Como se ha explicado anteriormente, el cálculo de la derivada en el algoritmo de control, se obtiene aplicando la ecuación en diferencias de la derivada del error:

La ecuación en diferencias de la derivada del error, basándose en la aproximación numérica del método de Euler hacia atrás es la siguiente:

$$u(t) = e_{nT} - e_{(n-1)T}$$

Con lo cual, en el algoritmo de control se aplica de la siguiente manera:

error_dif = (error_pos - error_pos_ant);

La señal de la derivada del error es igual a la resta del error de posición actual menos el error de posición anterior.

Cálculo de la integral del error:

Como en uno de los modos de funcionamiento se pretende poder variar las diferentes acciones del controlador PID a través de la interfaz de usuario, mientras el sistema está ejecutándose, con esto si se aplica la ecuación en diferencias de la integral del error tal y como se conoce en el algoritmo de control, se generan baches indeseables en la señal de control cuando se modifican las diferentes acciones del controlador PID mientras el sistema se ejecuta.

La ecuación en diferencias del cálculo de la integral del error:

$$u_{nT} = u_{(n-1)T} + e_{nT}$$

Entonces, la ecuación en diferencias del término integral es la siguiente:

$$u(t) = K_I * (u_{(n-1)T} + e_{nT})$$

La señal del término integral es igual a la suma de la señal del término integral anterior más el error de posición actual. Con esto el término integral se va ir acumulando en el tiempo, y todo esto se multiplica por la constante de la acción integral K_I.

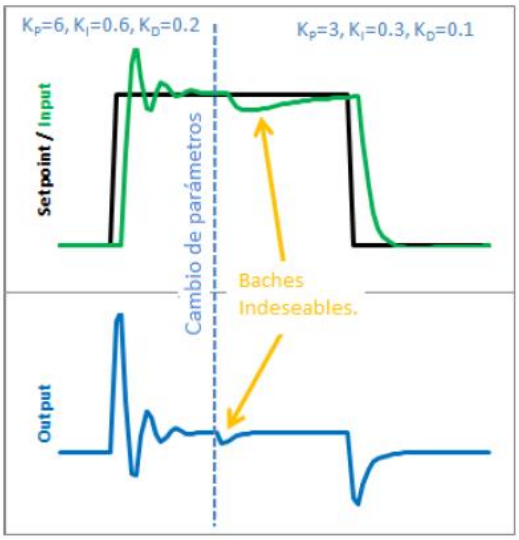


Figura 4.30 Como se generan los baches indeseables al modificar las acciones del PID.

Como se puede ver en la figura, en la cual "Input" es la posición de la bola medida por el sensor y "Output" la salida del PID, en este caso la salida del PID es la señal del ángulo de giro del servomotor (θ, en rad), se puede ver como se generan unos baches indeseables en la señal de control al modificar las acciones del PID mientras el sistema se está ejecutando.

El culpable de este bache en la señal de salida del PID es el término integral, porque es el único término que cambia drásticamente cuando la señal de control se modifica. Esto sucede debido a la interpretación de la integral. Esta interpretación funciona bien hasta que K_I cambia de valor; de repente, la suma de todos los errores se multiplica con el valor de K_I, esto no es lo que se necesita. Solo se quiere que afecte a los valores que estén por delante en el tiempo. Por ejemplo: Si se modifica K_I, en un

tiempo t = 5s, se necesita que el impacto de este cambio solo afecte a valores de K_I que se modifican en un tiempo mayor a t = 5s.

Entonces la solución para eliminar los baches indeseables que se generan en la señal de control al modificar sobre todo la acción integral es, en lugar de tener el término K_I fuera del cálculo de la integral del error, se introduce dentro del cálculo multiplicando al error de posición actual.

$$u(t) = u_{(n-1)T} + K_I * e_{nT}$$

Gracias a esto en el algoritmo de la programación ahora se toma el error de posición y se multiplica por el valor K_I en ese momento, luego se almacena la suma de los diferentes errores multiplicados por la constante K_I en la variable "*error_int*". La instrucción que va a realizar el cálculo de la integral del error queda de la siguiente manera:

*error_int = error_int + Ki * error_pos;*

Gracias a esto se eliminan los baches indeseables que se generaban en la señal de control al modificar sobre todo la acción integral, mientras el sistema estaba en ejecución. Ahora la señal de control es una señal suave y sin sobresaltos.

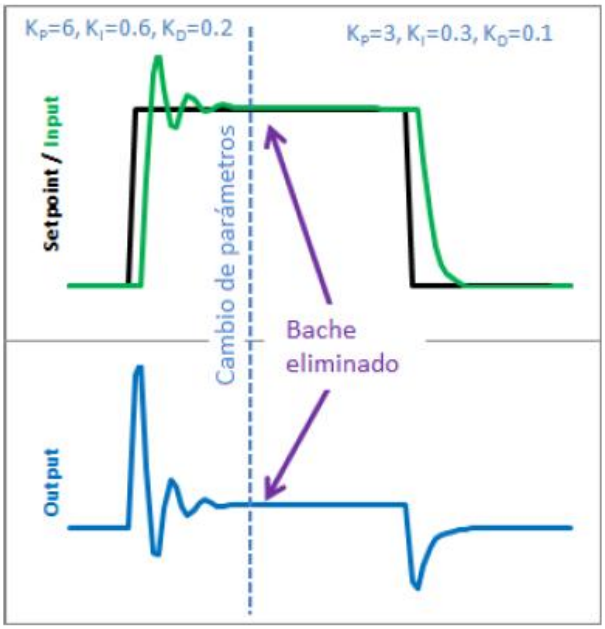


Figura 4.31 Se puede ver como los baches de la señal de salida del PID se han eliminado al multiplicar la constante K_I al error de posición actual.

Con esta modificación realizada, los cambios en la acción integral mientras el sistema se está ejecutando, ya no afectan al rendimiento del sistema.

Wind-up

El efecto wind-up aparece al arrancar el sistema o en cualquier otra situación, donde aparece un error muy grande durante un tiempo prolongado. Esto hará que el término integral aumente para reducir el error, pero si el actuador utilizado tiene límites físicos, que es el caso, se saturará, pero el término integral seguirá creciendo. Cuando el error se reduce, la parte integral también comenzara a reducirse, pero desde un valor muy alto llevando mucho tiempo hasta que logre estabilizarse, generando fluctuaciones muy grandes. En definitiva, cuando ocurre el efecto wind-up el sistema se hace muy lento debido a los retrasos que se generan e inestable, puesto que el término integral tiene demasiado error que corregir o ha acumulado demasiado error.

Figura 4.32 Los efectos del wind-up cuando el actuador del sistema se satura.

En la figura se puede ver que el valor de salida del PID está muy por encima de lo que realmente envía debido a los límites físicos del actuador del sistema. En este caso, los límites físicos del actuador van a estar provocados por el ángulo de giro del servomotor, debido a que solo puede girar 90º en un sentido y en el otro.

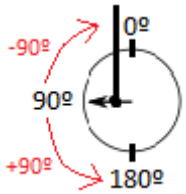


Figura 4.33 Límites físicos del servomotor utilizado.

Cuando empieza el control del sistema a ejecutarse la barra se establece en su punto de equilibrio (también llamado reposo en la programación) para poder realizar el control de posición de la bola de manera eficaz, para ello, el ángulo de giro del servomotor se pone a 90º (OCR5A = 1400), con lo cual, el punto de equilibrio o reposo de la barra es cuando el ángulo de giro del servomotor está a 90º, por eso, como se sabe que el rango de movilidad del servomotor utilizado es de 0º a 180º, el servomotor solo se va a poder mover 90º en un sentido y en el otro, con lo cual, los límites físicos del actuador del sistema van a ser de -90º a +90º.

Hay varias formas para mitigar el efecto del wind-up, en esta programación se han aplicado 2, que son las siguientes:

1. Se han establecido unos límites para poder aplicar la acción integral, es decir, para que el error integral que es el que acumula el error de posición en el tiempo, no se acumule excesivamente y no tenga que corregir demasiado error, lo que lleva a retrasos en el sistema como se puede ver en la figura anterior y a inestabilidades.

   Entonces, cuando los valores del error de posición estén dentro de los límites establecidos para la acción integral, se aplica el cálculo de la integral del error, es decir, se aplica la acción integral. Se acota la acción integral en las proximidades del equilibrio, entre 5 y 40, es decir, si el error de posición está entre 5 y 40mm tanto positivos como negativos se realiza el cálculo de la integral del error de posición. Cuando el error de posición sea inferior a 5mm, se da por cumplido el objetivo y se deja de integrar.

   En cambio, si el error de posición esta fuera de los límites de la acción integral, se realiza un controlador PD (I = 0), para que la acción integral no acumule demasiado error y el sistema no sufra retrasos y no se haga tan lento e inestable.

Todo esto se escribe en la programación software de Arduino de la siguiente manera:

```
int lim_inf = 5;
int lim_sup = 40;

  if(abs(error_pos)>lim_inf && abs(error_pos)<lim_sup){
   error_int = error_int + Ki * error_pos;
  }
  else {
   error_int = 0;
  }
```

A esto se le suele llamar anti wind-up. Como conclusión cabe destacar, que hay que acotar convenientemente la acción integral porque de lo contrario conlleva demasiada inestabilidad.

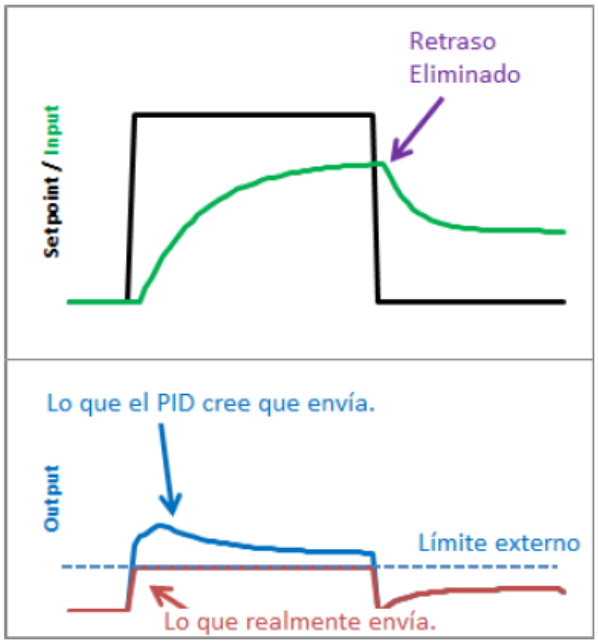


Figura 4.34 Aplicando los límites establecidos para la acción integral en la programación software, se consiguen eliminar los retrasos de la señal de posición de la bola medida por el sensor.

Como se puede ver en la figura, gracias a esta limitación de la acción integral, se consiguen eliminar los posibles retrasos que se puedan producir en la señal de posición de la bola.

2. Como se ha visto en la anterior figura, se ha eliminado el retraso inducido por el wind-up, sin embargo no se han mitigado todos sus efectos, puesto que, todavía hay una diferencia entre lo que el PID piensa que está enviando, y lo que realmente sucede, debido a los límites físicos que tiene el actuador, en este caso el servomotor, es decir, por ejemplo la señal de control dice que el servomotor se posicione con un ángulo de giro de 250º para establecer la bola en el valor de referencia asignado, pero el servo debido a sus límites físicos no puede girar más de 180º, con lo cual, el PID cree que es posible posicionar el servo con un ángulo de giro de 250º, pero lo que realmente sucede es que el servo llega a su límite físico y se queda con un ángulo de giro de 180º.
   Para que no ocurra todo esto, además de acotar el término integral, también hay que acotar la señal de salida del controlador PID para que se mantenga dentro de los límites físicos de funcionamiento del actuador. En este caso, se limita el duty cycle de la señal PWM que controla el ángulo de giro del servo, para mantenerlo dentro de los límites de trabajo del servo (500-2300), es decir, para mantenerlo dentro del rango de movilidad del servo que va de 0 a 180º. Esto se aplica en la programación software de Arduino de la siguiente manera:

   *salida_PID = Kp * error_pos + Kd * error_dif + error_int;*

   DC = reposo + salida_PID;

   DC = constrain(DC,500,2300); // Se acota el duty cycle de la señal PWM que va controlar el ángulo de giro del servo, entre los valores en los cuales el servo trabaja, es decir, se acota entre los límites del rango de trabajo del servo (rango de movilidad del servo de 0º a 180º).

   OCR5A = DC;

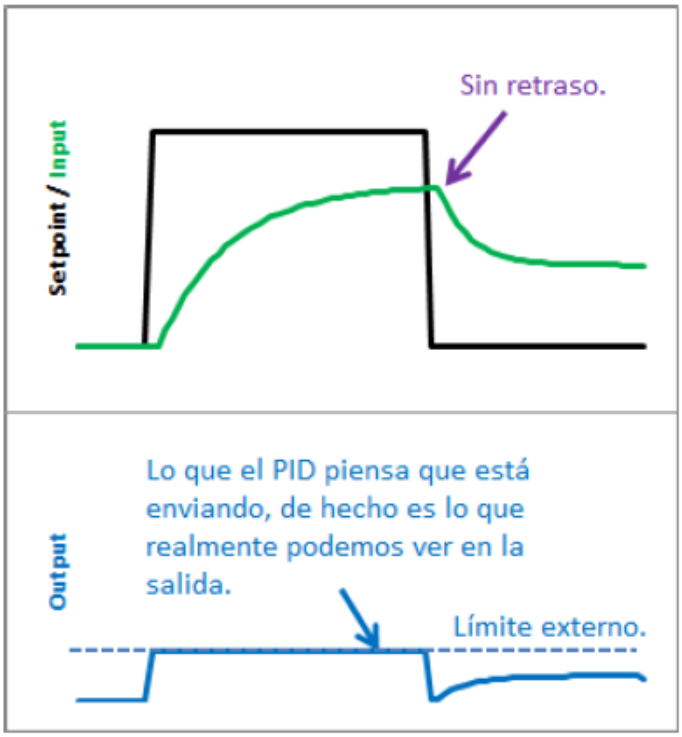


Figura 4.35 Aplicando la limitación de la acción integral y acotando la señal de salida del controlador PID en el rango de funcionamiento del actuador, se elimina el fenómeno wind-up.

Como se puede ver en la figura, aplicando la limitación impuesta a la acción integral y acotando la señal de salida del controlador PID en el rango de funcionamiento del servomotor se consigue eliminar el fenómeno wind-up. Se puede ver como la señal de salida del PID permanece dentro del rango de funcionamiento del actuador, en este caso del servomotor. Con esto se sabe que cualquier señal de salida del PID va a ser físicamente realizable por el servomotor.

Por último, cabe destacar la siguiente instrucción del algoritmo de control del controlador PID:

float reposo = 1400; // El reposo es el valor en el cual la barra está horizontal, es decir, es el punto de equilibrio de la barra, ángulo de giro del servo a 90º.

DC = reposo + salida_PID;

Mediante esta última instrucción se obtiene el valor del duty cycle de la señal PWM que controla el ángulo de giro del servo necesario para establecer la bola en la posición deseada de la barra. Para ello se consigue, realizando la suma entre el ángulo de giro de equilibrio de la barra 90º (en la programación OCR5A = 1400), y el ángulo de giro obtenido por la señal de salida del PID que tiene que ir adaptando para controlar la posición de la bola.

Figura 4.36 Barra en equilibrio, ángulo de giro del servo a 90º, OCR5A = 1400.

Es decir, lo que se hace es sumar o restar el valor de la salida del PID con la del reposo, para que si la salida del PID es negativa (normalmente cuando la salida del PID da negativa es porque la medida de posición es más grande que la consigna, dicho de otra manera, porque el error de posición es negativo) la barra se incline hacia un lado, en este caso al ser negativa la salida del PID, se inclinará hacia la derecha, es decir, la biela se moverá hacia arriba en sentido horario, con lo cual, el ángulo de giro del servomotor será inferior a 90º.

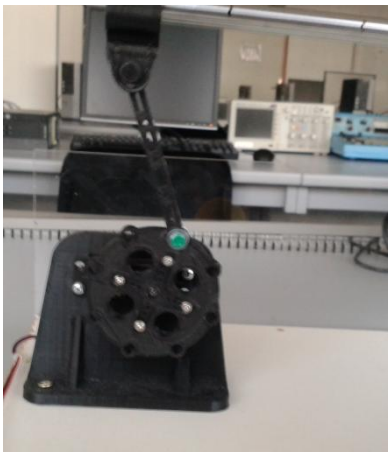


Figura 4.37 Ángulo de giro del servo a 0º, OCR5A = 500.

En cambio, si la salida del PID es positiva (error de posición positivo, esto es, valor de consigna superior al de la medida de posición) la barra se inclina hacia el otro lado, en este caso hacia la izquierda, es decir, la biela se moverá hacia abajo en sentido antihorario, por lo tanto, el ángulo de giro del servomotor será superior a 90º.

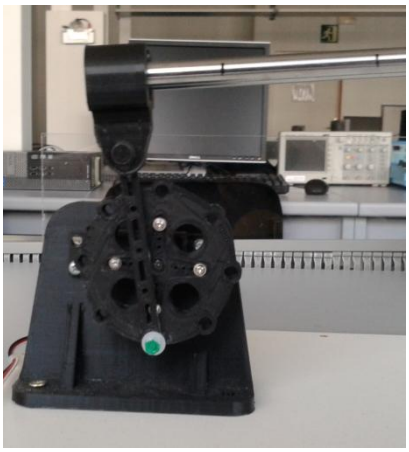


Figura 4.38 Ángulo de giro del servo a 180º, OCR5A = 2300.

Por ejemplo, si la medida de posición de la bola es de 300mm y la consigna está establecida en 250mm, se necesita que la bola se mueva a la derecha para establecerse en 250mm; para conseguir esto, todo sucede de la siguiente manera: el error de posición será negativo, con lo cual, la señal de salida del PID será negativa, entonces se resta esa señal negativa del PID al valor en el cual la barra está horizontal, en reposo, 1400 = 90º, entonces el valor del duty cycle de la señal PWM será un valor inferior al del reposo < 1400, con lo cual, el ángulo de giro del servo será inferior a 90º, esto significa que el

servomotor va a girar en sentido horario partiendo del punto de equilibrio, por lo tanto la biela va estar situada más arriba, inclinando la barra hacia la derecha, y a así, moviendo la bola hacia la derecha, que es lo que requiere el control. El proceso para llegar a situar la bola en el valor de consigna asignado, va a realizarse de esta manera en cada periodo de muestreo (cada 60ms), hasta que el error se reduzca a un valor cercano a cero.

4.4.5 Programación software de Arduino final.

Con lo aprendido en los apartados anteriores (*4.4 Programación software mediante Arduino para el control del sistema*), para el control de los diferentes dispositivos que conforman el sistema, aplicando todo ello en la programación, ya se realiza un control de posición de la bola en la maqueta, con el inconveniente de que no se conocen las constantes del PID idóneas para lo que se quiere, con lo cual, en este momento probando unas constantes al azar, si realiza el control de posición de la bola pero con unas características malas, puesto que el error de posición de la bola es muy grande y el tiempo de establecimiento del mismo es demasiado alto, por todo esto es necesario obtener unas constantes del controlador PID que realicen un control de posición de la bola óptimo.

Como no conozco los valores óptimos del controlador PID, se va simular el sistema, por medio del software Matlab Simulink para obtener una primera aproximación de las constantes, para ello, es necesario realizar el modelado de la planta, puesto que es imprescindible obtener la función de transferencia de la planta para la realización de la simulación. En los siguientes apartados se va hablar sobre ello.

4.5 Maqueta sistema barra y bola

En este apartado se expone, por una parte, el proceso de obtención del modelado de la planta del sistema barra y bola, y por otra parte, la implementación en el software Matlab Simulink del control de posición de la bola para simular el sistema, y así obtener una primera aproximación de los parámetros de los controladores.

Los pasos a seguir en este apartado son los siguientes:

1. Realizar el modelado de la planta del sistema barra y bola para su correspondiente simulación a través de la herramienta Matlab Simulink.
2. Diseñar un modelo de simulación, mediante el software Matlab Simulink.
3. Realizar el modelado de la planta en tiempo discreto, mediante el software Matlab Simulink.
4. Desarrollar un controlador PID en tiempo discreto que realice un control de posición de la bola óptimo.
5. Implementar estos valores del controlador PID (z) en tiempo discreto en la maqueta, y comparar la respuesta del sistema con la de la simulación.

4.5.1 Modelado de la planta

Lo que se quiere es realizar la simulación de este sistema con el objetivo de simularlo antes de implementarlo en el sistema físico, para obtener las constantes del PID que realicen un control de posición de la bola óptimo, o por lo menos obtener una primera aproximación de las constantes; para ello, lo primero que se debe hacer es obtener el modelado de la planta, en este caso el modelado de la maqueta barra y bola.

El modelado se puede definir como un modelo matemático que representa la dinámica de un sistema en cuestión, por medio de un conjunto de ecuaciones diferenciales. Puede haber diferentes modelos para un mismo sistema en función de las necesidades del problema de control.

Para obtener el modelado de la planta se utilizan las leyes de la física, para lograr una ecuación diferencial que represente la dinámica del sistema. Teniendo en

cuenta lo analizado en el apartado *4.1 Análisis del control del sistema*, se puede saber que, la entrada y la salida de la función de transferencia de la planta son las siguientes:

Figura 4.39 Entrada-salida de la función de transferencia de la planta del sistema.

Con lo cual, la función de transferencia de la planta del sistema, es decir, de la maqueta barra y bola es la siguiente:

$$G(s) = \frac{X(s)}{\theta(s)}$$

Sabiendo que la entrada es la señal del ángulo de giro del servomotor en radianes, θ(s) [rad], y la salida de la planta es la señal de posición de la bola en la barra en metros, X(s) [m], lo primero que hay que hacer para constituir el modelo matemático es definir el número de grados de libertad que posee, así como la definición de cada uno. En este caso se encuentran dos grados de libertad que son: la posición de la bola (m) y el ángulo de inclinación de la barra (radianes).

Estas serán tomadas como las variables en función de las cuales se van a obtener las ecuaciones dinámicas que gobiernan el sistema. Para que dichas ecuaciones resulten más simples y fáciles de usar, se supondrá que la bola rueda sin deslizarse y que la fricción entre la bola y la barra es insignificante (se desprecia).

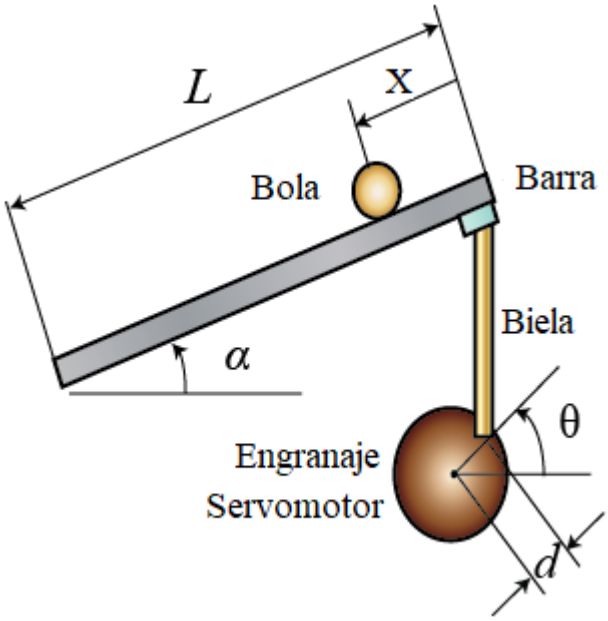


Figura 4.40 Esquema del sistema para el modelado.

Para obtener los parámetros físicos de la maqueta se han medido y se han pesado las siguientes constantes:

- m: Masa de la bola → m = 41,1g → m = 0,0411kg
- R: Radio de la bola → R = 27,5mm (d = 55mm) → R = 0,0275m
- d: Desplazamiento de la biela → d = 3cm → d = 0,03m
- g: aceleración gravitacional (gravedad) → g = - 9,8m/s^2
- L: Longitud de la barra → L = 49,5cm → L = 0,495m
- J: Momento de inercia de la bola →

  Sabiendo que para una esfera el momento de inercia con respecto al centro se expresa como: $J = \frac{2}{5} \times m \times R^2 \rightarrow J = 1{,}243 \times 10^{-5}\ kg.m^2$
- x: Coordenada de posición de la bola (a la salida de la planta en "m", a la hora de calcular el error, en "mm").
- α: Coordenada del ángulo de inclinación de la barra [rad].
- θ: Ángulo de giro del servomotor [rad].

Se va a partir de la formulación lagrangiana y no de las ecuaciones de movimiento de Newton debido a que para este caso concreto supone una simplificación importante. La

formulación lagrangiana simplifica considerablemente muchos problemas físicos. Así, mientras que con el procedimiento de Newton se coloca el énfasis sobre el agente exterior que actúa sobre el cuerpo (la fuerza), con el de Lagrange se manejan magnitudes asociadas al cuerpo (energía cinética y potencial). Este hecho es muy importante ya que hace que la lagrangiana de un sistema sea invariante ante los cambios de coordenadas. Esto permite pasar del espacio ordinario (en el que las ecuaciones de movimiento pueden ser muy complicadas) a un espacio de configuraciones elegido de tal forma que de una simplificación máxima.

La conocida ecuación de Lagrange se va a utilizar para obtener el modelado de la planta, que se representa de la siguiente manera:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} + \frac{\partial F}{\partial \dot{q}_i} = Q_i$$

Siendo:

q_i = Cada una de las coordenadas generalizadas.

L → La lagrangiana que será → L = T – V, donde T es la energía cinética y V la potencial.

F = Fuerza de disipación de Rayleigh.

Q = Fuerza generalizada.

Como coordenadas generalizadas se eligen la posición de la bola en la barra "q_1" (x) y el ángulo de inclinación de la barra con respecto a la horizontal "q_2" (α).

Sabiendo todo esto, ahora se va a realizar el desarrollo para obtener la ecuación o ecuaciones diferenciales que representen la dinámica del sistema.

Se empieza calculando la energía cinética total del sistema que será la que aporte la barra más la que aporte la bola, puesto que son los 2 elementos en movimiento:

$$T = T_{barra} + T_{bola}$$

$$T_{barra} = T_{rot} = \frac{1}{2} J_{barra}\, \omega_{barra}^2 = \frac{1}{2} J_{barra}\, \dot{q}_2^2$$

$$T_{bola} = T_{tras} + T_{rot} = \frac{1}{2} m\, v^2 + \frac{1}{2} J\, \omega^2$$

La velocidad lineal de la bola con respecto a las coordenadas generalizadas se calcula de la siguiente forma:

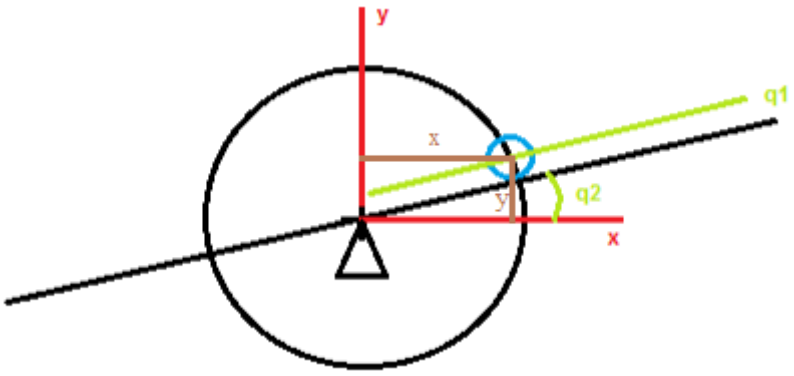


Figura 4.41 Representación de la trayectoria de la bola en la barra.

Tal y como puede verse en la figura, la bola se desplaza en horizontal por su propio movimiento y en vertical gracias al movimiento de la barra. Este movimiento compuesto queda encerrado dentro de un triángulo rectángulo y por lo tanto se puede decir que:

$$v^2 = \dot{x}^2 + \dot{y}^2$$

$$x = q_1 \cos q_2 \rightarrow \dot{x} = \dot{q_1} \cos q_2 - q_1 \dot{q_2} \operatorname{sen} q_2$$

$$y = q_1 \operatorname{sen} q_2 \rightarrow \dot{y} = \dot{q_1} \operatorname{sen} q_2 + q_1 \dot{q_2} \cos q_2$$

$$v^2 = \dot{q}_1^2 cos^2 q_2 - 2\dot{q}_1 q_1 \dot{q}_2 \operatorname{sen} q_2 \cos q_2 + q_1^2 \dot{q}_2^2 sen^2 q_2 + \dot{q}_1^2 sen^2 q_2 + 2\dot{q}_1 q_1 \dot{q}_2 \operatorname{sen} q_2 \cos q_2 + q_1^2 \dot{q}_2^2 cos^2 q_2 =$$

$$\dot{q}_1^2 (cos^2 q_2 + sen^2 q_2) + q_1^2 \dot{q}_2^2 (sen^2 q_2 \, cos^2 q_2) \rightarrow$$

$$v^2 = \dot{q}_1^2 + q_1^2 \dot{q}_2^2$$

La velocidad angular de la bola se calcula de forma sencilla, a través de la condición de rodadura sin deslizar, como:

$$v = R\,\omega \rightarrow \omega = \frac{v}{R} \rightarrow \omega = \frac{\dot{q}_1}{R} \rightarrow \omega^2 = \left(\frac{\dot{q}_1}{R}\right)^2$$

Con todo esto, la energía cinética total del sistema, queda de la siguiente manera sustituyendo estas velocidades en la ecuación:

$$T = \frac{1}{2} J_{barra}\, \dot{q}_2^2 + \frac{1}{2} m\, (\dot{q}_1^2 + q_1^2\, \dot{q}_2^2) + \frac{1}{2} J \left(\frac{\dot{q}_1}{R}\right)^2 \rightarrow$$

$$T = \frac{1}{2} J_{barra}\, \dot{q}_2^2 + \frac{1}{2} m\, \dot{q}_1^2 + \frac{1}{2} m\, q_1^2\, \dot{q}_2^2 + \frac{1}{2} \frac{J}{R^2}\, \dot{q}_1^2$$

$$\boxed{T = \frac{1}{2} J_{barra}\, \dot{q}_2^2 + \frac{1}{2} m\, \dot{q}_1^2 + \frac{1}{2} m\, q_1^2\, \dot{q}_2^2 + \frac{1}{2} \frac{J}{R^2}\, \dot{q}_1^2}$$

En cuanto a la energía potencial del sistema, siguiendo el mismo procedimiento que para la energía cinética se puede calcular la energía potencial de la barra, tomando como punto de referencia la posición horizontal de la misma.

$$V = V_{barra} + V_{bola}$$

$$V_{barra} = m_{barra}\, g\, r\, cos\, q_2 \quad y \quad V_{bola} = m\, g\, r\, q_1\, sen\, q_2$$

Donde g es la aceleración de la gravedad y se considera positiva para evitar signos en la ecuación y r es la distancia desde la línea que pasa por el centro de la bola hasta el eje del servomotor, la cual se ha considerado despreciable y, por tanto, $V_{barra} = 0$.

$$\boxed{V = m\, g\, q_1\, sen\, q_2}$$

Para poder aplicar la función lagrangiana falta por calcular la fuerza de disipación Rayleigh, que en este caso se considera nula por no tener en cuenta ninguna fuerza disipativa.

$$\boxed{F = 0}$$

Como se tienen dos coordenadas generalizadas, se obtendrán dos ecuaciones que rigen el comportamiento del sistema, donde la fuerza generalizada asociada a la coordenada de posición de la bola en la barra (q_1) es nula, y la fuerza generalizada asociada a la coordenada del ángulo de inclinación de la barra (q_2) es el par motor, τ.

$$\boxed{Q_1 = 0} \quad \text{y} \quad \boxed{Q_2 = \tau}$$

Con todo esto, aplicando la ecuación de Lagrange anteriormente expuesta, así quedaría definido el sistema dinámicamente:

$$\boxed{\left(m + \frac{J}{R^2}\right)\ddot{q}_1 - m\,q_1\,\dot{q}_2^2 + m\,g\,sen\,q_2 = 0}$$

$$\boxed{(J_{barra} + m\,q_1^2)\ddot{q}_2 + 2\,m\,q_1\,\dot{q}_1\,\dot{q}_2 + m\,g\,q_1\cos q_2 = \tau}$$

En el presente caso, se ha despreciado la segunda ecuación, la basada en la coordenada del ángulo de inclinación de la barra q_2, pues no se va a cerrar un lazo de control sobre el ángulo de inclinación de la barra ya que no se dispone de su medida.

Para obtener el modelado de la planta en función de transferencia se parte de la ecuación lagrangiana obtenida para la coordenada de la posición de la bola en la barra q_1, que viene dada por la siguiente ecuación como se ha visto antes; las coordenadas generalizadas se han sustituido por su valor real, es decir, $q_1 = x$ y $q_2 = \alpha$:

$$\left(\frac{J}{R^2} + m\right) * \ddot{x} + m * g * \sin\alpha - m * x * \dot{\alpha}^2 = 0$$

Para obtener la función de transferencia a partir de una ecuación diferencial, esta tiene que ser obligatoriamente lineal y, en este caso, no lo es. Con lo cual, lo que hay que hacer es linealizarla en torno a un punto de equilibrio estable, que en este caso el único posible es cuando la barra está horizontal, $\alpha = 0$ (ángulo de inclinación de la barra = 0). Cabe recordar que cuando se linealiza una ecuación no lineal, el modelo que se obtiene solo se ajusta bien para valores cercanos al punto de equilibrio.

La ecuación linealizada queda de la siguiente forma, asumiendo que cuando $\alpha \approx 0$ el $\sin\alpha$ se puede aproximar por el ángulo α a $\sin\alpha \approx \alpha$:

$$\left(\frac{J}{R^2} + m\right) * \ddot{x} = -m * g * \alpha$$

La ecuación que relaciona el ángulo de inclinación de la barra con el ángulo de giro del servomotor se puede aproximar como lineal mediante la siguiente ecuación:

$$\alpha = \frac{d}{L} * \theta$$

Sustituyendo esto en la ecuación anterior, se obtiene:

$$\left(\frac{J}{R^2} + m\right) * \ddot{x} = -m * g * \frac{d}{L} * \theta$$

Aplicando, ahora que es lineal, la transformada de Laplace a la ecuación, para pasar de ecuación diferencial a ecuación polinómica que es más fácil de resolver, se obtiene:

$$\left(\frac{J}{R^2} + m\right) * s^2 * X(s) = -m * g * \frac{d}{L} * \theta(s)$$

Como lo que relaciona está función de transferencia de la planta del sistema barra y bola es la posición de la bola en la barra X(s) según el ángulo de giro del servomotor θ(s):

Figura 4.42 Entrada-salida del bloque de la planta.

Siendo G(s) la función de transferencia de la planta de la maqueta:

$$\boxed{G(s) = \frac{X(s)}{\theta(s)} = -\frac{m * g * d}{\left(\frac{J}{R^2} + m\right) * L} * \frac{1}{s^2} \left[\frac{m}{rad}\right]}$$

Como se puede ver, cabe señalar que la función de transferencia de la planta tiene un polo doble en el origen, con lo cual, ya se puede prever, que va a ser críticamente estable, y en cuanto al amortiguamiento (sistema de 2º orden) va a ser críticamente amortiguado, δ = 1, porque tiene un polo doble en el origen. Como el comportamiento

de esta función de transferencia es críticamente estable en bucle abierto, como era de esperar y se había mencionado anteriormente, es necesario cerrar un lazo de control que lo lleve a la estabilidad, aplicando la función de transferencia de un controlador PID.

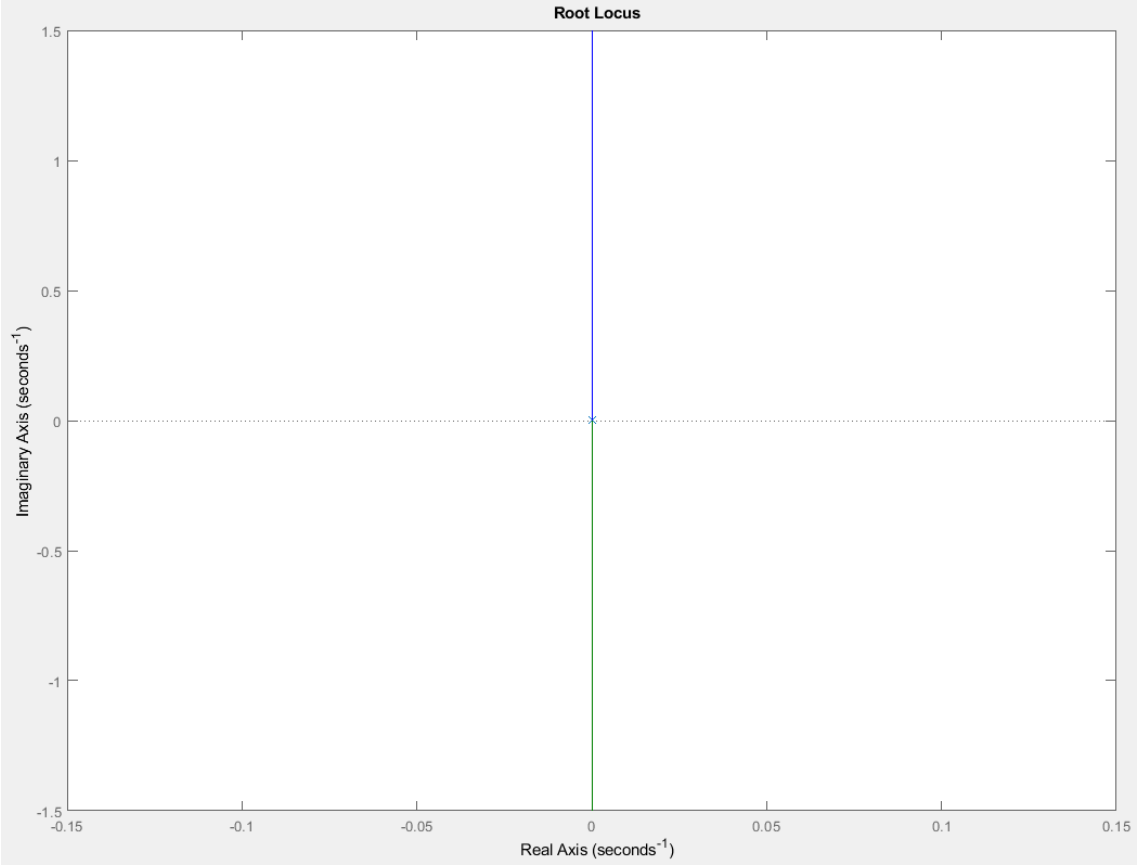


Figura 4.43 Lugar de las raíces de la función de transferencia de la planta.

Aplicando los parámetros físicos de la maqueta anteriormente expuestos, se obtiene la función de transferencia de la planta del sistema:

$$\boxed{\mathrm{G(s)} = \frac{X(s)}{\theta(s)} = \frac{0{,}424269}{s^2} \left[\frac{m}{rad}\right]}$$

Para analizar la estabilidad de esta función de transferencia se va aplicar un método clásico en el cual a partir del modelo se va a aplicar una entrada tipo escalón, y se observará si la salida acaba alcanzando un valor estable en régimen permanente, es decir, para una entrada acotada la salida es acotada.

Para realizar esto, el comando de Matlab que hay que utilizar es *step(G(S))*, el cual automáticamente representa la gráfica de la respuesta, o también se puede realizar mediante Simulink añadiéndole al bloque de la función de transferencia de la planta una entrada escalón.

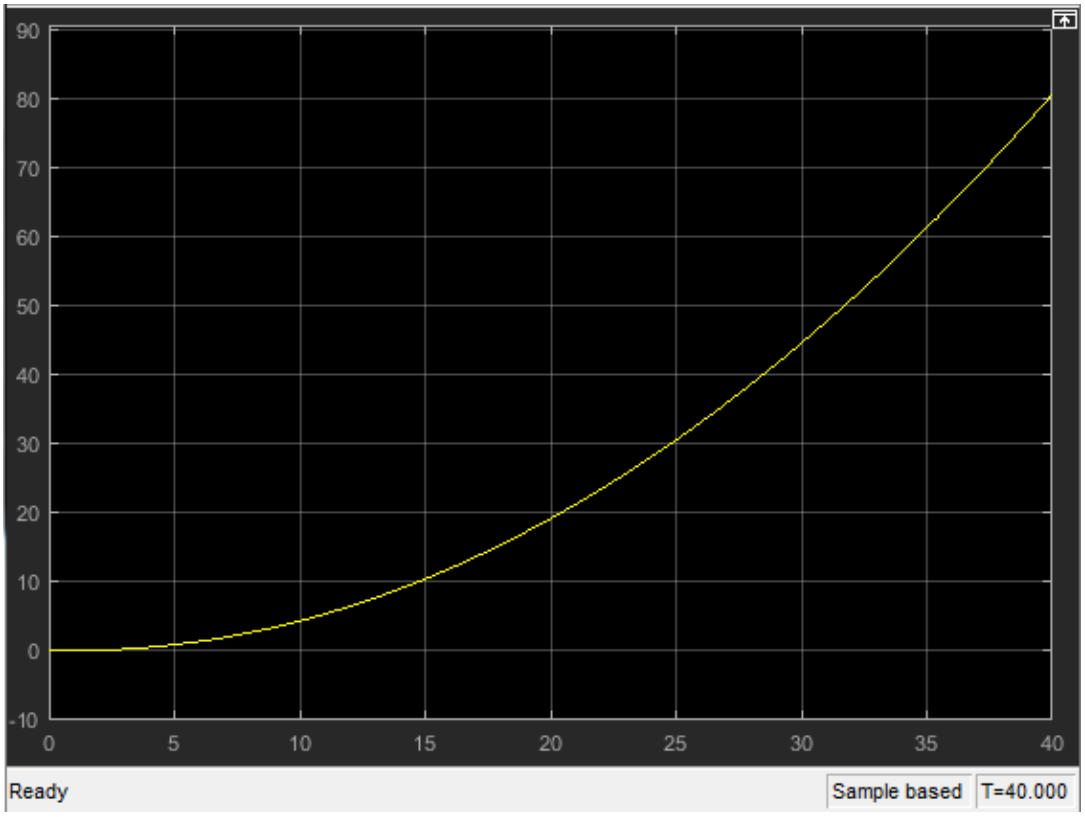


Figura 4.44 Respuesta de la F.T de la planta en lazo abierto ante una entrada escalón.

Como se puede ver la salida tiende a infinito, con lo que se comprueba como se habia mencionado antes que el modelo de la planta no es estable, puesto que tiene un polo doble en el origen. Si se piensa en el sistema físico como tal, puede deducirse que sabiendo que la entrada es la tension del motor y se introduce un valor fijo, la barra girará hacia alguno de los dos lados y la bola rodará sin posibilidad de volver a su punto de equilibrio o reposo.

Con lo cual, como ya se ha adelantado antes, queda clara la necesidad de cerrar un lazo de control y de implementar un controlador PID que estabilice el sistema.

4.5.2 Simulación del sistema

En este apartado se van a obtener las constantes del controlador PID mediante la realización de la simulación del sistema por medio del software Matlab Simulink, las cuales van a cumplir unas especificaciones técnicas impuestas para que el control del sistema corrija de manera eficaz las perturbaciones que se puedan producir en el control de posición de la bola.

En definitiva, lo que se quiere es realizar la simulación de este sistema con el objetivo de simularlo antes de implementarlo en el sistema físico, para obtener una primera aproximación de las constantes del PID, para luego, en el sistema físico tener unos valores de partida mediante los cuales poder realizar un ajuste fino.

En este caso el control del sistema se debe realizar en tiempo discreto (z), puesto que, se utiliza un dispositivo digital (Arduino) para controlar el sistema, como se ha visto en otros apartados.

Como se ha mencionado en el apartado *4.1 Análisis del control del sistema*, los sistemas en los que el control está basado en un procesador digital tienen una parte del sistema que opera en tiempo continuo (s) y otra parte en tiempo discreto (z); como es sabido la parte que opera siempre en tiempo continuo es la planta, puesto que siempre es analógica, sin embargo, como el control del sistema está basado en un procesador digital los controladores operan en tiempo discreto. Dicho esto, el diagrama de bloques del sistema es el siguiente:

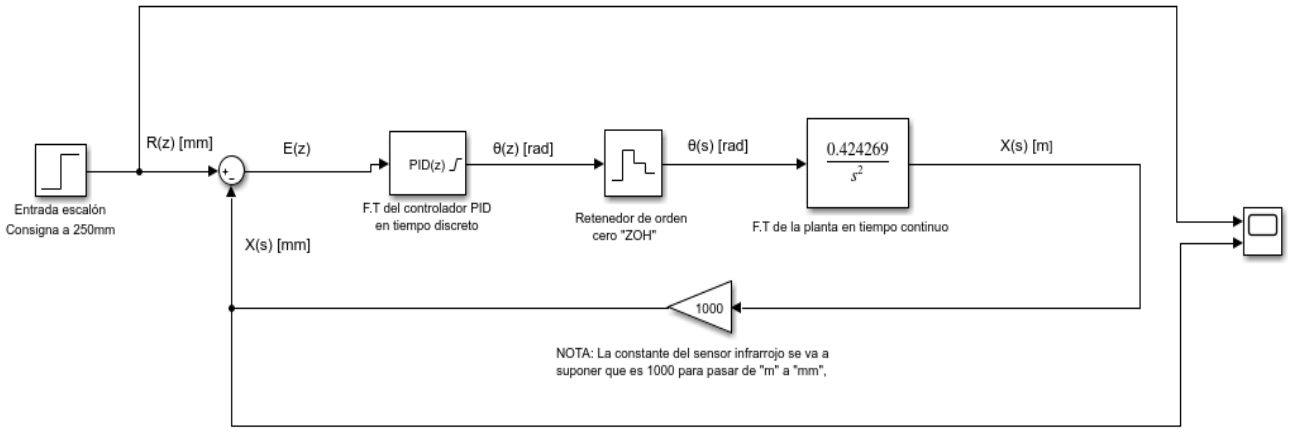


Figura 4.45 Diagrama de bloques del sistema antes de discretizar la planta.

El bloque "ZOH" es un retenedor de orden cero, el cual es el retenedor más sencillo que existe. Un retenedor trabaja como un convertidor digital-analógico, es decir, es un dispositivo que convierte una señal discreta (z) en una señal analógica (s). A este proceso se le denomina reconstrucción de señales. La función de transferencia del retenedor de orden cero (zoh) es:

$$G_{ZOH} = \frac{1 - e^{-Ts}}{s}$$

Como se puede ver en el diagrama de bloques, cabe destacar el bloque del sensor que está en la realimentación del sistema constituido por una constante de 1000, esto se realiza para pasar de "m" a "mm", porque como la función de transferencia de la planta está obtenida en "m" y el valor de la consigna está asignado en "mm" es necesario realizar está conversión de unidades a través del bloque del sensor para poder obtener el error de posición de manera correcta, es decir, error = mm - mm.

Una vez visto esto, es necesario discretizar la función de transferencia de la planta, puesto que para poder realizar la simulación del sistema es imprescindible que el diagrama de bloques entero este en tiempo discreto, porque para controlar el sistema se utiliza un microcontrolador, es decir, un dispositivo digital, con lo cual, hay que discretizar el modelo de la planta para pasarlo a tiempo discreto; para ello, se cogen tanto el modelo de la planta como la función de transferencia del retenedor de orden cero "zoh".

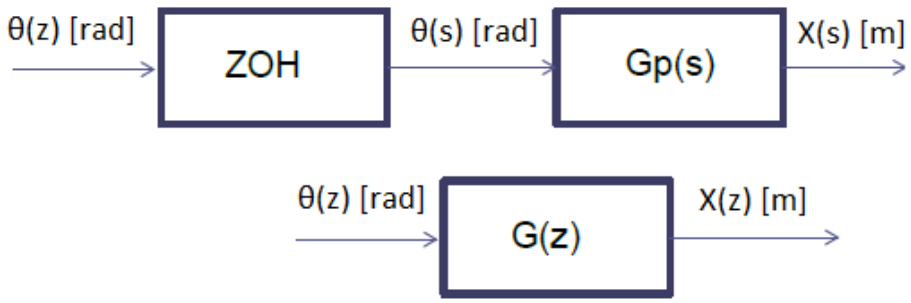


Figura 4.46 Fig. 8. Proceso de discretización de la planta.

Este proceso de discretización del modelo de la planta se ha realizado de dos formas diferentes, una mediante el software Matlab Simulink, y la otra teóricamente, obteniendo el mismo resultado, como se va a ver a continuación:

- Mediante el software Matlab Simulink:

  Para discretizar la planta mediante el software Matlab Simulink, se ha utilizado una función de Matlab denominada "c2d", la cual convierte una función de transferencia continua (s) en una función de transferencia discreta (z) utilizando la operación de retención de orden cero, además de esto también se introduce en esta función el periodo de muestreo seleccionado. La sintaxis básica para esto en Matlab es: *Gd(s) = c2d (G, Ts, 'zoh')*.

  Entonces para discretizar la planta se ha realizado un script en Matlab, en el cual, se ha obtenido la función de transferencia de la planta en tiempo continuo (s) mediante su fórmula, y utilizando esa función de transferencia mediante la función de Matlab "c2d" anteriormente explicada, en la cual se ha introducido el periodo de muestreo seleccionado, Ts = 60ms y como convertidor digital-analógico se ha aplicado el retenedor de orden cero "zoh", con todo esto se ha obtenido la función de transferencia de la planta en tiempo discreto.

```
% Obtención de la función de transferencia de la planta del sistema barra y bola (maqueta)
% EN TIEMPO DISCRETO:
%*************************************************************************************************

pi = 3.141592654; % Valor del número pi, que se utiliza en el bloque de saturación.
m = 0.0411; % masa de la bola [kg]
R = 0.0275; % Radio de la bola [m]
d = 0.03; % Desplazamiento de la biela [m]
g = 9.8; % Aceleración gravitacional (gravedad) [m/s^2]
L = 0.495; % Longitud de la barra [m]
J = 1.243*10^-5; % Momento de inercia de la bola [kg.m^2]
% "x" coordenada de posición de la bola.
% "alfa" coordenada del ángulo de inclinación de la barra.
% "theta" Ángulo de giro del servomotor.

% Aplicación de la función de transferencia de la planta obtenida:

s = tf('s');
Planta = m * g * d / L / (J/R^2 + m) / s^2  % [m/rad]


% Discretización de la función de transferencia de la planta:
Ts = 0.06;  %Periodo de muestreo de 60ms
Planta_discreta = c2d (Planta, Ts, 'zoh' )
```

Figura 4.47 Script con el que se ha discretizado la función de transferencia de la planta.

La función de transferencia de la planta obtenida en tiempo discreto (z) es la siguiente:

$$\text{Planta}_{\text{discreta}}(z) = G(z) = \frac{X(z)}{\theta(z)} = \frac{0{,}0007637z + 0{,}0007637}{z^2 - 2z + 1} \left[\frac{m}{rad}\right]$$

- Discretización realizada teóricamente:

Sabiendo que la función de transferencia de la planta en tiempo continuo y la función de transferencia del retenedor de orden son las siguientes, respectivamente, $G(s) = \frac{X(s)}{\theta(s)} = \frac{0{,}424269}{s^2}\left[\frac{m}{rad}\right]$, $G_{ZOH} = \frac{1 - e^{-Ts}}{s}$. Y sabiendo que el periodo de muestreo del sistema es de Ts = 60ms = 0,06s, se discretiza el modelo de la planta de la siguiente manera:

$$G(z) = z\left[\frac{1 - e^{-Ts}}{s} * G(s)\right]$$

Sabiendo que $z = e^{Ts}$:

$$G(z) = (1 - z^{-1}) * z\left[\frac{1}{s} * \frac{0{,}424269}{s^2}\right]$$

Se realiza el método de expansión en fracciones parciales, y se obtiene:

$$G(z) = \frac{z-1}{z} * z\left[\frac{0{,}424269}{s^3}\right]$$

Utilizando la tabla de la transformada z, se transforma de tiempo continuo (s) a tiempo discreto (z):

$$G(z) = \frac{z-1}{z} * 0{,}424269 * \left[\frac{T^2 .\ z(z+1)}{2 .\ (z-1)^3}\right] \rightarrow$$

$$\rightarrow G(z) = 0{,}424269 * \left[\frac{T^2 .\ (z+1)}{2 .\ (z-1)^2}\right] = 0{,}2121345 * \left[\frac{T^2 .\ (z+1)}{(z-1)^2}\right]$$

Aplicando el periodo de muestreo del sistema, Ts = 0,06s queda de la siguiente manera:

$$G(z) = \left[\frac{0{,}0007637\ .\ (z+1)}{(z-1)^2}\right]$$

Con lo cual, la función de transferencia de la planta en tiempo discreto (z) es la siguiente:

$$\boxed{G(z) = \frac{X(z)}{\theta(z)} = \frac{0{,}0007637z + 0{,}0007637}{z^2 - 2z + 1}\ \left[\frac{m}{rad}\right]}$$

Como se ha podido ver, se ha obtenido el mismo resultado en ambas formas, con lo cual, se pude decir que el modelo de la planta en tiempo discreto (z) es el siguiente:

$$\boxed{G(z) = \frac{X(z)}{\theta(z)} = \frac{0{,}0007637z + 0{,}0007637}{z^2 - 2z + 1}\ \left[\frac{m}{rad}\right]}$$

Una vez discretizada la planta, se consigue que el diagrama de bloques entero este en tiempo discreto (z), con lo cual, gracias a discretizar la planta se puede realizar la simulación del sistema. El diagrama de bloques utilizado para realizar la simulación del sistema mediante la plataforma Simulink, es el siguiente:

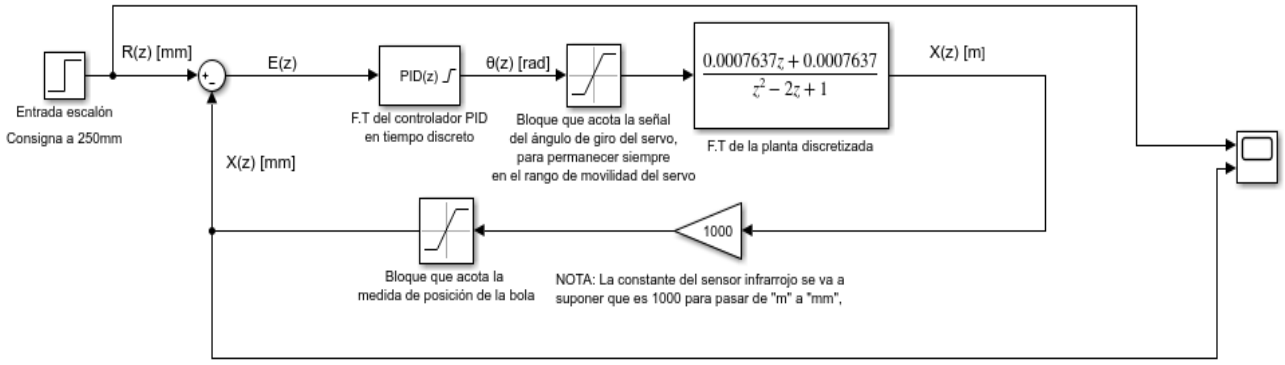


Figura 4.48 Diagrama de bloques utilizado para realizar la simulación del sistema.

Ahora se va a explicar el diagrama de bloques utilizado en la simulación, bloque a bloque:

- Bloque de consigna: Como consigna se ha utilizado un bloque de entrada escalón de 250mm por defecto, puesto que más o menos es el centro de la barra.

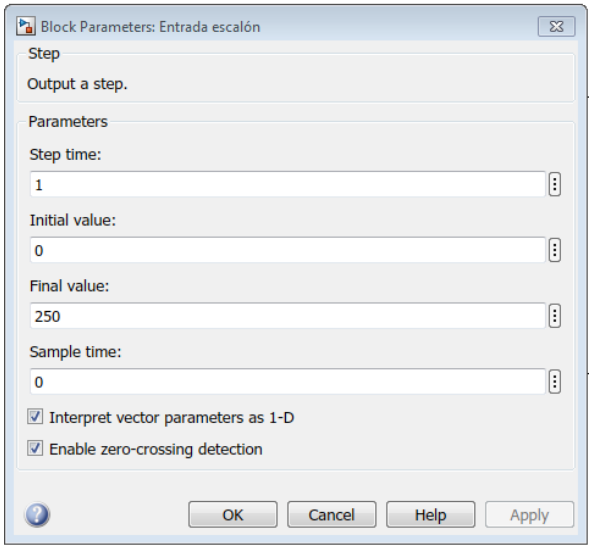


Figura 4.49 Bloque consigna.

- Bloque PID: Se ha establecido el bloque del controlador PID paralelo en tiempo discreto (z), discretizado por medio del método de Euler hacia atrás (Backward Euler), con un periodo de muestreo de 60ms establecido en la programación Arduino.

  La función de transferencia del controlador PID en tiempo discreto basándose en el método de aproximación numérica de Euler hacia atrás, se obtiene de la siguiente manera, como se ha visto en el apartado *4.4.4 Implementación del controlador PID en Arduino*:
  Sabiendo que la ecuación característica de un controlador PID es la siguiente:

$$u(t) = K_P * e(t) + K_I * \int_0^t e(t)dt + K_D * \frac{de(t)}{dt}$$

  Y la función de transferencia en tiempo continuo (s) es:

$$G_{PID}(s) = \frac{U(s)}{E(s)} = K_P + K_I * \frac{1}{s} + K_D * s$$

  Entonces se va a discretizar la integración basándose en el método de Euler hacia atrás:

$$u(t) = \int_0^t e(t)dt \rightarrow u_{nT} = T \sum_{i=1}^{n} e_{iT} \rightarrow u_{nT} = T \sum_{i=1}^{n-1} e_{iT} + Te_{nT} \rightarrow$$

Se obtiene la ecuación en diferencias de la integral basándose en el método de Euler hacia atrás:

$$u_{nT} = u_{(n-1)T} + Te_{nT}$$

Aplicando a esta ecuación en diferencias la transformada Z se obtiene:

$$U(z) = U(z) * z^{-1} + T * E(z)$$

Con lo cual:

$$\frac{U(z)}{E(z)} = \frac{1}{s} = \frac{T}{1 - z^{-1}} \rightarrow s = \frac{z-1}{T \, . \, z}$$

Discretizando el término diferencial por el método de Euler hacia atrás, el resultado es exactamente igual a la discretización del término integral:

$$u(t) = \frac{de(t)}{dt} \rightarrow u(t) = \frac{e_{nT} - e_{(n-1)T}}{T} \rightarrow \frac{U(z)}{E(z)} = s = \frac{z-1}{T \, . \, z}$$

Con lo cual, extrapolando lo obtenido en la discretización del término integral y del término derivativo a la función de transferencia del controlador PID, se obtiene la función de transferencia del controlador PID en tiempo discreto (z) utilizando el método de aproximación numérica de Euler hacia atrás:

$$G_{PID}(z) = \frac{U(z)}{E(z)} = \frac{\theta(z)}{E(z)} = K_P + K_I * \frac{T \, . \, z}{z-1} + K_D * \frac{z-1}{T \, . \, z}$$

- Bloque de la planta: Se ha establecido el bloque de la función de transferencia de la planta discretizada, aplicando un periodo de muestreo de 60ms (el establecido en la programación Arduino), se ha realimentado para realizar el control en lazo cerrado.

  La función de transferencia de la planta en tiempo discreto (z) introducida en el bloque, es la siguiente:

$$G(z) = \frac{X(z)}{\theta(z)} = \frac{0{,}0007637z + 0{,}0007637}{z^2 - 2z + 1} \left[\frac{m}{rad}\right]$$

Se puede ver como se ha introducido en el bloque en la siguiente figura:

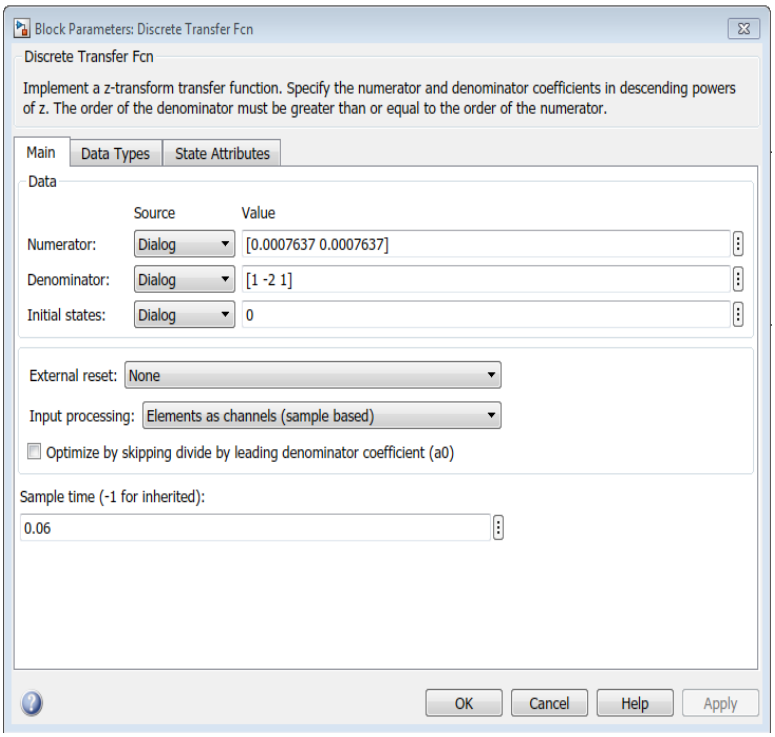


Figura 4.50 Bloque planta.

- Bloque del sensor: El bloque del sensor está en la realimentación del sistema constituido por una constante de 1000, como se ha explicado anteriormente esto se realiza para pasar de "m" a "mm", porque como la función de transferencia de la planta está obtenida en "m" y el valor de la consigna está asignado en "mm" es necesario realizar está conversión de unidades a través del bloque del sensor para poder obtener el error de posición de manera correcta, es decir, error = mm - mm.

- Bloque del scope: Se ha añadido un "scope" para poder visualizar la señal de referencia junto con la señal de medida del sensor para poder obtener conclusiones acerca del control del sistema.

Una vez comentado esto, se va a explicar los bloques de saturación establecidos en el diagrama de bloques de simulación, y el anti wind-up impuesto al bloque del PID:

- Para habilitar el anti wind-up en el bloque del controlador PID se va a la pestaña "PID Advanced" del diálogo del bloque PID y se selecciona limitar salida para

poder establecer los límites de saturación del actuador del sistema. El método anti wind-up utilizado es "clamping". Lo explicado anteriormente se puede ver en la siguiente figura:

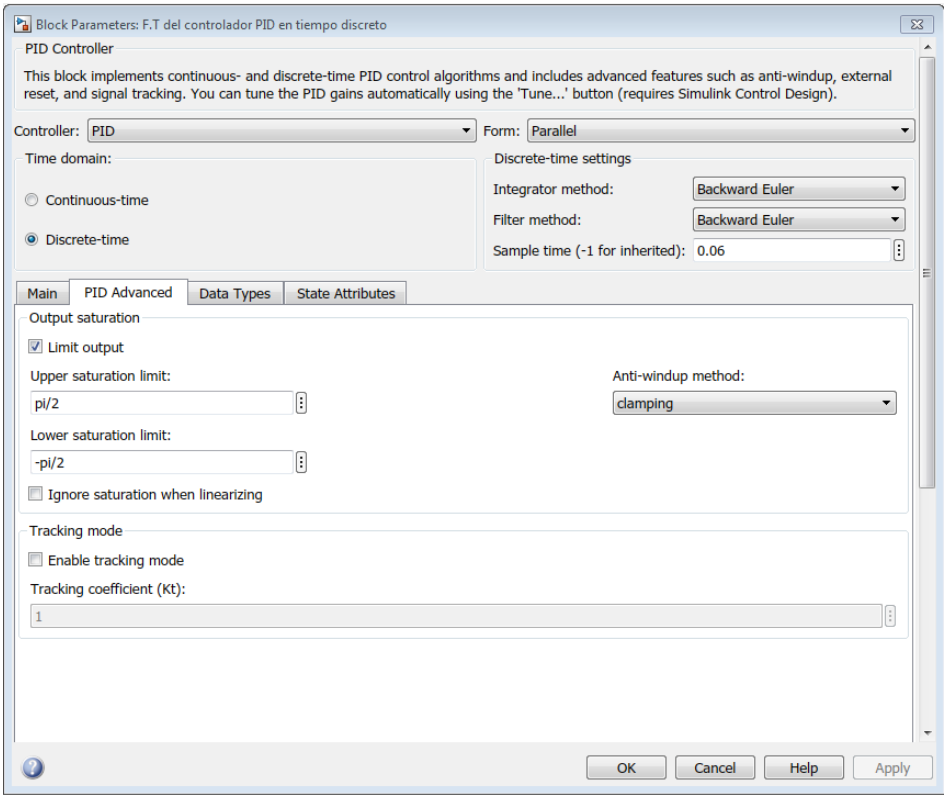


Figura 4.51 Pestaña del bloque PID, en la cual se habilita el anti wind-up.

En el anti wind-up se han introducido los límites de saturación del actuador del sistema, que en este caso, van a estar provocados por el ángulo de giro del servomotor, debido a que solo puede girar 90º en un sentido y en el otro. Esto se debe poner en radianes, puesto que la señal de salida del PID (señal del ángulo de giro del servo θ(z)) está en radianes. Los valores introducidos como los límites de saturación del servomotor en radianes son los siguientes:

$$\left(-\frac{\pi}{2} \div \frac{\pi}{2}\right) \text{[rad]}$$

Los límites de saturación del ángulo de giro del servomotor van a ser de -90º a +90º, puesto que cuando empieza el control del sistema a ejecutarse la barra se sitúa en su punto de equilibrio (horizontal), es decir, el ángulo de giro del servomotor se sitúa a 90º (en la simulación esta situación, se supone con un

valor de 0), por eso, el servomotor solo se va a poder mover 90º en un sentido y en el otro, con lo cual, los límites de saturación del ángulo de giro del servomotor van a ser de -90º a +90º, puesto que se sabe que el rango de movilidad del servomotor utilizado es de 0º a 180º.

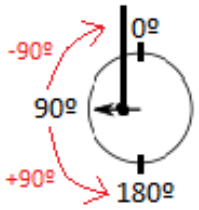


Figura 4.52 Límites de saturación del ángulo de giro del servomotor utilizado.

- Se ha establecido un bloque de saturación a la salida del PID, puesto que en la programación Arduino se acota el ángulo de giro del servo en su rango de disponibilidad, esto es, de 0º a 180º, después de calcular la señal de control y restarla o sumarla al valor de reposo de la barra, es decir, donde se calcula el duty cycle de la señal PWM del ángulo de giro del servo, como se puede ver en esta instrucción de la programación software de Arduino:

DC = constrain(DC,500,2300);

Esto se realiza para que la señal de salida del PID, o lo que es lo mismo, para que la señal del ángulo de giro del servo (θ(z)) este siempre dentro del rango de movilidad del servo. Como se ha dicho antes en el anti wind-up, los límites de saturación del ángulo de giro del servomotor van a ser de -90º a +90º, con lo cual, en radianes que es como se debe establecer los límites en el bloque de saturación, puesto que la señal del ángulo de giro del servo θ(z) está en radianes, van a ser de $-\frac{\pi}{2} \div \frac{\pi}{2}$ [rad] como se ha visto en el anterior bloque, y se puede ver en la siguiente figura:

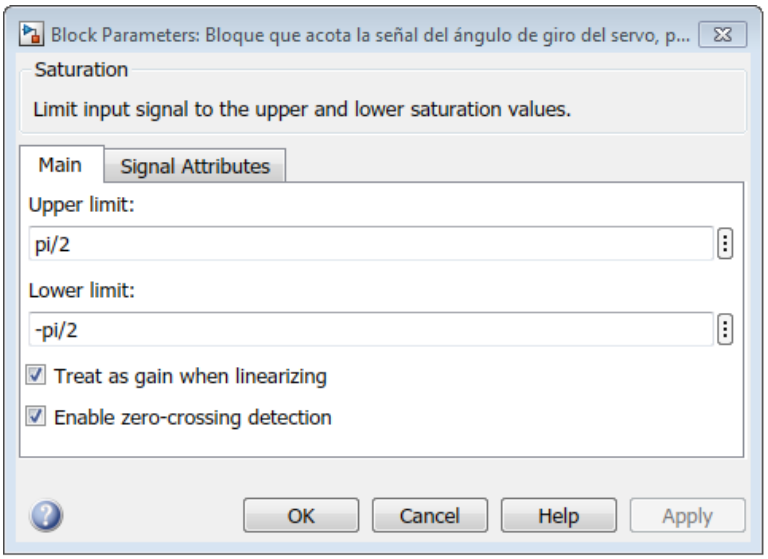


Figura 4.53 Bloque de saturación a la salida del PID.

- Por último, se ha puesto un bloque de saturación a la salida de la planta, es decir, después de que realice la medición el sensor, puesto que en la programación Arduino se realiza esto para que las medidas que se obtengan estén dentro del rango de medida optimo del sensor infrarrojo y de la barra, es decir, son las medidas mínima y máxima que se utilizaron para realizar la linealización del sensor, como se puede ver en la siguiente instrucción de la programación software de Arduino:

  int dist_cal [] = {100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450};

  Y una vez calculada la posición de la bola, se acota entre el mínimo y el máximo valor del array de las distancias, utilizadas para realizar la linealización del sensor:

  posicion = constrain(posicion,dist_cal[0],dist_cal[14]);

  Por todo ello, se pone un bloque de saturación después de la conversión del bloque del sensor de “m” a “mm” antes de llegar al cálculo del error de posición. Como se ha visto en la programación Arduino las medidas de posición de la bola

hay que acotarlas entre 100 y 450mm, como se puede ver en el diálogo del bloque de saturación:

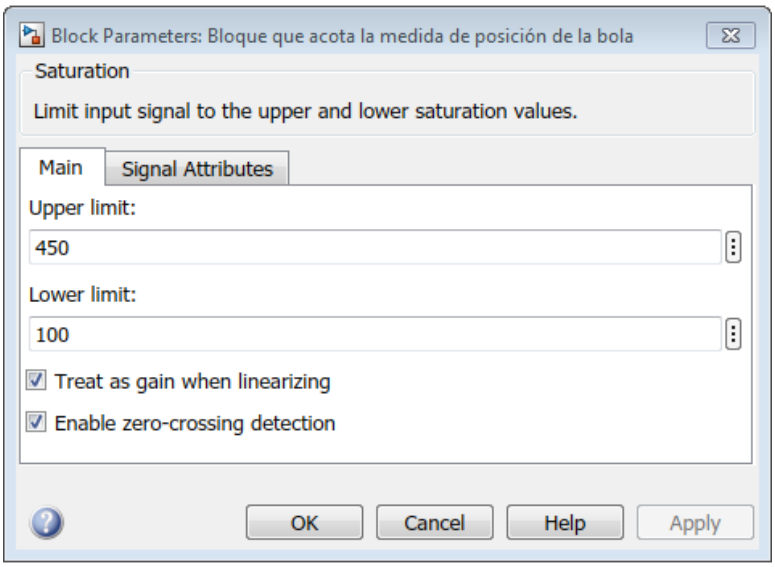


Figura 4.54 Bloque que satura la medida de posición de la bola.

Una vez analizado todo el diagrama de bloques de la simulación, se pasa a obtener las constantes discretas que realicen un control óptimo del sistema, para ello se va al bloque del controlador PID discreto y mediante la aplicación "PID tuning tool" se consiguen unas constantes discretas que realizan el control de posición de la bola de manera óptima.

Para obtener esas constantes de manera adecuada se han establecido unas especificaciones técnicas, las cuales se han establecido viendo las necesidades del control del sistema:

- Rebose < 10% (régimen transitorio) → Que el control del sistema no sea brusco.
- Tiempo de establecimiento < 5seg. (régimen transitorio) → Que el control sea rápido.
- Error en estado estacionario mínimo (régimen permanente) → Es la característica fundamental que debe cumplir este sistema, el error de posición del sistema tiene que ser mínimo, es un sistema que requiere de precisión para establecer la bola en la posición de referencia deseada.

Entonces se quieren obtener unas constantes discretas que cumplan con las especificaciones técnicas impuestas, para ello, se va a la aplicación "PID tuning tool" del bloque del PID y dentro de ella se van moviendo las 2 barras de la parte superior, tanto la del tiempo de respuesta como la del comportamiento transitorio, hasta que se encuentren unas constantes discretas que cumplan con las especificaciones técnicas impuestas para este sistema. En este caso el resultado es el siguiente:

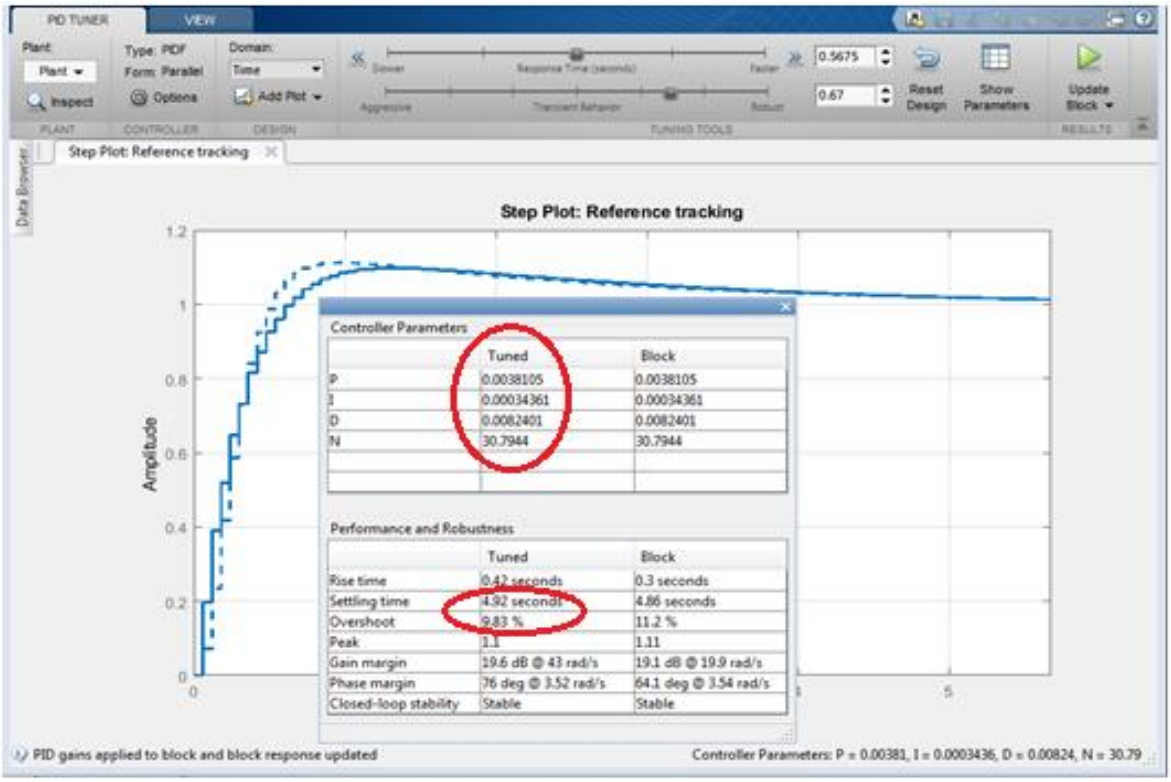


Figura 4.55 Constantes discretas que cumplen las especificaciones técnicas impuestas.

Las constantes discretas obtenidas para el cumplimiento de las especificaciones técnicas establecidas para el control del sistema, son las siguientes como se puede ver en la figura anterior:

- $K_P = 0{,}0038105$
- $K_I = 0{,}00034361$
- $K_D = 0{,}0082401$

Estas constantes son las que se establecen en el bloque del controlador PID discreto, para realizar la simulación del sistema:

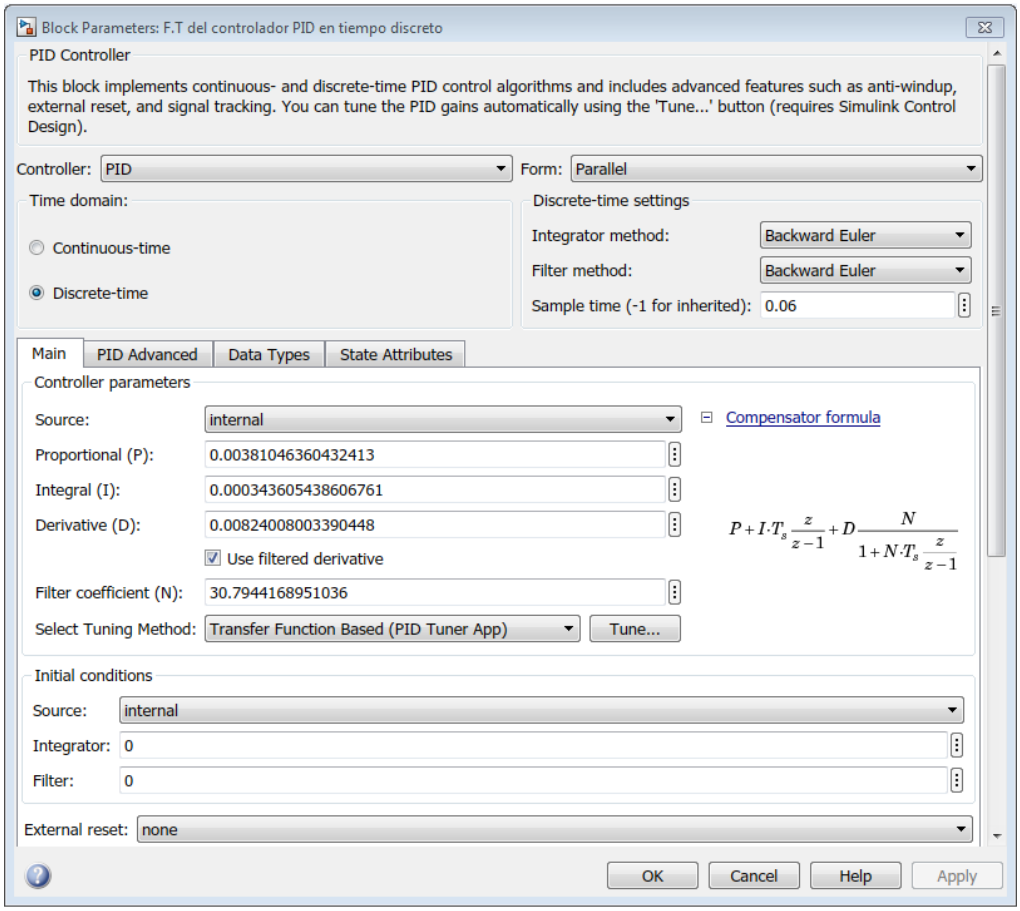


Figura 4.56 Bloque del PID discreto con las constantes discretas establecidas.

Como se puede ver el bloque del controlador PID paralelo en tiempo discreto (z), está discretizado por medio del método de Euler hacia atrás (Backward Euler), con un periodo de muestreo de 60ms establecido en la programación Arduino.

Con todo esto se realiza la simulación del sistema barra y bola obteniendo los siguientes resultados:

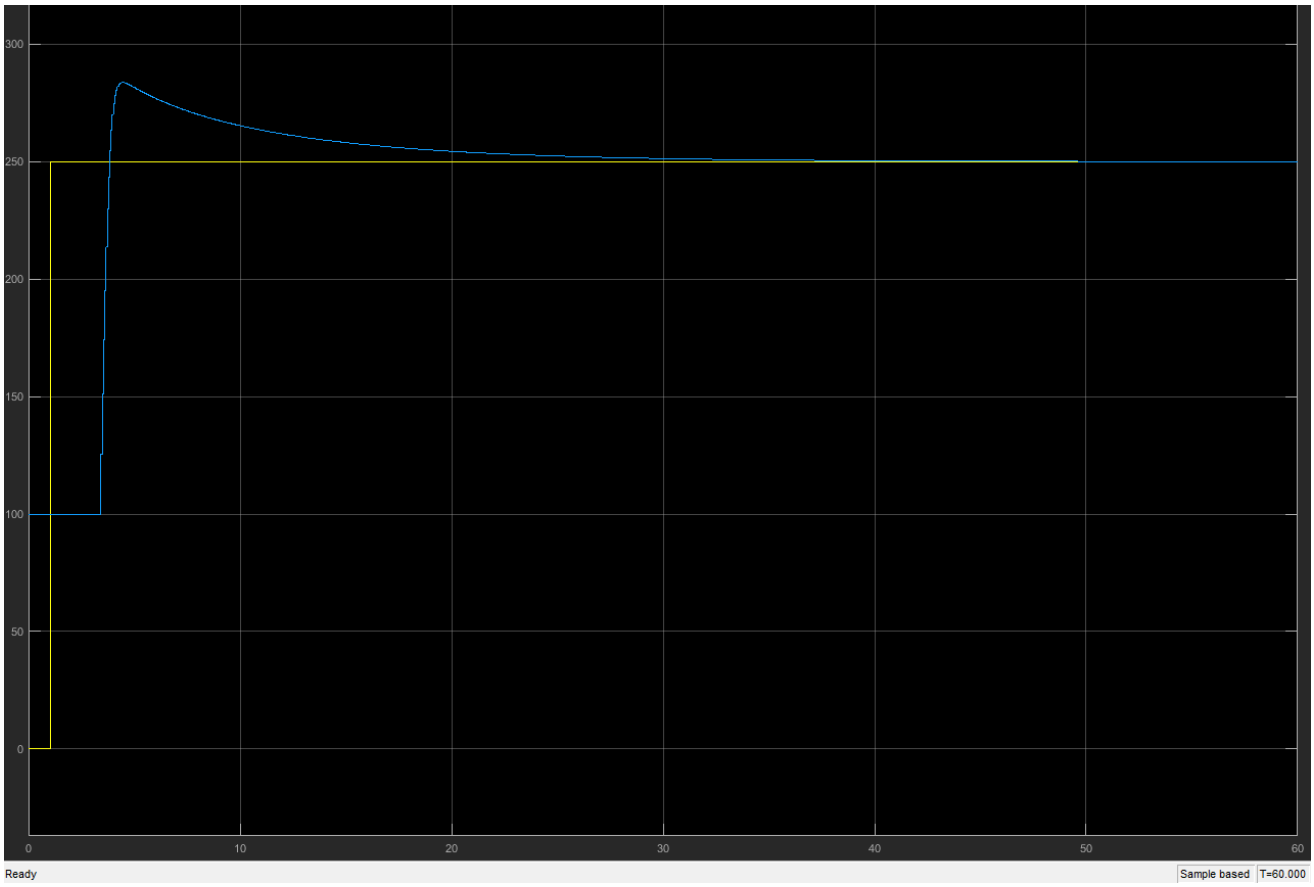


Figura 4.57 Señal de salida del diagrama de bloques en tiempo discreto.

Como se puede ver en la gráfica, el comportamiento del sistema es óptimo, se puede ver como el control del sistema no es brusco, puesto que el rebose es lo suficientemente pequeño, y como la señal de salida (azul) respecto a la señal de referencia (amarilla) se corrige con el paso del tiempo hasta conseguir que el error de posición de la bola sea prácticamente nulo, se puede decir que la simulación realizada tiene un comportamiento aceptable.

4.5.3 Comparación de la respuesta del sistema al implementar las constantes obtenidas en la simulación, en la maqueta

Al probar estos valores del controlador PID en tiempo discreto (z) obtenidos mediante la simulación, en la maqueta, se puede ver como la maqueta prácticamente no reacciona, puesto que los valores obtenidos del PID son demasiado pequeños, con lo cual, se puede

intuir que la respuesta de la simulación solo es posible con el modelo y que no es físicamente realizable por la maqueta.

Se puede decir, que las simulaciones son adecuadas para hacerse una idea de cómo va a reaccionar el sistema y para obtener una primera aproximación de los parámetros de los controladores, en este caso sin éxito, pero que lo mejor es obtener las constantes del controlador PID discreto, probando en la propia maqueta, realizando el método de ajuste por ensayo-error hasta que se obtengan unas constantes que realicen de manera adecuada el control de posición de la bola con un error de posición mínimo. Al final una simulación nunca te va a dar el valor exacto, siempre va a dar una ligera aproximación, porque las simulaciones nunca se van a parecer a la realidad, solamente se pueden asemejar un poco, puesto que en los sistemas hay muchas variables que la simulación no las contempla, como por ejemplo, el uso y funcionamiento de los diferentes componentes que conforman el sistema.

Es importante tener en cuenta, que a la hora de obtener el modelado de la planta se asumieron varias simplificaciones y además se tuvo que linealizar la ecuación de la posición de la bola en torno a un punto de equilibrio estable, en este caso cuando la barra está en reposo, horizontal; con lo cual, cuanto más se aleje la bola del punto de equilibrio menos se parecerá el modelo al sistema físico real.

Por todo esto, en la simulación realizada como era de esperar, no se obtienen los valores aproximados de las constantes, puesto que, al obtener el modelado de la planta se realizaron varias simplificaciones y aproximaciones lineales en la ecuación obtenida con la coordenada de la posición de la bola, las cuales, se tuvieron que realizar para poder obtener el modelado porque si no de lo contrario, las ecuaciones se volvían demasiado complejas como para obtener el modelado. Al realizar las simplificaciones y al linealizar la ecuación, el modelo de la planta obtenido pierde suficiente semejanza con el de la vida real, por este motivo, las constantes obtenidas en la simulación no arrojan buenos resultados en el sistema real.

De todas formas, hay una posibilidad que se ha contemplado para obtener el modelado del sistema completo, y a partir de este, sacar las constantes que realizan un control de posición de la bola óptimo, pero no se ha llevado a cabo por falta de tiempo, además de

que en los ordenadores del laboratorio no está instalada esta aplicación de Matlab necesaria para ello. La aplicación que se debe utilizar para esto es la aplicación de Matlab denominada "System identification toolbox", la cual como se ha dicho no está instalada en los ordenadores del laboratorio. Con esta aplicación lo que se pretende es obtener el modelo dinámico del sistema completo, partiendo de los datos experimentales de entrada y salida extraídos del funcionamiento de la maqueta, y así, poder mediante esta aplicación "System identification toolbox" identificar el modelo del sistema completo. Por medio de este modelo se podrían extraer las constantes discretas del controlador PID que consiguen realizar un control de posición de la bola óptimo.

4.6 Obtención de las constantes en tiempo discreto y análisis de diferentes tipos de reguladores PID

4.6.1 Obtención de las constantes en tiempo discreto

En este apartado se va a explicar cómo se han obtenido las constantes discretas del controlador PID paralelo que realizan el control de posición de la bola de la mejor manera posible.

Como se ha visto en el anterior apartado *4.5.3 Comparación de la respuesta del sistema al implementar las constantes obtenidas en la simulación, en la maqueta*, a la hora de ajustar los controladores se ha tenido problemas, puesto que las constantes obtenidas con la simulación del modelado no arrojaban buenos resultados a la hora de la implementación en la maqueta. Por este motivo se han ajustado las constantes de los controladores, por medio, del método de ajuste por ensayo-error.

La mejor manera para obtener las constantes del controlador PID discreto, una vez los valores obtenidos en la simulación no muestran buenos resultados, es probando en la propia maqueta, realizando el método de ajuste por ensayo-error (probando con diferentes valores de las constantes y viendo cómo se comporta el sistema) hasta que se obtengan las constantes que realizan de manera óptima el control de posición de la bola.

El método de ajuste por ensayo-error consiste en modificaciones sucesivas de los parámetros de control hasta conseguir las especificaciones deseadas. Presenta el inconveniente, de que en algunas ocasiones se necesita invertir un tiempo considerable en el ajuste de dichas constantes.

El objetivo es ajustar los tres parámetros, es decir, que los tres parámetros se complementen para lograr que el bucle de control corrija eficazmente, con la mayor estabilidad posible, en el mínimo tiempo y con un error mínimo los efectos de la perturbación.

El proceso seguido para ajustar las constantes mediante el método de ajuste por ensayo-error aplicado en la maqueta, ha sido el siguiente:

- Primero se prueba solo, con la constante proporcional:

  La constante proporcional, K_P, controla la posición de la bola, es decir, cuanto más lejos este la bola del valor deseado de posición más inclinará la barra para acercar la bola al valor deseado, la salida del sistema con solamente la constante proporcional es oscilatoria. Para asignar un valor a K_P se comienza asignándole un valor cualquiera, y se observa el comportamiento del sistema, se puede ver como el sistema inclina la barra en el sentido adecuado:

  - K_P = 1. Demasiado lento. Con poco efecto en las proximidades de la consigna (consigna establecida a 250mm, más o menos el centro de la barra).
  - K_P = 5. Demasiado rápido y brusco, es decir, demasiado rebose y poco amortiguado. Acelera demasiado la bola.
  - K_P = 3 o K_P = 4. Son los valores adecuados para este sistema. Ni demasiado lento ni demasiado rápido.

  Como se busca el valor más pequeño que sea suficiente para inclinar la barra cuando la bola está cerca de la consigna, el valor adecuado es 3. Como se sabe, solo con el término proporcional, la bola nunca se estabiliza.

Se puede ver que si se aumenta el término proporcional el control del sistema se hace más rápido, pero pierde estabilidad, puesto que aumenta el rebose (control más brusco).

- Ahora se prueba solo, con la constante derivativa:

  La constante derivativa, K_D, opera sobre la diferencia de posiciones entre el ciclo actual y el anterior, es decir, controla la velocidad de la bola. Lo que hace es inclinar la barra oponiéndose a la velocidad de la bola, es decir, cuanto más rápido se mueve la bola más se inclina la barra para detenerla; esto se produce, puesto que la acción derivativa predice que va a producirse un rebose y lo ataja antes de que suceda.

  Para asignar un valor a K_D se comienza asignándole un valor cualquiera, y se observa cómo reacciona la maqueta, si detiene la bola de manera rápida o no:

  - $K_D = 5$. No reacciona lo suficiente como para detener la bola, es un valor insuficiente.
  - $K_D = 30$. No reacciona, lo mismo que antes algo mejor pero sigue sin reaccionar bien.
  - $K_D = 50$. Detiene la bola pero de manera muy lenta, es un valor insuficiente.
  - $K_D = 150$. Sobre-reacciona y hace al sistema inestable, oscila la barra de un lado para el otro de manera infinita, con lo cual, nunca detiene la bola. Valor demasiado alto.
  - $K_D = 100$. Reacciona bastante bien, detiene la bola de manera rápida cuando coge velocidad. Es el valor adecuado para este sistema.

  Se puede ver cómo según se aumenta el término derivativo la maqueta reacciona de manera más rápida deteniendo la bola antes. Si se introduce un valor excesivo en el término derivativo se puede ver como el sistema oscila de manera infinita, es decir, nunca detiene la bola, sistema inestable.

- Ahora se prueba la acción conjunta de los 2 términos seleccionados anteriormente, $K_P = 3$ y $K_D = 100$:

El objetivo es detener la bola en el valor deseado de posición. Como se ha visto anteriormente, la constante proporcional inclina más la barra cuanto más lejos este la bola del valor deseado, para acercar la bola a ese valor, en cambio, la constante derivativa inclina más la barra cuanto más rápido se mueve la bola para detenerla debido a que se anticipa a un error futuro antes de que se haga muy grande e incontrolable, como se ha explicado en el apartado *3.5.3 Controlador PID*.

Al probar este control PD con las constantes seleccionadas, se puede ver como la bola se detiene cerca del valor deseado de posición, el término derivativo da estabilidad al sistema, sin embargo sigue habiendo un error de posición que hay que subsanar (error en régimen permanente). En este caso con el control PD aplicado, no se subsana, puesto que, cuando la bola se detiene cerca de la consigna, la velocidad es 0, con lo que el término diferencial no actúa, y como está cerca de la consigna el término proporcional inclina la barra de manera muy sutil y no es suficiente como para que la bola se mueva.

La solución para eliminar ese error de posición de la bola es añadir el término integral a lo visto anteriormente.

- Ahora se prueba añadiendo un término integral, para realizar un controlador PID, y así poder reducir el error de posición que tenía en el régimen permanente cuando se ha aplicado un control PD:

  La característica fundamental de la constante integral, K_I, es que elimina el error en estado estacionario, lo cual es muy importante en este sistema, puesto que el objetivo de control más destacado de este sistema es la precisión. El término integral considera la posición de la bola y cuanto tiempo lleva allí, cuanto más tiempo pase alejado del valor de la consigna más se irá incrementando el término integral debido a que funciona como un acumulativo del error, e inclinando cada vez más la barra con el paso del tiempo, hasta que reduzca lo máximo posible el error de posición.

  Para asignar un valor a K_I se comienza asignándole un valor cualquiera, y se observa cómo reacciona la maqueta, se sabe que normalmente el término integral es un valor muy pequeño:

- $K_P = 3$, $K_D = 100$, $K_I = 0{,}1$. Lo hace bien pero de manera muy brusca. Añadiendo este término con este valor, el rebose del sistema aumenta, con lo cual pierde estabilidad.
- $K_P = 3$, $K_D = 100$, $K_I = 1$. La maqueta reacciona mal, sistema inestable.
- $K_P = 3$, $K_D = 100$, $K_I = 0{,}05$. La maqueta reacciona bien, menos bruscos los movimientos, y el error disminuye.
- $K_P = 3$, $K_D = 100$, $K_I = 0{,}03$. La maqueta reacciona muy bien, buen equilibrio entre brusquedad y suavidad, con lo cual, sistema rápido y estable con un error mínimo. Valores adecuados para este sistema.

Se puede ver cómo según se aumenta la constante integral, K_I, aumenta el rebose del sistema, con lo cual, la maqueta realiza movimientos cada vez más bruscos llegando a inestabilizar el sistema, por otra parte, cuanto más término integral se le ponga al sistema más preciso será, es decir, el error de posición será prácticamente nulo. Por eso en un controlador PID es fundamental que las 3 acciones del controlador (K_P, K_I y K_D) se complementen de la mejor manera posible, para realizar un control óptimo del sistema.

Por último cabe destacar, que una vez obtenidas las constantes se realizó un pequeño ajuste fino en una de las acciones, más específicamente en la acción proporcional estableciendo un valor de 4 en vez de 3, puesto que realizando ensayos en la maqueta se pudo ver qué era lo más oportuno para el control del sistema.

Finalmente, las constantes discretas del controlador PID paralelo que se han introducido en la programación software de Arduino, son:

- ✓ $K_P = 4$
- ✓ $K_I = 0{,}03$
- ✓ $K_D = 100$

La combinación de estas constantes consigue realizar un control de posición de la bola óptimo, con un error de posición mínimo que es lo que se quiere, esto se ha obtenido

realizando diferentes ensayos en la maqueta para obtener el mejor control posible. Hay que decir que no solo hay una combinación de las constantes que realicen un control óptimo, sino que hay varias.

El comportamiento del sistema implementando estas constantes ante una perturbación es el siguiente:

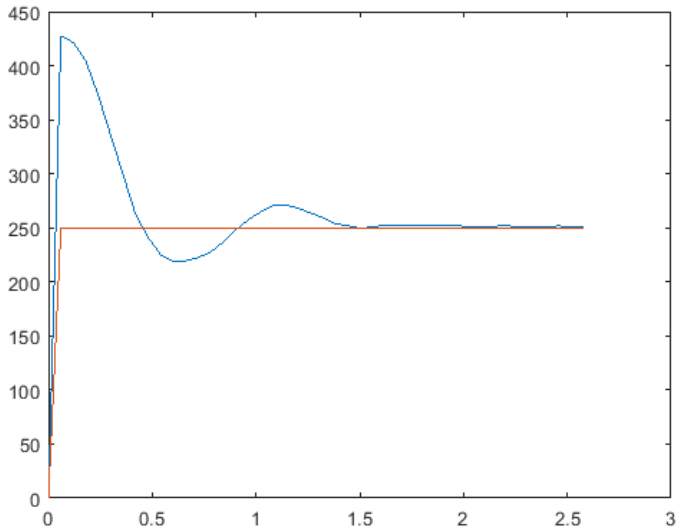


Figura 4.58 Comportamiento del sistema ante una perturbación.

Se puede ver como el control del sistema lleva la bola a la posición deseada, en este caso a 250mm, de manera rápida y amortiguada, con un rebose pequeño del 10% ya que el rebose es igual a 100*(275-250)/250 = 10%, es decir, la maqueta no va a realizar movimientos bruscos, y con un error de posición mínimo. De este comportamiento se puede extraer que corrige la perturbación de manera rápida, puesto que el tiempo de establecimiento está en torno a 1,6 segundos. Como se ha dicho antes, en régimen permanente el error en estado estacionario es mínimo. En la gráfica, la posición de la bola es la línea azul y la consigna, es decir, el valor deseado de posición de la bola es la línea roja. Hay que decir, que dependiendo del efecto de la perturbación, curva inicial en la figura, el tiempo de establecimiento y el rebose pueden ser diferentes a lo visto antes en el sistema de control.

Con todo esto se puede decir, que el control realizado es un control rápido, bastante estable gracias al filtro que se aplica al cálculo de la acción derivativa y con buena precisión, puesto que, el error es pequeño. Se puede decir que la acción

proporcional es la que le da rapidez al control, la acción integral es la que le da precisión, y por último, la acción derivativa es la que le da estabilidad al control del sistema.

4.6.2 Análisis de diferentes tipos de reguladores PID

En este apartado se van a exponer los diferentes tipos de reguladores PID que existen. Además, se va a explicar cómo se ha implementado el controlador PID en serie en la programación software de Arduino.

Como se sabe hay tres topologías principales en un controlador PID que son: la topología paralela, la ideal y la topología serie.

- Controlador PID en paralelo, es la configuración PID que más se suele utilizar:

  El controlador PID en paralelo se ha implementado en la programación software de Arduino como algoritmo de control del sistema.
  La ecuación del controlador PID en paralelo es la ecuación más característica y más conocida de un controlador PID:

$$u(t) = K_P * e(t) + K_I * \int_0^t e(t)dt + K_D * \frac{de(t)}{dt}$$

  Aplicando la transformada de Laplace a la ecuación del controlador PID en paralelo, se obtiene la siguiente función de transferencia:

$$F.T = \frac{U(s)}{E(s)} = K_P + K_I * \frac{1}{s} + K_D * s$$

  Con lo cual, el diagrama de bloques de la topología paralela del PID es el siguiente:

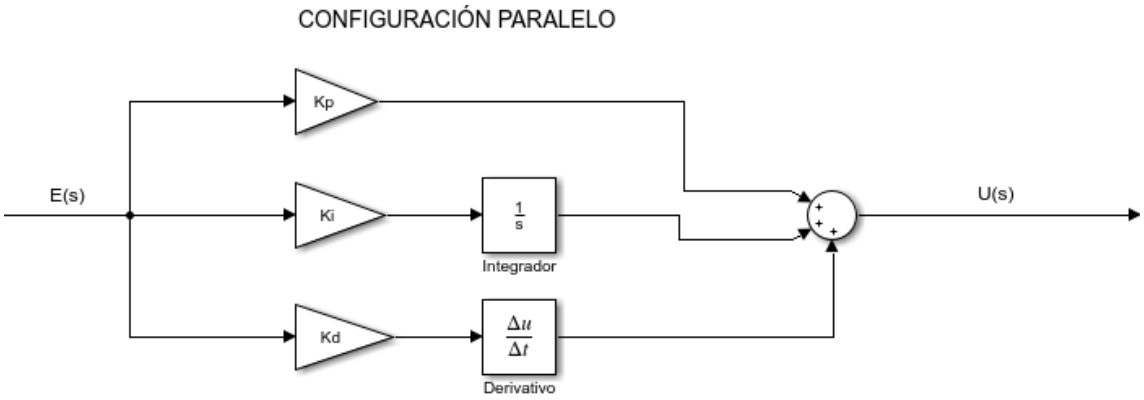


Figura 4.59 Diagrama de bloques de la configuración paralela del PID.

- Controlador PID ideal:

  Hay que decir que el controlador PID ideal es prácticamente lo mismo que el PID paralelo, es decir, el control del sistema se realiza exactamente igual que en el PID paralelo, pero expresado de diferente forma, en este caso utiliza las constantes de tiempo en vez de las constantes integral y derivativa. Por todo esto, el controlador PID en estructura ideal no se ha implementado en la programación software de Arduino, puesto que la señal de control es exactamente igual en ambas topologías. La ecuación del controlador PID ideal es la siguiente:

$$u(t) = K_P * (e(t) + \frac{1}{T_I} * \int_0^t e(t)dt + T_D * \frac{de(t)}{dt})$$

  Sabiendo que:

$$K_I = \frac{K_P}{T_I} \quad ; \quad K_D = K_P * T_D$$

  T_I = Constante de tiempo integral.

  T_D = Constante de tiempo derivativa.

  Las constantes de tiempo integral T_I y derivativa T_D representan el tiempo de respuesta del sistema por medio de la acción integral y derivativa respectivamente. De tal forma que, si T_I es alta la acción integral necesitará un tiempo largo de acumular error para actuar sustancialmente, y si T_D es alta la acción derivativa actúa más rápido y se anticipa con mayor velocidad, amortiguando el sistema y aumentando el tiempo de establecimiento.

Aplicando la transformada de Laplace a la ecuación del controlador PID ideal, se obtiene:

$$F.T = \frac{U(s)}{E(s)} = K_P * (1 + \frac{1}{T_I * s} + T_D * s)$$

Entonces, el diagrama de bloques de la configuración ideal del PID es el siguiente:

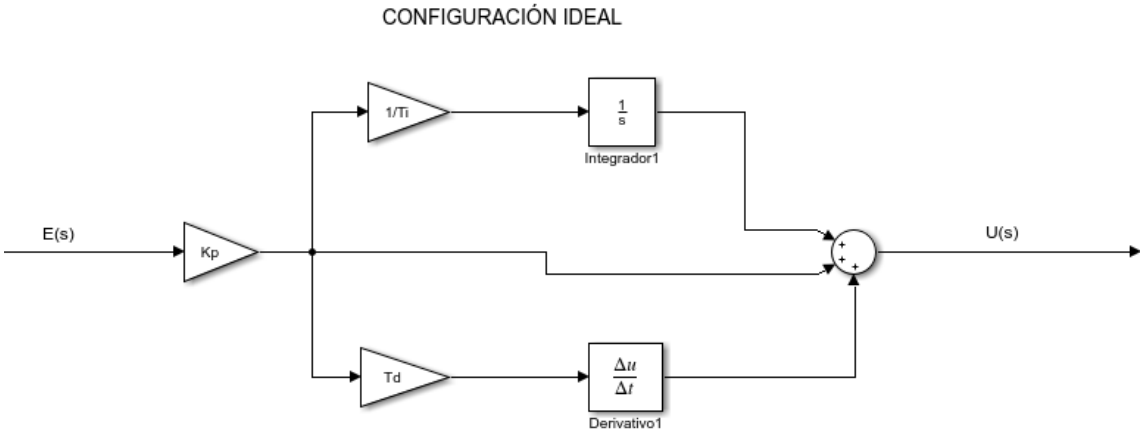


Figura 4.60 Diagrama de bloques de la configuración ideal del PID.

- Controlador PID en serie, también llamado controlador PID clásico, real o interactivo:

  El controlador PID en serie se va a implementar en la programación software de Arduino, además del controlador PID en paralelo como se ha visto en los demás apartados. El controlador PID en serie se va a implementar en la programación para dar la opción al usuario, por medio de la interfaz de usuario de poder seleccionar diferentes configuraciones del PID.

  La función de transferencia del controlador PID en serie es la siguiente:

$$F.T_{PIDserie} = \frac{U(s)}{E(s)} = K_{Ps} * \left(1 + \frac{1}{T_{Is} * s}\right) * (1 + T_{Ds} * s)$$

  Con lo cual, el diagrama de bloques de la topología serie del PID es el siguiente:

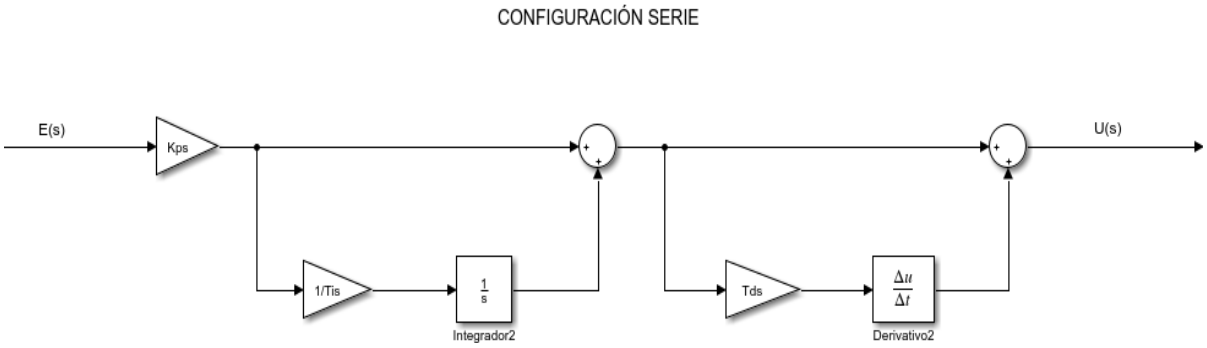


Figura 4.61 Diagrama de bloques de la configuración serie del PID.

Conociendo la función de transferencia del controlador PID en serie, se obtiene la ecuación característica del controlador PID serie en el dominio del tiempo, que es la que se implementa en la programación software de Arduino transformándola en ecuaciones en diferencias (tiempo discreto), puesto que, los procesadores digitales (Arduino) utilizan las ecuaciones en diferencias para implementar los algoritmos de control, en este caso el algoritmo de control de un controlador PID serie:

$$F.T_{PIDserie} = \frac{U(s)}{E(s)} = K_{Ps} * (1 + \frac{1}{T_{Is} * s}) * (1 + T_{Ds} * s) \rightarrow$$

$$\rightarrow U(s) = K_{Ps} * E(s) * (1 + T_{Ds} * s + \frac{1}{T_{Is} * s} + \frac{T_{Ds} * s}{T_{Is} * s}) \rightarrow$$

$$\rightarrow U(s) = K_{Ps} * (E(s) + T_{Ds} * E(s) * s + \frac{1}{T_{Is} * s} * E(s) + \frac{T_{Ds}}{T_{Is}} * E(s))$$

Aplicando la antitransformada de Laplace L^{-1} a esta ecuación:

$$u(t) = K_{Ps} * (e(t) + T_{Ds} * \frac{de(t)}{dt} + \frac{1}{T_{Is}} * \int_0^t e(t)dt + \frac{T_{Ds}}{T_{Is}} * e(t)) \rightarrow$$

Se obtiene la ecuación característica del controlador PID en serie:

$$\boxed{u(t) = K_{Ps} * (e(t) * (1 + \frac{T_{Ds}}{T_{Is}}) + T_{Ds} * \frac{de(t)}{dt} + \frac{1}{T_{Is}} * \int_0^t e(t)dt)}$$

Sabiendo que:

$$K_{Is} = \frac{K_{Ps}}{T_{Is}} \quad ; \quad K_{Ds} = K_{Ps} * T_{Ds}$$

T_{Is} = Constante de tiempo integral del PID en serie.

T_{Ds} = Constante de tiempo derivativa del PID en serie.

Conociendo la ecuación característica del controlador PID serie, ya se puede implementar en la programación software de Arduino transformándola en ecuaciones en diferencias (tiempo discreto).

El controlador PID en serie se implementa en la programación software de Arduino exactamente igual que el controlador PID paralelo visto en el apartado *4.4.4 Implementación del controlador PID en Arduino*, modificando solo con respecto a la implementación del PID paralelo, el cálculo de la integral del error y la ecuación de salida del PID, además de establecer las ecuaciones de las constantes de tiempo:

- El cálculo de la integral del error se establece en la programación de la siguiente manera:

  error_int = error_int + error_pos; //Para el PID serie, se calcula el error integral, que es el error integral anterior más la suma del error de posición actual.

  Se puede ver que con respecto al PID paralelo se ha eliminado el término K_I de la instrucción, puesto que, el controlador PID serie utiliza las constantes de tiempo en vez de las constantes integral y derivativa en su algoritmo de control.

- Ecuación de salida del PID en serie:

  La ecuación de salida del PID en serie se ha establecido en la programación software de Arduino transformando la ecuación característica del controlador PID serie en ecuaciones en diferencias, mediante el método de aproximación numérica de Euler hacia atrás:

  *salida_PID = Kps * (error_pos * (1 + Tds/Tis) + Tds * error_dif + (1/Tis) * error_int);* // Fórmula para obtener la señal de salida del controlador PID en serie (también llamado controlador PID interactivo).

- Las ecuaciones de las constantes de tiempo se declaran en la programación:

  Como el controlador PID en serie utiliza las constantes de tiempo integral y derivativa en su algoritmo de control, se declaran las ecuaciones de las constantes de tiempo en función de las constantes del PID:

  float Tis = Kps/Kis; // Constante de tiempo integral PID serie.
  float Tds = Kds/Kps; // Constante de tiempo derivativa del PID serie.

Una de las diferencias más significativas entre los algoritmos del controlador PID paralelo y del PID serie, es que el controlador PID paralelo tiene una ganancia proporcional K_P, mientras que el algoritmo del PID en serie tiene una ganancia del controlador K_{Ps}. La ganancia del controlador K_{Ps} afecta a los tres términos del PID (proporcional, integral y derivativo) en serie, es decir, si se modifica la ganancia del controlador se alteran los tres términos del PID. Por otro lado, la ganancia proporcional K_P del controlador PID en paralelo solo afecta al término proporcional, por lo que al modificarla solo se altera el término proporcional y los demás términos (integral y derivativo) permanecen inalterados.

4.6.2.1 Obtención de las constantes del controlador PID en serie

Como se ha contemplado en el apartado *4.6.1 Obtención de las constantes en tiempo discreto*, se han obtenido las constantes del controlador PID en paralelo que realizan un control de posición de la bola óptimo, con un error de posición mínimo que es lo que se quiere; con lo cual es necesario obtener las constantes del controlador PID en serie que realicen un control óptimo del sistema. Para ello se van a obtener partiendo de las constantes de la configuración paralela del PID que realizan un control óptimo, para así obtener el mismo control del sistema con diferente configuración del PID (PID en serie):

Las constantes del controlador PID paralelo que realizan un control óptimo del sistema son las siguientes:

- $K_P = 4$
- $K_I = 0{,}03$
- $K_D = 100$

Con esto se obtiene los valores de las constantes de tiempo:

$$T_I = \frac{K_P}{K_I} = 133{,}3333$$

$$T_D = \frac{K_D}{K_P} = 25$$

Entonces, como se sabe que los parámetros de un controlador PID tipo serie equivalente a uno tipo paralelo son estos, se obtiene:

$$K_{Ps} = K_P * K$$

$$T_{Is} = T_I * K$$

$$T_{Ds} = T_D / K$$

Donde:

$$K = 0{,}5 * \left(1 + \sqrt{1 - \frac{4 * T_D}{T_I}}\right)$$

K: es el factor de conversión del PID paralelo al PID serie.

Aplicando las fórmulas anteriormente expuestas, se obtiene:

$$K = 0{,}5 * \left(1 + \sqrt{1 - \frac{4 * 25}{133{,}3333}}\right) \rightarrow K = 0{,}75$$

$$K_{Ps} = 4 * 0{,}75 \rightarrow \boxed{K_{PS} = 3}$$

$$T_{Is} = 133{,}3333 * 0{,}75 \rightarrow T_{Is} = 100$$

Sabiendo que:

$$T_{Is} = \frac{K_{PS}}{K_{IS}} \rightarrow K_{Is} = \frac{3}{100} \rightarrow \boxed{K_{IS} = 0{,}03}$$

$$T_{Ds} = \frac{25}{0{,}75} \rightarrow T_{Ds} = 33{,}3333$$

Sabiendo que:

$$T_{Ds} = \frac{K_{Ds}}{K_{Ps}} \rightarrow K_{Ds} = 33{,}3333 * 3 \rightarrow \boxed{K_{Ds} = 100}$$

Con lo cual, las constantes discretas del controlador PID serie que realizan un control óptimo del sistema son:

- ✓ $K_{Ps} = 3$
- ✓ $K_{Is} = 0{,}03$
- ✓ $K_{Ds} = 100$

Estas constantes discretas son las que se han introducido en la programación software de Arduino, las cuales realizan un control del sistema exactamente igual al del PID paralelo, es decir, realizan un control de posición de la bola óptimo, con un error de posición mínimo que es lo que se quiere.

El control realizado es un control rápido, bastante estable gracias al filtro que se aplica a la acción derivativa y con buena precisión, puesto que, el error es pequeño.

Se puede ver que lo único que cambia de valor respecto a las constantes del controlador PID paralelo es la constante proporcional, también denominada ganancia del controlador K_{Ps}, y como consecuencia de esto también cambian los valores de las constantes de tiempo, hay que decir que el funcionamiento de la ganancia del controlador es diferente a la acción proporcional del PID paralelo, como se ha visto en el anterior apartado..

4.7 Diseño, desarrollo y realización de la interfaz de usuario

Como se ha dicho en el apartado *3.6 Interfaz de usuario a realizar*, el objetivo de esta interfaz es permitir al usuario poder interactuar con el control de la maqueta y con los diferentes parámetros de los controladores implementados. Para ello, se ha diseñado una

placa PCB, la cual va a ser el núcleo de la interfaz, se ha diseñado con el software KiCad de filosofía open source.

Está interfaz de usuario va a permitir seleccionar distintos tipos de control, y en algunos de los tipos se va dar la opción al usuario de poder variar las constantes del controlador PID (K_p, K_i, K_d), y de poder establecer la referencia de posición de la bola, además se va a poder visualizar en todo momento las variables del sistema, incluido el error de posición de la bola mediante una pantalla LCD; todo esto para que el usuario que utilice está interfaz pueda visualizar en tiempo real y de manera clara como afectan cada una de las acciones de los distintos PID en la respuesta del sistema al modificarlas, y así poder obtener conclusiones del funcionamiento de cada una de las configuraciones del PID, y de los distintos parámetros de cada una de ellas.

En definitiva, se va a realizar una interfaz de usuario, en la cual se va a utilizar un panel de control diseñado para conseguir una comunicación interactiva entre usuario y control del sistema, con las funciones de transmitir el valor de la consigna y los valores de las constantes de los PID implementados, visualizar los resultados por medio de una pantalla LCD y obtener los datos del error de posición de la bola en tiempo real.

4.7.1 Funcionamiento de la interfaz de usuario

Como se ha dicho, la interfaz va a disponer de distintos tipos de control, los cuales van a estar constituidos por los siguientes modos de funcionamiento a elegir por el usuario:

- Modo 1: En el modo 1 de funcionamiento se implementa el controlador PID en su configuración paralela con sus constantes óptimas obtenidas en el apartado *4.6.1 Obtención de las constantes en tiempo discreto*.
- Modo 2: En el modo 2 de funcionamiento se implementa el controlador PID en su configuración serie con sus constantes óptimas obtenidas en el apartado *4.6.2.1 Obtención de las constantes del controlador PID en serie*.
- Modo 3: Se implementa el controlador PID en su configuración paralela, pero en este caso se puede modificar cada una de las acciones del controlador. Este modo inicia la selección de los valores de las constantes con las constantes

óptimas, para partir de los valores correctos y ver cómo cambia la respuesta del sistema al modificar cada una de las acciones.

- Modo 4: Exactamente lo mismo que el modo 3, pero implementando el controlador PID en su configuración serie.

NOTA: En todos los modos de funcionamiento se puede ajustar la consigna, esto es, la posición deseada de la bola.

Una vez se elige el modo de funcionamiento deseado o el control del sistema deseado y se ajusta el valor de la consigna, para los modos en los cuales no se puede modificar las acciones del PID, se establecen en la pantalla los valores designados o aplicados a cada una de las variables del sistema: las tres constantes del PID, la consigna y el error de posición de la bola, el cual se visualiza como va cambiando a medida que se va ejecutando la señal de control, y es el momento en el cual se inicia la ejecución de la señal de control.

En cambio, para los modos de funcionamiento en los cuales se puede modificar los parámetros del PID, se va a una pantalla en la cual se pueden ir modificando cada una de las acciones del PID, en está pantalla es el momento en el cuál el control del sistema empieza a ejecutarse, es decir, durante la selección de los valores de las constantes, el control del sistema va estar en ejecución, esto se ha realizado así, para que el usuario pueda visualizar de una manera más clara y rápida como cambia la respuesta del sistema al modificar cada una de las acciones en tiempo real. Una vez que el usuario queda conforme con el control aplicado, pasa a la siguiente pantalla y es en está donde se visualizan todas las variables seleccionadas durante todo el proceso, además de visualizar el error de posición de la bola como se ha dicho antes.

En todos los modos, cuando se está en esta ultima pantalla y se pulsa el pulsador de “ok” se vuelve a la pantalla de implementación del control deseado o de modo de funcionamiento, y se inicia de nuevo todo el proceso, para ello se deben poner a 0 todas las variables de las constantes para que el sistema empiece de nuevo con las condiciones iniciales, puesto que para las constantes de los PID implementados esto es muy importante para que el sistema una vez que se inicia de nuevo el proceso de selección de tipo de control, el sistema no siga en ejecución, como se verá en la programación

software de Arduino, *Anexo I* y en el diagrama de flujo de la programación de la interfaz, apartado *4.7.5*.

4.7.2 Diseño y componentes para realizar el panel de control de la interfaz de usuario

El panel de control de la interfaz estará constituido por una caja de plástico para que sea la parte exterior del panel, llamado de otra manera, para que sea el chasis del panel de control de la interfaz, y aislé todas las conexiones internas del panel, puesto que la placa PCB y el microcontrolador van en el interior de la caja de plástico. Por eso a la hora de seleccionar la caja de plástico se tuvo en cuenta las medidas del Arduino MEGA ADK utilizado, las cuales son conocidas, entonces teniendo en cuenta esas medidas de la placa Arduino se seleccionó una caja de plástico con unas medidas superiores a las de la placa Arduino, teniendo en cuenta que la placa PCB también va dentro de la caja, por todo esto se seleccionó una caja de plástico con las siguientes medidas: longitud: 191mm, anchura: 110mm y altura: 61mm.

Figura 4.62 Caja de plástico utilizada como panel de control.

La placa PCB va situada en el interior de la caja, y va conectada al Arduino MEGA ADK, a través de unas clavijas macho de 2,54mm el paso, puesto que la distancia entre los pines de Arduino es de 100mils = 0,1pulgadas = 2,54mm.

La placa PCB es el elemento más importante de la interfaz de usuario, debido a que es el componente al que todos los demás dispositivos que conforman el sistema, incluido el sensor infrarrojo y el servomotor se conectan, puesto que es el componente que conecta cada dispositivo al pin de Arduino seleccionado para ello.

La caja de plástico consta de una tapa en la cual van a ir los siguientes componentes situados en su superficie, que van a ser utilizados por el usuario para transmitir órdenes, visualizar los resultados y obtener una situación del proceso/máquina en tiempo real:

- Una pantalla LCD de 20x4 + módulo I2C, la cual se ha visto su funcionamiento, sus conexiones y su programación en el apartado *3.6.1 Pantalla LCD*. Con la utilización del módulo I2C se simplifica tanto el montaje de la pantalla, como el diseño de la placa PCB. La función de la pantalla LCD en está interfaz de usuario es la siguiente: es en donde se van a visualizar todas las variables del sistema: las tres constantes del PID, la consigna, el error de posición de la bola, los diferentes modos de control y los diferentes mensajes explicativos que van aparecer durante la navegación.

- 4 encoders rotativos con pulsador: Se van a utilizar para poder variar el valor de cada una de las constantes de los controladores PID implementados (solo en los modos de funcionamiento 3 y 4), para ello, se van a utilizar 3 encoders. Además se va a utilizar otro encoder rotativo con pulsador para poder variar el valor de la señal de referencia (posición deseada de la bola), asimismo el pulsador de este encoder se va a utilizar como un pulsador de "ok", esto es, se va a utilizar para validar la selección, tanto de las constantes del PID, como del valor de la consigna, también valida la selección del modo de control seleccionado; además de esto, va a ser el pulsador que se va a utilizar para pasar de pantalla en pantalla, y el que en la última pantalla si es pulsado, finalice la ejecución del sistema de control, y reinicie de nuevo la interfaz de usuario con la elección del modo de funcionamiento. Al igual que la pantalla LCD en el apartado teórico de la interfaz, apartado *3.6.2 Encoder rotativo con pulsador*, se ha explicado su funcionamiento, sus conexiones y su programación.

- 2 pulsadores normalmente abiertos: Estos pulsadores se utilizan para navegar por los diferentes modos de control. Un pulsador será llamado como "up" y el otro como "down", puesto que un pulsador se utiliza para cambiar de modo de manera ascendente y el otro para cambiar de modo de manera descendente.

Figura 4.63 Pulsador utilizado en la interfaz de usuario.

- 4 diodos LEDs verdes: Estos diodos LEDS verdes se utilizan para indicar de manera visual el modo de control seleccionado durante todo el proceso. Si se implementa en el sistema el Modo 1 se enciende un LED, si se implementa el Modo 2 se encienden los 2 primeros LEDs, y así respectivamente.
- 1 diodo LED rojo: Este diodo LED rojo indica cuando se da por bueno el error de posición de la bola en el sistema o cuando se da por cumplido el objetivo de control, es decir, se enciende cuando el error de posición de la bola es de 10mm, tanto positivos como negativos, o es inferior a 10mm, cuando ocurre esto, se da por bueno el error de posición de la bola y se enciende el LED rojo.

Figura 4.64 Diodo LED rojo y diodos LEDs verdes utilizados en la interfaz de usuario.

Teniendo en cuenta los componentes que van a conformar el panel de control de la interfaz y sus respectivas funciones, el esquema eléctrico de todo el sistema es el siguiente, incluyendo tanto el sensor infrarrojo como el servomotor, los cuales están situados en la maqueta:

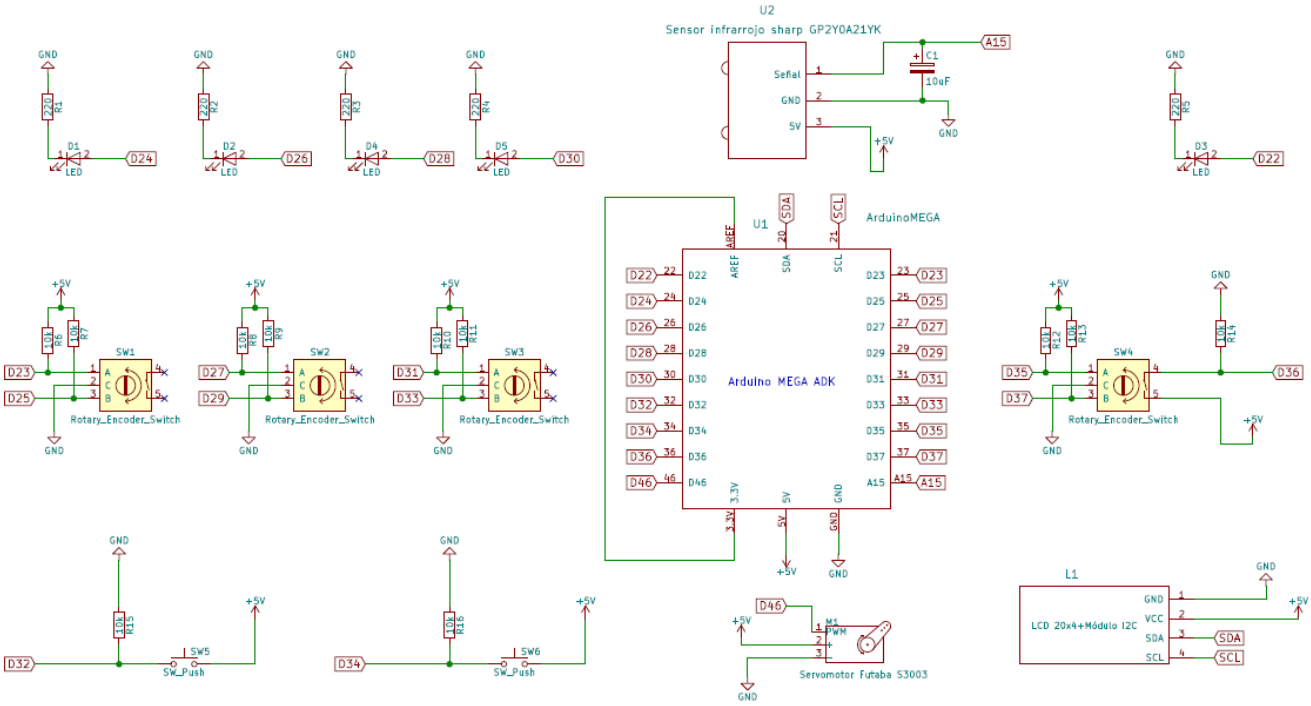


Figura 4.65 Esquema eléctrico de todo el sistema.

Como se puede ver en el esquema eléctrico de todo el sistema, las conexiones hardware de los componentes utilizados con el Arduino MEGA ADK son las siguientes:

- Conexiones de la pantalla LCD de 20x4 + módulo I2C con el Arduino son las siguientes: SDA (pin 20), SCL, (pin 21), GND y VCC (+5V).
- Las conexiones de los encoders rotativos para las constantes son las siguientes: para K_P 23 y 25, para K_I 27 y 29, y por ultimo para K_D 31 y 33. Estos pines se van a declarar en la programación como pines digitales de entrada.
- Las conexiones del encoder rotativo con pulsador para establecer el valor de consigna, y para el pulsador llamado “ok” son las siguientes: se utilizan tanto para el encoder como para el pulsador pines digitales de entrada, para el encoder de la consigna 35 y 37, y para el pulsador “ok” se utiliza el pin 36.
- Las conexiones de los pulsadores son las siguientes: para el pulsador llamado “up” se utiliza el pin 32 del Arduino, y para el pulsador “down” se utiliza el pin 34 del Arduino. Los dos pines se declaran como pines digitales de entrada en la programación.

- Las conexiones utilizadas por los diodos LEDs son las siguientes: para los diodos LEDs verdes se utilizan los pines 24, 26, 28 y 30, y para el diodo LED rojo se utiliza el pin 22. Todos estos pines se declaran como pines digitales de salida en la programación.
- Para la señal de salida del sensor infrarrojo se utiliza el pin analógico de entrada A15.
- Por último, la señal de control del servomotor está conectada al pin digital de salida 46 (timer 5 (OC5A)) de la placa Arduino.

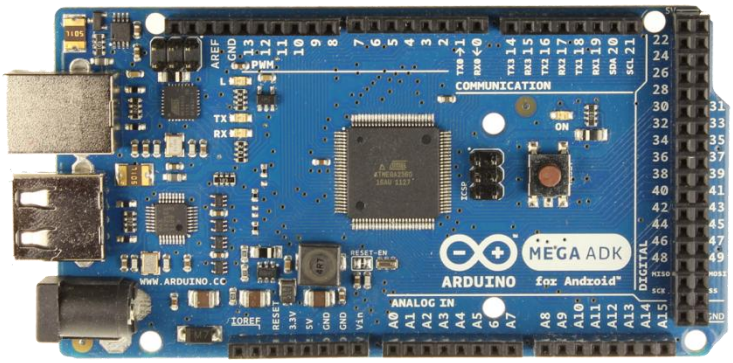


Figura 4.66 Disposición de los pines que se utilizan del Arduino MEGA ADK.

Como se puede ver, solamente se va a utilizar el lado derecho de la placa, con lo cual, se ha diseñado una placa PCB lo más pequeña posible para que tape los mínimos pines posibles del Arduino, y así, queden libres los demás pines, para si es necesario o para posibles ampliaciones futuras del proyecto poder utilizarlos.

4.7.3 Diseño y desarrollo de la placa PCB con el software KiCad

Como se ha dicho antes, la placa PCB va ir conectada a los pines del Arduino, puesto que es la que va a conectar a los demás componentes al Arduino. Entonces, debido a esto y a que los componentes del panel de control de la interfaz van a ir en la superficie de la caja de plástico, se debe diseñar un esquema electrónico para la placa PCB, en el cual, los componentes del esquema anterior se sustituyan por conectores, puesto que ninguno de los componentes utilizados va a estar o va a conformar la placa PCB.

Teniendo en cuenta lo dicho anteriormente, el esquema electrónico de la placa de circuito impreso (PCB) es el siguiente:

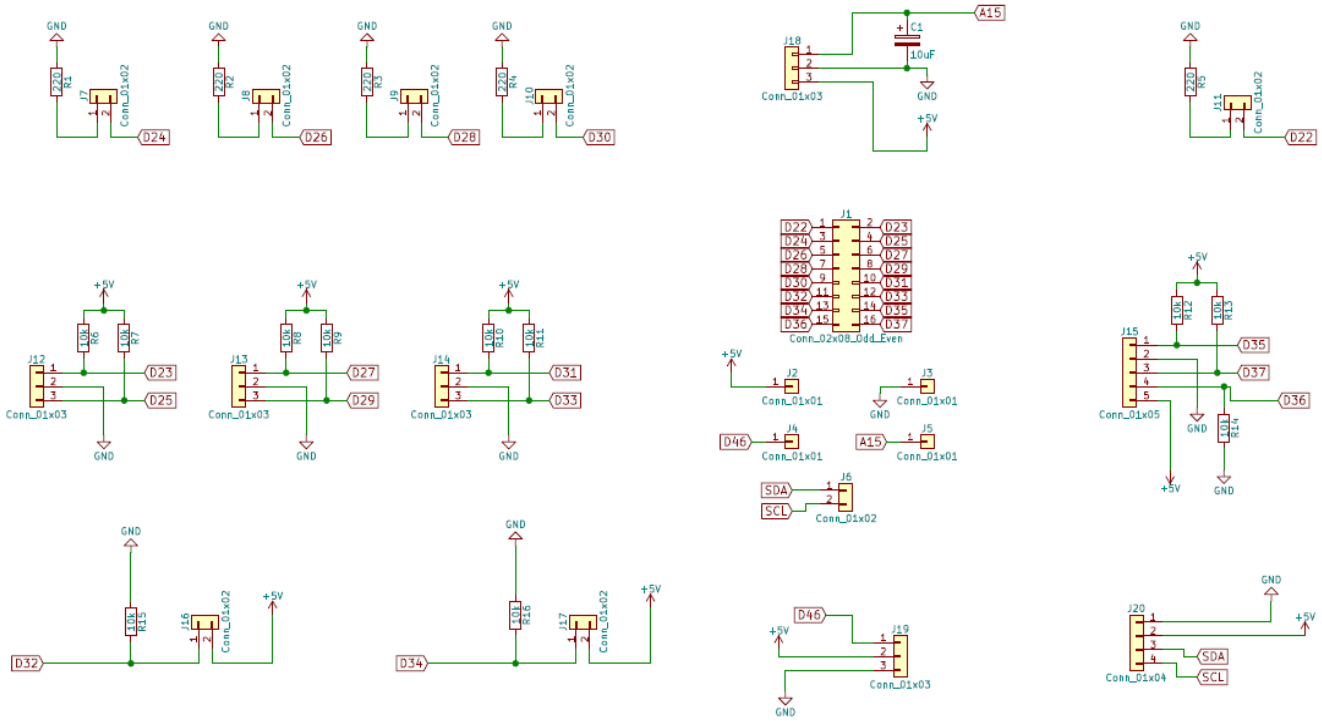


Figura 4.67 Diseño del esquema electrónico de la placa PCB.

Como se sabe, dentro de la caja de plástico va a ir el Arduino MEGA ADK y la placa PCB diseñada conectada a sus pines, además de las conexiones entre los componentes situados en la superficie y la placa PCB. La placa PCB como se puede ver en el diseño del esquema electrónico, va estar conformada por los siguientes componentes:

- 5 resistencias de 220Ω.
- 11 resistencias de 10kΩ.
- 1 condensador electrolítico de 10µF.
- 7 conectores montables (macho/hembra) de 2 pines, paso de 2,54mm (100mils).
- 5 conectores montables (macho/hembra) de 3 pines, paso de 2,54mm (100mils).
- 1 conector montable (macho/hembra) de 4 pines, paso de 2,54mm (100mils).
- 1 conector montable (macho/hembra) de 5 pines, paso de 2,54mm (100mils).

  NOTA: Los conectores montables constan de dos partes, la parte que va en la placa PCB es un conector macho, y la parte que se monta en los cables es un conector hembra.

- Tira de clavijas macho de 2,54mm el paso, para conectar la placa PCB a los pines del Arduino. Se utilizan 22 clavijas macho, puesto que, en el sistema se van a utilizar 22 pines del Arduino para conectar los distintos componentes que conforman el sistema barra y bola.

A continuación se van a explicar los pasos realizados con el software KiCad de filosofía open source para el diseño y desarrollo de la placa de circuito impreso (PCB):

1. Una vez creado y diseñado el esquema electrónico de la placa PCB como se ha visto anteriormente, se comprueban las conexiones del esquema con el botón ERC (control de las reglas eléctricas) situado en la barra superior de tareas de la aplicación de diseño de esquemas de KiCad:

Figura 4.68 Botón para el control de las reglas eléctricas de KiCad.

2. Una vez comprobado el esquema electrónico diseñado, se genera el "netlist" o lo que es lo mismo una lista de redes para que se especifiquen las conexiones de todos los componentes en un archivo, y así poder exportarlas a otro programa, en este caso, para poder exportarlas a la aplicación "Pcnew" de KiCad para el diseño de la placa PCB. Con esto ya se puede asociar a cada componente del esquemático su "footprint" (huella), las cuales son las que se van a utilizar para poder diseñar la placa PCB en el entorno de diseño.

3. Importar la lista de redes (netlist) generada en la aplicación "CvPcb" situada en la barra de tareas del esquema, la cual se utiliza para asociar a cada componente del esquemático su huella. La huella es el espacio que va a ocupar cada componente en la placa PCB, en donde van estar los *pads* situados. Para asignar a cada componente su respectiva huella se mide el componente real o se mira en su datasheet, y con las medidas se selecciona la huella que mejor se ajuste a sus características y medidas.

```
 1      C1 -            10uF : Capacitors_THT:CP_Radial_D5.0mm_P2.50mm
 2      J1 - Conn_02x08_Odd_Even : Pin_Headers:Pin_Header_Straight_2x08_Pitch2.54mm
 3      J2 -       Conn_01x01 : Pin_Headers:Pin_Header_Straight_1x01_Pitch2.54mm
 4      J3 -       Conn_01x01 : Pin_Headers:Pin_Header_Straight_1x01_Pitch2.54mm
 5      J4 -       Conn_01x01 : Pin_Headers:Pin_Header_Straight_1x01_Pitch2.54mm
 6      J5 -       Conn_01x01 : Pin_Headers:Pin_Header_Straight_1x01_Pitch2.54mm
 7      J6 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
 8      J7 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
 9      J8 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
10      J9 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
11     J10 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
12     J11 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
13     J12 -       Conn_01x03 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
14     J13 -       Conn_01x03 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
15     J14 -       Conn_01x03 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
16     J15 -       Conn_01x05 : Pin_Headers:Pin_Header_Straight_1x05_Pitch2.54mm
17     J16 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
18     J17 -       Conn_01x02 : Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
19     J18 -       Conn_01x03 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
20     J19 -       Conn_01x03 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
21     J20 -       Conn_01x04 : Pin_Headers:Pin_Header_Straight_1x04_Pitch2.54mm
22      R1 -              220 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
23      R2 -              220 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
24      R3 -              220 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
25      R4 -              220 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
26      R5 -              220 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
27      R6 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
28      R7 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
29      R8 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
30      R9 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
31     R10 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
32     R11 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
33     R12 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
34     R13 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
35     R14 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
36     R15 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
37     R16 -              10k : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
```

Figura 4.69 Lista de componentes con sus huellas asociadas.

Hay que mencionar, que todos los componentes utilizados son de tecnología THT, es decir, de orificio pasante, por eso, todas las huellas asociadas son de tecnología THT.

Una vez asignada la huella a cada componente se guarda el archivo de la aplicación “CvPcb”, el cual al guardarlo introduce la huella asignada a cada componente en el esquemático.

4. Se genera de nuevo la lista de redes para actualizarla, y que aparezcan en ella las huellas de cada componente, y se importa en la aplicación “Pcbnew” utilizada para el diseño del circuito impreso.

5. Una vez importada la lista de redes en la aplicación “Pcbnew”, aparecen todos los componentes con sus respectivas huellas y sus correspondientes conexiones, es decir, lo que se ha hecho es realizar la conversión del esquema a la placa de circuito impreso. Una vez obtenido esto, lo primero que hay que realizar en la aplicación “Pcbnew” es dimensionar la superficie de la placa PCB. Una vez

hecho esto, se sitúan los componentes en la placa de la mejor manera posible, intentando que las líneas amarillas se crucen lo menos posible para poder enrutarlo bien.

En este caso, teniendo en cuenta que el objetivo de la placa PCB es el de transferir las conexiones de todos los dispositivos del sistema a la placa Arduino, como se ha dicho antes es necesario utilizar 22 clavijas macho de 2,54mm (100mils) el paso para conectar la placa PCB al Arduino, el paso de las clavijas es de 2,54mm (100mils), puesto que la distancia entre los pines de Arduino es de 2,54mm (100mils). Teniendo en cuenta todo esto en el diseño de la placa PCB, los pads de las clavijas anteriormente mencionadas deben colocarse mediante las coordenadas "x" e "y" en la aplicación "Pcbnew", para establecer entre ellas unas distancias adecuadas, para que luego se conecten perfectamente todas las clavijas en los pines que se quieran utilizar del Arduino.

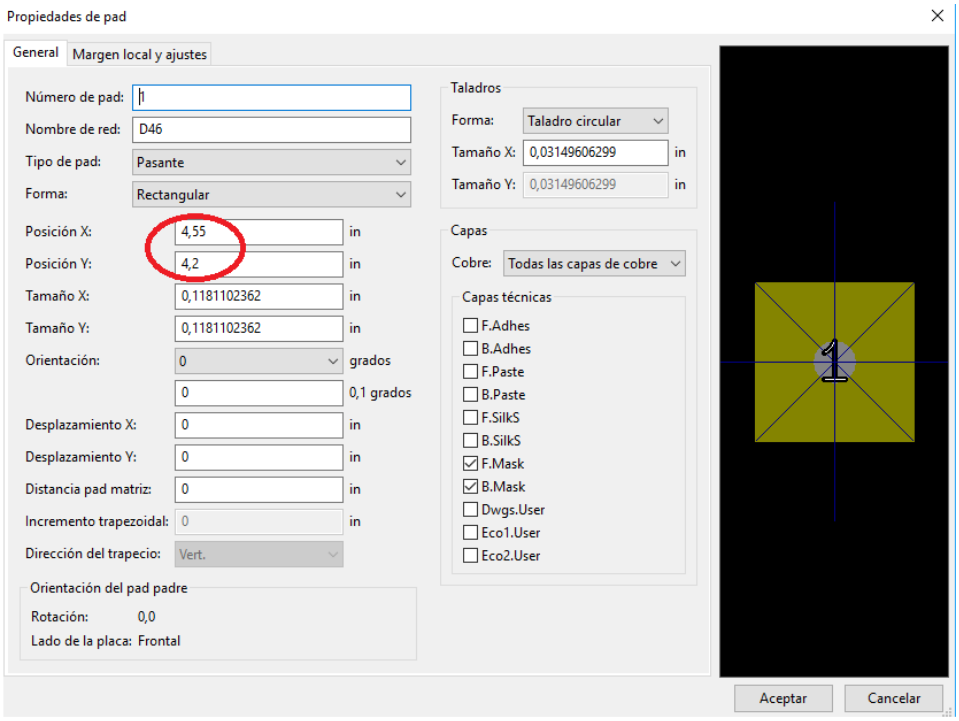


Figura 4.70 Pad de las clavijas situado, introduciendo sus coordenadas "x" e "y".

Todo este proceso se ha realizado teniendo en cuenta las dimensiones de la placa Arduino MEGA 2560, la cual es exactamente igual al Arduino MEGA ADK utilizado.

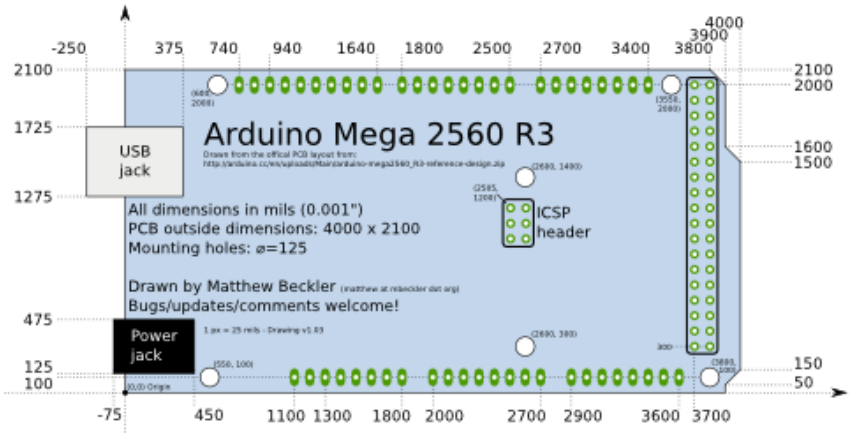


Figura 4.71 Dimensiones entre los pines de la placa Arduino MEGA.

6. Después de situar los componentes en la placa PCB se enrutan las pistas de los distintos componentes, es decir, se diseñan las pistas de cobre de cada interconexión. Antes de enrutar la placa PCB se configuran las reglas de diseño, en las cuales, se establecen el margen entre pistas, la anchura de las pistas, el diámetro de las vías y el taladro a utilizar en las vías en mm. Las pistas de alimentación (+5V y GND) tiene una anchura superior al resto de pistas, puesto que es por donde va a transitar una mayor tensión, porque son las pistas de potencia.

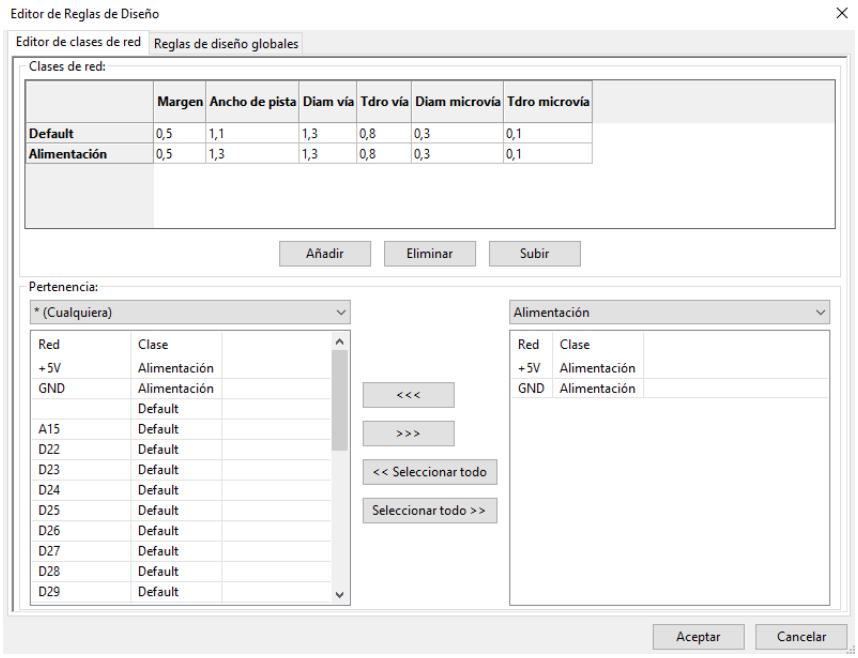


Figura 4.72 Reglas de diseño utilizadas para la fabricación de la placa PCB.

Además de esto, hay que introducir el grosor de los pads a 2mm o más, puesto que con un grosor inferior se pueden tener problemas a la hora de taladrar los pads, puesto que te puedes comer toda la superficie de cobre del pad y esto puede dar problemas; también se establece la broca que se va a utilizar para taladrar los pads, en este caso se van a taladrar todos los pads incluido las vías con una broca de 0,8mm.

En cuanto a las capas hay que decir, que solo se ha utilizado la capa inferior de la placa PCB, en la cual se han enrutado todas las pistas, puesto que todos los componentes utilizados como se ha dicho antes son de tecnología THT de orificio pasante, con lo cual, las pistas van a ir en la capa inferior, pero los componentes en la superior.

A la hora de enrutar la placa PCB en un principio tenía muchos problemas, debido a que las conexiones hardware utilizadas con el posicionamiento de cada componente hacia el enrutamiento de las pistas demasiado complejo, para ello, se modificó las conexiones hardware de cada uno de los componentes con los pines del Arduino, para que así, según el posicionamiento de cada componente el enrutamiento de las pistas fuese un poco más fácil, a pesar de que una de las pistas no se pudo enrutar, por lo que se utilizaron 2 vías. Como una de las pistas (una pista de masa) no se pudo enrutar se puenteo la parte de la placa en la cual los componentes no estaban conectados a masa, con la parte de la placa en la cual los componentes si lo estaban, con ello se consiguió que todos los componentes estuviesen conectados a masa, para ello, como se ha dicho antes se utilizaron 2 vías, para luego puentearlas con un cable llamado “jumper”.

A continuación se va a mostrar la placa PCB una vez enrutada:

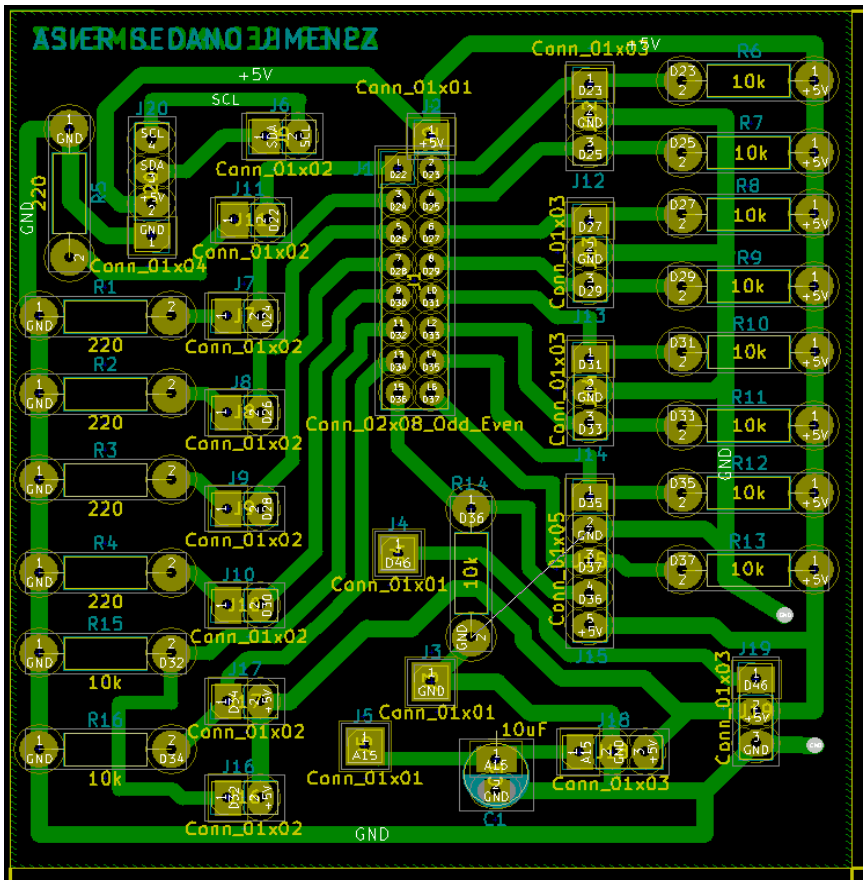


Figura 4.73 Diseño de la placa de circuito impreso (PCB) utilizada.

7. Planos de cobre. Se va a aplicar un plano de masa (GND) en la capa inferior (BOTTOM) de la placa PCB, es decir, donde están situadas todas las pistas de la placa. Esto se realiza para que a la hora de fabricar la placa PCB a mano, el ácido solo tenga que eliminar los huecos de cobre que queden en el diseño de la placa PCB.

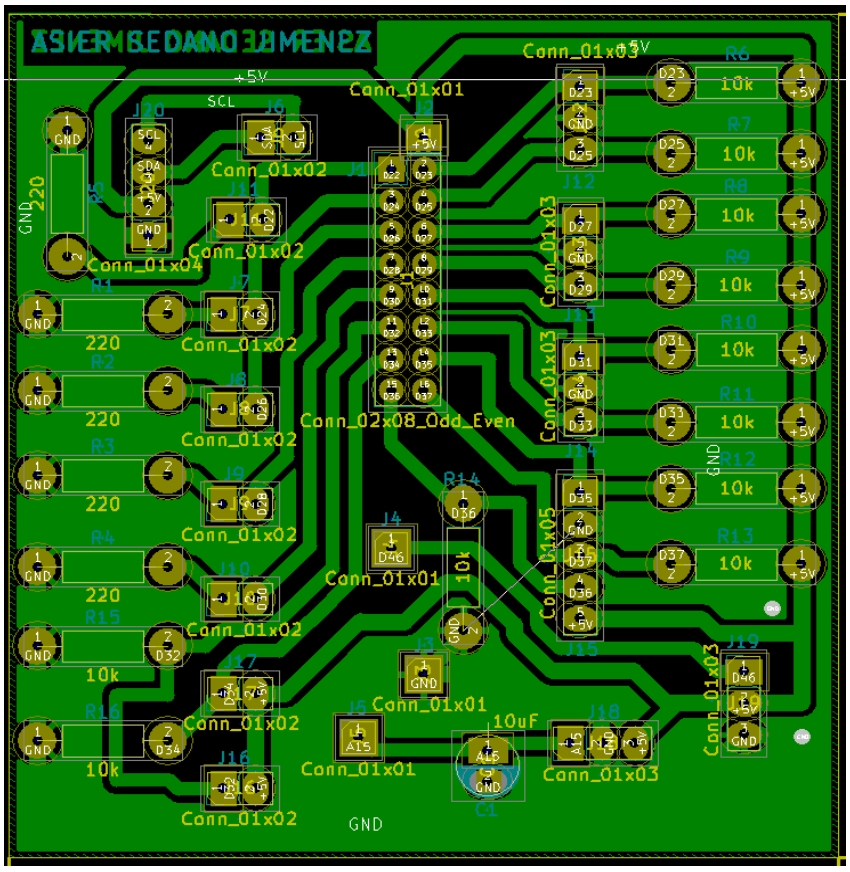


Figura 4.74 Diseño de la placa PCB con la aplicación del plano de masa.

8. Una vez realizados todos estos pasos se consigue el diseño final de la placa PCB, la cual ya está lista para su fabricación, por lo que, se va a generar el pdf para poder fabricarla a mano.

   A continuación voy a mostrar los 2 pdfs generados para la fabricación, aunque uno de ellos no se utilizada, pero se muestra para poder visualizar su diseño:

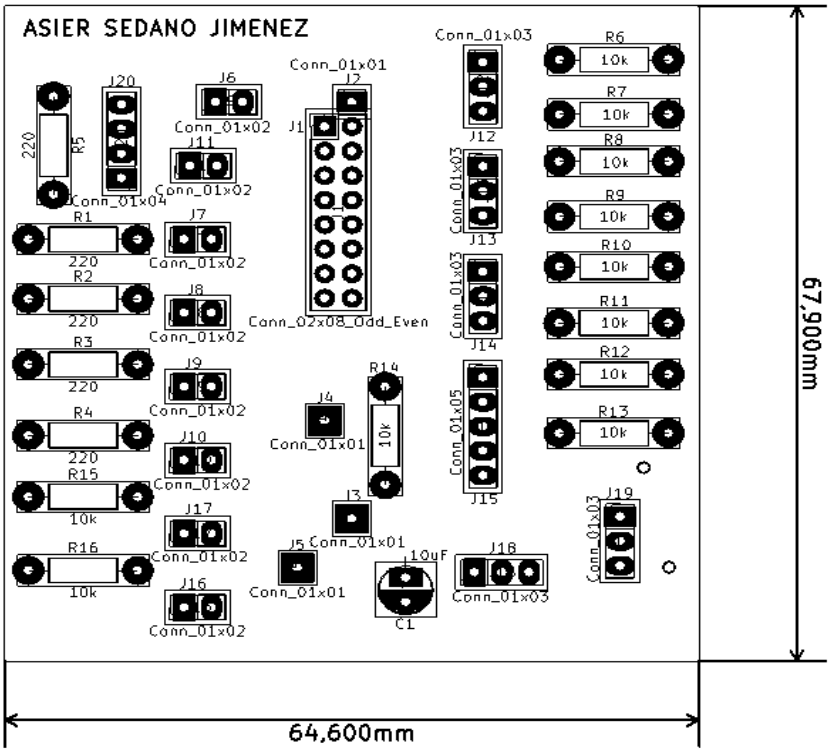


Figura 4.75 Diseño final de la PCB capa superior.

En este caso como solo se utiliza la capa inferior, esta capa no se utiliza, pero se muestra porque se pueden ver bastante bien las huellas y la colocación de los componentes. Hay que decir que está capa superior, normalmente cuando solo se utiliza la capa inferior para las pistas de cobre, solo se implementa en la placa cuando se manda a fabricar, es decir, cuando se realiza la fabricación a mano como es el caso, esta capa no se implementa.

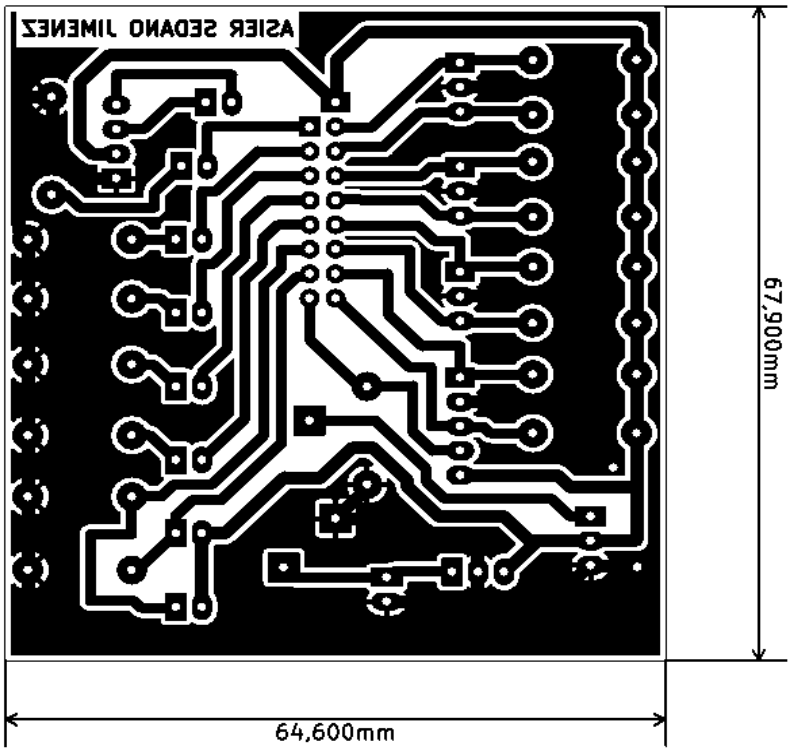


Figura 4.76 Diseño final de la PCB capa inferior, la utilizada para la fabricación a mano.

Como se puede ver, este es el diseño final de la capa inferior de la PCB, la cual se va a utilizar para la fabricación a mano de la PCB. Este diseño también va aparecer en el *Anexo II. Esquemas y Diseño de PCB.*

Cabe destacar, como se puede ver en la figura las medidas de la placa PCB diseñada, puesto que como se mencionó anteriormente en el apartado *4.7.2 Diseño y componentes para realizar el panel de control de la interfaz de usuario*, se ha diseñado una placa PCB lo más pequeña posible para que tape los mínimos pines posibles del Arduino, y así, queden libres los demás pines, para si es necesario o para posibles ampliaciones futuras del proyecto poder utilizarlos.

9. Fabricación de la placa PCB a mano. Explicación breve del proceso: una vez generado el pdf del diseño de la PCB, los fotolitos del diseño y la placa se introducen en una insoladora durante un tiempo, una vez haya pasado ese tiempo, la placa con los fotolitos ya implementados se revela con el liquido revelador (disolución de sosa cáustica más agua), cuando se ha revelado la placa se mete en el liquido atacador (agua oxigenada + agua fuerte + agua) y se

elimina el cobre de la placa, es decir, los huecos de cobre de la placa mencionados antes, quedando la placa lista para su uso.

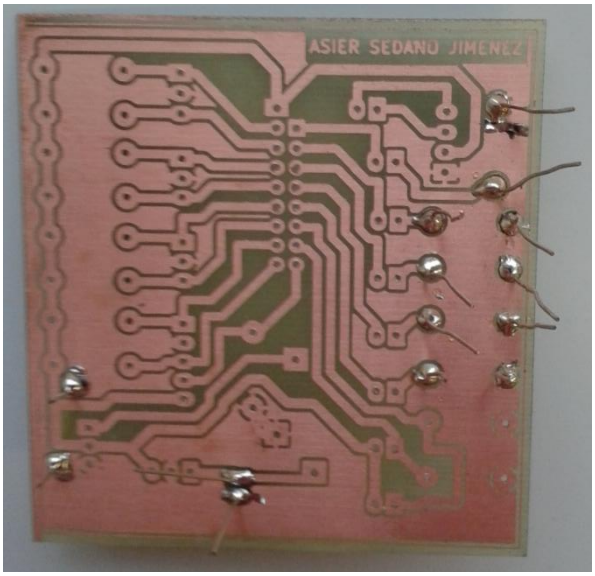


Figura 4.77 Placa PCB fabricada.

4.7.4 Montaje del panel de control de la interfaz de usuario

Una vez fabricada la placa PCB de la interfaz de usuario, se pasa a realizar el montaje del panel de control que va a ser utilizado por la interfaz de usuario para conseguir la comunicación interactiva entre el usuario y el control de la maqueta. Para ello, se han seguido los siguientes pasos:

1. Se comienza taladrando los pads de la placa PCB con una broca de métrica 0,8mm.

2. Una vez taladrados todos los pads de la PCB, se sueldan los componentes que conforman la PCB en los agujeros realizados en cada uno de los pads teniendo en cuenta el diseño realizado de la PCB, para no equivocarse con la colocación de los componentes. Se sueldan cada uno de los componentes en su correspondiente lugar para ensamblarlos a la PCB. Como se puede ver en las 2 siguientes figuras, en la capa superior de la placa PCB van a ir los componentes y en la capa inferior todas las pistas de cada uno de los componentes con la

soldadura realizada. Cabe destacar el puente realizado, por medio de un "jumper" amarillo para unir las masas de los distintos componentes.

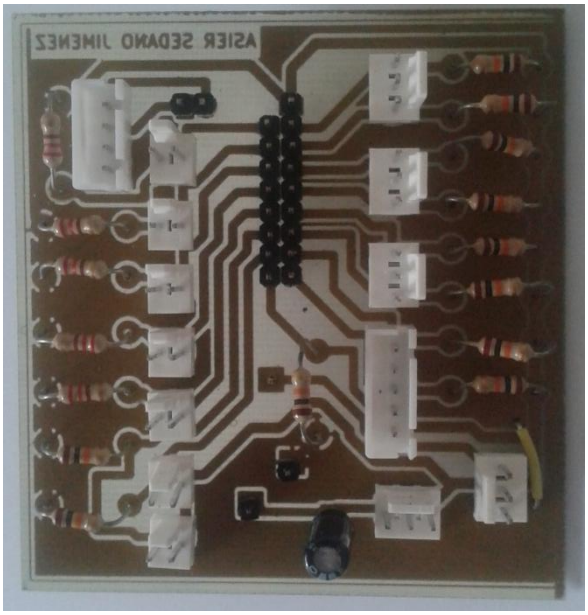


Figura 4.78 Placa PCB soldada, capa superior.

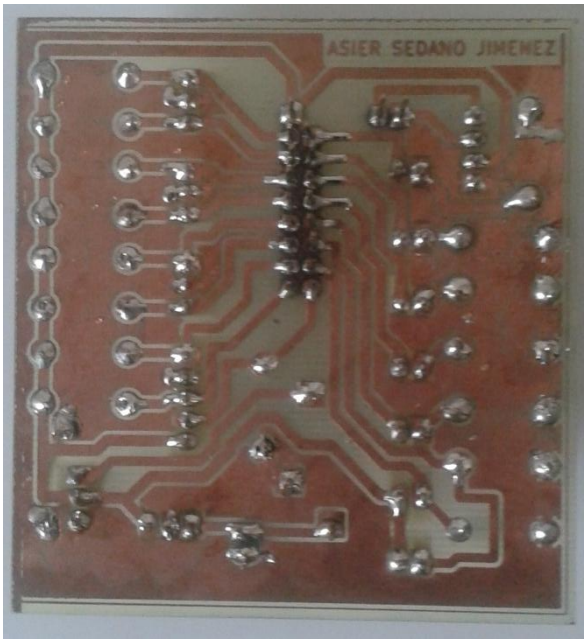


Figura 4.79 Placa PCB soldada, capa inferior.

Con esto la placa PCB ya estaría lista para su uso, faltando por conectar en sus conectores los distintos dispositivos que conforman todo el sistema, incluyendo

tanto el sensor infrarrojo como el servomotor que vienen desde fuera del panel de control, es decir, que vienen desde su posición en la maqueta.

3. Realizar los agujeros en la tapa de la caja de plástico, para poder introducir en ellos todos los componentes del panel de control, los agujeros se han realizado teniendo en cuenta las medidas de cada uno de los componentes para que queden bien situados. El diseño del panel de control, se ha realizado pensando en la comodidad del usuario, dejando espacios entre cada uno de los componentes.

Figura 4.80 Realización de los agujeros en la tapa de la caja.

Cabe destacar que se ha realizado un agujero en la parte inferior trasera de la caja para poder conectar el cable USB del Arduino al ordenador o a cualquier dispositivo, además se utiliza también para que los cables que conectan el sensor infrarrojo y el servomotor de la placa PCB, salgan a través de ese agujero y se conecten o se empalmen con los cables de ambos dispositivos. En definitiva, con esto se consigue que los elementos de fuera del panel estén conectados también a la placa PCB, y a su vez al Arduino.

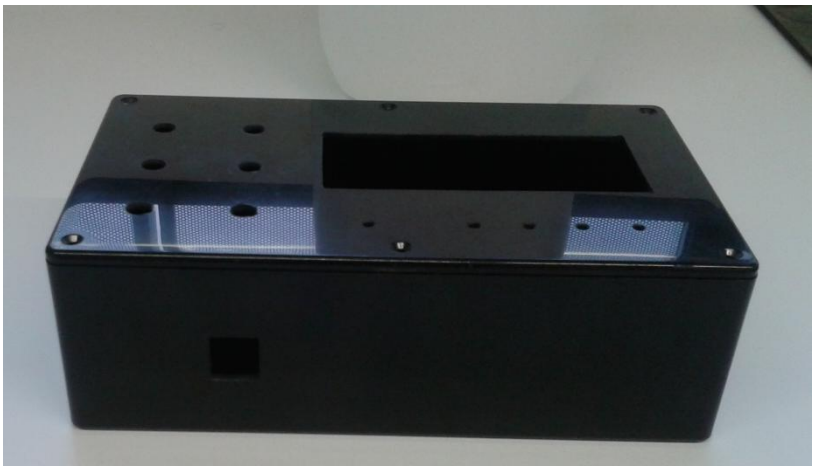


Figura 4.81 Parte trasera de la caja.

4. Montar los conectores montables tipo hembra, puesto que los ensamblados en la PCB son de tipo macho, en los diferentes cables que se van a utilizar, es decir, si para un encoder es necesario tres cables se coge un conector de 3 pines de paso 2,54mm y se monta cada uno de los tres cables con una pieza flexible en uno de sus extremos, se engancha al cable o se crimpan, y luego el cable con esa pieza se introduce, para que no deje salir el cable de ese conector hembra y para que el cable haga contacto con el conector macho de la placa PCB, así para los tres cables en el caso de un encoder. Se realiza este proceso una y otra vez para todos los componentes que conforman el panel de control.

5. Una vez montados todos los cables e introducidos en sus respectivos conectores hembra, el otro extremo del cable se suelda en su correspondiente patilla de su respectivo componente. Este proceso se realiza para cada uno de los componentes que conforman el panel de control, teniendo en cuenta el diseño del esquema electrónico.

6. Una vez hecho los agujeros en la caja y soldado todos los cables a los distintos componentes, y en el otro extremo de los cables haber situado sus respectivos conectores, se pasa a realizar el montaje de toda la interfaz de usuario.

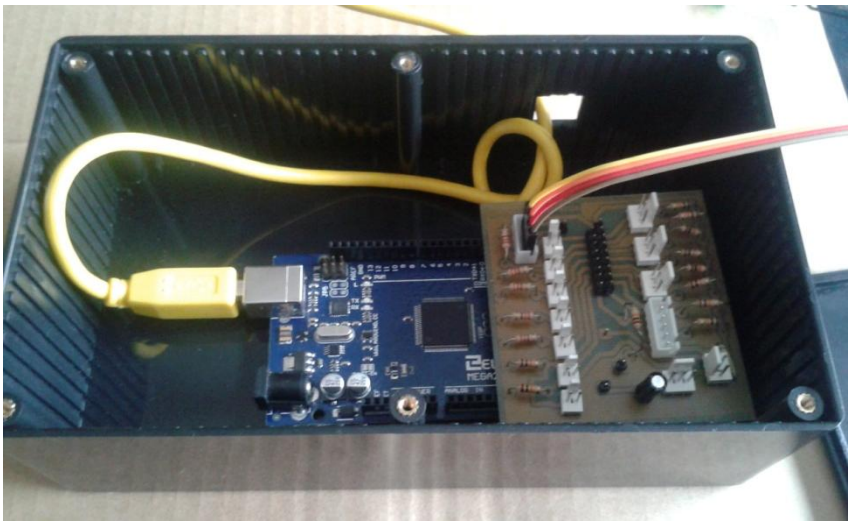


Figura 4.82 Montaje del interior del panel de control.

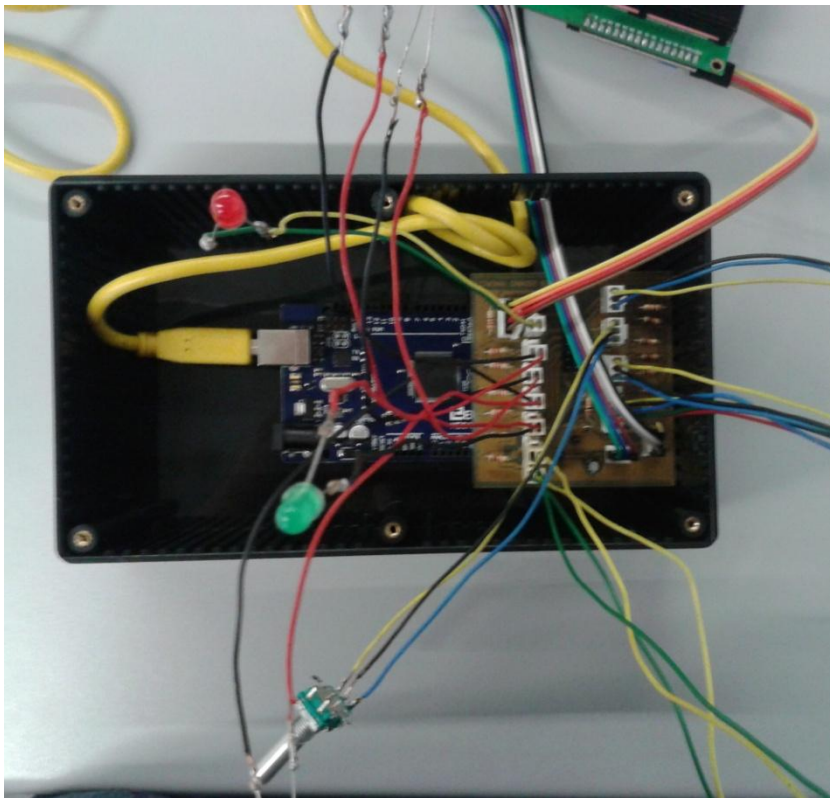


Figura 4.83 Montaje del interior del panel de control, parte 2.

Una vez realizado todo el montaje y ensamblado todo el panel de control diseñado para realizar la interfaz de usuario del sistema, queda de la siguiente manera:

Figura 4.84 Panel de control diseñado para la interfaz de usuario.

Cabe destacar, que cada mando del panel de control sea etiquetado para que el usuario sepa en donde está ubicada cada acción, y para entender cuál es el funcionamiento de cada uno de los mandos del panel.

7. Una vez realizado el montaje del panel de control, se conectan tanto el cable USB del Arduino al ordenador, como los distintos cables que sobresalen del panel a los cables del sensor y del servomotor, con el objetivo de probar a ver si realiza realmente su función de interfaz de usuario.

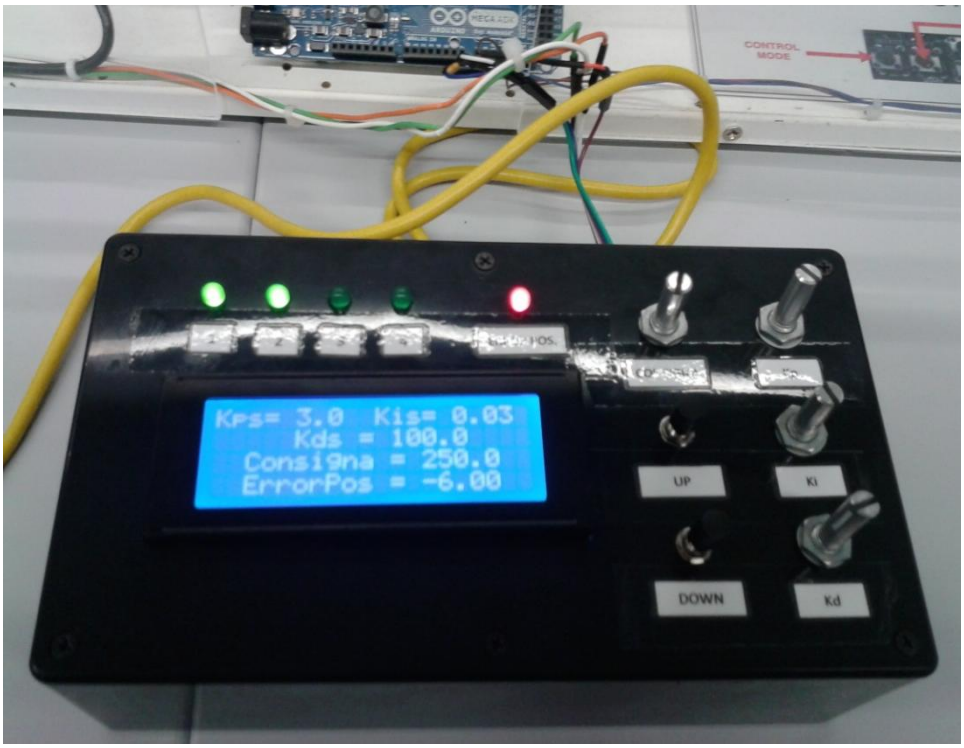


Figura 4.85 Interfaz de usuario implementada en el sistema.

Como se puede ver en la figura de arriba la interfaz de usuario implementada en el sistema funciona correctamente, con esto se puede decir que se ha cumplido uno de los objetivos de este proyecto.

A continuación se va a mostrar el sistema completo implementado, tanto la maqueta con sus dispositivos, como la interfaz de usuario con su panel de control:

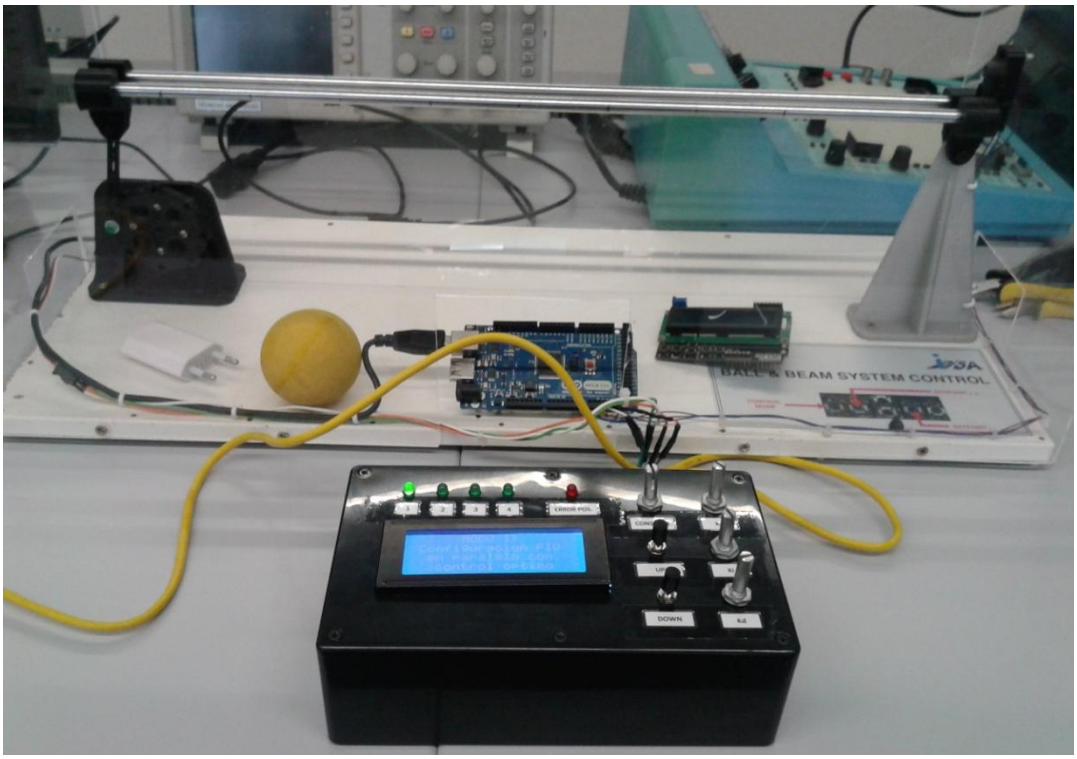


Figura 4.86 Sistema completo implementado.

4.7.5 Programación software de la interfaz de usuario

Se va a mostrar la programación software realizada mediante Arduino para el control de la interfaz de usuario implementada en el sistema. Se ha realizado un diagrama de flujo para ello:

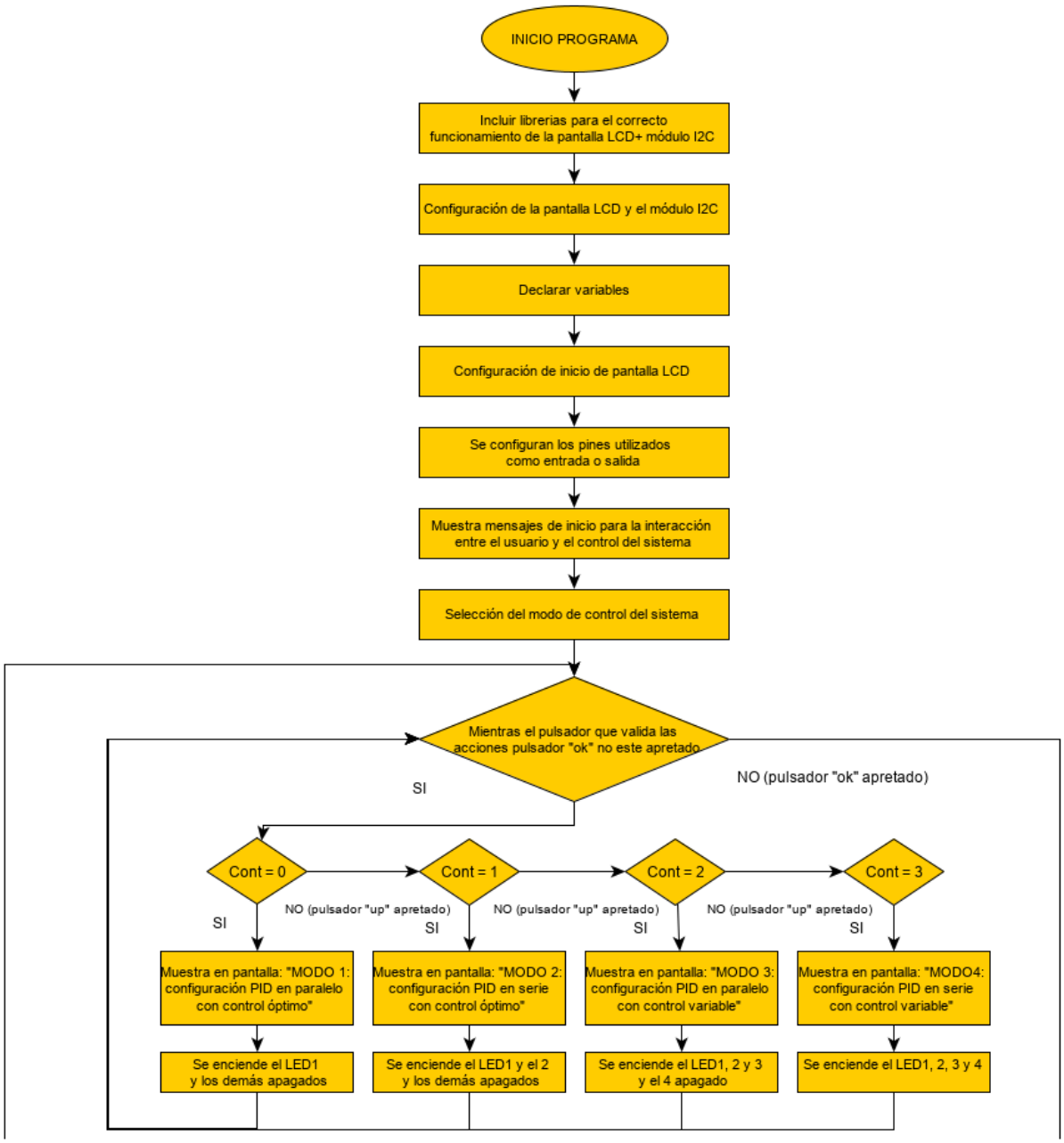


Figura 4.87 Diagrama de flujo de la programación de la interfaz de usuario, parte 1.

De esta primera parte del diagrama de flujo cabe destacar que cuando se está en el MODO 4 de control y se pulsa el pulsador “up”, el sistema pasa al MODO 1; a su vez cuando se está en el MODO 1 y se pulsa el pulsador “down”, el sistema pasa al MODO 4. Esto se realiza para que la navegación con la interfaz de usuario sea rápida, para que si estas en el MODO 1 y quieres establecer el MODO 4 no se tenga que pasar por todos los modos de control.

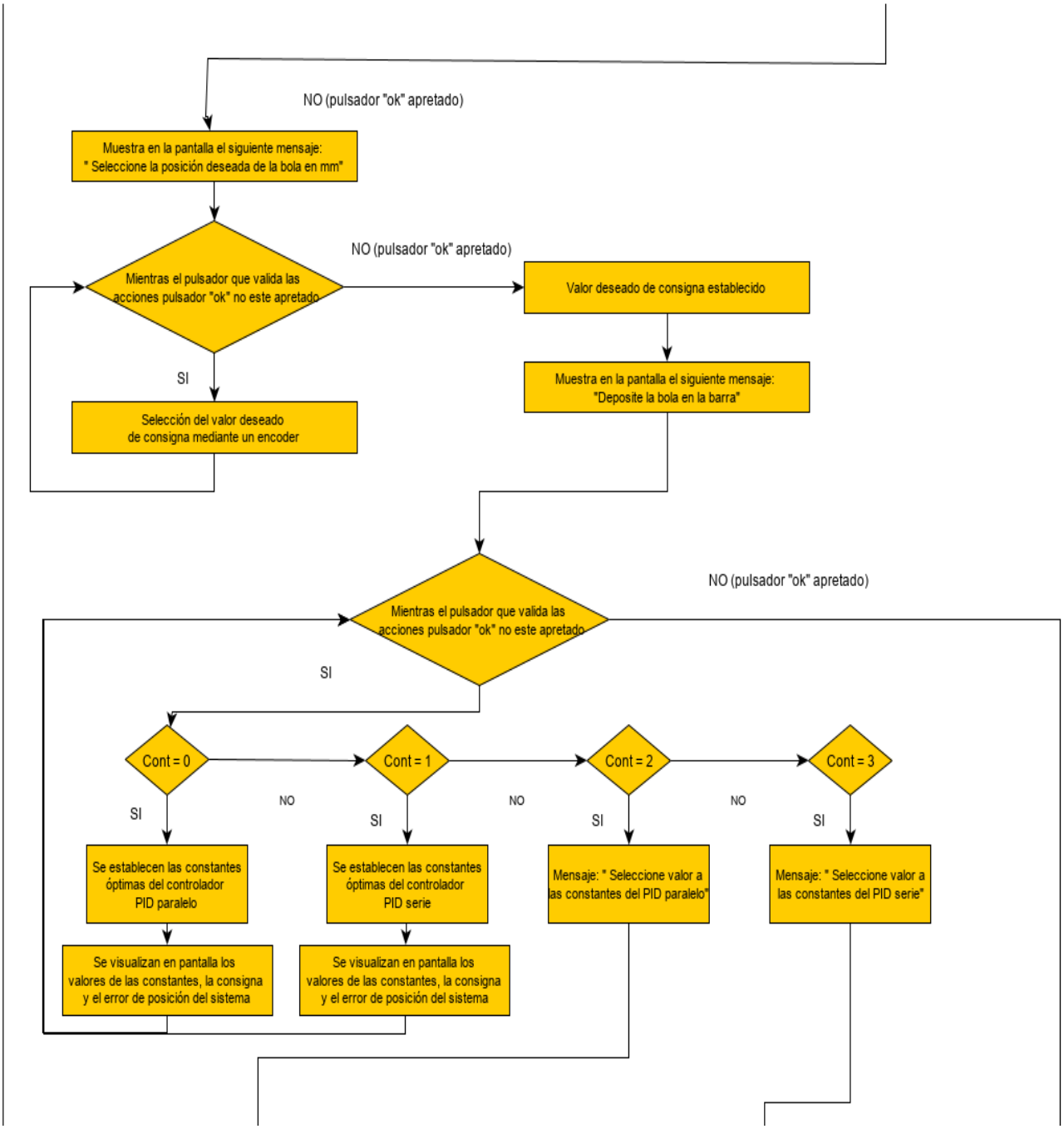


Figura 4.88 Diagrama de flujo de la programación de la interfaz de usuario, parte 2.

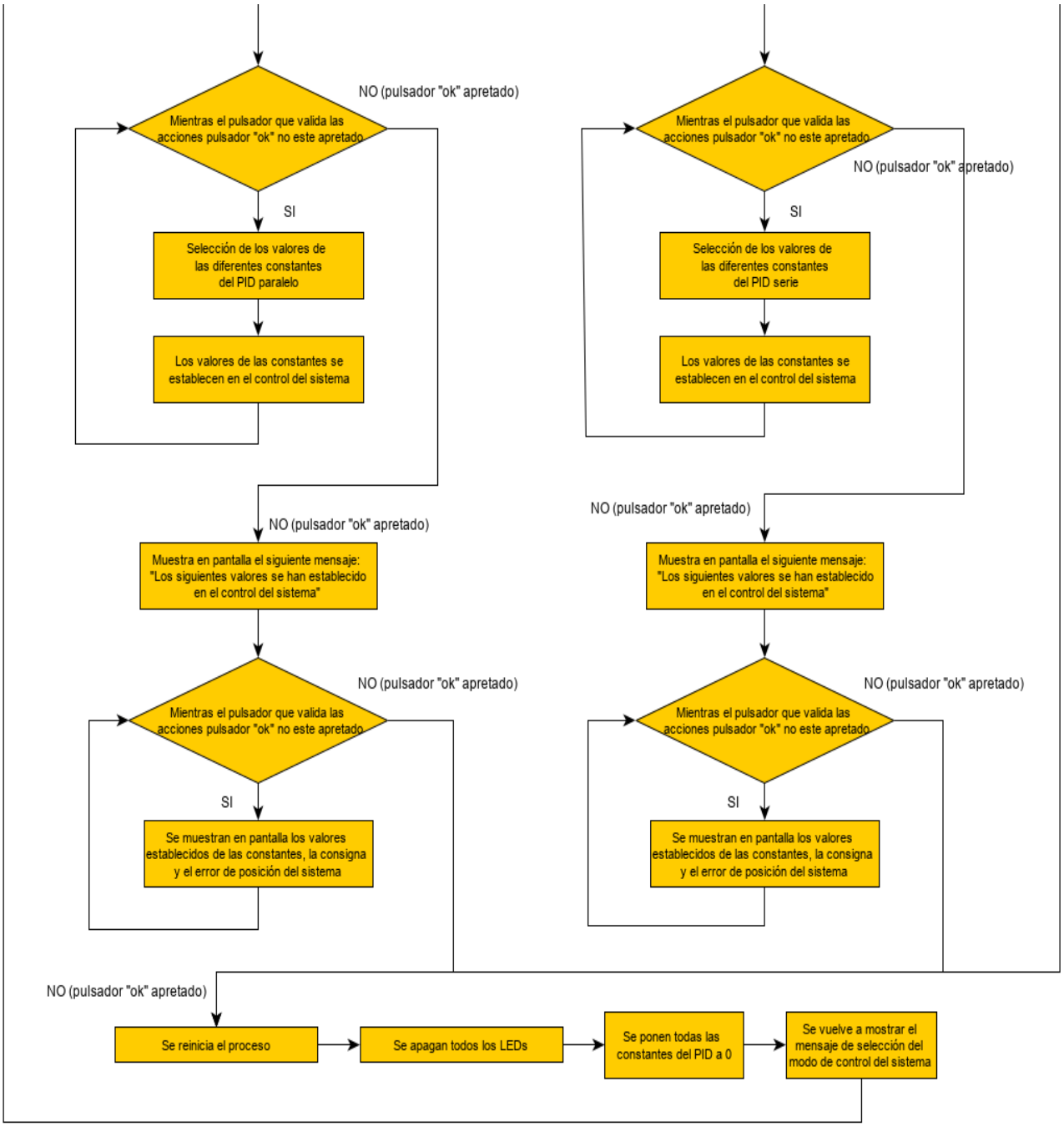


Figura 4.89 Diagrama de flujo de la programación de la interfaz de usuario, parte 3.

De esta última parte del diagrama de flujo cabe destacar que se ha establecido que el control del sistema se empiece a ejecutar, tanto en el MODO 3, como en el MODO 4 en la pantalla de selección de los valores de las constantes, esto se realiza para que durante la selección de las constantes el control del sistema este en ejecución para que se pueda ver de una manera más clara y rápida como cambia la respuesta del sistema al modificar

cada una de las acciones, en esta pantalla se parte de las constantes óptimas de cada uno de los controladores PID implementados.

5 METODOLOGÍA SEGUIDA EN EL DESARROLLO DEL TRABAJO

En este apartado se van a describir las tareas realizadas para la consecución de los objetivos impuestos en el inicio de este documento. Además de esto, cada una de las tareas se va a exponer en un diagrama de Gantt.

5.1 Descripción de tareas, fases, equipos y/o procedimientos

En este apartado se van a describir cada una de las tareas realizadas para la consecución de los objetivos impuestos en este proyecto:

1. Estudio inicial de la maqueta: Como el punto de partida que se tenía era la maqueta, la cual es un sistema barra y bola, se empezó investigando acerca de los sistemas barra y bola para poder sacar conclusiones tanto de las necesidades hardware de la maqueta (como pueda ser, que necesita de un sensor, de un actuador, de un microcontrolador, puesto que es un sistema inestable en lazo cerrado, etc.), como del objetivo de control de la misma.

2. Estudio del microcontrolador Arduino: Dentro del estudio del microcontrolador, la primera tarea que se realizó fue ver, que microcontrolador era el más idóneo, más barato y sencillo de utilizar. Por ello se ha utilizado una placa Arduino. Dentro de las placas Arduino, se investigó cual era la más conveniente para poder controlar el sistema barra y bola, teniendo en cuenta las necesidades del sistema.

   Una vez seleccionado el microcontrolador, se empezó a estudiar tanto su programación software como su plataforma hardware.

3. Estudio del sensor: Lo primero que se hizo fue seleccionar el sensor adecuado para la maqueta, para ello, se vieron los distintos tipos de sensores que existen en el mercado, y se pudo percibir el tipo de sensores que se necesitaba. El tipo de sensor que se necesitaba para el correcto funcionamiento del control, obviamente eran los sensores de proximidad, para poder detectar la bola. Dentro de estos sensores de proximidad se pudo deducir, cuáles eran los más idóneos para este sistema. Con esto ya se sabía que tanto el sensor infrarrojo como el sensor ultrasonidos eran las opciones más viables. Llegado a este punto, se hizo un análisis profundo investigando tanto las características de uno como las características del otro, viendo finalmente que por la naturaleza de la maqueta la opción más factible y más eficaz para medir la posición de la bola era la de utilizar un sensor infrarrojo.

   Una vez adquirido, se hicieron diferentes pruebas con él, para conocer tanto sus conexiones hardware como su programación en el software Arduino. Una vez conocida su programación, y viendo en sus hojas de características que el sensor no era lineal en sus medidas, se procedió a realizar la linealización del sensor, para lo cual se ensayaron dos técnicas diferentes. Para realizar ambas técnicas de linealización se realizaron varias medidas de la bola a lo largo de la barra, con diferentes distancias de la bola, esto se realizó para obtener las lecturas del sensor a diferentes distancias, para luego poder realizar la curva de linealización.

   Se probaron ambas técnicas en la maqueta y se vio cuál de ellas era la que más se acercaba a la realidad en sus medidas de posición, por ello, se decidió utilizar en la programación software de Arduino del sistema, el método de linealización del sensor por regresión lineal, puesto que era el método que más se aproximaba a los valores reales de posición de la bola.

4. Estudio del servomotor: Como era necesario para el sistema utilizar un actuador, se barajaron distintas opciones de utilización, llegando a la conclusión de que por la naturaleza del sistema, la opción idónea, la cual se ha implementado en el sistema era la de utilizar un servomotor para el control de su ángulo de giro. Dentro de los diferentes servomotores que hay en el mercado se eligió uno adecuado y barato para su uso en el sistema.

Una vez obtenido el servomotor, se investigó como se conectaba y controlaba dicho dispositivo en Arduino. Para ello, se vieron diferentes opciones de control, analizándolas y realizando alguna prueba con algunas de las posibilidades de control, se percibió cuál de ellas era el método adecuado para el control del servomotor, puesto que, era la opción de control más eficaz, debido a que era la quedaba una mayor resolución a la señal PWM generada, por lo que el ángulo de giro del servo se colocaba con una mayor precisión. Este método de control utilizado para el servo, se aplica utilizando los registros de los timers.

Una vez hecho todo esto y conociendo las características del servomotor utilizado, el *Futaba S3003*, mediante la realización de unas pruebas, con el control realizado mediante la configuración de los registros de los timers en Arduino se obtuvo tanto el rango de trabajo del servo, como el valor en el cuál la barra se sitúa en reposo. Todo esto como se sabe, se consiguió cambiando y probando una y otra vez el duty cycle de la señal PWM generada a través del pin del timer, para el control necesario del servo (control de su ángulo de giro).

5. Estudio de la pantalla LCD: Como se iba a implementar una interfaz de usuario para interactuar con el control de la maqueta, obviamente la utilización de este dispositivo para visualizar las diferentes variables era indispensable. Por ello, se empezó a adquirir información acerca de su programación y de sus conexiones hardware. Investigando un poco se vio como la utilización de un módulo I2C para conectar la pantalla LCD al Arduino era un paso importante, puesto que, simplificaba notablemente sus conexiones con el Arduino.

   Después de esto, se estudió su programación realizando pequeños proyectos, para así, adquirir los conocimientos necesarios para poder programarlo correctamente.

6. Estudio de los registros de los timers: En esta actividad básicamente lo que se realizó fue adquirir información acerca de los distintos timers de Arduino y de de la configuración de los registros de los mismos, y se eligió el timer de 16 bits adecuado tanto para el control del servo, como para la generación de interrupciones.

En esta actividad, se estudió todo lo relativo a los timers, tanto sus registros, como los bits más significativos de cada uno de sus registros, para elegir la mejor configuración de los registros de los timers utilizados, para la mejor generación de la señal PWM, utilizada para el control del ángulo de giro del servo y para la mejor generación de las interrupciones cada cierto periodo de tiempo.

En cuanto a las interrupciones, también se configuró uno de los timers para la generación de una interrupción periodo de muestreo, la cual se realiza para que tanto la señal de control, como la medida del sensor se muestreen cada cierto periodo de tiempo.

7. Estudio del sistema de control: Se analizó el sistema de control de la maqueta, para tener las ideas claras antes de pasar a realizar la programación software de todo el conjunto. Se analizaron tanto los bloques necesarios para realizar el control, como las variables que conforman todo el sistema de control. Con esto se consiguió tener una idea clara de cuál es la señal que manipula el controlador para que la variable de control corrija su posición, además, de cómo se debía calcular la señal de error en la programación. Teniendo el diagrama de bloques del sistema en mente, se pasó al estudio del controlador PID.

8. Estudio del controlador PID: En esta actividad se analizó las distintas acciones del PID, con ello, se pudo deducir que para realizar el control de posición de una bola eran necesarias aplicar las tres acciones del PID (la proporcional, la integral y la derivativa) en el sistema de control.

   Como se utiliza un microcontrolador para controlar el sistema, esto es, como se utiliza un dispositivo digital para controlar el sistema, el controlador se debe implementar en tiempo discreto (z). Teniendo en cuenta esto, se implementó el controlador PID en tempo discreto en la programación software de Arduino. Para ello, se tuvo que transformar la ecuación diferencial del PID en una ecuación en diferencias (tiempo discreto), esto se realizó aplicando el método de aproximación numérica de Euler hacia atrás, tanto para la integración como para la diferenciación. Con esto, y con las diferentes instrucciones para la

implementación de un PID en Arduino, las cuales se estudiaron y se entendieron, se consiguió realizar un control de posición de la bola adecuado.

Cabe destacar, que en la implementación del PID se introdujeron, tanto un filtro paso bajo para el cálculo de la derivada, como distintos limitadores introducidos, para el cálculo de la integral del error y para la señal de salida del PID, limitándola con los límites físicos del servomotor, estas dos limitaciones sirven para mitigar el efecto producido por el wind-up, con lo cual, a estas limitaciones se las conoce como antiwind-up.

9. Estudio de las interrupciones en Arduino: Como en el estudio del controlador PID se vio que la mejor opción para realizar la función de un periodo de muestreo en la programación de Arduino, era generar una interrupción cada "x" microsegundos en la programación, necesaria para el correcto funcionamiento del control, puesto que, el controlador PID implementado es un controlador digital, por lo que necesita de un periodo de muestreo, para muestrear su señal de control cada cierto periodo de tiempo. Teniendo en cuenta esto, se analizaron y se vieron las distintas posibilidades que existen en la programación Arduino para la generación de interrupciones.

   Además de esto, se realizaron distintas comprobaciones con el sensor infrarrojo para la selección del periodo de muestreo, puesto que los periodos de muestreo se seleccionan en base al tiempo que necesita el sensor para realizar la lectura.

   Como anteriormente se estudiaron los registros de los timers para la generación de la PWM utilizada para el control del ángulo de giro del servo, se vio como uno de los modos de operación es utilizado para la generación de interrupciones, con lo cual, viendo las diferentes posibilidades para generar una interrupción, se decidió utilizar también los registros de uno de los timers de la placa Arduino para la generación de una interrupción cada cierto periodo de tiempo. Para que tanto la medida del sensor como la señal del actuador se muestreen cada cierto periodo de tiempo.

   Como se estudiaron los registros de configuración de los timers en la actividad anterior, se implementó lo aprendido en esa tarea para configurar los registros del timer 4 utilizado para la generación de interrupciones, para generar una

interrupción cada 60ms, puesto que el periodo de muestreo utilizado es de 60ms, como se obtuvo o se seleccionó en esta actividad.

10. Realización de la programación software de todo el sistema: Teniendo en cuenta las actividades anteriores, se implementaron en la programación de Arduino las distintas instrucciones o los distintos algoritmos aprendidos para el control de cada una de las partes estudiadas. Una vez introducidas todas las instrucciones de cada una de las partes anteriores, se podía ver que el control del sistema controlaba la posición de la bola, pero de manera inadecuada, puesto que el error en régimen permanente era muy grande. Esto es debido a que las constantes introducidas para el PID no eran las adecuadas, puesto que eran unas introducidas al azar, para poder ver el funcionamiento de la maqueta.

11. Obtener el modelado de la planta y simulación del sistema: Para obtener unas constantes adecuadas para el control de la maqueta, se realizó la simulación del sistema mediante el software Matlab Simulink. Para realizar la simulación es necesario obtener el modelado de la planta, es decir, la función de transferencia de la planta, para después de esto, una vez realizada la simulación, implementar las constantes obtenidas en el sistema real, para comparar ambas respuestas.

    Sabiendo esto, el proceso seguido para la consecución de esta tarea es el siguiente:

    Primero se obtuvo el modelado de la planta, aplicando la conocida ecuación de Lagrange, una vez aplicada tanto para la posición de la bola como para el ángulo de inclinación de la barra, se consiguió las dos ecuaciones diferenciales que rigen el comportamiento del sistema, una de las cuales se despreció, prestando atención a una de ellas, la obtenida mediante la coordenada de la posición de la bola. En esta ecuación diferencial obtenida, se realizaron varias simplificaciones y linealizaciones, para poder manipularla, porque si no era demasiado compleja como para resolverla. Una vez aplicadas las simplificaciones y linealizaciones a la ecuación diferencial se realizó la transformada de Laplace, para transformarla en ecuaciones polinómicas, y así obtener la función de transferencia de la planta.

Una vez obtenido el modelado de la planta, se realizó la simulación del sistema mediante el software Matlab Simulink, para ello se diseñó un modelo de simulación, en el cual se tuvo que discretizar la función de transferencia de la planta, puesto que el dispositivo de control utilizado es un dispositivo digital, para ello, se discretizó de 2 formas, por medio de la utilización de un script en Matlab, y de forma teórica.

Para obtener unos valores correctos de las constantes, se impusieron unas especificaciones técnicas que el control de la simulación debía cumplir. Con ello se obtuvieron unas constantes que en la simulación realizan un control adecuado cumpliendo las especificaciones técnicas impuestas, pero que a la hora de implementarlas en la maqueta no reaccionaban. Es decir, después de realizar la simulación, se comparó la respuesta del sistema implementándolas en la maqueta, sin éxito.

12. Obtención de las constantes adecuadas: Para obtener los valores de las constantes del PID adecuadas para el control, se tuvo que realizar el método de ajuste por ensayo-error en la maqueta descrito en el apartado *4.6.1 Obtención de las constantes en tiempo discreto*, obteniendo así dichos valores de las constantes. Con estas constantes se puede ver que el control del sistema es óptimo, con un error mínimo.

    Una vez obtenidas las constantes óptimas para el PID paralelo, se realizó un análisis de los diferentes tipos de reguladores PID que existen, viendo las diferencias entre cada uno de ellos.

    Una vez visto esto, se decidió implementar en la programación Arduino un controlador PID en configuración serie, puesto que tiene un comportamiento diferente al PID paralelo, para ello se tuvo que obtener su ecuación diferencial, para luego transformarla a ecuaciones en diferencias (tiempo discreto). También se decidió implementarla para que la interfaz de usuario dispusiera de más de un tipo de control, puesto que me parecía interesante disponer de diferentes controles para el control de la maqueta, para sacar diferentes conclusiones acerca de uno y de otro. Para comparar ambas respuestas del sistema.

Una vez implementado en la programación Arduino, se obtuvieron las distintas constantes óptimas partiendo de las constantes óptimas del PID paralelo, aplicando las diferentes fórmulas de equivalencia entre el PID serie y el paralelo.

13. Pruebas para ajustar la programación: Una vez realizadas todas estas tareas, se realizó un ajuste fino de la programación para conseguir la mayor eficacia posible.

14. Estudio y realización de la interfaz de usuario: Estudio de la interfaz de usuario adquiriendo conocimientos sobre ella, estableciendo como se va realizar la interfaz de usuario, que ordenes se van transmitir, que se quiere visualizar, que se quiere modificar en su funcionamiento, y como se va realizar el panel de control de ella. Una vez establecido esto se seleccionaron los componentes que iban a conformar el panel de control de la interfaz, probando su programación en Arduino, y se vio que había algún componente que no hacia bien su función, por lo que se cambio por otro que si la hacía, es decir, al principio se pensó en utilizar un potenciómetro para variar el valor de las constantes, pero se vio que no era adecuado, puesto que sus valores estaban limitados al ser analógicos, debido al convertidor analógico digital de Arduino, por ello, se utilizó unos encoders para ese cometido con buen resultado.

    Una vez ajustada la programación de la interfaz de usuario, se diseño una placa PCB a través del software KiCad, la cuál es la que va hacer de elemento intermedio entre los componentes del panel control y el Arduino, con lo cual se diseño teniendo en cuenta esto; diseñando primero el esquema, estableciendo a cada uno de los componentes su huella, y con ello, diseñando finalmente la placa PCB.

    Una vez diseñada se fabricó la placa PCB a mano, por medio de una insoladora, después de esto se realizó todo el montaje de la placa PCB: taladrando sus pads y soldando sus componentes. Una vez montada la placa PCB, se soldaron diferentes cables a los componentes de la superficie del panel de control, y a cada cable se le montó sus correspondiente conector, para así, conectarse a la placa PCB. Una vez hecho esto, se realizaron los agujeros a la caja para poder

introducir los componentes en ella, con todo esto, se terminó el montaje final del panel de control de la interfaz de usuario.

15. Pruebas finales de todo el conjunto: Una vez listo el montaje del panel de control de la interfaz, se conectó la maqueta a ella, es decir, los componentes que forman parte de la maqueta como el sensor y el servomotor al panel de la interfaz, y se realizaron diferentes pruebas, con los diferentes controladores PID implementados, para así analizarlos, y obtener conclusiones acerca de cada uno de ellos.

16. Redacción del TFG: Realización de la documentación del TFG. A lo largo del proyecto se fue redactando cada parte del trabajo, pero fue al final cuando se fueron uniendo cada parte del trabajo en un solo documento llamado memoria, corrigiendo y mejorando cada parte del proyecto.

Todo el TFG se ha realizado siguiendo la filosofía Open Source: hardware, software y conocimiento libre.

5.2 Diagrama de Gantt/cronograma

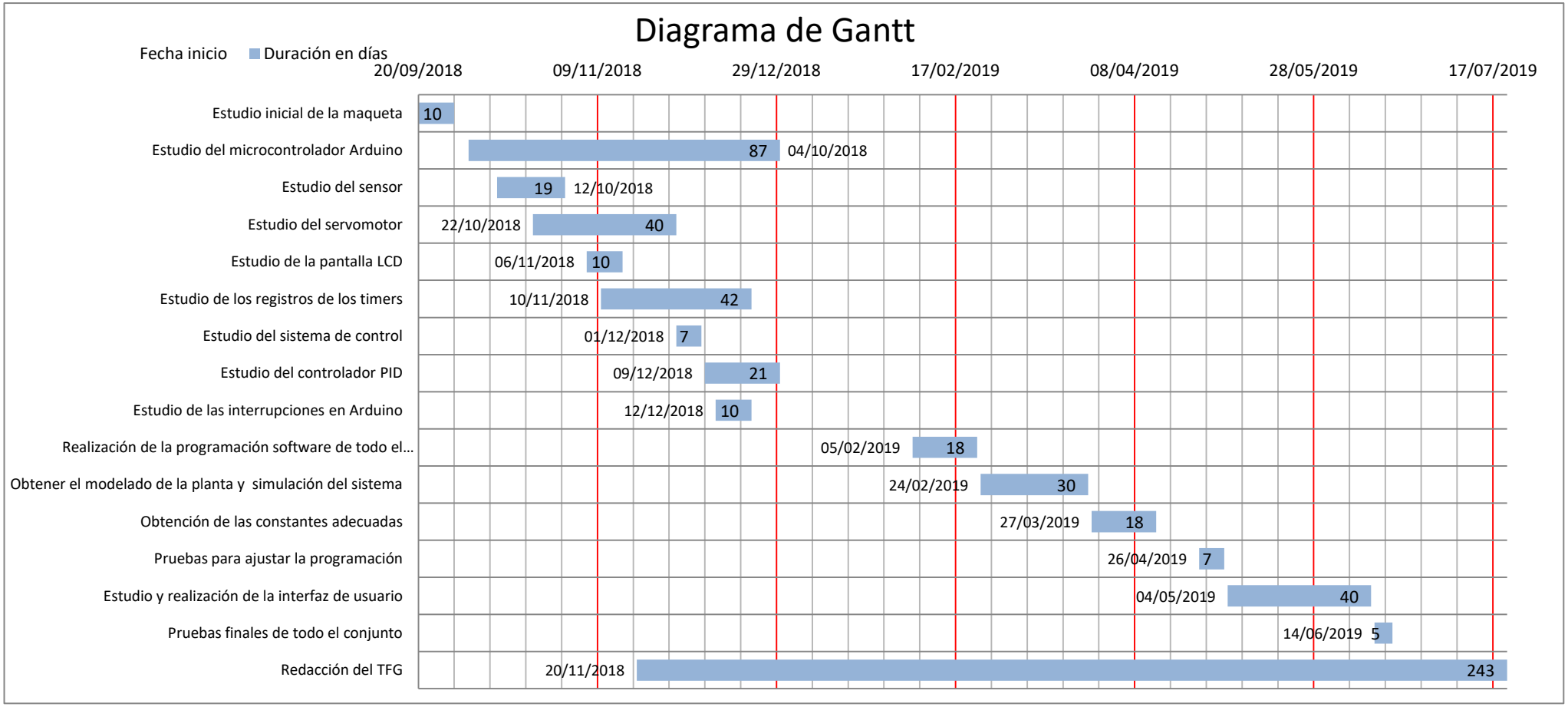


Figura 5.1 Diagrama de Gantt de las tareas del proyecto.

6 PRESUPUESTO

En este apartado se realiza el presupuesto de los materiales empleados:

Tabla 6.1 Presupuesto empleado en materiales.

Componente	Unidades	Precio Unidad (€)	Precio total (€)
Arduino MEGA	1	13,99	13,99
Sensor infrarrojo Sharp GP2Y0A21YK	1	12,62	12,62
Servomotor Futaba S3003	1	8,71	8,71
Caja de plástico de ABS Hammond 191x110x61mm	1	14,58	14,58
Pantalla LCD de 20x4 + módulo I2C	1	10,41	10,41
Encoder rotativo con pulsador	4	4,91	19,64
Diodo LED verde	4	0,24	0,96
Diodo LED rojo	1	0,52	0,52
Pulsador de panel de acción momentánea	2	3,67	7,34
Resistencia fija de 220Ω	5	0,128	0,64
Resistencia fija de 10kΩ	11	0,166	1,826
Condensador electrolítico de aluminio de 10µF	1	0,221	0,221
Tira de clavijas macho de 2,54mm el paso, pack de 30 unidades	1	4,68	4,68
Kit de conectores montables de 2,54mm el paso	1	9,99	9,99
Pack de cables	1	4,62	4,62
COSTE TOTAL			**110,747€**

7 RESULTADOS

7.1 Análisis de resultados

Se han realizado diferentes pruebas con la interfaz de usuario en el control del sistema para obtener conclusiones acerca de lo que sucede en la respuesta del sistema al modificar los controladores PID implementados con sus parámetros:

- Prueba 1: Partiendo de las constantes óptimas del controlador PID paralelo, se aumenta la acción proporcional: $K_P = 20$, $K_I = 0{,}03$ y $K_D = 100$:

  La velocidad del control aumenta, es decir, reacciona más rapido a la posición de la bola, con lo que lleva el control a la inestabilidad. En definitiva, aumentando la acción proporcional, el sistema sobrereacciona haciéndolo inestable. Se puede decir que al aumentar la acción integral aumenta la velocidad del sistema, pero disminuye la estabilidad.

- Prueba 2: Se aumenta la acción integral: $K_P = 4$, $K_I = 0{,}2$, $K_D = 100$:

  Pasa exactamente igual que en la prueba anterior, con la salvedad que en este caso solo sobrereacciona en las proximidades del valor de consigna, es decir, en las proximidades del valor de referencia la bola oscila haciendo movimientos cortos, con lo cual, se puede decir que el sistema es críticamente estable por el polo en el origen. Basicamente lo que sucede aumentndo la acción integral, es que el sistema sobrereacciona alrededor del valor deseado, por lo que el valor integral introducido es excesivo. Cuanto más acción integral se introduce, más se reduce el error en el régimen permanente, pero si te pasas de valor se inestabiliza, sobrereacciona.

- Prueba 3: Se aumenta la acción derivativa: $K_P = 4$, $K_I = 0{,}03$, $K_D = 150$:

  En este caso al aumentar la acción derivativa, el sistema de control funciona de manera adecuada, se puede decir que funciona incluso mejor a los valores establecidos para el control óptimo, puesto que el error es ínfimo, aunque si que es verdad que el control es un poco más lento, es decir, su tiempo de

establecimiento es superior a las constantes establecidas, puesto que al aumentar el termino derivativo el amortiguamiento del sistema aumenta, haciéndolo más estable, pero a su vez más lento.

- Prueba 4: Probando con los mismos valores de las constantes para el PID paralelo y para el PID serie, se puede ver que el comportamiento del sistema es diferente. Esto es debido a que la señal de salida del PID para los mismos valores de las constantes es diferente, esto se debe a la ecuación característica de cada uno de los controladores PID implementados (configuración serie y paralelo).

- Prueba 5: Al modificar los valores de las constantes del PID serie, el sistema no se comporta igual que a los cambios del PID paralelo, puesto que al modificar el K_{Ps} es como modificar todas las acciones del PID paraleo, es decir, al modificar la acción proporcional en el PID serie, todos los demás términos también quedan afectados. Se puede decir que para ajustar el PID serie solamente es necesario ajustar la constante K_{Ps}.

  Como se ha dicho, la ganancia del controlador K_{Ps} afecta a los tres términos del PID (proporcional, integral y derivativo) en serie, es decir, si se modifica la ganancia del controlador se alteran los tres términos del PID, sin embargo, la ganancia proporcional K_P del controlador PID en paralelo solo afecta al término proporcional.

7.2 Conclusiones

Como se puede ver en el desarrollo del TFG, el objetivo de control del sistema barra y bola mediante la utilización de un Arduino se ha cumplido. Gracias a esto, se han aprendido a utilizar diferentes técnicas de programación, como la utilización de los timers y de las interrupciones en la programación a través de Arduino, así como a implementar controladores PID en la programación de Arduino para el control de sistemas.

Se ha visto que el modelo de la planta obtenido no es el adecuado para este sistema. Cabe destacar, que por mucho que se prueben este tipo de maquetas es altamente improbable conseguir un control óptimo que clave la posición de la bola deseada, ya que, como ayuda para obtener una primera aproximación de las variables de los controladores, solo se tiene el realizar la simulación del sistema, y para ello es necesario el modelo de la planta, el cual al obtenerlo suele tener bastantes imprecisiones comparándolo con el de la vida real, puesto que, al utilizar ciertas simplificaciones y al linealizar las ecuaciones necesarias para obtener el modelo de la planta porque si no las ecuaciones son demasiado complejas como para obtenerlo, el modelo no se parece al sistema físico real, y por eso las constantes obtenidas en él, no arrojan buenos resultados al implementarlos en la maqueta.

Al implementar distintos tipos de controladores PID, se ha podido ver las diferencias que existen en cuanto al manejo de control entre la configuración paralela y serie del PID.

Se ha aprendido a diseñar, desarrollar y realizar una interfaz de usuario, para ello se ha aprendido a utilizar el software KiCad para el diseño y desarrollo de placas PCB, y se ha podido construir un panel diseñado para conseguir una comunicación interactiva entre usuario y sistema de control.

Con la realización de este proyecto se ha podido ver cómo afecta cada una de las acciones del controlador PID en el comportamiento del sistema al modificarlas, a través de la interfaz de usuario implementada en el sistema. Se puede decir que con esto se ha aprendido a intuir que parámetros hay que tocar si se detecta que la respuesta del sistema es muy lenta o, muy rápida. Del mismo modo, se ha empezado a saber que parámetros hay que aumentar o disminuir, si por ejemplo el sistema tiene mucho rebose hay que aumentar el termino derivativo para adelantarse al error, y así, disminuir el rebose y hacer el control del sistema más amortiguado, o si por ejemplo el error es muy grande se debe añadir acción integral al sistema de control para disminuirlo. Todo ese tipo de cosas de la teoría de los sistemas de control se han aprendido con la realización de este proyecto.

También se ha visto que cuantas más pruebas se realicen y más casos, favorables o desfavorables, se busquen más se aprende sobre la teoría de control, poniendo a prueba los conceptos estudiados teóricamente, para ver si se reflejan en la práctica.

En cuanto a la elaboración de un material claro y comprensible para facilitar el estudio de distintos aspectos de la regulación y control de sistemas, pienso que se ha cumplido.

Con el cumplimiento de todos estos objetivos se da por cumplimentado el objetivo principal de este proyecto, con la realización del cual se quería poder acercar a la gente que no conoce la teoría de control o que se adentra por primera vez en este mundillo de la regulación como se puede controlar un sistema de manera real, para que acudiendo a la interfaz de la maqueta, hiciese pruebas con los distintos parámetros de los controladores implementados y pudiese sacar conclusiones de cómo afectan cada uno de ellos en el comportamiento del sistema.

7.3 Líneas de desarrollo futuras

Una vía de desarrollo puede ser la de optimizar las posibilidades de control que ofrece este sistema utilizando alguna de las otras técnicas para la regulación, y así afinar un poco más el control del sistema para que el error de posición de la bola sea prácticamente nulo; así como modificar geometrías o utilizar distintos sensores y materiales si se tuvieran más a mano por cualquier circunstancia.

Como posible ampliación de este proyecto sería la de mejorar la interfaz de usuario, para que, en la cual, se puedan introducir y variar diferentes variables del sistema como pueda ser el periodo de muestreo. Además, se puede implementar una interfaz gráfica donde se pueda visualizar la gráfica del comportamiento del sistema en tiempo real, comparándola con la señal de referencia, para así, poder ver de manera más clara todas las especificaciones técnicas de la respuesta del sistema al implementar diferentes controladores, como el rebose, el tiempo de establecimiento, el régimen permanente...

Se podría probar a obtener las constantes discretas del controlador PID, por medio de la aplicación de Matlab denominada “System identification toolbox” mencionada en el apartado *4.5.3 Comparación de la respuesta del sistema al*

implementar las constantes obtenidas en la simulación, en la maqueta, en la cual, lo que se pretende es obtener el modelo dinámico del sistema completo, partiendo de los datos experimentales de entrada y salida extraídos del funcionamiento de la maqueta, y así, por medio de este modelo se podrían extraer las constantes discretas del controlador PID que consigan realizar un control de posición de la bola óptimo o por lo menos obtener una primera aproximación de las constantes, para luego, mediante el método de ajuste por ensayo-error realizar un ajuste fino de los parámetros del controlador.

8 REFERENCIAS

[1] Anónimo, «Github,» [En línea]. Available: https://github.com/johnrickman/LiquidCrystal_I2C. [Último acceso: 2018].

[2] Anónimo, «Prometec,» [En línea]. Available: https://www.prometec.net/bus-i2c/. [Último acceso: 2018].

9 BIBLIOGRAFÍA

[1] Anonimo, «Github,» [En línea]. Available: https://github.com/johnrickman/LiquidCrystal_I2C. [Último acceso: 2018].

[2] Anónimo, «Prometec,» [En línea]. Available: https://www.prometec.net/bus-i2c/. [Último acceso: 2018].

[3] A. Espeso, «Estudio roble,» [En línea]. Available: https://roble.uno/control-pid-barra-y-bola-arduino/. [Último acceso: 2019].

[4] Microchip, «Microchip,» 2019. [En línea]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.

[5] Arduino, «Arduino,» 2019. [En línea]. Available: https://www.arduino.cc/en/Hacking/PinMapping2560.

[6] EIB, «Apuntes relativos al grado estudiado».

[7] Anónimo. [En línea]. Available: https://www.electricaplicada.com/diferencias-sensores-infrarrojos-ultrasonicos-doble-tecnologia/. [Último acceso: 2019].

[8] Anónimo. [En línea]. Available: http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Gu%C3%ADa-de-uso-PID-para-Arduino.pdf. [Último acceso: 2019].

[9] Anónimo, «Matlab,» [En línea]. Available: http://ctms.engin.umich.edu/CTMS/index.php?aux=Home. [Último acceso: 2019].

[10] Anónimo, «Matlab,» [En línea]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=BallBeam&section=System

Modeling. [Último acceso: 2019].

[1 Anónimo, «Blog opticontrols,» [En línea]. Available:
1] https://blog.opticontrols.com/archives/124. [Último acceso: 2019].

[1 V. M. Alfaro. [En línea]. Available:
2] https://www.researchgate.net/publication/260058601_ECUACIONES_PARA_CONTROLADORES_PID_UNIVERSALES. [Último acceso: 2019].

[1 Arduino, «Arduino,» [En línea]. Available: https://store.arduino.cc/arduino-mega-
3] adk-rev3. [Último acceso: 2018].

[1 Arduino, «Arduino,» 2019. [En línea]. Available:
4] https://www.arduino.cc/reference/en/.

[1 Prometec, «Prometec,» 2019. [En línea]. Available: https://store.prometec.net/.
5]

[1 *Tutoriales MATLAB SIMULINK: Mathworks.*
6]

[1 *Tutoriales KiCad.*
7]

ANEXOS

ANEXO I. PROGRAMA COMPLETO DEL SISTEMA

```
//*****Programación completa de todo el sistema con interfaz de
usuario linealizando el sensor por regresión lineal **************


/*
CONEXIONES HARDWARE:
 * Conexiones de la pantalla LCD 20x4 + módulo I2C: SDA (pin 20), SCL
(pin 21), GND, VCC(+5V)
 * Conexiones de los encoders rotativos para las constantes: Pines
digitales de entrada, Kp: 23 y 25, Ki: 27 y 29, Kd: 31 y 33
 * Conexión del encoder rotativo con pulsador para la consigna: Pines
digitales de entrada 35, 37 y para el pulsador de "ok" 36
 * Conexiones de los pulsadores: Pines digitales de entrada 32(up) y
34(down)
 * Conexiones de los diodos LED: Pines digitales de salida 22 (LED
rojo), 24, 26, 28, 30

 * Salida de la PWM que controla la posición (ángulo de giro) del
servo por el pin 46 (servomotor conectado al pin digital de salida 46)
 * La señal de salida del sensor infrarrojo está conectada al pin
analógico de entrada A15.
 */

// Librerias para poder programar la pantalla LCD con el módulo I2C
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Constructor de la librería de LCD 20x4
// Aqui se configuran la dirección del I2C obtenida mediante el
programa "I2C scan" y las columnas y filas de la LCD
LiquidCrystal_I2C lcd(0x3F, 20, 4);

int sensorPin = 15;   //Pin Analogico donde esta conectada la señal
del sensor de distancia

int lim_inf = 5;    // Esto es para limitar la accion integral, porque
si tiene mucho error que corregir, el sistema se hace muy lento e
inestable
int lim_sup = 40;   // Se puede decir que esto es un anti-wind up (los
valores en los cuales acotamos la acción integral son entre 5 y 40mm)

int periodo = 60000;  // Periodo de muestreo en microsegundos (periodo
de muestreo de 60ms)
float Kp = 0;        // Constante Proporcional (4)
float Ki = 0;     // Constante Integral (0.03)
```

```
float Kd = 0;      // Constante Derivativa (100)

float Kps = 0;        // Constante Proporcional PID serie (3)
float Kis = 0;     // Constante Integral PID serie (0.03)
float Kds = 0;      // Constante Derivativa PID serie (100)
float Tis = Kps/Kis;    // constante de tiempo integral PID serie
float Tds = Kds/Kps;    // Constante de tiempo derivativa del PID
serie

int n = 8;          // Numero de valores con los que se hace la media
de la lectura de la posición de la bola en mm
int filtro[8];     // Array con el que se implementa el filtro pasa
bajo
int m = 5;          // Numero de valores con los que se hace la media
de la derivada del error
int filtroD[5];    // Array con el que se implementa el filtro pasa
bajo de la acción derivativa


float medida;             // Lectura del convertidor analogico digital
de 10 bits de Arduino procedente del sensor
float voltaje;            // Tensión de salida del sensor infrarrojo
(la que entra al Arduino procedente del sensor)
double posicion;          // Posición de la bola tras linealizar y
filtrar
float error_pos;          // Diferencia entre la consigna y la
posición
float error_pos_ant;      // Aqui se guarda el error de posición
anterior (el del periodo de muestreo anterior)
float error_dif;          // Derivada del error_pos
float error_int;          // Integral del error_pos
float salida_PID;         // Salida del algoritmo PID
float consigna = 250;     // Posición deseada de la bola (250)


float reposo = 1400;     // Valor que mantiene la barra horizontal
int DC;                 //Duty Cycle de la PWM que va controlar la
posición (angulo de giro) del servo

// DECLARACIÓN VARIABLES PARA LA INTERFAZ DE USUARIO:
int pulsadorup = 32;  // El pulsador que cambia de modo hacia arriba
está conectado en el pin digital 32
int estadopulsadorup = 0; // Variable en donde se va a almacenar el
estado del pulsadorup
int pulsadordown = 34;  // El pulsador que cambia de modo hacia abajo
está conectado en el pin digital 34
int estadopulsadordown = 0;  // Variable en donde se va a almacenar el
estado del pulsadordown
int contador = 0;  // Variable que se va a utilizar para pasar de un
modo a otro
int pulsadorok = 36;  // El pulsador que va a validar las acciones
está conectado en el pin digital 36 (Es el pulsador del encoder
utilizado para establecer la consigna)
int estadopulsadorok = 0;  // Variable en donde se va a almacenar el
estado del pulsadorok

int outputA_cons = 35; // Salida A del encoder que establece el valor
de la consigna, está conectada al pin digital 35
int outputB_cons = 37; // Salida B del encoder que establece el valor
de la consigna, está conectada al pin digital 37
```

```
int Aestado_actual_cons; // Variable que se utiliza para almacenar el
estado actual de la salida A del encoder de la consigna
int Aestado_ant_cons; // Variable que se utiliza para almacenar el
estado anterior de la salida A del encoder de la consigna
float cont_cons = 250; // Variable utilizada como contador de pulsos
en el encoder que establece el valor de la consigna (Valor inicial,
por defecto la mitad de la barra 250mm)
int outputA_Kp = 23; // Salida A del encoder que establece el valor de
la acción proporcional, está conectada al pin digital 23
int outputB_Kp = 25; // Salida B del encoder que establece el valor de
la acción proporcional, está conectada al pin digital 25
int Aestado_actual_Kp; // Variable que se utiliza para almacenar el
estado actual de la salida A del encoder de la acción proporcional
int Aestado_ant_Kp; // Variable que se utiliza para almacenar el
estado anterior de la salida A del encoder de la acción proporcional
float cont_p = 0; // Variable utilizada como contador de pulsos en el
encoder que establece el valor de la acción proporcional
int outputA_Ki = 27; // Salida A del encoder que establece el valor de
la acción integral, está conectada al pin digital 27
int outputB_Ki = 29; // Salida B del encoder que establece el valor de
la acción integral, está conectada al pin digital 29
int Aestado_actual_Ki; // Variable que se utiliza para almacenar el
estado actual de la salida A del encoder de la acción integral
int Aestado_ant_Ki; // Variable que se utiliza para almacenar el
estado anterior de la salida A del encoder de la acción integral
float cont_i = 0; // Variable utilizada como contador de pulsos en el
encoder que establece el valor de la acción integral
int outputA_Kd = 31; // Salida A del encoder que establece el valor de
la acción derivativa, está conectada al pin digital 31
int outputB_Kd = 33; // Salida B del encoder que establece el valor de
la acción derivativa, está conectada al pin digital 33
int Aestado_actual_Kd; // Variable que se utiliza para almacenar el
estado actual de la salida A del encoder de la acción derivativa
int Aestado_ant_Kd; // Variable que se utiliza para almacenar el
estado anterior de la salida A del encoder de la acción derivativa
float cont_d = 0; // Variable utilizada como contador de pulsos en el
encoder que establece el valor de la acción derivativa

int led0 = 22;  // Diodo LED rojo que se va a utilizar para saber si
la bola está dentro del rango de error de posición establecido (se
enciende cuando el error de posición es <= 10mm), está conectado al
pin digital 22
// Conjunto de diodos verdes que se van a utilizar para poder saber
que modo de funcionamiento se está aplicando
int led1 = 24;  // Diodo verde 1 está conectado al pin digital 24
int led2 = 26;  // Diodo verde 2 está conectado al pin digital 26
int led3 = 28;  // Diodo verde 3 está conectado al pin digital 28
int led4 = 30;  // Diodo verde 4 está conectado al pin digital 30


int dist_cal [] = {100, 125, 150, 175, 200, 225, 250, 275, 300, 325,
350, 375, 400, 425, 450};  // Las distancias medidas con sus
correspondientes lecturas del sensor,
int lect_cal [] = {768, 624, 538, 457, 403, 355, 312, 277, 247, 229,
209, 191, 178, 166, 154};  // ,para la linealización del sensor (Estas
son las medidas recabadas para la curva de linealización del sensor,
N° de muestras recogidas 15)

void setup()
{
  lcd.init();       // Se inicializa la LCD
```

```
  lcd.backlight();  // Se enciende la retroiluminación
  lcd.clear();  // Limpia toda la pantalla LCD
  lcd.setCursor(3,0);  // cursor a la cuarta posición de la pantalla y
primera fila (3, 0)
  lcd.print("PID Regulator");
  lcd.setCursor(3,2);
  lcd.print("Ball and Beam");
  lcd.setCursor(7,3);
  lcd.print("system");
  Serial.begin(9600);          // Inicializo la interface de la
comunicación serie
  analogReference(EXTERNAL);  // AREF conectado a 3.3V, esto es para
referenciar la tensión del Arduino a 3,3V, en vez de los 5V por
defecto, esto se realiza para obtener una mayor resolución de medida
en el sensor.
  delay(6000);
  lcd.clear();
  lcd.setCursor(2,1);  // cursor a la tercera posición de la pantalla
y segunda fila (2, 1)
  lcd.print("Seleccione modo");
  lcd.setCursor(1,2);
  lcd.print("de funcionamiento");
  delay(4000);
  lcd.clear();


  // Para la interface de usuario:
  pinMode(outputA_cons, INPUT);
  pinMode(outputB_cons, INPUT);
  pinMode(outputA_Kp, INPUT);
  pinMode(outputB_Kp, INPUT);
  pinMode(outputA_Ki, INPUT);
  pinMode(outputB_Ki, INPUT);
  pinMode(outputA_Kd, INPUT);
  pinMode(outputB_Kd, INPUT);
  pinMode(pulsadorup,INPUT);
  pinMode(pulsadordown,INPUT);
  pinMode(pulsadorok,INPUT);
  pinMode(led0,OUTPUT);
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
  // Se lee el valor inicial de la salida A de todos los encoders y se
guarda en la variable de "estado anterior de la salida A"
  Aestado_ant_cons = digitalRead(outputA_cons);
  Aestado_ant_Kp = digitalRead(outputA_Kp);
  Aestado_ant_Ki = digitalRead(outputA_Ki);
  Aestado_ant_Kd = digitalRead(outputA_Kd);

  //Se emplea el Timer 5 para la generacion de la PWM que va a
controlar el ángulo de giro del servo (pin de salida 46 - OC5A)
  pinMode(46, OUTPUT);  // Configuro el pin 46 (servo) como un pin de
salida digital
  TCCR5A = B10000000;   //Modo de comparación de salida no invertido
en OC5A (pin 46) y modo de generación de forma de onda-> PWM phase and
frequency correct estableciendo el valor TOP con el registro ICR5
  TCCR5B = B00010010;   //Preescaler del reloj del micro a 8 y modo de
generación de forma de onda-> PWM phase and frequency correct
estableciendo el valor TOP con el registro ICR5
```

```
  ICR5 = 20000;      // Establece el valor TOP de la señal PWM, es
decir, mediante este registro se establece el periodo de la señal PWM
que va ser de 20ms (periodo de trabajo del servo)
  OCR5A = reposo;   // Para que empiece el sistema con la barra
horizontal (ángulo de giro del servo a 90°). Esto es necesario para
poder realizar el control de posición de una bola de manera eficaz.

  //Se emplea el timer 4 para generar una interrupción cada 60ms, esto
se hace para realizar la labor de un periodo de muestreo, para que
cada 60ms se mida la posición de la bola y se ejecute la señal de
control (pin 6-OC4A)
  cli();
  TCCR4A = B00000000;    //Modo de generación de forma de onda-> Modo
CTC con comparación con el registro OCR4A y modo de comparación de
salida normal
  TCCR4B = B00001010;    //Preescaler del reloj del micro a 8 y Modo
de generación de forma de onda-> Modo CTC con comparación con el
registro OCR4A
  OCR4A = periodo-1;     //Ts=(1/Fosc)*2*PS*(1+OCR4A). Establece cada
cuanto tiempo se quiere que se genere una interrupción, es decir,
mediante este registro se establece el periodo de muestreo, que va ser
de 60ms
  TIMSK4 = B00000010;    //Habilita la interrupción por comparación
igualada de salida, poniendo a 1 el bit OCIE4A
  sei();
}

void loop()
{
  // Hasta que no se seleccione el MODO de funcionamiento no va a
salir de este bucle
  while(estadopulsadorok == LOW){
    // digitalRead () comprueba si hay voltaje en el pin o no
    estadopulsadorup = digitalRead(pulsadorup);
    estadopulsadordown = digitalRead(pulsadordown);

    if (estadopulsadorup == HIGH) {
      contador++;
      delay(1000); // Se pone 1 segundo de espera para que solo sume
uno al contador
      lcd.clear(); // Se limpia la pantalla al pasar a otro modo
    }
    if (estadopulsadordown == HIGH) {
      contador--;
      delay(1000);
      lcd.clear();
    }
    if(contador > 3) {
      contador = 0;
    }
    if(contador < 0) {
      contador = 3;
    }
    estadopulsadorok = digitalRead(pulsadorok);
    Serial.println(estadopulsadorok);

    switch (contador) {
      case 0:
        lcd.setCursor(6,0);  // cursor a la sexta posición de la
pantalla (6, 0)
        lcd.print("MODO 1:");
```

```
        lcd.setCursor(1,1);
        lcd.print("Configuracion PID");
        lcd.setCursor(2,2);
        lcd.print("en paralelo con");
        lcd.setCursor(3,3);
        lcd.print("control optimo");
        Serial.println("MODO 1");
        digitalWrite(led1, HIGH);
        digitalWrite(led2, LOW);
        digitalWrite(led3, LOW);
        digitalWrite(led4, LOW);
        break;

      case 1:
        lcd.setCursor(6,0);  // cursor a la sexta posición de la
pantalla (6, 0)
        lcd.print("MODO 2:");
        lcd.setCursor(1,1);
        lcd.print("Configuracion PID");
        lcd.setCursor(4,2);
        lcd.print("en serie con");
        lcd.setCursor(3,3);
        lcd.print("control optimo");
        Serial.println("MODO 2");
        digitalWrite(led1, HIGH);
        digitalWrite(led2, HIGH);
        digitalWrite(led3, LOW);
        digitalWrite(led4, LOW);
        break;

      case 2:
        lcd.setCursor(6,0);  // cursor a la sexta posición de la
pantalla (6, 0)
        lcd.print("MODO 3:");
        lcd.setCursor(1,1);
        lcd.print("Configuracion PID");
        lcd.setCursor(2,2);
        lcd.print("en paralelo con");
        lcd.setCursor(2,3);
        lcd.print("control variable");
        Serial.println("MODO 3");
        digitalWrite(led1, HIGH);
        digitalWrite(led2, HIGH);
        digitalWrite(led3, HIGH);
        digitalWrite(led4, LOW);
        break;

      case 3:
        lcd.setCursor(6,0);  // cursor a la sexta posición de la
pantalla (6, 0)
        lcd.print("MODO 4:");
        lcd.setCursor(1,1);
        lcd.print("Configuracion PID");
        lcd.setCursor(4,2);
        lcd.print("en serie con");
        lcd.setCursor(2,3);
        lcd.print("control variable");
        Serial.println("MODO 4");
        digitalWrite(led1, HIGH);
        digitalWrite(led2, HIGH);
        digitalWrite(led3, HIGH);
```

```
        digitalWrite(led4, HIGH);
        break;
    }
  }

  lcd.clear();
  delay(2000);
  estadopulsadorok = digitalRead(pulsadorok);
  Serial.println(estadopulsadorok);

  // Visualización consigna:
  lcd.setCursor(3,0);  // cursor a la tercera posición de la pantalla
(3, 0)
  lcd.print("Seleccione la");
  lcd.setCursor(2,1);
  lcd.print("posicion deseada");
  lcd.setCursor(1,2);
  lcd.print("de la bola en mm:");
  consigna = cont_cons; // El valor de la consigna es igual al valor
del contador del encoder de la consigna
  lcd.setCursor(7,3);
  lcd.print(consigna,1);
  // Bucle while que se utiliza para establecer el valor de la
consigna, hasta que no se pulse el pulsador ok no se valida la
consigna y no sale del bucle
  while (estadopulsadorok == LOW){
    Aestado_actual_cons = digitalRead(outputA_cons); // Lee el estado
"actual" de la salida A del encoder de la consigna
    // Si la lectura actual y la anterior de la salida A es diferente,
eso significa que ha llegado un pulso, se ha girado el encoder de la
consigna
    if (Aestado_actual_cons != Aestado_ant_cons) {
      // Si el estado de la salida B es diferente al estado actual de
la salida A, eso significa que el encoder de la consigna se ha girado
en sentido horario
      if (digitalRead(outputB_cons) != Aestado_actual_cons) {
        cont_cons ++;
        seleccionarconsigna();
      }
      // Si no, el encoder se ha girado en sentido antihorario
      else {
        cont_cons --;
        seleccionarconsigna();
      }
      Aestado_ant_cons = Aestado_actual_cons; // Se actualiza la
variable del "estado anterior" de la salida A de la consigna
    }
    estadopulsadorok = digitalRead(pulsadorok);
    Serial.println(estadopulsadorok);
  }

  lcd.clear();
  delay(2000);
  estadopulsadorok = digitalRead(pulsadorok);
  Serial.println(estadopulsadorok);
  lcd.setCursor(2,1);
  lcd.print("Deposite la bola");
  lcd.setCursor(5,2);
  lcd.print("en la barra");
  delay(5000);
  lcd.clear();
```

```
  // Bucle donde se realizan las diferentes acciones de cada modo de
funcionamiento, cuando se pulsa el pulsador ok vuelve a empezar el
algoritmo del "void loop",
  //, es decir, se vuelve a la selección del modo de funcionamiento
  while (estadopulsadorok == LOW){
    switch (contador) {
      case 0:
        Kp = 4;
        Ki = 0.03;
        Kd = 100;
        lcd.setCursor (1,0);
        lcd.print("Kp= ");
        lcd.print(Kp,1);
        Serial.print("Kp = ");
        Serial.print(Kp);
        lcd.setCursor (10,0);
        lcd.print("Ki= ");
        lcd.print(Ki,2);
        Serial.print(" Ki= ");
        Serial.print(Ki);
        lcd.setCursor(5,1);
        lcd.print("Kd = ");
        lcd.print(Kd,1);
        Serial.print("  Kd = ");
        Serial.println(Kd);
        lcd.setCursor(2,2);
        lcd.print("Consigna = ");
        lcd.print(consigna,1);
        Serial.print("Consigna = ");
        Serial.println(consigna);
        lcd.setCursor(2,3);
        lcd.print("ErrorPos = ");
        lcd.setCursor(13,3);
        lcd.print("       ");  //Para que no se queden numeros
fantasma en la pantalla LCD
        lcd.setCursor(13,3);
        lcd.print(error_pos);
        Serial.print("Error = ");
        Serial.println(error_pos);
        estadopulsadorok = digitalRead(pulsadorok);
        Serial.println(estadopulsadorok);
        Serial.print("salida_PID =  ");
        Serial.println(salida_PID);
        break;

      case 1:
        Kps = 3;
        Kis = 0.03;
        Kds = 100;
        Tis = Kps/Kis;    // constante de tiempo integral del PID
serie
        Tds = Kds/Kps;    // Constante de tiempo derivativa del PID
serie
        lcd.setCursor (0,0);
        lcd.print("Kps= ");
        lcd.print(Kps,1);
        Serial.print("Kps = ");
        Serial.print(Kps);
        lcd.setCursor (10,0);
        lcd.print("Kis= ");
```

```
        lcd.print(Kis,2);
        Serial.print("  Kis = ");
        Serial.print(Kis);
        lcd.setCursor(4,1);
        lcd.print(" Kds = ");
        lcd.print(Kds,1);
        Serial.print("  Kds = ");
        Serial.println(Kds);
        lcd.setCursor(2,2);
        lcd.print("Consigna = ");
        lcd.print(consigna,1);
        Serial.print("Consigna = ");
        Serial.println(consigna);
        lcd.setCursor(2,3);
        lcd.print("ErrorPos = ");
        lcd.setCursor(13,3);
        lcd.print("       ");  //Para que no se queden numeros
fantasma en la pantalla LCD
        lcd.setCursor(13,3);
        lcd.print(error_pos);
        Serial.print("Error = ");
        Serial.println(error_pos);
        estadopulsadorok = digitalRead(pulsadorok);
        Serial.println(estadopulsadorok);
        Serial.print("salida_PID =  ");
        Serial.println(salida_PID);
        break;

      case 2:
        lcd.setCursor (2,0);
        lcd.print("Seleccione valor");
        lcd.setCursor(2,1);
        lcd.print("a las constantes");
        lcd.setCursor(2,2);
        lcd.print("del PID paralelo");
        delay(4000);
        lcd.clear();
        //Inicializo la selección con los valores de las constantes
del PID paralelo optimas:
        cont_p = 4;
        cont_i = 3;
        cont_d = 100;
        // Visualización de constantes:
        Kp = cont_p; // El valor de la constante Kp es igual al valor
del contador del encoder de la acción proporcional
        Ki = cont_i/100; // El valor de la constante Ki es igual al
valor del contador del encoder de la acción integral dividido por 100.
        Kd = cont_d; // El valor de la constante Kd es igual al valor
del contador del encoder de la acción derivativa.
        lcd.setCursor (5,0);
        lcd.print("Kp = ");
        lcd.print(Kp,1);
        Serial.print("Kp = ");
        Serial.print(Kp);
        lcd.setCursor(5,1);
        lcd.print("Ki = ");
        lcd.print(Ki,2);
        Serial.print("  Ki = ");
        Serial.print(Ki);
        lcd.setCursor(5,2);
        lcd.print("Kd = ");
```

```
        lcd.setCursor(10,2);
        lcd.print(Kd,1);
        Serial.print("  Kd = ");
        Serial.println(Kd);
        // En este bucle se seleccionan los valores de las diferentes
acciones del PID paralelo, hasta que no se valide la selección con el
pulsador ok no sale del bucle
        while (estadopulsadorok == LOW) {
          Aestado_actual_Kp = digitalRead(outputA_Kp); // Lee el
estado "actual" de la salida A del encoder de la acción proporcional
          Aestado_actual_Ki = digitalRead(outputA_Ki); // Lee el
estado "actual" de la salida A del encoder de la acción integral
          Aestado_actual_Kd = digitalRead(outputA_Kd); // Lee el
estado "actual" de la salida A del encoder de la acción derivativa
          // Para el encoder que establece el valor de la acción
proporcional:
          // Si la lectura actual y la anterior de la salida A es
diferente, eso significa que ha llegado un pulso, se ha girado el
encoder de la acción P
          if (Aestado_actual_Kp != Aestado_ant_Kp) {
            // Si el estado de la salida B es diferente al estado
actual de la salida A, eso significa que el encoder de la acción P se
ha girado en sentido horario
            if (digitalRead(outputB_Kp) != Aestado_actual_Kp) {
              cont_p ++;
              seleccionarKp();
            }
            // Si no, el encoder de la acción P se ha girado en
sentido antihorario
            else {
              cont_p--;
              seleccionarKp();
            }
            Aestado_ant_Kp = Aestado_actual_Kp; // Se actualiza la
variable del "estado anterior" de la salida A del encoder de la acción
P.
          }
          // Lo mismo para el encoder que establece el valor de la
acción integral:
          if (Aestado_actual_Ki != Aestado_ant_Ki) {
            if (digitalRead(outputB_Ki) != Aestado_actual_Ki) {
              cont_i ++;
              seleccionarKi();
            }
            else {
              cont_i--;
              seleccionarKi();
            }
            Aestado_ant_Ki = Aestado_actual_Ki;
          }
          // Lo mismo para el encoder que establece el valor de la
acción derivativa:
          if (Aestado_actual_Kd != Aestado_ant_Kd) {
            if (digitalRead(outputB_Kd) != Aestado_actual_Kd) {
              cont_d ++;
              seleccionarKd();
            }
            else {
              cont_d--;
              seleccionarKd();
            }
```

```
            Aestado_ant_Kd = Aestado_actual_Kd;
          }
          estadopulsadorok = digitalRead(pulsadorok);
          Serial.println(estadopulsadorok);
        }

        lcd.clear();
        delay(2000);
        estadopulsadorok = digitalRead(pulsadorok);
        Serial.println(estadopulsadorok);
        lcd.setCursor (3,0);
        lcd.print("Los siguientes");
        lcd.setCursor(4,1);
        lcd.print("valores sean");
        lcd.setCursor(2,2);
        lcd.print("establecido en el");
        lcd.setCursor(0,3);
        lcd.print("control del sistema");
        delay(5000);
        lcd.clear();
        // Bucle en donde se visualizan en la LCD la consigna y los
valores de las constantes del PID seleccionados, si se pulsa el
pulsador ok se sale y vuelve a la selección del modo de funcionamiento
        while (estadopulsadorok == LOW) {
          lcd.setCursor (1,0);
          lcd.print("Kp= ");
          lcd.print(Kp,1);
          Serial.print("Kp = ");
          Serial.print(Kp);
          lcd.setCursor (10,0);
          lcd.print("Ki= ");
          lcd.print(Ki,2);
          Serial.print("  Ki = ");
          Serial.print(Ki);
          lcd.setCursor(5,1);
          lcd.print("Kd = ");
          lcd.print(Kd,1);
          Serial.print("  Kd = ");
          Serial.println(Kd);
          lcd.setCursor(2,2);
          lcd.print("Consigna = ");
          lcd.print(consigna,1);
          Serial.print("Consigna = ");
          Serial.println(consigna);
          lcd.setCursor(2,3);
          lcd.print("ErrorPos = ");
          lcd.setCursor(13,3);
          lcd.print("       ");  //Para que no se queden numeros
fantasma en la pantalla LCD
          lcd.setCursor(13,3);
          lcd.print(error_pos);
          Serial.print("Error = ");
          Serial.println(error_pos);
          estadopulsadorok = digitalRead(pulsadorok);
          Serial.println(estadopulsadorok);
          Serial.print("salida_PID =  ");
          Serial.println(salida_PID);
        }
        break;

      case 3:
```

```
        lcd.setCursor (2,0);
        lcd.print("Seleccione valor");
        lcd.setCursor(2,1);
        lcd.print("a las constantes");
        lcd.setCursor(3,2);
        lcd.print("del PID serie");
        delay(4000);
        lcd.clear();
        //Inicializo la selección con los valores de las constantes
del PID SERIE optimas:
        cont_p = 3;
        cont_i = 3;
        cont_d = 100;
        // Visualización de constantes:
        Kps = cont_p; // El valor de la constante Kps es igual al
valor del contador del encoder de la acción proporcional
        Kis = cont_i/100; // El valor de la constante Kis es igual al
valor del contador del encoder de la acción integral dividido por 100.
        Kds = cont_d; // El valor de la constante Kds es igual al
valor del contador del encoder de la acción derivativa.
        Tis = Kps/Kis;  // Constante de tiempo integral del PID serie
        Tds = Kds/Kps;  // Constante de tiempo derivativa del PID
serie
        lcd.setCursor (5,0);
        lcd.print("Kps = ");
        lcd.print(Kps,1);
        Serial.print("Kps = ");
        Serial.print(Kps);
        lcd.setCursor(5,1);
        lcd.print("Kis = ");
        lcd.print(Kis,2);
        Serial.print("  Kis = ");
        Serial.print(Kis);
        lcd.setCursor(5,2);
        lcd.print("Kds = ");
        lcd.setCursor(11,2);
        lcd.print(Kds,1);
        Serial.print("  Kds = ");
        Serial.println(Kds);
        // En este bucle se seleccionan los valores de las diferentes
acciones del PID serie, hasta que no se valide la selección con el
pulsador ok no sale del bucle
        while (estadopulsadorok == LOW) {
          Aestado_actual_Kp = digitalRead(outputA_Kp); // Lee el
estado "actual" de la salida A del encoder de la acción proporcional
          Aestado_actual_Ki = digitalRead(outputA_Ki); // Lee el
estado "actual" de la salida A del encoder de la acción integral
          Aestado_actual_Kd = digitalRead(outputA_Kd); // Lee el
estado "actual" de la salida A del encoder de la acción derivativa
          // Para el encoder que establece el valor de la acción
proporcional:
          // Si la lectura actual y la anterior de la salida A es
diferente, eso significa que ha llegado un pulso, se ha girado el
encoder de la acción P
          if (Aestado_actual_Kp != Aestado_ant_Kp) {
            // Si el estado de la salida B es diferente al estado
actual de la salida A, eso significa que el encoder de la acción P se
ha girado en sentido horario
            if (digitalRead(outputB_Kp) != Aestado_actual_Kp) {
              cont_p ++;
              seleccionarKps();
```

```
            }
            // Si no, el encoder de la acción P se ha girado en
sentido antihorario
            else {
              cont_p--;
              seleccionarKps();
            }
            Aestado_ant_Kp = Aestado_actual_Kp; // Se actualiza la
variable del "estado anterior" de la salida A del encoder de la acción
P.
          }
          // Lo mismo para el encoder que establece el valor de la
acción integral:
          if (Aestado_actual_Ki != Aestado_ant_Ki) {
            if (digitalRead(outputB_Ki) != Aestado_actual_Ki) {
              cont_i ++;
              seleccionarKis();
            }
            else {
              cont_i--;
              seleccionarKis();
            }
            Aestado_ant_Ki = Aestado_actual_Ki;
          }
          // Lo mismo para el encoder que establece el valor de la
acción derivativa:
          if (Aestado_actual_Kd != Aestado_ant_Kd) {
            if (digitalRead(outputB_Kd) != Aestado_actual_Kd) {
              cont_d ++;
              seleccionarKds();
            }
            else {
              cont_d--;
              seleccionarKds();
            }
            Aestado_ant_Kd = Aestado_actual_Kd;
          }
          Tis = Kps/Kis;  // Constante de tiempo integral del PID
serie
          Tds = Kds/Kps;  // Constante de tiempo derivativa del PID
serie
          estadopulsadorok = digitalRead(pulsadorok);
          Serial.println(estadopulsadorok);
        }

        lcd.clear();
        delay(2000);
        estadopulsadorok = digitalRead(pulsadorok);
        Serial.println(estadopulsadorok);
        lcd.setCursor (3,0);
        lcd.print("Los siguientes");
        lcd.setCursor(4,1);
        lcd.print("valores sean");
        lcd.setCursor(2,2);
        lcd.print("establecido en el");
        lcd.setCursor(0,3);
        lcd.print("control del sistema");
        delay(5000);
        lcd.clear();
```

```
        // Bucle en donde se visualizan en la LCD la consigna y los
valores de las constantes del PID serie seleccionados, si se pulsa el
pulsador ok se sale y vuelve a la selección del modo de funcionamiento
        while (estadopulsadorok == LOW) {
          lcd.setCursor (0,0);
          lcd.print("Kps= ");
          lcd.print(Kps,1);
          Serial.print("Kps = ");
          Serial.print(Kps);
          lcd.setCursor (10,0);
          lcd.print("Kis= ");
          lcd.print(Kis,2);
          Serial.print("  Kis = ");
          Serial.print(Kis);
          lcd.setCursor(4,1);
          lcd.print(" Kds = ");
          lcd.print(Kds,1);
          Serial.print("  Kds = ");
          Serial.println(Kds);
          lcd.setCursor(2,2);
          lcd.print("Consigna = ");
          lcd.print(consigna,1);
          Serial.print("Consigna = ");
          Serial.println(consigna);
          lcd.setCursor(2,3);
          lcd.print("ErrorPos = ");
          lcd.setCursor(13,3);
          lcd.print("       ");  //Para que no se queden numeros
fantasma en la pantalla LCD
          lcd.setCursor(13,3);
          lcd.print(error_pos);
          Serial.print("Error = ");
          Serial.println(error_pos);
          estadopulsadorok = digitalRead(pulsadorok);
          Serial.println(estadopulsadorok);
          Serial.print("salida_PID =  ");
          Serial.println(salida_PID);
        }
        break;
    }

    delay(200);
  }

  // Esto se realiza para que pueda empezar correctamente de nuevo, a
partir de la selección del modo de funcionamiento
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  lcd.clear();
  delay(2000);
  estadopulsadorok = digitalRead(pulsadorok);
  Serial.println(estadopulsadorok);
  contador = 0;  // Se pone a 0, para que cuando se vuelva a la
selección del modo de funcionamiento el contador este a 0, es decir,
MODO 1
  cont_cons = 250; // Se pone a 250 (valor inicial, por defecto mitad
de la barra) para que cuando se vuelva a asignar el valor de la
consigna, el contador empiece de manera correcta
```

```
  Kp = 0;  // Constante Proporcional a 0, para que cuando empiece el
programa no se ejecute la señal de control establecida anteriormente
  Ki = 0;  // Constante Integral a 0, para que cuando empiece el
programa no se ejecute la señal de control establecida anteriormente
  Kd = 0;  // Constante Derivativa a 0, para que cuando empiece el
programa no se ejecute la señal de control establecida anteriormente
  Kps = 0; // Constante Proporcional PID serie a 0, para que cuando
empiece el programa no se ejecute la señal de control establecida
anteriormente
  Kis = 0; // Constante Integral PID serie a 0, para que cuando
empiece el programa no se ejecute la señal de control establecida
anteriormente
  Kds = 0; // Constante Derivativa PID serie a 0, para que cuando
empiece el programa no se ejecute la señal de control establecida
anteriormente
  lcd.setCursor(2,1);  // cursor a la segunda posición de la pantalla
y segunda fila (2, 1)
  lcd.print("Seleccione modo");
  lcd.setCursor(1,2);
  lcd.print("de funcionamiento");
  delay(4000);
  lcd.clear();
}


// Subrutinas para poder visualizar en todo momento las variables,
cuando se está realizando el proceso de elegir los valores
void seleccionarconsigna()
{
  cont_cons = constrain(cont_cons, 100,450); // Acota el valor de la
consigna entre el minimo y el maximo valor posible de asignación,
puesto que son los valores entre los cuales se ha realizado la
linealización del sensor.
  consigna = cont_cons; // El valor de la consigna es igual al valor
del contador del encoder de la consigna
  lcd.setCursor(7,3);
  lcd.print(consigna,1);
  Serial.print("Consigna = ");
  Serial.println(consigna);
}

void seleccionarKp()
{
  Kp = cont_p; // El valor de la constante Kp es igual al valor del
contador del encoder de la acción proporcional
  lcd.setCursor (5,0);
  lcd.print("Kp = ");
  lcd.print("     ");  //Para que no se queden numeros fantasma en la
pantalla
  lcd.setCursor(10,0);
  lcd.print(Kp,1);
  Serial.print("Kp = ");
  Serial.print(Kp);
}

void seleccionarKi()
{
  Ki = cont_i/100; // El valor de la constante Ki es igual al valor
del contador del encoder de la acción integral dividido por 100.
  lcd.setCursor(5,1);
  lcd.print("Ki = ");
```

```
  lcd.print("     ");  //Para que no se queden numeros fantasma en la
pantalla
  lcd.setCursor(10,1);
  lcd.print(Ki,2);
  Serial.print("  Ki = ");
  Serial.print(Ki);
}

void seleccionarKd()
{
  Kd = cont_d; // El valor de la constante Kd es igual al valor del
contador del encoder de la acción derivativa.
  lcd.setCursor(5,2);
  lcd.print("Kd = ");
  lcd.print("     ");  //Para que no se queden numeros fantasma en la
pantalla
  lcd.setCursor(10,2);
  lcd.print(Kd,1);
  Serial.print("  Kd = ");
  Serial.println(Kd);
}

void seleccionarKps()
{
  Kps = cont_p; // El valor de la constante Kps es igual al valor del
contador del encoder de la acción proporcional
  lcd.setCursor (5,0);
  lcd.print("Kps = ");
  lcd.print("     "); //Para que no se queden numeros fantasma en la
pantalla
  lcd.setCursor(11,0);
  lcd.print(Kps,1);
  Serial.print("Kps = ");
  Serial.print(Kps);
}

void seleccionarKis()
{
  Kis = cont_i/100; // El valor de la constante Kis es igual al valor
del contador del encoder de la acción integral dividido por 100.
  lcd.setCursor(5,1);
  lcd.print("Kis = ");
  lcd.print("     "); //Para que no se queden numeros fantasma en la
pantalla
  lcd.setCursor(11,1);
  lcd.print(Kis,2);
  Serial.print("  Kis = ");
  Serial.print(Kis);
}

void seleccionarKds()
{
   Kds = cont_d; // El valor de la constante Kds es igual al valor del
contador del encoder de la acción derivativa.
   lcd.setCursor(5,2);
   lcd.print("Kds = ");
   lcd.print("     ");  //Para que no se queden numeros fantasma en la
pantalla
   lcd.setCursor(11,2);
   lcd.print(Kds,1);
   Serial.print("  Kds = ");
```

```
   Serial.println(Kds);
}



ISR(TIMER4_COMPA_vect){  // LLama a la rutina de servicio de
interrupción cada 60ms con el timer 4 (se emplea una interrupción de
tipo de comparación igualada de salida)

    cli();
    error_pos_ant = error_pos;    //Guardamos la anterior lectura de
error de posicion como anterior, para utilizarla en el siguiente ciclo
de programa.

    //******************** CÁLCULO DE LA POSICIÓN DE LA BOLA
**********************

    medida = analogRead(sensorPin);  // En la variable "medida"
almacena el valor de la lectura del sensor infrarrojo.
    medida = constrain(medida, lect_cal[14], lect_cal[0]);  //Acota la
lectura medida por el sensor entre el mínimo y el máximo valor del
array de "lectura calculada" (Son los valores medidos para poder
realizar la curva de linealización del sensor)

    voltaje = medida * (3.3 / 1023.0);  // Mediante la lectura del
sensor (que va de 0 a 1023), se obtiene el valor de la tensión de
salida del sensor (0 a 3.3 V), se hace una regla de 3

    posicion =   44.313 * pow(voltaje, 6)
               - 434.95 * pow(voltaje, 5)
               + 1765.7 * pow(voltaje, 4)
               - 3819.8 * pow(voltaje, 3)
               + 4723.8 * pow(voltaje, 2)
               - 3351.3 * voltaje
               + 1323.2;     // Regresión lineal (fórmula) obtenida
por medio de la curva de linealización, que se utiliza para calcular
la distancia (en mm) en función de la tensión de salida del sensor.

    posicion = constrain(posicion,dist_cal[0],dist_cal[14]);   //
Acota la señal de posición de la bola entre el mínimo y el máximo
valor del array de "distancia calculada" (Son los valores medidos para
poder realizar la curva de linealización del sensor)


     // Ahora se va aplicar un FILTRO DIGITAL PASO BAJO, para mejorar
la precisión de la medida de posición de la bola
    for (int i=0; i<n-1; i++){    //Desplazo el array a la izquierda,
eliminando la lectura de posición más antigua
      filtro[i]=filtro[i+1];      // Esto es para que se vaya
actualizando el array, se desplaza todo el array un lugar a la
izquierda, dejando libre el ultimo hueco para poner la lectura de
posición nueva
    }
    filtro[n-1] = posicion;      //La posicion actual la guardo al
final del array (posición[7])

    posicion=0;      // Pone la variable posición a 0

    for (int i=0; i<n; i++){
      posicion = posicion + filtro[i];  // Aqui se suman todas las
medidas de posición que tengo en el array(8)
```

```
    }
    posicion = posicion / n;      // Y se saca la media de las medidas
de Posición de la bola
     // Esto se realiza para hacer el efecto de un filtro paso bajo,
para despreciar las posibles medidas de posición defectuosas o
incoherentes y para mejorar la precisión de las medidas
    // Aqui se acaba el filtro paso bajo


    //************************************ CONTROLADOR PID
*******************************************************

    // Se implementa un controlador PID para establecer la bola en la
posición especificada en la consigna ante perturbaciones externas,
    // es decir, se va utilizar un controlador PID para corregir las
posibles desviaciones que ocasione la perturbación.
    // El controlador PID se implementa en Arduino implementandolo en
ecuaciones en diferencias (tiempo discreto (z)), en este caso,
basándose en el metodo de Euler hacia atrás,
    // ,que es el metodo más utilizado.

    error_pos = consigna - posicion;    //El error entre la consigna y
la posicion es la entrada al PID


    //************************************ Calculo de la derivada del
error de posición

    // Se va a realizar un FILTRO DIGITAL PASO BAJO, puesto que se ha
visto que si se realiza este filtro en la acción derivativa el control
del sistema es más estable, es decir, la bola se queda quieta y el
servo no se mueve tanto.
    for (int i=0; i<m-1; i++){  //Desplazo el array a la izquierda,
eliminando la derivada del error más antigua
      filtroD[i] = filtroD[i+1];  // Esto es para que se vaya
actualizando el array, se desplaza todo el array un lugar a la
izquierda, dejando libre el ultimo hueco para poner el calculo nuevo
    }
    filtroD[m-1] = (error_pos - error_pos_ant); // La derivada del
error actual la guardo al final del array (posición[4])
    // La derivada del error o el error diferencial se calcula
realizando la resta del error de posición actual menos el error de
posición anterior

    error_dif = 0;  // Se pone la variable de la derivada del error a
0

    for (int i=0; i<m; i++){ // Se calcula la media
      error_dif = error_dif + filtroD[i];  // Aqui se suman todos los
calculos de la derivada del error almacenadas en el array(5)
    }
    error_dif = error_dif / m;  // Se saca la media de las medidas de
la derivada del error de posición
   // Con esto se mejora la precisión del cálculo de la derivada del
error (acción derivativa)
   // Aqui acaba el filtro paso bajo digital


    //************************************ Cálculo de la integral del
error de posición
```

```
    if(abs(error_pos)>lim_inf && abs(error_pos)<lim_sup){   // Solo se
aplica la acción integral, cuando los valores del error de posición
esten dentro de los limites establecidos para la acción integral. Se
puede decir que esto es un anti-wind up.
      // Se acota la acción integral en las proximidades del
equilibrio, entre 5 y 40, es decir, si el error de posición está entre
5 y 40mm tanto positivos como negativos se aplica la acción integral.
      // Esto es para limitar la accion integral, porque si tiene
demasiado error que corregir o si se acumulado demasiado error en la
acción integral, el sistema se hace muy lento debido a los retrasos
que se generan e inestable.
      if(contador == 0 || contador == 2) {  // PID paralelo
        error_int = error_int + Ki * error_pos; // Aqui se calcula el
error integral, que es el error integral anterior más la suma del
error de posición actual multiplicado por el valor de Ki.
      }
      else {  // PID SERIE
        error_int = error_int + error_pos;   // PARA EL PID SERIE,
Aqui se calcula el error integral, que es el error integral anterior
más la suma del error de posición actual.
      }
    }
    else {
      error_int = 0;   // Si el error de posición esta fuera de los
limites de la accion integral, se realiza un controlador PD, para que
el sistema no sufra retrasos y no se haga tan lento e inestable
    }

    //************************************ Acción PID

    if(contador == 0 || contador == 2) {  // PID paralelo
      salida_PID = Kp * error_pos + Kd * error_dif + error_int;
 //Formula para obtener la señal de salida del controlador PID
paralelo (para obtener el ángulo de giro del servo necesario)
    }
    else {  // PID SERIE
      salida_PID = Kps * (error_pos * (1 + Tds/Tis) + Tds * error_dif
+ (1/Tis) * error_int); //Formula para obtener la señal de salida del
controlador PID en serie (Tambien llamado controlador PID
interactivo), para obtener el ángulo de giro del servo necesario
    }

    DC = reposo + salida_PID;     // El reposo es el valor en el cual
la barra está horizontal,
                                  // ,entonces lo que se hace es sumar
o restar ese valor de reposo con la salida del PID,
                                  // esto se realiza para que la barra
se mueva en ambos sentidos dependiendo de la polaridad del error de
posición .

     // Un ejemplo en cuanto a lo anterior, por ejemplo si la medida
de posición da 300 y la consigna la tienes a 250, se necesita que la
bola se mueva a la derecha para establecerse en 250;
     //para conseguir esto, todo sucede de la siguiente manera:
     // el error de posición será negativo, con lo cual, la señal de
salida del PID será negativa, entonces se resta esa señal negativa del
PID al valor en el cual la barra está horizontal (90°),
     //, entonces el valor de duty cycle de la señal PWM será un valor
inferior al de reposo, con lo cual, el ángulo de giro del servo será
inferior a 90°,
```

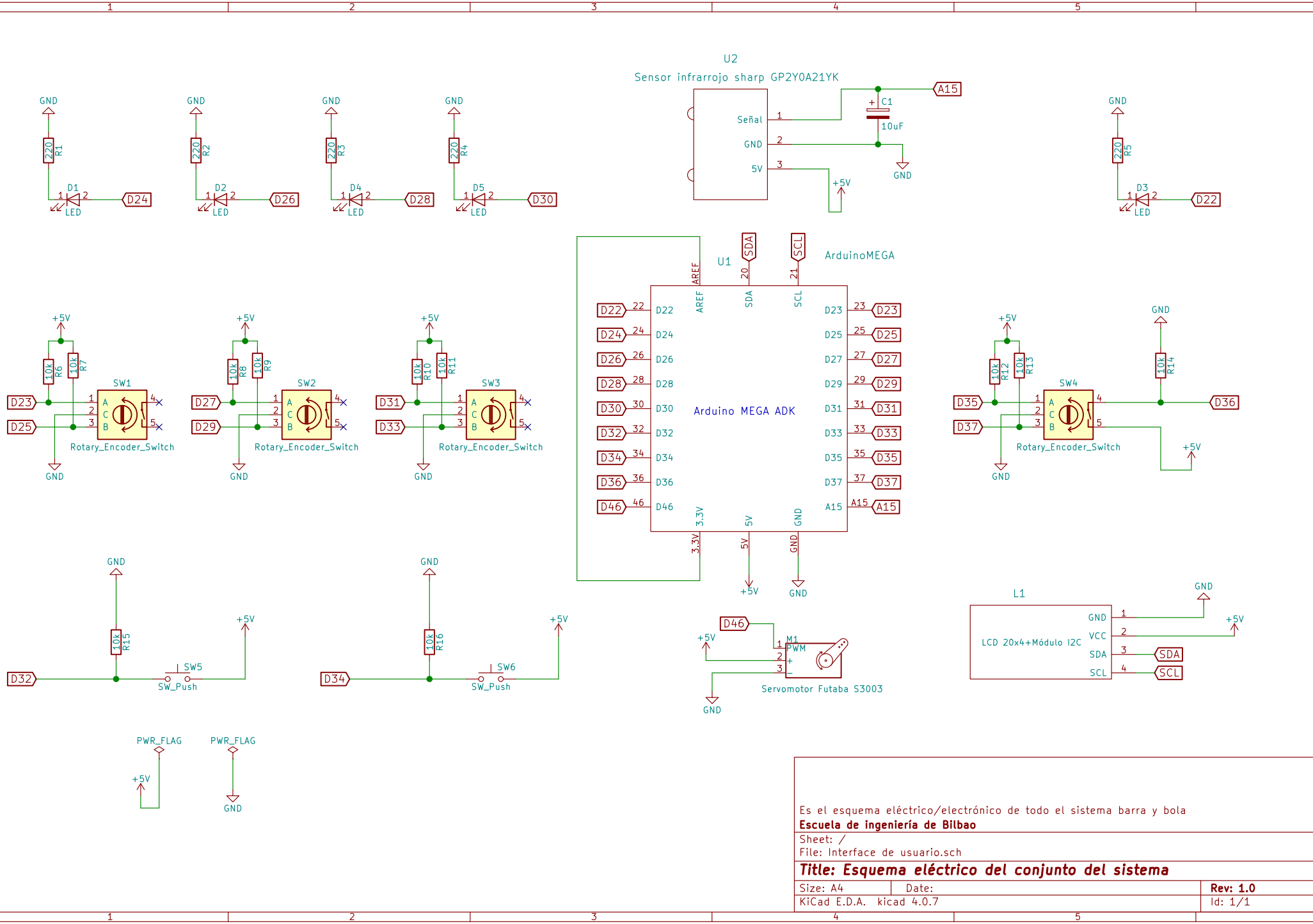
```
     //, esto significa que la barra se va a inclinar hacia la derecha
moviendo la bola hacia la derecha, que es lo que se quiere.
     // El proceso para llegar a establecer la bola en el valor de
consigna asignado va a realizarse de está manera en cada periodo de
muestreo (cada 60000 microsegundos), hasta que el error se reduzca a
cero.

    DC = constrain(DC,500,2300);  // Acota el duty cycle de la señal
PWM que va controlar el ángulo de giro del servo, entre los valores en
los cuales el servo trabaja,
                                  //es decir, lo acota en el rango de
trabajo del servo (rango de movilidad del servo de 0° a 180°).

    OCR5A = DC;  // El registro de comparación de salida del timer 5
es igual al valor del duty cycle calculado,
                 // este valor es el que va hacer que el ángulo de giro
del servo se posicione en la posición necesaria para que el error de
posición desaparezca.

    if(abs(error_pos) <= 10){ // Si el error de posición es inferior o
igual a 10mm
      digitalWrite(led0,HIGH); // Se enciende el LED rojo
    }
    else{
      digitalWrite(led0,LOW); // Si no, no se enciende
    }
    sei();
}
```

ANEXO II. ESQUEMAS Y DISEÑO DE PCB



Es el esquema eléctrico/electrónico de todo el sistema barra y bola

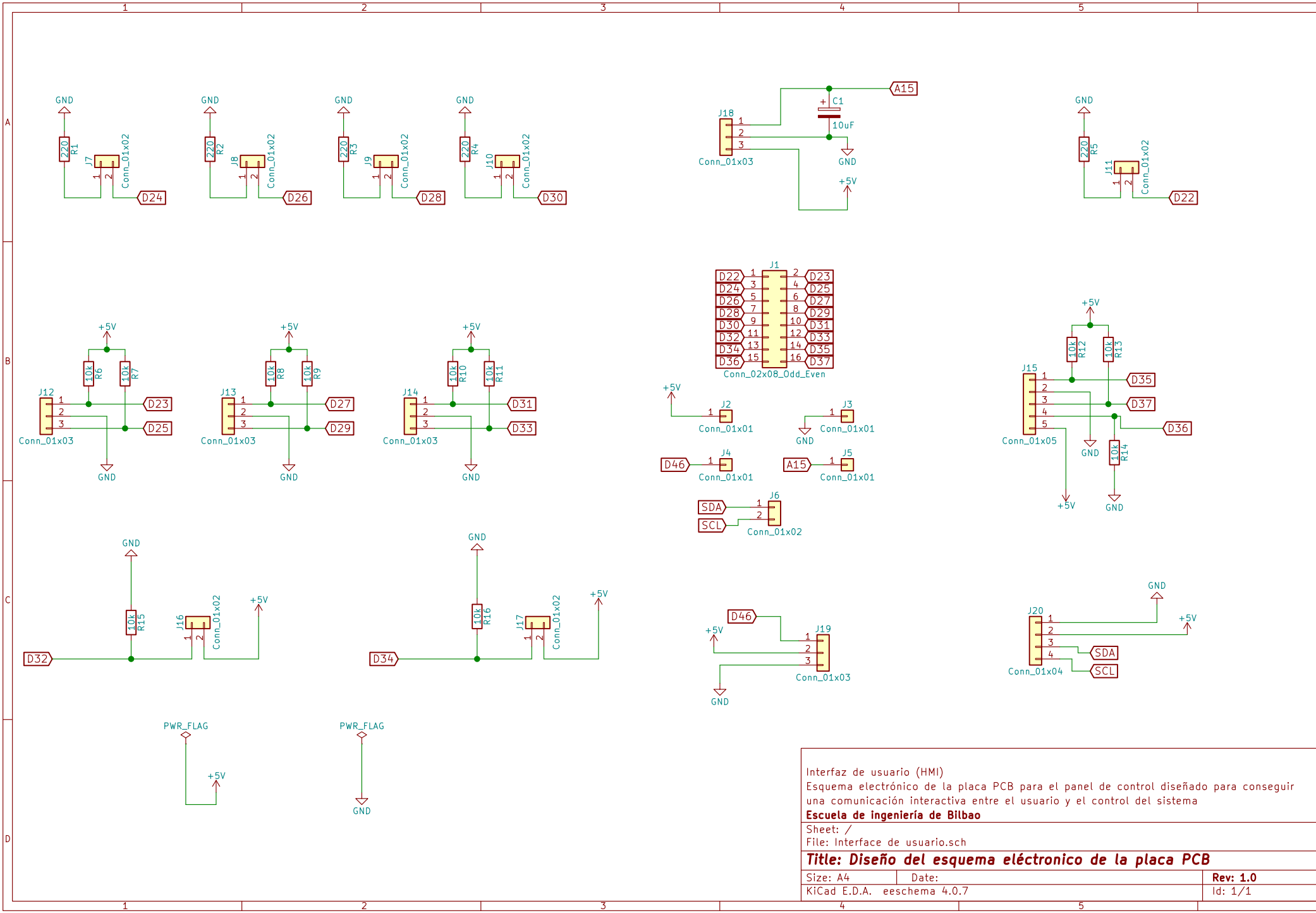
Escuela de ingeniería de Bilbao

Sheet: /

File: Interface de usuario.sch

Title: Esquema eléctrico del conjunto del sistema

Size: A4	Date:	**Rev: 1.0**
KiCad E.D.A. kicad 4.0.7		Id: 1/1



Interfaz de usuario (HMI)
Esquema electrónico de la placa PCB para el panel de control diseñado para conseguir una comunicación interactiva entre el usuario y el control del sistema
Escuela de ingeniería de Bilbao

Sheet: /
File: Interface de usuario.sch

Title: Diseño del esquema eléctronico de la placa PCB

Size: A4	Date:	**Rev: 1.0**
KiCad E.D.A. eeschema 4.0.7		Id: 1/1

ASIER SEDANO JIMENEZ

ANEXO III. DOCUMENTACIÓN TÉCNICA

Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

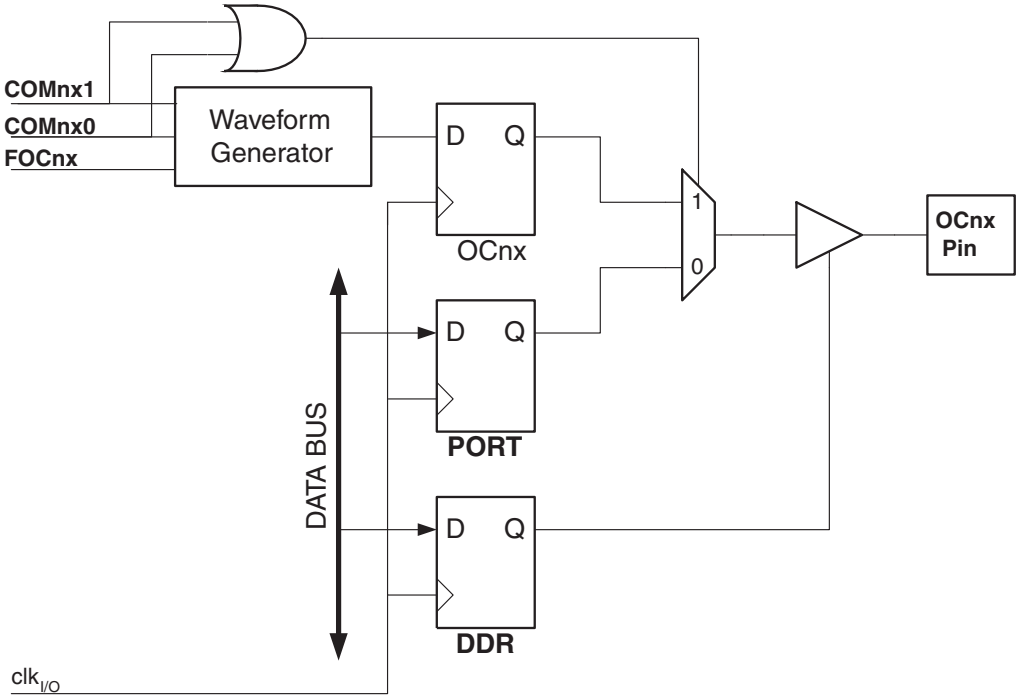
8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels
- JTAG (IEEE® std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green
- Temperature Range:
  - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V: 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
  - ATmega640V/ATmega1280V/ATmega1281V:
    - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega2560V/ATmega2561V:
    - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega640/ATmega1280/ATmega1281:
    - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
  - ATmega2560/ATmega2561:
    - 0 - 16MHz @ 4.5V - 5.5V

Figure 17-5. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OCnx) from the Waveform Generator if either of the COMnx1:0 bits are set. However, the OCnx pin direction (input or output) is still controlled by the *Data Direction Register* (DDR) for the port pin. The Data Direction Register bit for the OCnx pin (DDR_OCnx) must be set as output before the OCnx value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to Table 17-3 on page 155, Table 17-4 on page 155 and Table 17-5 on page 155 for details.

The design of the Output Compare pin logic allows initialization of the OCnx state before the output is enabled. Note that some COMnx1:0 bit settings are reserved for certain modes of operation. See “Register Description” on page 154.

The COMnx1:0 bits have no effect on the Input Capture unit.

17.8.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COMnx1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COMnx1:0 = 0 tells the Waveform Generator that no action on the OCnx Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 17-3 on page 155. For fast PWM mode refer to Table 17-4 on page 155, and for phase correct and phase and frequency correct PWM refer to Table 17-5 on page 155.

A change of the COMnx1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOCnx strobe bits.

17.9 Modes of Operation

The mode of operation, that is, the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the *Waveform Generation mode* (WGMn3:0) and *Compare Output mode* (COMnx1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COMnx1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COMnx1:0 bits control whether the output should be set, cleared or toggle at a compare match. See “Compare Match Output Unit” on page 143.

Table 17-2. Waveform Generation Mode Bit Description[1]

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 152.

17.9.1 Normal Mode

The simplest mode of operation is the *Normal mode* (WGMn3:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the *Timer/Counter Overflow Flag* (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOVn Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

17.9.2 Clear Timer on Compare Match (CTC) Mode

In *Clear Timer on Compare* or CTC mode (WGMn3:0 = 4 or 12), the OCRnA or ICRn Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNTn) matches either the OCRnA (WGMn3:0 = 4) or the ICRn (WGMn3:0 = 12). The OCRnA or ICRn define the top value for the

17.11 Register Description

17.11.1 TCCR1A – Timer/Counter 1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.2 TCCR3A – Timer/Counter 3 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x90)	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.3 TCCR4A – Timer/Counter 4 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xA0)	COM4A1	COM4A0	COM4B1	COM4B0	COM4C1	COM4C0	WGM41	WGM40	TCCR4A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.4 TCCR5A – Timer/Counter 5 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50	TCCR5A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

The COMnA1:0, COMnB1:0, and COMnC1:0 control the output compare pins (OCnA, OCnB, and OCnC respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bits are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC1:0 bits are written to one, the OCnC output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OCnA, OCnB or OCnC pin must be set in order to enable the output driver.

When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMnx1:0 bits is dependent of the WGMn3:0 bits setting. Table 17-3 on page 155 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to a normal or a CTC mode (non-PWM).

- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

Combined with the WGMn3:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 17-2 on page 145. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. For more information on the different modes, see “Modes of Operation” on page 144.

Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Table 17-4 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode.

Table 17-4. Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. See “Fast PWM Mode” on page 146. for more details.

Table 17-5 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct and frequency correct PWM mode.

Table 17-5. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 =9 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting Set OCnA/OCnB/OCnC on compare match when downcounting
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting Clear OCnA/OCnB/OCnC on compare match when downcounting

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1//COMnC1 is set. See “Phase Correct PWM Mode” on page 148. for more details.

17.11.5 TCCR1B – Timer/Counter 1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.6 TCCR3B – Timer/Counter 3 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x91)	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.7 TCCR4B – Timer/Counter 4 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xA1)	ICNC4	ICES4	–	WGM43	WGM42	CS42	CS41	CS40	TCCR4B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.8 TCCR5B – Timer/Counter 5 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50	TCCR5B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNCn: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The input capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICESn: Input Capture Edge Select**

This bit selects which edge on the Input Capture Pin (ICPn) that is used to trigger a capture event. When the ICESn bit is written to zero, a falling (negative) edge is used as trigger, and when the ICESn bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register (ICRn). The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCRnB is written.

- **Bit 4:3 – WGMn3:2: Waveform Generation Mode**

See TCCRnA Register description.

- **Bit 2:0 – CSn2:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 17-10 and Figure 17-11 on page 152.

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

If external pin modes are used for the Timer/Countern, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

17.11.9 TCCR1C – Timer/Counter 1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

17.11.10 TCCR3C – Timer/Counter 3 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x92)	FOC3A	FOC3B	FOC3C	–	–	–	–	–	TCCR3C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

17.11.11 TCCR4C – Timer/Counter 4 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0xA2)	FOC4A	FOC4B	FOC4C	–	–	–	–	–	TCCR4C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

17.11.12 TCCR5C – Timer/Counter 5 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x122)	FOC5A	FOC5B	FOC3C	–	–	–	–	–	TCCR5C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOCnA: Force Output Compare for Channel A**
- **Bit 6 – FOCnB: Force Output Compare for Channel B**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**

The FOCnA/FOCnB/FOCnC bits are only active when the WGMn3:0 bits specifies a non-PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB/OCnC output is changed according to its COMnx1:0 bits setting. Note that the

17.11.30 ICR3H and ICR3L – Input Capture Register 3

Bit	7	6	5	4	3	2	1	0	
(0x97)	ICR3[15:8]								ICR3H
(0x96)	ICR3[7:0]								ICR3L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.31 ICR4H and ICR4L – Input Capture Register 4

Bit	7	6	5	4	3	2	1	0	
(0xA7)	ICR4[15:8]								ICR4H
(0xA6)	ICR4[7:0]								ICR4L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.32 ICR5H and ICR5L – Input Capture Register 5

Bit	7	6	5	4	3	2	1	0	
(0x127)	ICR5[15:8]								ICR5H
(0x126)	ICR5[7:0]								ICR5L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Input Capture is updated with the counter (TCNTn) value each time an event occurs on the ICPn pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 135.

17.11.33 TIMSK1 – Timer/Counter 1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.34 TIMSK3 – Timer/Counter 3 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x71)	–	–	ICIE3	–	OCIE3C	OCIE3B	OCIE3A	TOIE3	TIMSK3
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.35 TIMSK4 – Timer/Counter 4 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x72)	–	–	ICIE4	–	OCIE4C	OCIE4B	OCIE4A	TOIE4	TIMSK4
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.36 TIMSK5 – Timer/Counter 5 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x73)	–	–	ICIE5	–	OCIE5C	OCIE5B	OCIE5A	TOIE5	TIMSK5
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICIEn: Timer/Countern, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Input Capture interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 101) is executed when the ICFn Flag, located in TIFRn, is set.

- **Bit 3 – OCIEnC: Timer/Countern, Output Compare C Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare C Match interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 101) is executed when the OCFnC Flag, located in TIFRn, is set.

- **Bit 2 – OCIEnB: Timer/Countern, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 101) is executed when the OCFnB Flag, located in TIFRn, is set.

- **Bit 1 – OCIEnA: Timer/Countern, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 101) is executed when the OCFnA Flag, located in TIFRn, is set.

- **Bit 0 – TOIEn: Timer/Countern, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Overflow interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 101) is executed when the TOVn Flag, located in TIFRn, is set.

17.11.37 TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.38 TIFR3 – Timer/Counter3 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x18 (0x38)	–	–	ICF3	–	OCF3C	OCF3B	OCF3A	TOV3	TIFR3
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17.11.39 TIFR4 – Timer/Counter4 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x19 (0x39)	–	–	ICF4	–	OCF4C	OCF4B	OCF4A	TOV4	TIFR4
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

GP2Y0A21YK/ GP2Y0D21YK

General Purpose Type Distance Measuring Sensors

■ Features

1. Less influence on the color of reflective objects, reflectivity
2. Line-up of distance output/distance judgement type
   Distance output type (analog voltage) : **GP2Y0A21YK**
   Detecting distance : 10 to 80cm
   Distance judgement type : **GP2Y0D21YK**
   Judgement distance : 24cm
   (Adjustable within the range of 10 to 80cm [Optionally available])
3. External control circuit is unnecessary
4. Low cost

■ Applications

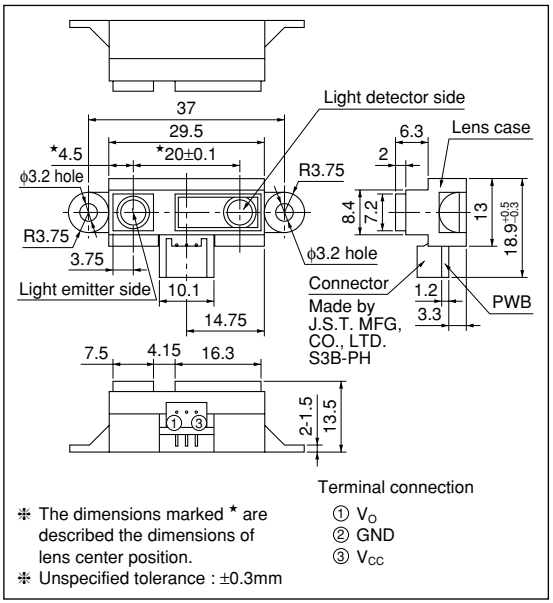
1. TVs
2. Personal computers
3. Cars
4. Copiers

■ Absolute Maximum Ratings (T_a=25°C, V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	−0.3 to +7	V
Output terminal voltage	V_O	−0.3 to V_{CC} +0.3	V
Operating temperature	T_{opr}	−10 to +60	°C
Storage temperature	T_{stg}	−40 to +70	°C

■ Outline Dimensions

(Unit : mm)



Terminal connection

① V_O
② GND
③ V_{CC}

❊ The dimensions marked * are described the dimensions of lens center position.
❊ Unspecified tolerance : ±0.3mm

Notice In the absence of confirmation by device specification sheets, SHARP takes no responsibility for any defects that may occur in equipment using any SHARP devices shown in catalogs, data books, etc. Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device.
Internet Internet address for Electronic Components Group http://sharp-world.com/ecg/

■ Recommended Operating Conditions

Parameter	Symbol	Rating	Unit
Operating supply voltage	V_{CC}	4.5 to +5.5	V

■ Electro-optical Characteristics

(T_a=25°C, V_{CC}=5V)

Parameter		Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Distance measuring range		ΔL	*1 *3	10	–	80	cm
Output terminal voltage	GP2Y0A21YK	V_O	L=80cm *1	0.25	0.4	0.55	V
	GP2Y0D21YK	V_{OH}	Output voltage at High *1	V_{CC} −0.3	–	–	V
		V_{OL}	Output voltage at Low *1	–	–	0.6	V
Difference of output voltage	GP2Y0A21YK	ΔV_O	Output change at L=80cm to 10cm *1	1.65	1.9	2.15	V
Distance characteristics of output	GP2Y0D21YK	V_O	*1 *4 *2	21	24	27	cm
Average Dissipation current		I_{CC}	L=80cm *1	–	30	40	mA

Note) L : Distance to reflective object

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

*2 We ship the device after the following adjustment : Output switching distance L=24cm±3cm must be measured by the sensor

*3 Distance measuring range of the optical sensor system

*4 Output switching has a hysteresis width. The distance specified by Vo should be the one with which the output L switches to the output H

Fig.1 Internal Block Diagram

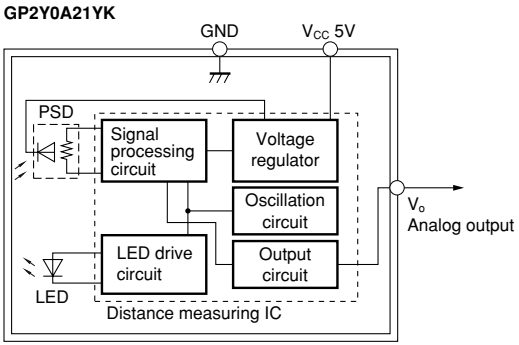


Fig.2 Internal Block Diagram

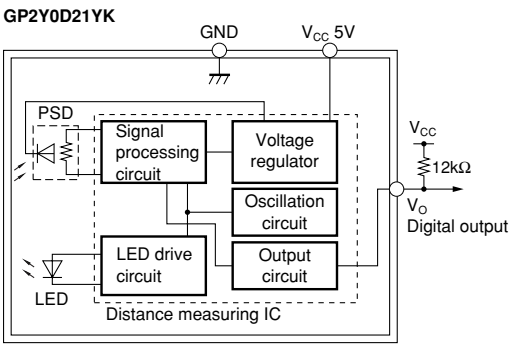


Fig.3 Timing Chart

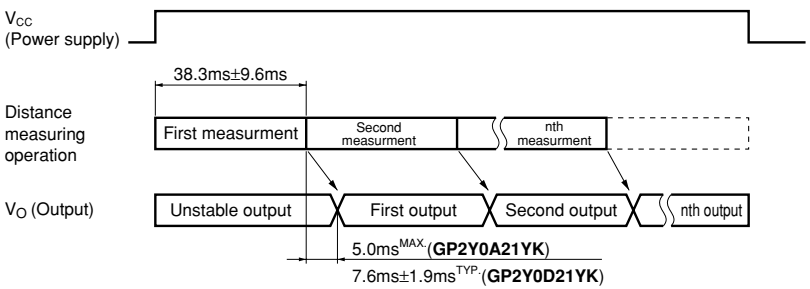


Fig.4 Distance Characteristics

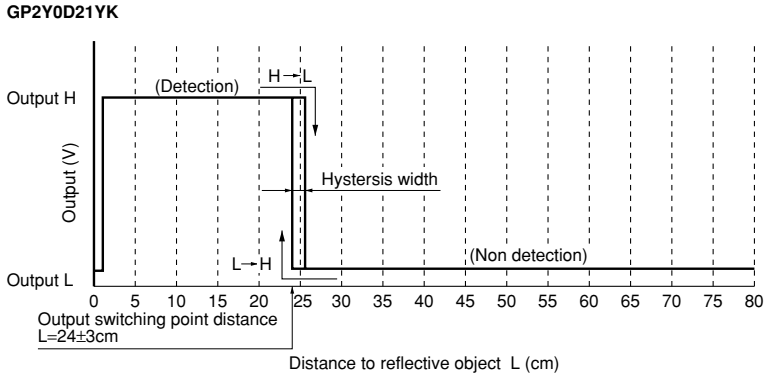
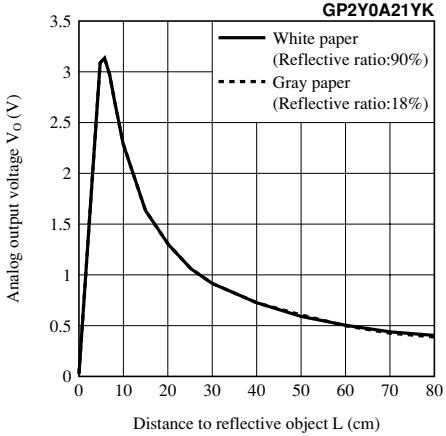
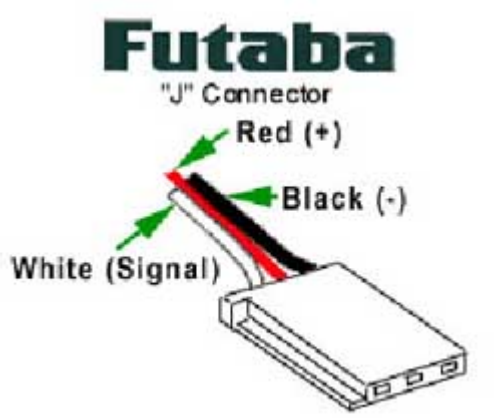


Fig.5 Analog Output Voltage vs. Distance to Reflective Object



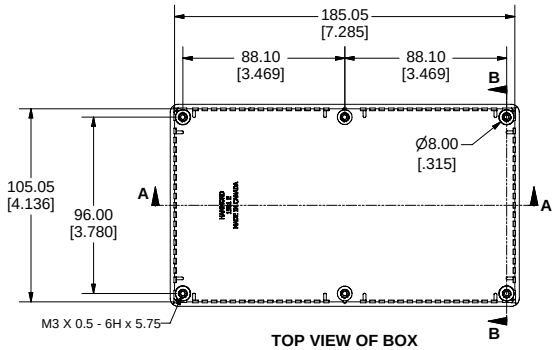
S3003 FUTABA SERVO



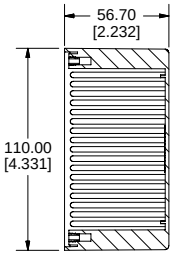
...S3003 FUTABA SERVO...

Detailed Specifications

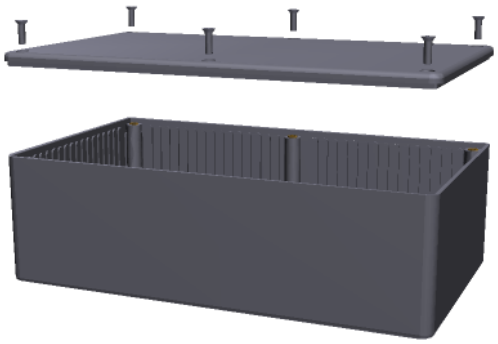
Control System:	+Pulse Width Control 1520usec Neutral	Current Drain (4.8V):	7.2mA/idle
Required Pulse:	3-5 Volt Peak to Peak Square Wave	Current Drain (6.0V):	8mA/idle
Operating Voltage:	4.8-6.0 Volts	Direction:	Counter Clockwise/Pulse Traveling 1520-1900usec
Operating Temperature Range:	-20 to +60 Degree C	Motor Type:	3 Pole Ferrite
Operating Speed (4.8V):	0.23sec/60 degrees at no load	Potentiometer Drive:	Indirect Drive
Operating Speed (6.0V):	0.19sec/60 degrees at no load	Bearing Type:	Plastic Bearing
Stall Torque (4.8V):	44 oz/in. (3.2kg.cm)	Gear Type:	All Nylon Gears
Stall Torque (6.0V):	56.8 oz/in. (4.1kg.cm)	Connector Wire Length:	12"
Operating Angle:	45 Deg. one side pulse traveling 400usec	Dimensions:	1.6" x 0.8"x 1.4" (41 x 20 x 36mm)
360 Modifiable:	Yes	Weight:	1.3oz. (37.2g)

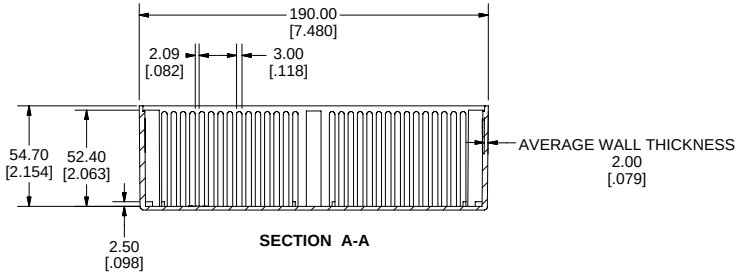


TOP VIEW OF BOX

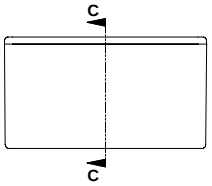


SECTION B-B

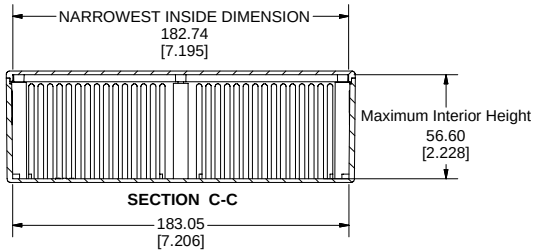




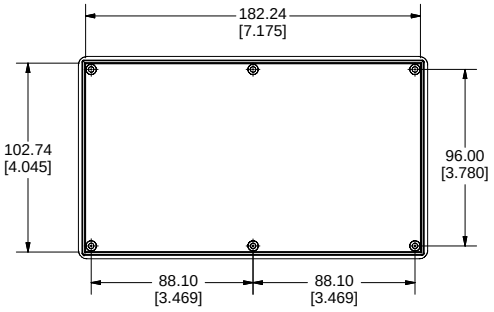
SECTION A-A



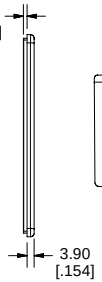
END VIEW OF ASSEMBLY



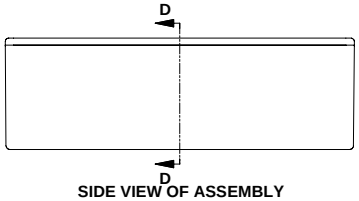
SECTION C-C



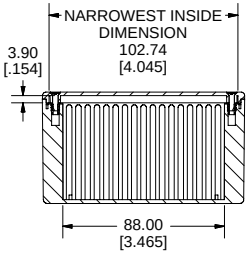
BOTTOM OF LID



SIDE OF LID



SIDE VIEW OF ASSEMBLY



SECTION D-D

Note:
Enclosures can be Factory Modified (Milling, Drilling, Printing etc.)
Contact Factory mjm@hammondmfg.com for quotes
Solid models of this enclosure available in STEP or IGES Format.

PART NUMBERS		
1591E Enclosures		
Material	**Colour**	**Part Number**
FR ABS UL94 - V0	Beige	1591EBG
	Black	1591EBK
	Blue	1591EBU
	Grey	1591EGY
GP ABS UL94 - HB	Black	1591ESBK
	Grey	1591ESGY
Translucent Polycarbonate	Ice Blue	1591ETBU
	IR Red	1591ETRD
	Crystal Clear	1591ETCL
Accessories		
	Description	**Part Number**
Universal Use	PC Board Card Adapters	See Accessories Section
	Clear Lids - clear, flame retardant polycarbonate - Lid ONLY	1591EC
FR ABS	Replacement Screws, Flat Head Phillips Nickel Plated Pkg of 100	1591MS100 1591MS100BK (Black)
GP ABS	Self Tapping Pkg of 100	1591TS100 1591TS100BK (Black)
Recomended Torque on Fasteners (no lubrication)		
	M3 - 0.5 x 10mm Screws	30-40 cN*m
	#4 x 0.5 Screws	30-40 cN*m

HAMMOND MANUFACTURING™

1591E

www.hammondmfg.com

Systronix 20x4 LCD
Brief Technical Data

July 31, 2000

Here is brief data for the Systronix 20x4 character LCD. It is a DataVision part and uses the Samsung KS0066 LCD controller. It's a clone of the Hitachi HD44780. We're not aware of any incompatabilities between the two - at least we have never seen any in all the code and custom applications we have done.

This 20x4 LCD is electrically and mechanically interchangeable with 20x4 LCDs from several other vendors. The only differences we've seen among different 20x4 LCDs are:

1) LED backlight brightness, voltage and current vary widely, as does the quality of the display

2) There is a resistor "Rf" which sets the speed of the LCD interface by controlling the internal oscillator frequency. Several displays we have evaluated have a low resistor value. This makes the display too slow. Looking at the Hitachi data sheet page 56, it appears that perhaps the "incorrect" resistor is really intended for 3V use of the displays.

At 5V the resistor Rf should be 91 Kohms. At 3V it should be 75 Kohms. Using a 3V display at 5V is acceptable from a voltage standpoint (the display can operate on 3-5V) but the oscillator will then be running too slowly. One fix is to always check the busy flag and not use a fixed time delay in your code, then it will work regardless of the LCD speed. The other option is to always allow enough delay for the slower display.

All Systronix 20x4 LCDs have the 91 Kohm resistor and are intended for 5V operation.

Thank you for purchasing Systronix embedded control products and accessories. If you have any other questions please email to support@systronix.com or phone +1-801-534-1017, fax +1-801-534-1019.

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Standard Value			Unit
		Min.	Typ.	Max.	
Supply Voltage for Logic	V_{DD-}	0	—	7.0	V
Supply Voltage for LCD Driver	V_{DD}-V_{EE}	—	—	13.5	V
Input Voltage	V_I	V_{SS}	—	V_{DD}	V
Operature Temp.	Topr	0	—	50	°C
Storage Temp.	Tstg	-20	—	70	°C

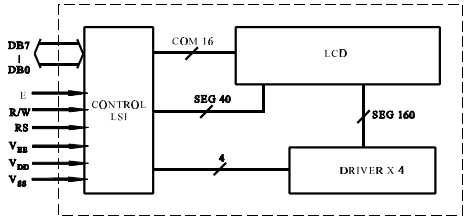
■ ELECTRICAL CHARACTERISTICS (REFLECTIVE TYPE)

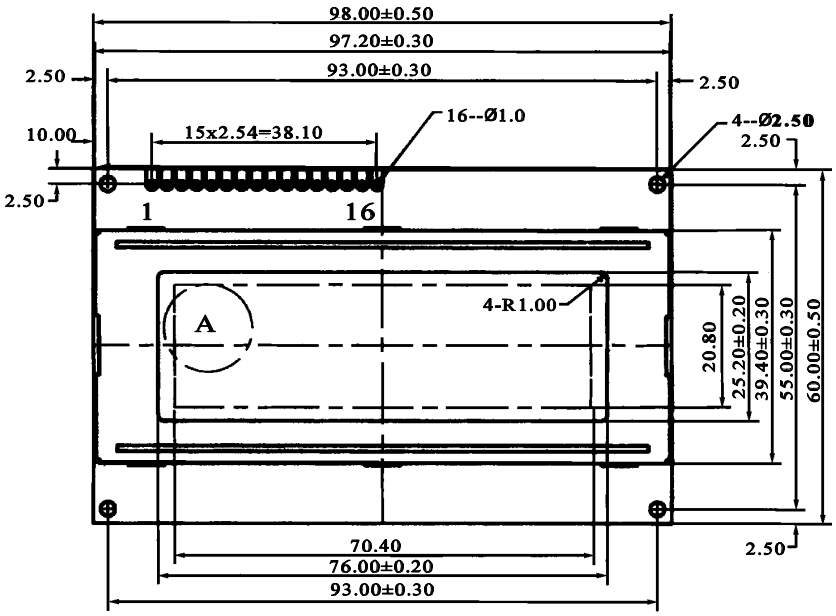
Item	Symbol	Test Condition	Standard Value			Unit
			Min.	Typ.	Max.	
Input "High" Voltage	V_{IH}	—	2.2	—	V_{EE}	V
Input "Low" Voltage	V_{IL}	—	—	—	0.6	V
Output "High" Voltage	V_{OH}	I_{OH}=0.2mA	2.2	—	—	V
Output "Low" Voltage	V_{OL}	I_{OL}=1.2mA	—	—	0.4	V
Supply Current	I_{DD}	V_{DD}=5.0A	—	2.5	4.0	mA

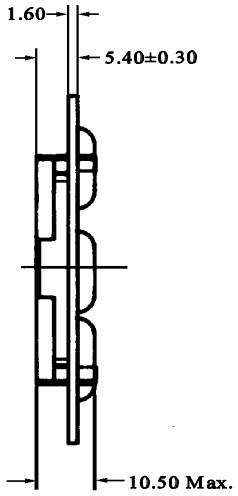
■ PIN FUNCTIONS

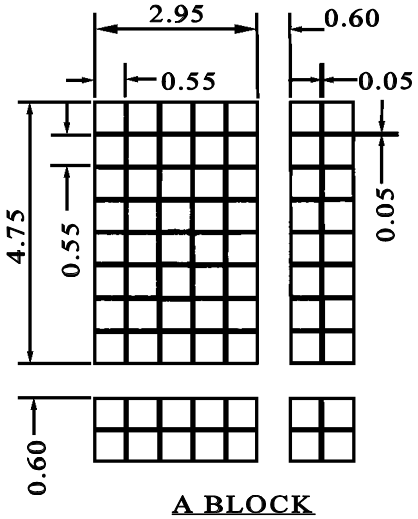
No	Symbol	Function	No	Symbol	Function
1	V_{SS}	GND, 0V	10	DB3	Data Bus
2	V_{DD}	+5V	11	DB4	—
3	V_{EE}	for LCD Drive	12	DB5	—
4	RS	Function Select	13	DB6	—
5	R/W	Read/Write	14	DB7	—
6	E	Enable Signal	15	LEDA	LED Power Supply
7-9	DB0-DB2	Data Bus Line	16	LEDA	

■ BLOCK DIAGRAM





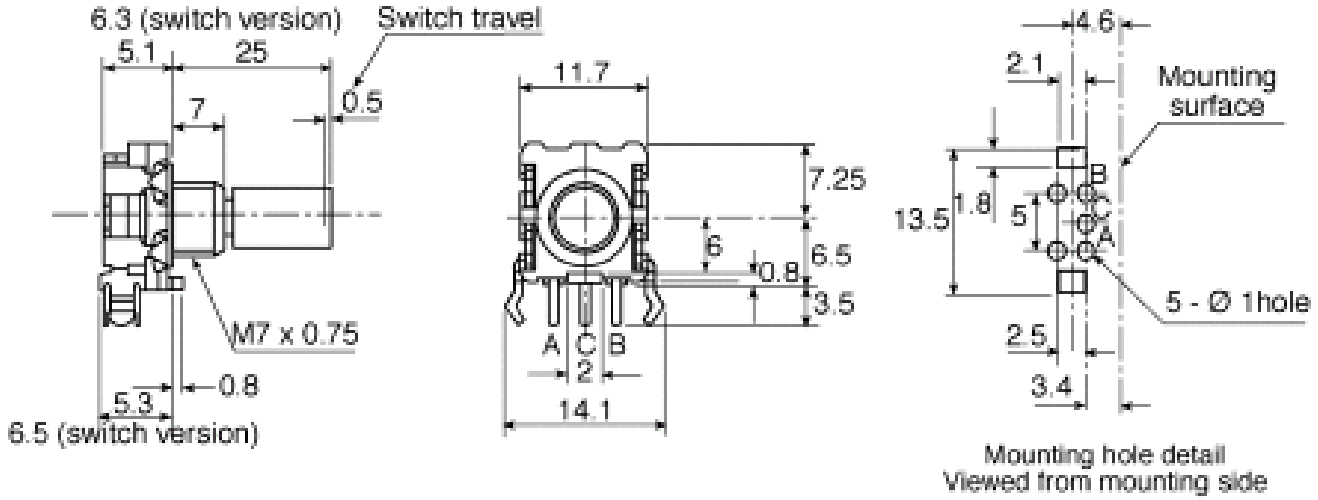




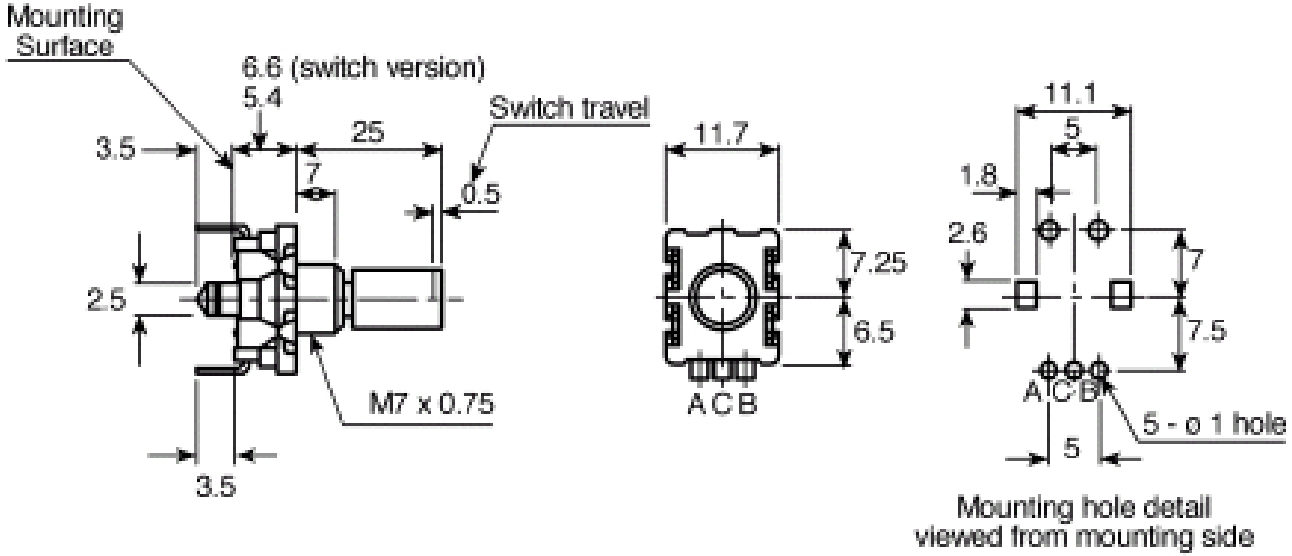
A BLOCK

Horizontal 263-2930, 265-2906 and Vertical 265-2928, 265-1717

Horizontal



Vertical



ENGLISH

Datasheet

SP-NO PCB Off-(On) Push Button Switch

RS Stock number 734-6735

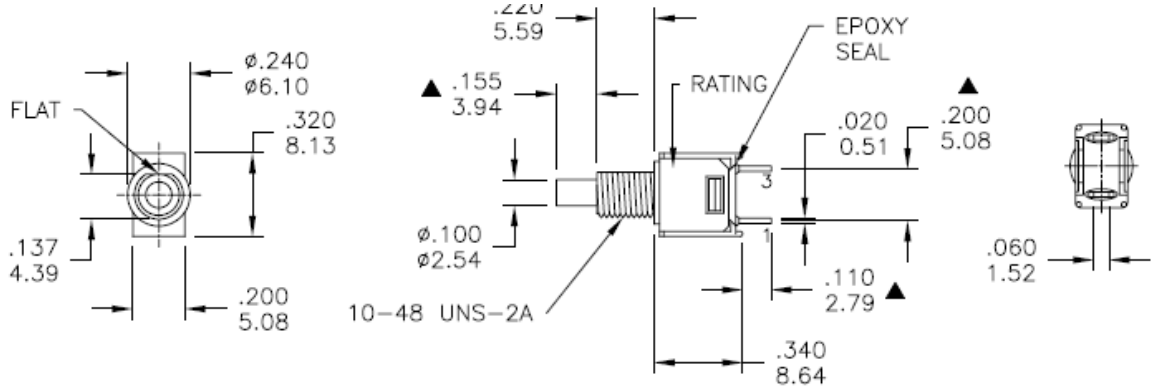
Specifications:

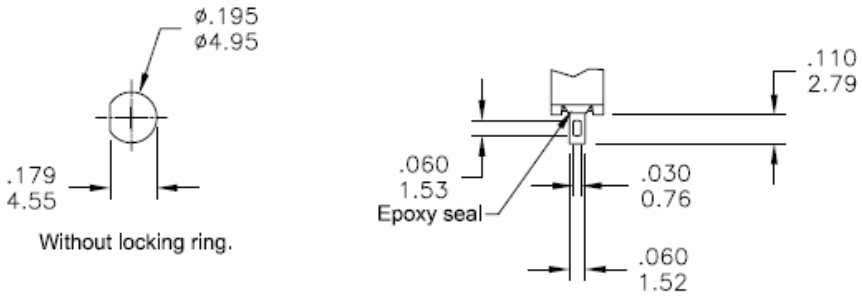
Mechanical Life:	60,000 make-and-break cycles
Contact Resistance:	20mΩ max. initial @ 2-4VDC, 100mA for both silver and gold plated contacts.
Insulation Resistance:	1,000MΩ min.
Dielectric Strength:	1,000 RMS @ sea level
Operating Temperature:	-30°C to 85°C
Case:	Diallyl phthalate (DAP) (UL94V-0)
Plunger:	Thermoplastic polyester-black
Bushing:	Brass, nickel plated
Housing:	Stainless steel
Contact:	Silver or gold plated
Terminal:	Brass, silver plated
Terminal Support:	Brass, electrotin plated
Rating:	R: 0.4VA max. @ 20V max. (AC or DC) Q: 1A @ 250VAC 3A @ 120VAC or 28VDC

RS, Professionally Approved Products, gives you professional quality parts across all products categories. Our range has been testified by engineers as giving comparable quality to that of the leading brands without paying a premium price.

ENGLISH

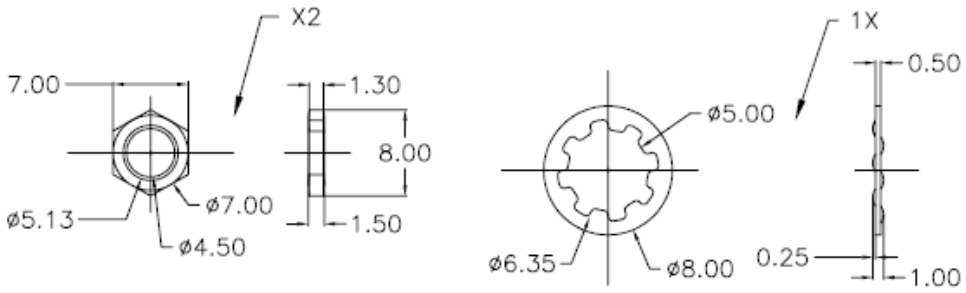
Dimensions – inch
mm





PANEL MOUNTING

SOLDER LUG



Model No.	POS.1	POS.2
8MS9	OFF	MOM
Term. Comm.	OPEN	1-3
SCHEMATIC		

SWITCH FUNCTION

RS, Professionally Approved Products, gives you professional quality parts across all products categories. Our range has been testified by engineers as giving comparable quality to that of the leading brands without paying a premium price.

L-53GD GREEN

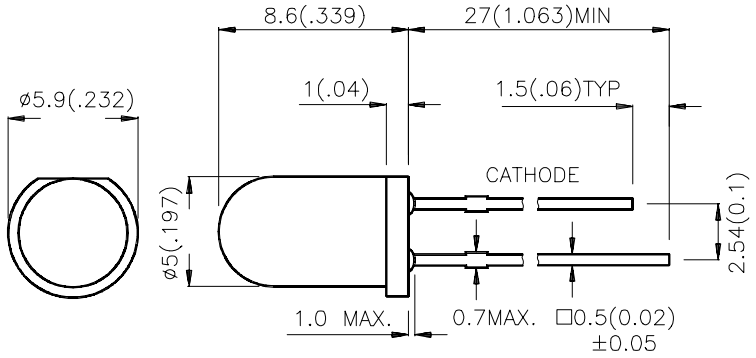
Features

- ●LOW POWER CONSUMPTION.
- ●POPULAR T-1 3/4 DIAMETER PACKAGE.
- ●RELIABLE AND RUGGED.
- ●LONG LIFE - SOLID STATE RELIABILITY.

Description

The Green source color devices are made with Gallium Phosphide Green Light Emitting Diode.

Package Dimensions



Notes:
1. All dimensions are in millimeters (inches).
2. Tolerance is ±0.25(0.01") unless otherwise noted.
3. Lead spacing is measured where the lead emerge package.
4. Specifications are subject to change without notice.