UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Dpto. de INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

INGENIERÍA INDUSTRIAL

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN TEATRO MÓVIL MEDIANTE ARDUINO E IMPRESORA 3D

AUTOR: SERGIO JIMÉNEZ CANALES
TUTOR: CONCEPCIÓN ALICIA MONJE MICHARET

Índice

Índice de figuras

Agradecimientos

En primer lugar quiero a gradecer a mi tutora Concha no sólo por ofrecerme la posibilidad de realizar este proyecto fin de carrera y ayudarme en todo momento con las dudas y obstáculos que he encontrado, sino también por haberme ofrecido la posibilidad de realizar otro proyecto fin de carrera anteriormente, dos trabajos dirigidos y una beca, la cual ha sido mi primera experiencia laboral relacionada con el mundo de la ingeniería. Gran parte de lo que he aprendido en la universidad se lo debo a ella.

A Marina Anaya, con quien ha sido un placer realizar dos proyectos fin de carrera y de la que soy un nuevo admirador de su trabajo.

A los profesores Alberto Valero Gómez y Juan González Gómez por ayudarme en los primeros pasos con la plataforma Arduino y la impresora 3D, y en especial a los técnicos Fernando San Deogracias Cecilia y Ángela Nombela Piqueras, por ayudarme a la realización de la placa y el layout de la caja.

Por último quería agradecer a mi familia, novia y amigos por apoyarme en estos 6 años y por hacerme ver que con paciencia y esfuerzo todo se consigue.

Capítulo

1. Introducción

1.1. Motivación

Una obra de teatro es una forma literaria normalmente constituida de diálogos entre personajes y que cuenta una historia con un cierto orden. Las primeras obras teatrales tal y como se conocen en la era moderna surgieron de la Grecia Antigua, gracias a grandes escritores como Esquilo, Sófocles y Eurípides.

Para poder escenificar la obra, los actores deben representar un guión teatral. Dicha escenificación también se debe apoyar en otros elementos que ayuden al espectador a posicionarse en el contexto, estos pueden ser los propios decorados, los efectos luminosos o ciertas características sonoras.

En esta línea, la ciencia es una buena aliada de los artistas. Gracias a la investigación, nuevos colores han sido creados para pintar o nuevos materiales se han obtenido para esculpir, lo cual ha proporcionado una mayor compresión del arte. Por lo tanto, desde los alquimistas hasta la nanotecnología actual, ciencia y arte han ido siempre de la mano, tal y como se relata en el libro “El artista en el laboratorio” escrito por el periodista y químico Xavier Durán.

Consiguientemente, teniendo como punto de referencia la fusión entre ciencia y arte, la motivación de este proyecto fin de carrera ha sido crear un pequeño teatro en el cual tanto los personajes como ciertos elementos del decorado estén automatizados y por lo tanto la representación sea capaz de realizarse sin la ayuda de una persona.

La elaboración de teatro móvil se ha realizado siguiendo la nueva filosofía DIY (Do It Yourself), que es una tendencia que anima a las personas a crear, modificar o reparar aquello se quiera sin la ayuda de expertos, en otras palabras, a conseguir lo que se quiere con medios propios, usando tecnología de bajo coste, accesible y abierta a todos.

Dentro de esta corriente DIY, la empresa Makerbot juega un papel importante en este proyecto fin de carrera, ya que con su impresora 3D permite a los usuarios crear e imprimir sus propios diseños tridimensionales de forma rápida y económica. Gracias a esta impresora se creará el prototipado necesario para la elaboración de la plataforma que albergará la obra teatral.

Así mismo, la plataforma Arduino ofrece la posibilidad de articular todos los movimientos y efectos luminosos que lleve el teatro gracias a su microprocesador de bajo coste y a su software libre.

1.2. Objetivos

El objetivo de este proyecto fin de carrera es la elaboración de un teatro móvil y automatizado que permita representar una pequeña obra teatral de aproximadamente un minuto de duración.

Para conseguir este objetivo general se deben realizar una serie de subobjetivos que se detallan a continuación:

- **Diseño de la plataforma:** el diseño de la plataforma tiene por función el dimensionamiento de la caja y de los mecanismos que son necesarios para el correcto funcionamiento de la misma.
- **Prototipado:** el prototipado de la plataforma se realiza mayormente gracias a la impresora 3D, la cual proporcionará aquellas piezas que son necesarias para soportar los personajes, motores y el resto de elementos que permiten el movimiento de las partes móviles de la plataforma.
- **Control:** el control tiene por objetivo gobernar y regular las señales analógicas o digitales que entran o salen del microprocesador Arduino de tal manera que la representación de la obra teatral se realice de manera precisa y de acuerdo con el guión planificado.

La idea conceptual de este proyecto es el encuentro de dos personajes que, partiendo de la lejanía, se unen en una caricia y en un latir de un corazón que crece con la cercanía. Todo ello acompañado del movimiento de un pájaro que revoloteará durante dicho encuentro.

Por lo tanto, a modo de resumen, lo que se busca conseguir es una plataforma que albergue un microprocesador mediante el cual se consiga el accionamiento de una serie de motores que a su vez permitan el movimiento de los personajes gracias a una serie de piezas tridimensionales diseñadas e impresas en una impresora 3D.

Además de los componentes nombrados anteriormente, se van a necesitar otra serie de elementos para completar la plataforma. Esto son interruptores, convertidores de tensión, una paca de circuito impreso, un led de alta luminancia y otros dispositivos auxiliares como botones, pulsadores o cables para la conexión de los distintos elementos electromecánicos.

Cabe destacar que el diseño de los personajes y demás elementos artísticos del teatro ha sido realizado por la artista Marina Anaya [1], inspiradora de la idea original y cuya colaboración ha sido intensa y fundamental para la consecución de este proyecto.

En la figura 1.1 se muestra una imagen de la plataforma final.

Figura 1.1: Plataforma

1.3. Estructura del documento

El capítulo 1 comienza con una breve introducción al proyecto y nos sitúa en el entorno en el que se va a desarrollar. Además, se expone los objetivos principales y la estructura del mismo.

El capítulo 2 introduce las herramientas que se van a usar para la realización de dicho proyecto, las cuales se dividen en dos categorías: la impresora 3D y el Arduino. Además, incluye pequeños manuales tanto para el uso de la impresora como para la programación del Arduino y de las piezas tridimensionales que se quieran crear.

En el capítulo 3 se describe el diseño hardware de la plataforma, el cual incluye esquema de la plataforma, la explicación de cómo se ha conseguido el movimiento lineal de los personajes a partir del movimiento circular de los servomotores y la creación de la placa de circuito impreso.

En el capítulo 4 se explica el diseño software de la plataforma, es decir, la programación del microprocesador así como la programación de las piezas tridimensionales que posteriormente se imprimirán con la impresora.

En el capítulo 5 se trata de describir el funcionamiento de la plataforma explicando cada una de las fases y mostrando un link con el que se puede acceder a un vídeo que muestra el funcionamiento de la misma.

El capítulo 6 muestra las conclusiones a las que se han llegado así como los posibles trabajos futuros que se pueden hacer relacionados con la propia plataforma y sus herramientas, la impresora 3D y la plataforma Arduino.

Capítulo

2. Herramientas para el desarrollo de la plataforma

Las herramientas básicas para la creación de la plataforma son la impresora 3D y el microprocesador Arduino.

Gracias a la impresora se van a poder crear todas las piezas tridimensionales que sean necesarias. Dicho edición incluye todas las fases que son necesarias para la creación de la pieza: el diseño, su compilación y su impresión.

Por otro lado, con el microprocesador Arduino se van a poder controlar todos los dispositivos electrónicos que permitan la representación de la obra teatral.

2.1. Impresora 3D de MarketBot Industries

La mayoría de las piezas que forman la plataforma han sido creadas mediante una impresora 3D. Dicha impresora 3D, también conocida como Replicator, es un producto de la empresa MarketBot Industries [2], fundada por Bre Pettis en Enero del 2009 junto con Adam Mayer y Zach "Hoeken", los cuales tenían por objetivo la creación de impresoras de tres dimensiones que se pudieran utilizar en el mismo hogar de una persona a un precio razonable. Como puede verse en la figura 2.1, esta impresora permite la creación de objetos tridimensiones mediante capas de plástico que previamente hayan sido diseñados mediante un programa de software específico.

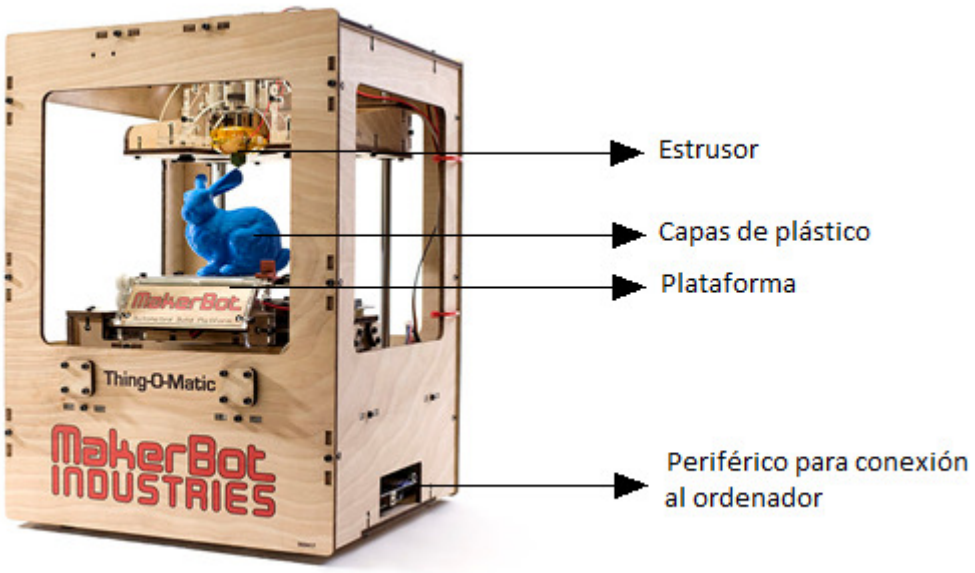


Figura 2.1: Impresora 3D de Makerbot Industries

La impresora Replicator permite en una plataforma de 10 cm^2 la creación de figuras formadas a partir de varias capas creadas con plástico fundido de 1.75 mm de espesor. Esto se realiza siguiendo los siguientes pasos:

1.- Diseño de la figura con el programa OpenScad, propio de MarketBot, o descarga del diseño de la página web de Thingiverse [3]. Cabe destacar que pueden usarse otros softwares para el diseño de la figura siempre y cuando permitan exportar el esquema como formato stl.

2.-Especificación de las características del diseño mediante el programa ReplicatorG.

3.- Impresión de la figura usando la impresora Replicator.

Como se ha indicado anteriormente, es importante destacar la existencia de la comunidad Thingiverse, cuya página web [3] permite a todos aquellos interesados compartir los diseños realizados con la agrupación para que todas aquellas personas que quieran puedan descargarse dichos diseños para imprimirlos o para utilizarlos como parte de diseños posteriores. Esta segunda opción ha sido utilizada en este proyecto de fin carrera, ya que se ha partido de la cadena del móvil Orugator, diseñado en esta misma universidad, para conseguir un movimiento rectilíneo a partir de un movimiento circular proporcionado por un motor.

2.1.1. OpenScad

La generación de las piezas que posteriormente se imprimen con el software ReplicatorG se realiza con el programa OpenScad. Este es un programa CAD (Computer Aided Design) y de uso libre que permite la creación de objetos a través de tres comandos: esferas, paralepípedos y conos. Estos comandos permiten, a través de ciertas instrucciones, obtener cualquier figura física que se pueda plantear, además de la compilación de los diseños para que posteriormente puedan ser aceptados por ReplicatorG. Dicho programa OpenScad permite asimismo la extrusión de documentos Autocad de tipo DXF, es decir, permite dar volumen a aquellos diseños en dos dimensiones que hayan sido previamente diseñados por el programa Autocad.

Como puede verse en la figura 2.2, la interfaz gráfica de este programa es muy sencilla, constando de una ventana izquierda para la elaboración del código de la o las piezas y una ventana derecha en la que se puede ver gráficamente la forma tridimensional del objeto que se está construyendo.

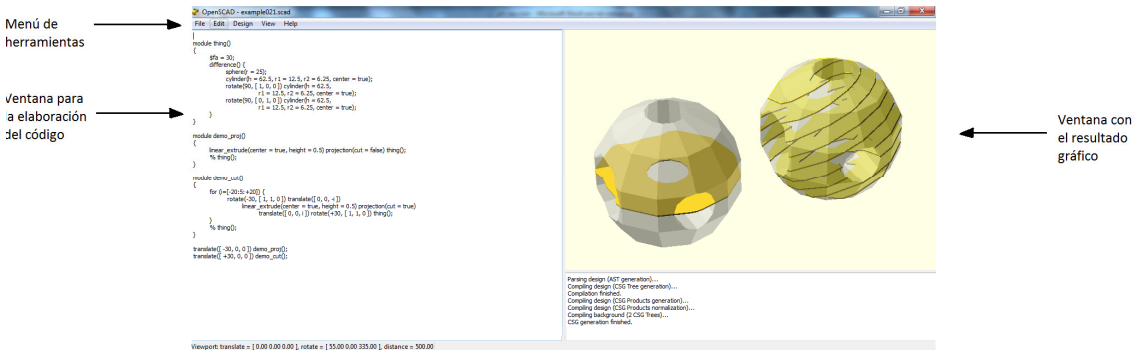


Figura 2.2: Interface de OpenScad

Como se ha comentado anteriormente, cualquier diseño se realiza a partir de las siguientes formas geométricas:

- **cube:** tiene la sintaxis *cube([x,y,z],center,$fn).* Ejemplo: cube([45, 5, 2], center = true, $fn=20). Permite definir un cubo en sus ejes *x*, *y, z.* La opción *center* es para indicar si el centro del objeto se encuentra en el centro de coordenadas (*true*) o si es uno de los

extremos del elemento el que se encuentra en el origen de coordenadas (*false*). La opción *$fn* sirve para indicar la resolución de la figura. En cuanto a las unidades, estas son en milímetros.

- **cylinder:** tiene la sintaxis *cylinder(h,r,center).* Ejemplo: cylinder(h = 2, r = 5, center = true). Permite definir un cilindro mediante su altura h y su radio r. La opción *center* y *$fn* tienen la misma funcionalidad que la explicada en la sentencia cube. El comando cylinder permite a su vez definir cilindro con bases inferior y superior de distintos tamaños mediante la sintaxis *cylinder(h,r1,r2,center).* En este caso r1 es el radio de la base del inferior y r2 el radio de la base superior. Cabe destacar que en el caso de que r1 o r2 valgan 0, la figura geométrica que se obtiene es un cono.
- **sphere:** tiene la sintaxis *sphere(r, center)*. Ejemplo: sphere(30, true). Permite definir esferas de un radio r. La opción *center* y *$fn* tienen la misma funcionalidad que la explicada en la sentencia cube.

Estos elementos pueden combinarse entre ellos mediante diversas operaciones para conseguir elementos más complejos. Esto se realiza mediante las funciones:

- **translate:** tiene la sintaxis *translate([x,y,z]).* Ejemplo: translate([-5,-5,2.5]). Permite desplazar los objetos en los ejes *x, y, z* respectivamente.
- **rotate:** tiene la sintaxis *rótate ([x,y,z])*. Ejemplo: rotate([45, 0, 0]). Permite rotar una figura en torno a los ejes *x, y, z* respectivamente según la regla de la mano derecha.
- **color:** tiene la sintaxis *color([RED,GREEN,BLUE])*. Ejemplo: color([255,2,15]). Permite dotar de color al diseño que se está realizando mediante el espacio de colores red, green, blue. Es importante indicar que aunque el diseño se haga de distintos colores la impresora sólo imprimirá en el color que tenga el plástico en ese momento. La función color se utiliza para distinguir mejor en la fase de diseño las diferentes partes que componen un objeto en su totalidad.
- **difference:** tiene la sintaxis *difference()*

  {figura 1;
  figura2;
  …}.

  Ejemplo:

```
difference()
{
        cylinder(r=20,h=10, center=false, $fn =50);
        translate(0,0,2.5)
        cylinder(r=15,h=15, center=false, $fn =50);
}
```

Permite restar a la figura 1 la figura 2. Esta función es muy útil a la hora de realizar huecos o cavidades en las formas geométricas.

- **unión:** tiene la sintaxis *unión()*

  {figura 1;
  figura2;
  …}.

  Ejemplo:

```
union()
 {
        cube(45, true);
        rotate([45, 0, 0]) cube(50, true);
        rotate([0, 45, 0]) cube(50, true);
        rotate([0, 0, 45]) cube(50, true);
}
```

Permite la unión de las figuras que se encuentren dentro de los corchetes.

- **intersection:** tiene la sintaxis *intersction()*
  {figura1;
  figurara2;
  …}

Ejemplo:

```
intersection()
        {
        rotate([0,20,0])
        cube([20,15,2],center=false);
        rotate([0,20,0])
        translate([10,5,0])
        color([1,0,0])
        cylinder(r1=3,r2=1,h=10, center=true, $fn =20);
        }
```

Permite obtener la figura resultante de la intersección de la figura 1 con la figura 2.

- **for:** tiene la sintaxis *for ([incio, incremento,final])*
  {
  Sentencia 1;
  Sentencia 2;
  …}

Ejemplo:

```
for (i=[0:0.1:5])
  {
  translate([5*i,0,0])
  cylinder(r=i,h=50, center=true, $fn =20);
  }
```

Bucle que permite ejecutar un conjunto de sentencias mientras se cumpla la condición indicada entre paréntesis.

Para poder exportar la figura a ReplicatorG una vez terminado el diseño, es necesario convertir el fichero a formato .stl mediante la opción Export as STL, la cual se encuentra en la opción Design del menú de herramientas.

En el capítulo 4 se describirá en detalle el diseño de todas las piezas realizadas para la consecución del proyecto.

2.1.2. ReplicatorG

La impresora Replicator se maneja a través del programa ReplicatorG, cuya interfaz se muestra en la figura 2.3. Este programa es un producto de software libre que permite definir las especificaciones del objeto ya diseñado como son el tamaño, la posición y orientación de la

entidad, la escala, el número de capas o el posible intermallado que constituye el interior de la pieza. Este programa permite también la generación del código *GCode,* el cual es el que almacena las instrucciones necesarias para el movimiento de la plataforma en los ejes X e Y, así como el movimiento vertical del extrusor.

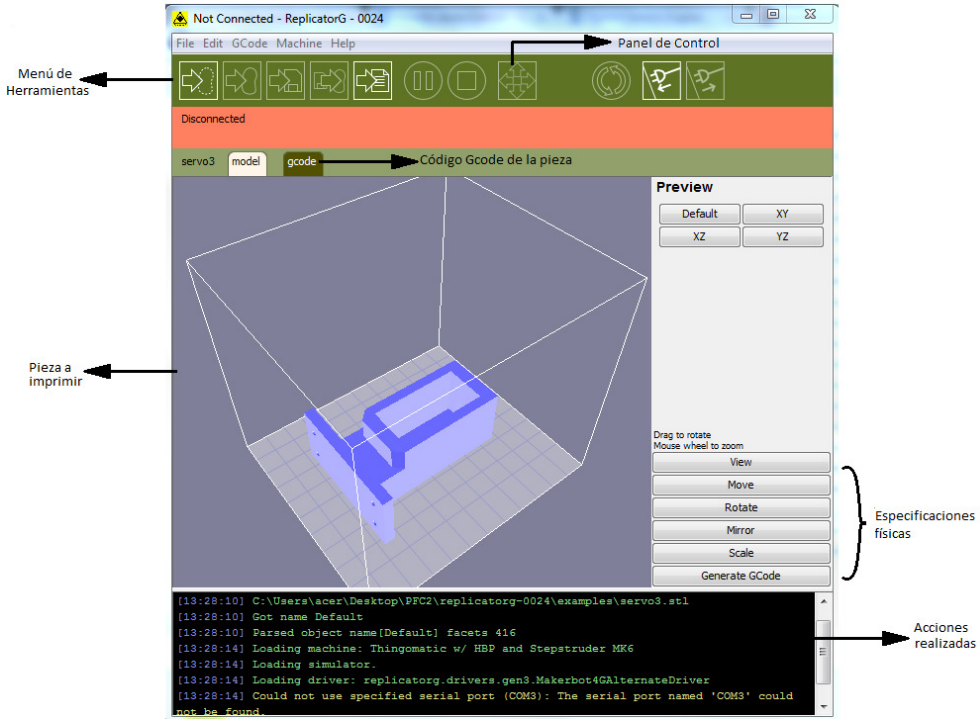


Figura 2.3: Interface de Replicator G

La interfaz de este software se compone de una ventana izquierda en la que se muestra la pieza que va a imprimir, la ventana derecha que permite definir las especificaciones físicas explicadas en el párrafo anterior, y un menú de herramientas que se encuentra en la parte superior y que permite acciones como conectar o desconectar la impresora, imprimir la pieza o parar la impresión. Además en este menú de herramientas se puede acceder a un panel de control en el cual se pueden realizar tres acciones: modificar la posición actual del extrusor en los ejes x, y, z, modificar la velocidad de giro del motor, modificando a su vez la velocidad con la que el plástico es absorbido, y registrar la temperatura de la plataforma y del extrusor. Estas acciones se suelen usar cuando se quiere retirar el rollo de plástico que se esté usando en el momento o cuando se quiere incluir uno nuevo.

Para imprimir una pieza lo primero que se debe hacer es abrir el archivo en formato stl que contiene a la pieza. Una vez abierto se debe fijar la pieza en la plataforma a través de la opción *Move→ Put on Platform*, además se recomienda centrarla mediante la opción *Move→ Center*. Una vez la pieza esté ubicada en la posición deseada se debe seleccionar la opción *Generate Gcode* para generar las acciones de la impresora. Como esta compilación puede tardar varios minutos es recomendable fijar previamente la temperatura de la plataforma a 230˚ y la del extrusor a 220˚ para que cuando se le dé a imprimir no tengan que empezar a calentar ambos componentes desde la temperatura ambiente. Por último, una vez generado el Gcode lo único que queda es pinchar el botón *Imprimir*.

2.2. Arduino

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares [5]. El microprocesador utilizado en este proyecto de fin de carrera es el Arduino Uno, una placa basada en el microcontrolador Atmega 328, cuyas especificaciones pueden verse en el anexo A.1.1.

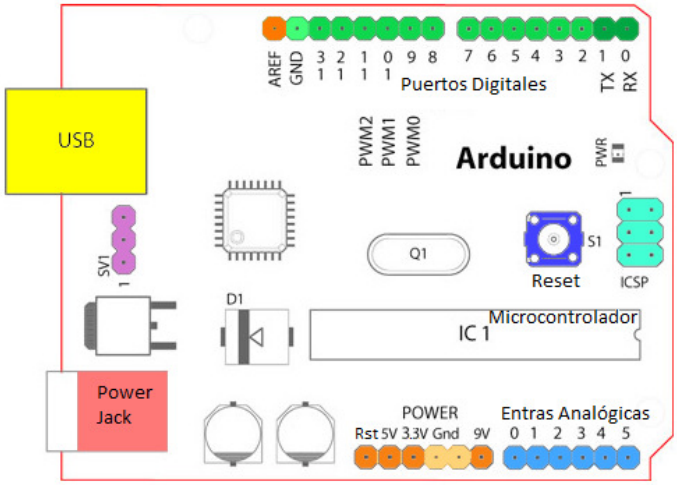


Figura 2.4: Arduino Uno

Como puede verse en la figura 2.4, el Arduino Uno consta de 14 entradas/salidas digitales (de los cuales 6 pueden ser utilizadas como salidas PWM), 6 entradas analógicas, un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, una cabecera de ICSP y un botón de Reset. De entre todos sus puertos, cabe destacar su puerto de salida de 5V, el cual proporciona la tensión necesaria para generar la PWM que permitirá mover los servomotores. Otros puertos que tiene este microprocesador son las 3 tomas de tierra, un puerto de reset aparte del botón que tiene mismo nombre y funcionalidad, un puerto de salida de 3.3V que permite una intensidad máxima de 50mA y el puerto Vin, el cual permite alimentar el propio microprocesador con una tensión que varía entre 7 y 12 V.

En cuanto a dispositivos de memoria, el microcontrolador consta de una memoria Flash de 32KB de la cual 0.5KB es utilizada por el dispositivo de arranque, una memoria SRAM de 2KB y una memoria EEPROM de 1KB de tipo no volátil, por lo que permite guardar información de variables aún con el microcontrolador apagado.

Por último, y aunque se haya mencionado parcialmente con anterioridad, este microprocesador puede ser alimentado desde tres puntos distintos:

- Power Jack, que permite conectarlo a cualquier alimentación monofásica de 220V de un hogar.
- Pin Vin, que hace posible su alimentación mediante pilas.
- Puerto USB, que permite su alimentación cuando el microprocesador está conectado al ordenador.

A través del puerto USB, la tensión es siempre constante e igual a 5V. En cambio, mediante el Power Jack y el pin Vin, la tensión de alimentación puede variar entre unos valores recomendados de 7 y 12 V, y con una tensión limite comprendida entre 6 y 20V.

En este proyecto fin de carrera se utilizarán las dos primeras opciones que permitirán alimentar la plataforma a través de 4 pilas de tipo AAA o a través de una fuente externa conectada a cualquier hogar.

En cuanto al pineado del microprocesador, a continuación se muestra en figura 2.5 el esquema del mismo. Es importante destacar que los pines cuya notación es únicamente numérica, corresponden a las 13 salidas digitales.

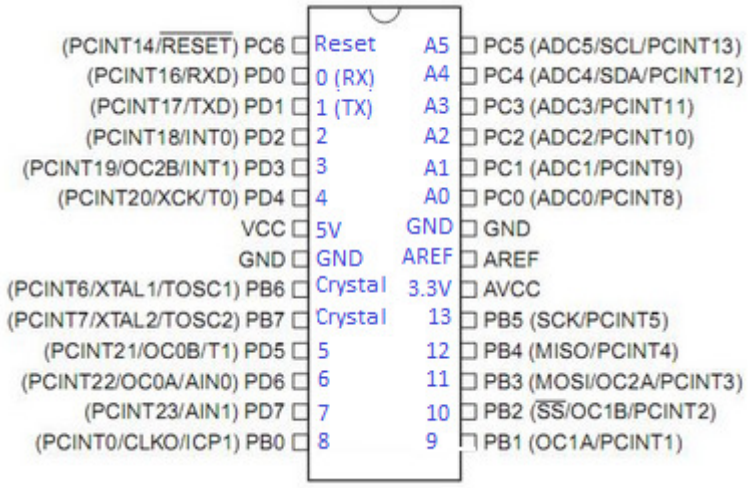


Figura 2.5: Pineado Atmega 328

2.2.1. Programación de Arduino

La programación del Arduino se realiza a través de un software de uso libre que se puede descargar de la propia página web de Arduino y que sólo necesita de la configuración de un puerto USB del ordenador para establecer la comunicación con el microprocesador. Así mismo, el software incluye una serie de programas iniciales, como por ejemplo el parpadeo de un diodo led, que permite al usuario probar el microprocesador y realizar unas primeras pruebas sencillas que le permitan familiarizarse con el Arduino.

Como puede verse en la figura 2.6, el programa es muy sencillo, pues está formado por la ventana principal en la que se escribe el código del programa y una serie de botones que se encuentran en la parte superior izquierda y que permiten al programador opciones tales como la compilación del código, guardado o carga al microprocesador del mismo.

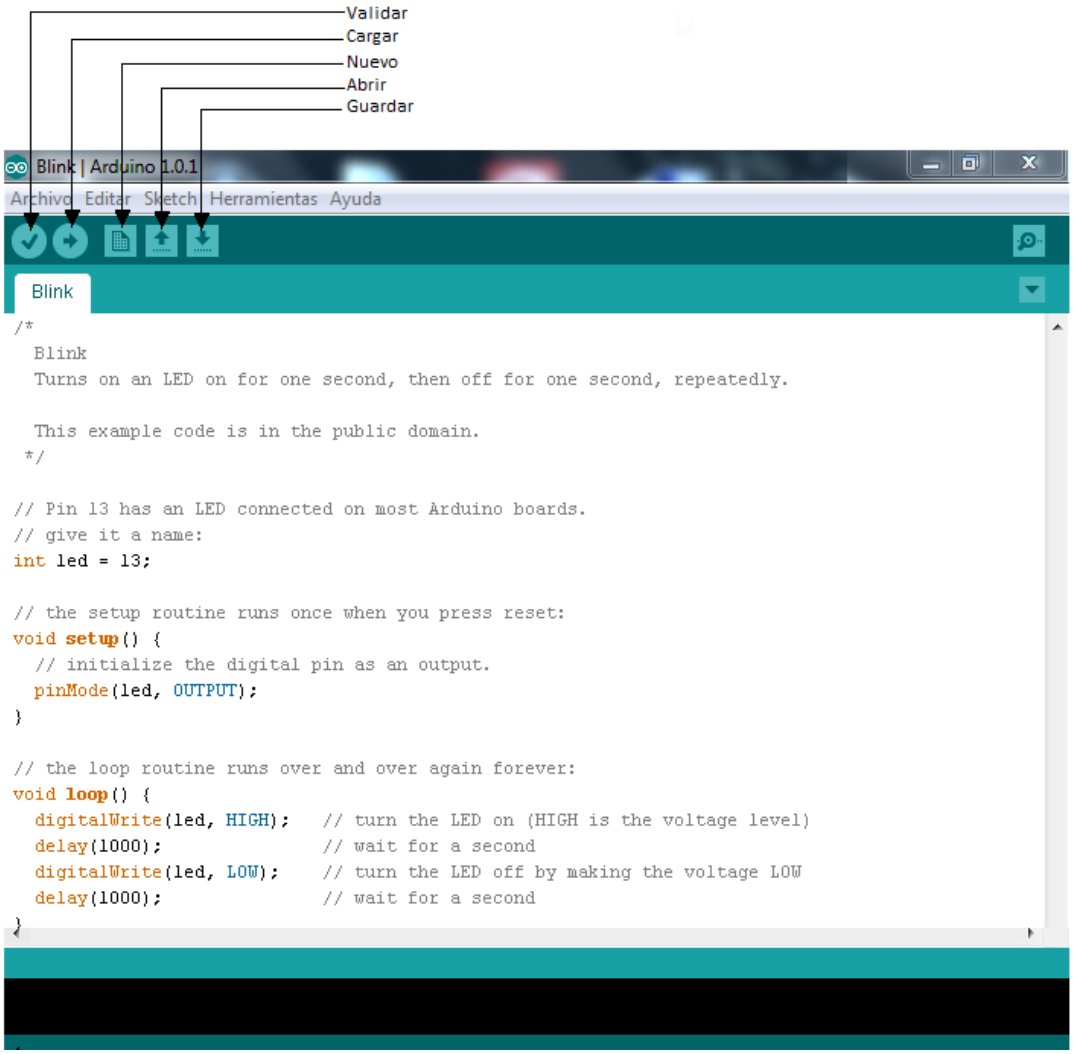


Figura 2.6: Software Arduino

Los programas hechos con Arduino se dividen en tres partes principales:

- Estructura o cuerpo, constituido por una serie de valores y funciones.
- Valores, que pueden ser variables, si éstas cambian a lo largo del proceso, o constantes, si éstas no cambian su valor.
- Funciones, conjunto de instrucciones que realizan una tarea específica.

En cuanto al lenguaje utilizado para programar el micro, es un lenguaje de programación basado C o C++, y por lo tanto incluye los típicos comandos utilizados para la codificación de instrucciones como son *if, if…else, for, switch case, while, int, byte, long, array,…*etc. Además, este lenguaje consta de funciones propias que le permiten interactuar con las entradas o salidas de los puertos como son las funciones *pinMode, digitalWrite, digitalRead, analogRead, analogWrite,…*etc.

Una ventaja que ofrece el Arduino es que, como puede verse en la figura 2.7, en la sección *Referencia de* su propia página web [6] se ofrece gran cantidad de ayuda a la programación del microprocesador, mostrando con claros ejemplos todas las estructuras, variables y funciones que componen la programación.

Figura 2.7: Referencia Arduino

En el capítulo 4 se describirá en detalle la programación de las distintas acciones de control necesarias para la consecución del proyecto.

Capítulo

3. Diseño hardware de la plataforma

El diseño hardware abarca tres aspectos: el dimensionamiento del soporte físico (caja) que albergará todos los elementos de la plataforma, el diseño estudiado para conseguir el movimiento lineal de los personajes y el diseño de la placa de circuito impreso. Cabe destacar que para poder conseguir el movimiento lineal es necesario compenetrar diversos elementos hardware como son los servomotores, las piezas impresas con la impresora 3D y el cableado necesario para la conexión con el microprocesador.

3.1. Soporte físico

La caja tiene unas dimensiones de 60x20x15 cm (ancho x largo x alto) y está formada por dos elementos diferenciados: la propia caja y la tapa.

Dicho soporte está fabricado con madera de DM, la cual se utiliza ampliamente en el ámbito de la maquetación y el modelismo. Esta madera ha sido elegida debido a las ventajas que ofrece de ligereza y facilidad de encolado. La estructura escogida para el diseño de la plataforma es el de una caja de zapatos, ya que esta forma posibilita la apertura de la caja en el caso de que sea necesario cambiar las pilas o revisar algún elemento.

3.1.1. Caja

La caja almacena todos los componentes que forman la plataforma y que no deben de ser vistos (motores, cadenas, microprocesador, PCB, así como el cableado necesario para la interconexión de los componentes). Tal y como se muestra en la figura 3.1, en su lateral derecho se encuentra toda la botonera necesaria para la activación del mecanismo, la cual está formada por el interruptor *Activar* (botón verde), para activar el movimiento de los personajes, el pulsador *Enganchar* (botón amarillo), para facilitar el enganche o desenganche de personajes, y un tercer botón *Alimentación* (botón negro) que permite elegir entre alimentación a través de la red o a través de 4 pilas de tipo AAA. El terminal que puede verse en la esquina inferior derecha sirve para conectar la plataforma con la fuente de alimentación externa.

Figura 3.1: Lateral de la caja

La caja presenta el esquema de componentes de la figura 3.2:

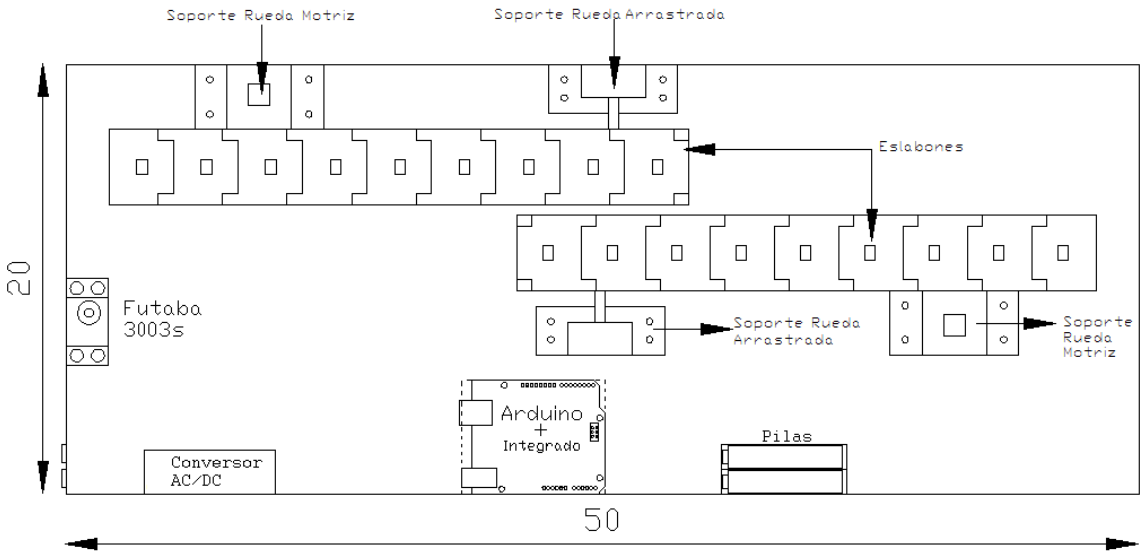


Figura 3.2: Layout plataforma

3.1.2. Tapa

La tapa sirve para cerrar la caja y posee una ranura en la parte superior, como puede verse en la figura 3.3, la cual sirve para poder enganchar los personajes con el sistema de movimiento y permitir el movimiento de ida y vuelta. El sistema elegido para cerrar la plataforma es el de una caja de zapatos, por lo que la tapa tiene forma de la parte superior de una caja de zapatos.

Figura 3.3: Tapa de la plataforma

Dicha tapa tiene unas dimensiones de 62x22x3 (ancho x largo x alto) y la ranura en la parte delantera tiene unas dimensiones de 55x3 (largo x ancho). El largo de dicho cajeado es el resultado del recorrido de cada uno de los personajes por dos más una distancia de seguridad.

$$Longitud\ ranura = 2 \times recorrido\ de\ cada\ muñeco + distancia\ de\ seguridad$$

La tapa posee también un agujero de 3 mm de diámetro para poder sacar el eje del movimiento de giro de un pájaro que forma parte de los elementos móviles del teatro.

3.2. Sistema de movimiento

En la plataforma se van a dar tres movimientos: un movimiento de los personajes principales, un movimiento del brazo del personaje chico y un movimiento de un pájaro que se encuentra en el decorado.

- El movimiento de los personajes principales es un movimiento lineal de vaivén que se consigue gracias a un motor y un sistema para pasar de movimiento circular a lineal.
- El movimiento del brazo del personaje chico es un movimiento circular que se consigue directamente gracias a un motor encajado en el hombro del chico.
- El movimiento del pájaro es un movimiento circular que se consigue encajando el eje de giro del pájaro en el eje del motor. Acompañado a este movimiento se encuentra el latido del corazón del personaje chica, conseguido gracias al parpadeo de un led de color rojizo.

El sistema usado para pasar de movimiento circular a movimiento lineal está basado en dos ruedas y una cadena que engrana en dichas ruedas. Este sistema permite convertir el movimiento circular de los servos en movimiento rectilíneo gracias a la rueda que engancha en el motor y la cadena, a la cual se le transmite el movimiento de giro de la rueda, tal y como se muestra en la figura 3.4.

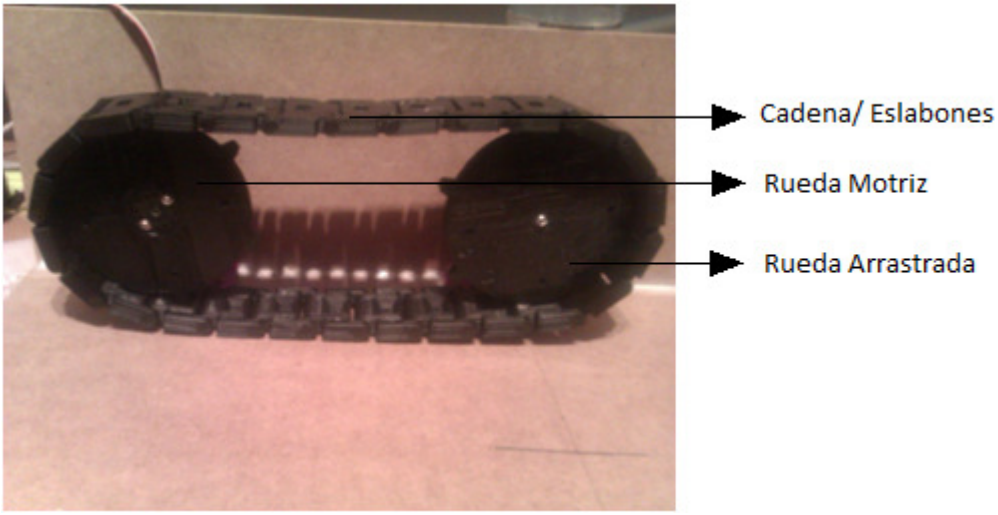


Figura 3.4: Sistema de conversión de movimiento lineal a circular

En un principio se pensaron como posibles opciones un sistema formado por un piñón y una cremallera o un sistema formado por una biela y una manivela. No obstante, el problema que estos dos sistemas presentaban era que tanto para el caso de la biela como el de la cremallera se necesitaba de dos soportes para sostener a ambos elementos a la altura necesaria, es decir, tanto la biela como la cremallera debían tener una elevación tal que permitiera enganchar a los personajes y, a su vez, tener el resto de su sistema dentro de la caja sin ser visto. Esto complicaba de manera considerable el diseño de la plataforma. Por esta razón se decidió que el sistema para pasar de movimiento circular a lineal fuera uno formado por dos ruedas y una cadena, ya que la altura a la que se encuentra la cadena se puede definir fácilmente mediante el radio de la ruedas.

Este sistema es utilizado dos veces en la plataforma, uno por cada personaje principal.

A continuación se detallan los componentes que forman parte del sistema de movimiento.

3.2.1. Motorización

La plataforma está formada por figuras estáticas, como los decorados del fondo, y figuras animadas, como los personajes principales y el pájaro. El movimiento de los personajes animados se ha conseguido mediante una serie de servomotores de la casa Futaba [7] que con su propio movimiento de giro o por su conversión a movimiento lineal han permitido vivificar el movimiento del teatro.

3.2.1.1. Motores para los personajes principales

Los servomotores utilizados para el movimiento de los personajes y el pájaro son el servo Futaba S3003, los cuales son muy utilizados para construir robots articulados o para la construcción de maquetas de helicópteros en el ámbito del modelismo. Como puede verse en la figura 3.5, este servomotor consta de 3 cables, los cuales permiten su alimentación (rojo), toma a tierra (negro) y recepción de la señal de control (blanco).

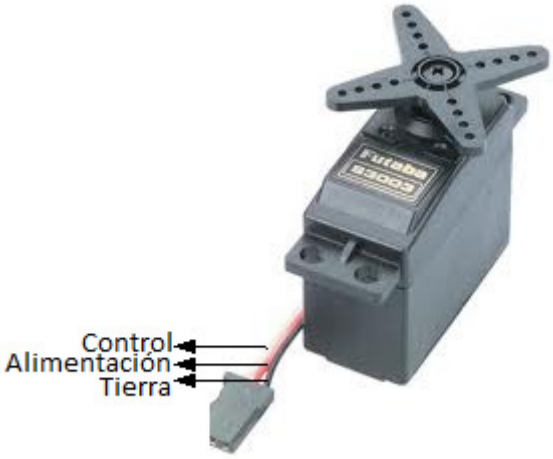


Figura 3.5: Esquema eléctrico del motor Futaba S3003

Estos servos se controlan aplicando una señal PWM (Pulse Width Modulation, Modulación por anchura de pulso) a través de su cable de control y con una frecuencia de 50 Hz (20 ms de periodo), tal y como se puede ver en la figura 3.6. Dichas señales PWM son digitales (pueden valer 0 ó 1) y permiten que usando un único pin de un microcontrolador podamos posicionar el servo. Esto es una gran ventaja, ya que en nuestro caso, permite controlar tres servos distintos con tres tipos de movimientos diferentes usando sólo tres pines.

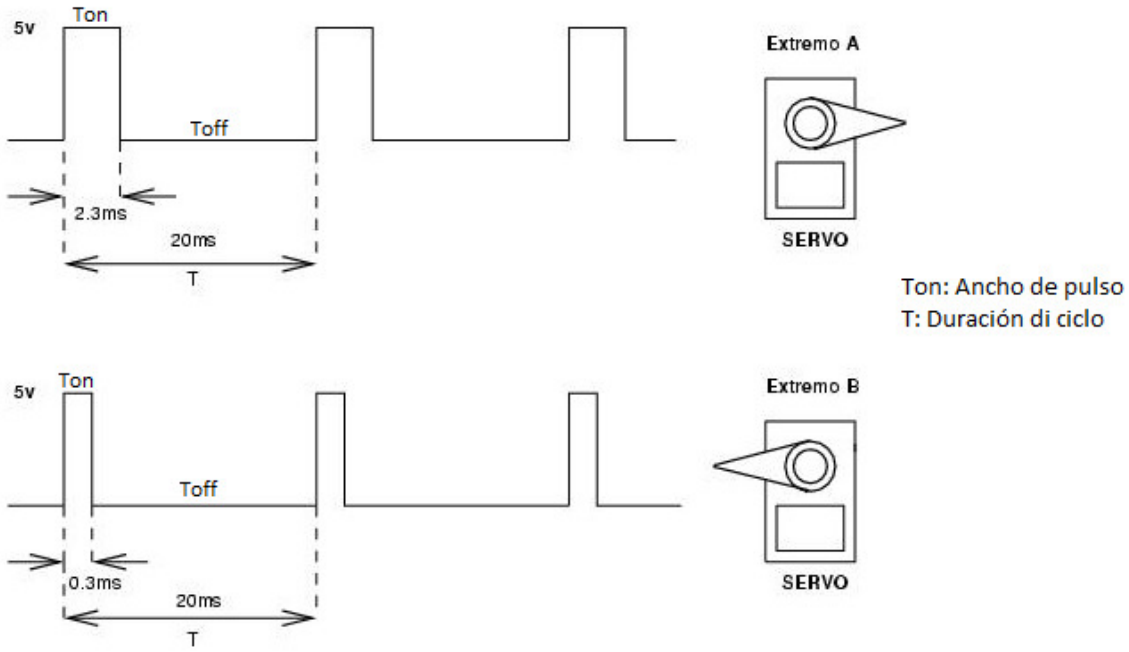


Figura 3.6: Funcionamiento de los servomotores

Para posicionar el servo hay que aplicar una señal periódica, de 50 Hz (20 ms de periodo). La anchura del pulso determina la posición del servo. Si la anchura es de 2.3 ms, el servo se sitúa en un extremo y si la anchura es de 0.3 ms se sitúa en el opuesto. Cualquier otra anchura entre 0.3 y 2.3 sitúa el servo en una posición comprendida entre un extremo y otro. Por ejemplo, si se quiere situar el servo exactamente en el centro, se debe aplicar una anchura de 1.3 ms [8]. En este proyecto de fin de carrera gracias a las funciones *Myservo.write(posición)* y *delay(time)* explicada en la sección 4.1.1 no nos hace falta realizar lo explicado anteriormente y sólo tenemos que indicar la duración del ancho de pulso y por lo tanto la velocidad de giro del motor.

Este motor proporciona un movimiento de giro que será transformado en movimiento lineal gracias a una rueda enganchada en el eje del motor y una cadena que a su vez engrana en dicha rueda.

Los planos y especificaciones de este tipo de servo se encuentran en los anexos A.3.1 y A.3.3, respectivamente.

3.2.1.2. Motor del brazo

Para el caso del movimiento del brazo de uno de personajes, se utiliza un motor de la misma casa pero del modelo S3114, el cual es más ligero y ofrece un par menor pero suficiente para el accionamiento del brazo. Este motor también funciona de la misma manera que el modelo S3003 y a la misma frecuencia, 50 Hz.

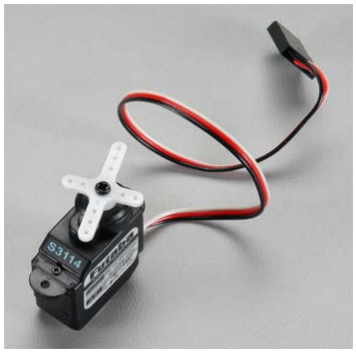


Figura 3.7: Motor Futaba S3114

Como puede verse en la figura 3.7, este motor también consta de tres cables, con el mismo esquema y funcionalidad que los del Futaba S3003 explicado anteriormente.

Este motor simula el movimiento de elevación y descenso de un brazo.

Los planos y especificaciones de este tipo de servo se encuentran en los anexos A.3.2 y A.3.4, respectivamente.

3.2.1.3. Motor del pájaro

El motor empleado para el movimiento del pájaro es un motor Futaba modelo S3003, al igual que en el caso de los motores que accionan el movimiento de los personajes principales. Este motor mantendrá su movimiento de giro, proporcionando al pájaro un movimiento de rotación en torno a sí mismo.

Los planos y especificaciones de este tipo de servo se encuentran en los anexos A.3.1 y A.3.3, respectivamente.

3.2.2. Piezas 3D

El sistema para transformar el movimiento circular de los motores a movimiento lineal se ha basado en el robot Orugator versión 1.1, construido por Olalla Bravo en esta misma universidad [4]. Como puede verse en la comparación de la figura 3.8, para la realización del proyecto fin de carrera se ha tomado del diseño del Orugator las dos ruedas (motriz y arrastrada) y los eslabones.

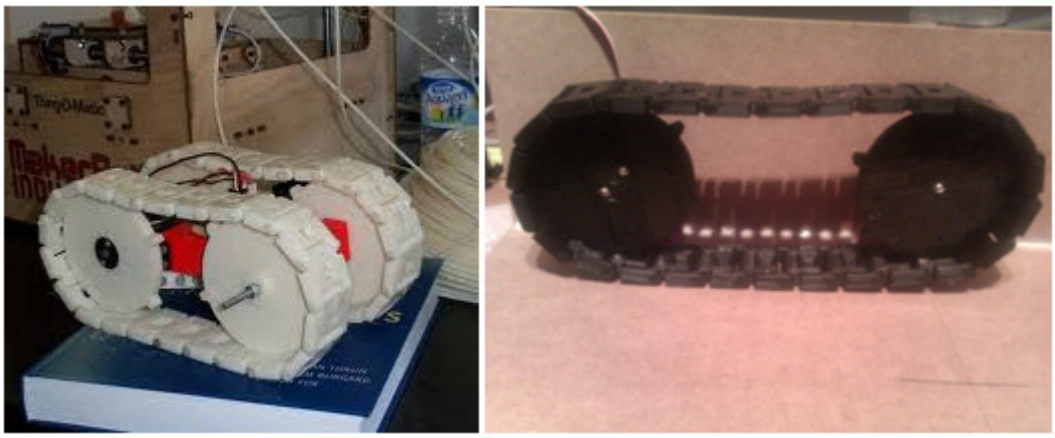


Figura 3.8: Comparación Orugator vs Sistema de movimiento

El esquema del movimiento lineal, representado en la figura 3.9, está formado por la cadena, de 32 eslabones, las ruedas dentadas y los soportes del servomotor y de la rueda arrastrada, todo realizado con la impresora 3D.

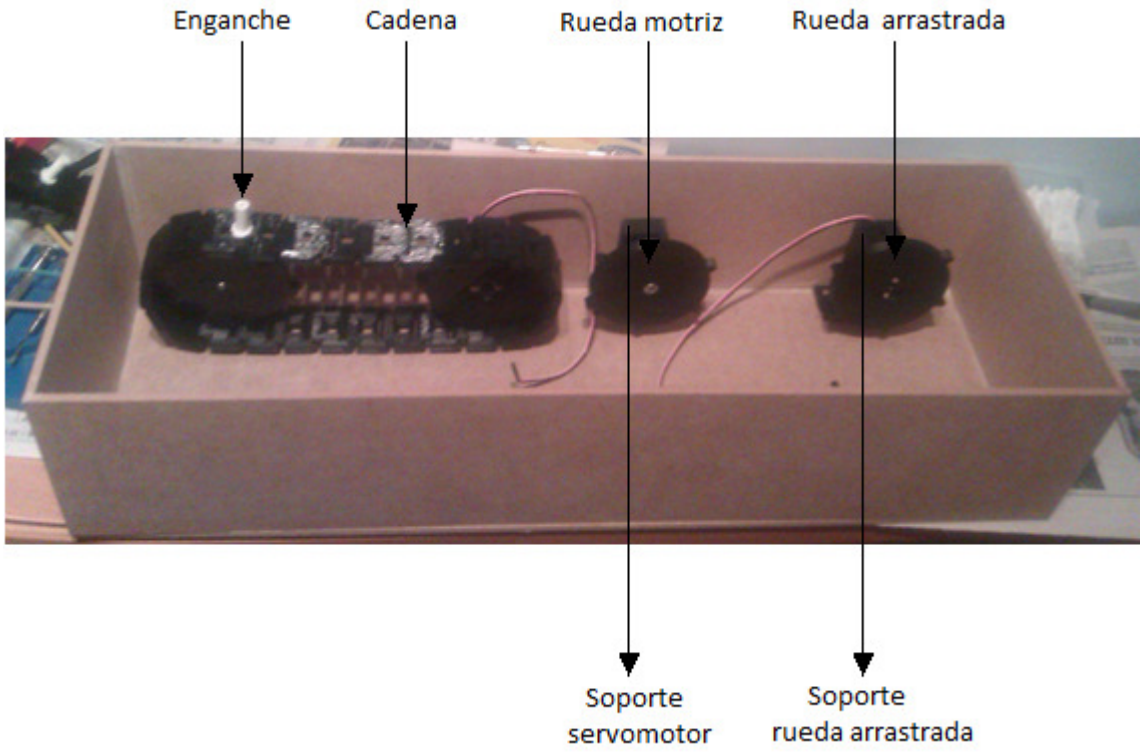


Figura 3.9: Sistema de movimiento

En el capítulo 4 se detalla el diseño CAD de estas piezas 3D y del resto que componen la plataforma.

3.2.2.1. Rueda motriz

Acoplada al eje de cada servomotor se encuentra una rueda motriz encargada de transmitir el movimiento del servo a los eslabones de la cadena por rozamiento, sin que se produzca deslizamiento. Esta rueda ha sido tomada del móvil Orugator versión 1.1 e impresa en Replicator.

La rueda lleva un intermallado que permite reducir el tiempo de fabricación de la misma considerablemente. Además, de esta manera se obtiene una pieza más ligera, consiguiendo que el par en el eje del servo sea menor y requiera así menos potencia.

En la figura 3.10 puede verse el esquema de la rueda motriz. Esta rueda tiene un diámetro exterior de 80 mm y uno interior de 70 mm, consta además de 6 dientes para poder enganchar los eslabones de la cadena a la que va a ir enganchada. Además, a diferencia de la rueda arrastrada, tiene un hueco circular de 21 mm para poder enganchar el servomotor, transmitiendo así el movimiento del mismo.

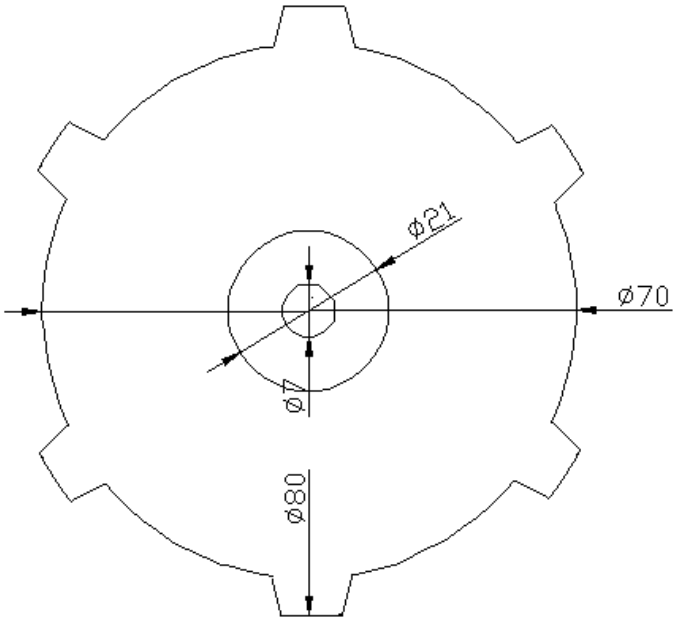


Figura 3.10: Rueda dentada motriz

El tiempo de fabricación de cada rueda está en torno a la hora y media.

3.2.2.2. Rueda arrastrada

La rueda arrastrada se encuentra en el extremo opuesto a la rueda motriz y tiene por objetivo cerrar el movimiento de la cadena y proporcionar la tensión adecuada. Esta rueda ha sido tomada del móvil Orugator versión 1.1 e impresa desde el Replicator. Su esquema se representa en la figura 3.11.

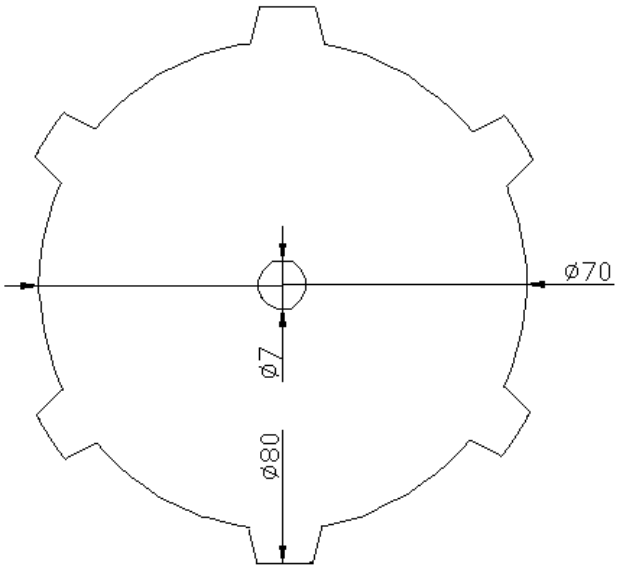


Figura 3.11: Rueda arrastrada

Como se ha comentado anteriormente, esta rueda no tiene el diámetro interior de 21 mm para el enganche del servo, sino uno de 7 mm para enganchar la rueda con el eje del sorporte. En cuanto al resto del diseño y medidas, es igual que la rueda motriz. Cabe destacar

también el intermallado que tiene en su interior por las mismas razones explicadas anteriormente.

El tiempo de fabricación de la rueda arrastrada es aproximadamente de 1 hora y media.

3.2.2.3. Soporte del servomotor

Como puede verse en la figura 3.12, este soporte permite la colocación del servomotor sin ninguna holgura y en posición vertical de manera que se eviten las posibles vibraciones que puedan darse a lo largo del movimiento de este. Dicho soporte ha sido diseñado desde cero e impreso en Replicator, como se detalla en el capítulo 4.

Figura 3.12: Soporte de los servomotores

Las piezas no son macizas sino que se han fabricado con un intermallado para que las piezas sean más ligeras y que el tiempo para fabricarlas sea menor.

Este soporte, a diferencia de las piezas extraidas del Orugator, no es una pieza parametrizable ya que su diseño se ha realizado exclusivamente para este proyecto de fin de carrera.

El tiempo de fabricación de estas piezas es de 2 horas y 30 minutos.

3.2.2.4. Soporte de la rueda arrastrada

Este soporte tiene por objetivo soportar la rueda arrastrada y permitir su libre movimiento para que ésta pueda girar al mismo paso y velocidad que la rueda motriz. Dicho soporte ha sido diseñado e impreso por el Replicator desde cero y puede verse en la figura 3.13.

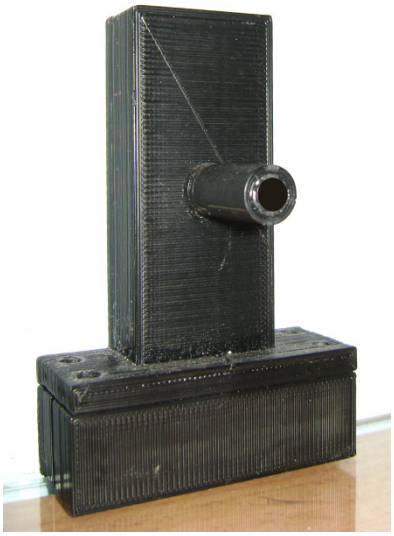


Figura 3.13: Soporte rueda arrastrada

Este soporte posee un mástil horizontal a la altura adecuada para que la rueda motriz y la arrastrada estén al mismo nivel y se respete la horizontalidad. Este soporte también se ha fabricado con intermallado por las mismas razones explicadas anteriormente y no es parametrizable por las mismas razones explicadas en la sección 3.2.1.1.

Su tiempo de fabricación es de 1hora y 40 minutos.

3.2.2.5. Cadena

La cadena está formada por una serie de eslabones que se unen cada uno con el siguiente y que tienen por objetivo soportar el peso de los personajes principales y desplazarlos entre su posición inicial y fina. El diseño de la cadena también ha sido tomado del Orugator versión 1.1 e impreso por la impresora 3D.

Como puede verse en la figura 3.14, los eslabones que forman la cadena tienen forma rectangular de 30 x 35 mm con los huecos disponibles para poder enganchar con los eslabones delantero y trasero sin ningún tipo de problema.

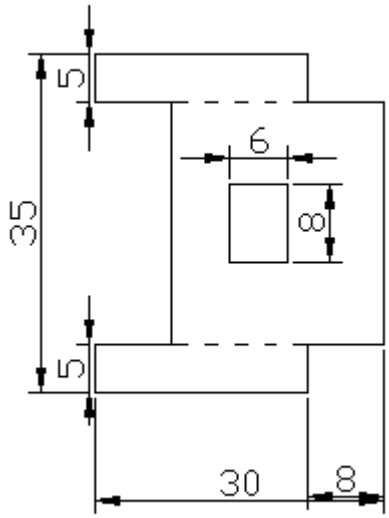


Figura 3.14: Eslabones de la cadena

Los eslabones se han fabricado en grupos de 4, opción disponible en el software ReplicatorG, para reducir los tiempos fijos de calentamiento del estrusor, que puede alcanzar los 20 minutos por ciclo de fabricación. Cabe destacar que la unión entre los eslabones no se realiza con ningún tipo de pieza fabricada sino con el mismo plástico que se utiliza para fabricar las piezas.

Tanto el diseño de las ruedas como los piñones del orugator son parametrizables. Esto quiere decir que las dimensiones de las piezas no son fijas y pueden ajustarse a las medidas requeridas en el diseño.

El tiempo de fabricación por cada 4 piezas es de 45 minutos.

3.2.2.6. Enganche

Los enganches permiten unir los personajes del teatro con el resto del sistema mecánico por medio de unas varillas. Estos enganches han sido diseñados e impresos por el Replicator.

Los enganches de las varillas se han fabricado desde cero e independientemente de los eslabones en los que van montado. Estos enganches están formados por un cilindro en el que se une la varilla y una base o superfice de apoyo ligeramente mayor que la del cilindro para aumentar el área de contacto del enganche con el eslabón, aumentando así su robustez. El hueco de los enganches en el que se encajan las varillas es rectangular y no cilíndrico para evitar los posibles movimientos de rotación de los personajes debido a las vibraciones que se producen en los desplazamientos de los mismos.

Este soporte no es parametrizable por las mismas razones explicadas en apartados anteriores.

La fabricación de este sistema de sujección se ha realizado en un tiempo de 17 minutos.

3.3. PCB

La placa PCB permitirá interconectar todos los elementos electrónicos gracias a su superficie constituida por caminos o pistas de material conductor laminadas sobre un sustrato no conductor. Gracias a esta placa de circuito impreso podemos conseguir que todos los servos, leds, botones y demás componentes electrónicos estén conectados al pin de +5V y GND del microprocesador, así como de conseguir de un sólo puerto de salida las señales de control que mueve a los dos personajes a la vez. La placa utilizada en este proyecto es la que se muestra en la figura 3.15.

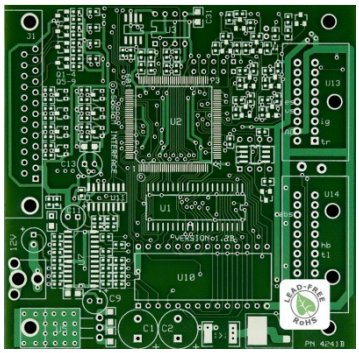


Figura 3.15: PCB

El diseño del circuito ha sido realizado con el programa Pspice mientras que el diseño de la placa se ha elaborado mediante el programa Orcad PCB editor.

3.4. Botonera de la plataforma

La plataforma consta de tres botones colocados en el lateral derecho de la caja (figura 3.1):

- **Botón Activar:** este botón permite activar el teatro, es decir, encender o poner en movimiento los elementos que forman la plataforma. Este botón funciona por niveles, es decir, cuando esté pulsado (nivel alto) la plataforma funcionará, mientras que cuando no esté pulsado (nivel bajo) esta está en estado de reposo.

- **Botón Enganchar:** este botón sirve para posicionar los personajes en un punto en el que puedan ser enganchados o desenganchados sin combar la cadena. Este botón funciona por pulsos: al apretarlo por primera vez (primer flanco de subida), el servo se posiciona en la posición 180, ubicando la cadena en un lugar que permita la manipulación de los personajes, y al apretar de nuevo el botón (segundo flanco de subida), el motor vuelve a su estado original.

- **Botón Alimentación:** este botón sirve para elegir entre los dos tipos de alimentación que puede tener la plataforma, pilas o alimentación de la red. Este botón tiene 3 posibles posiciones, dos sirven para elegir el tipo de alimentación mencionado anteriormente, mientras que el tercero sirve para que el microprocesador no reciba ningún tipo de alimentación, reduciendo así el consumo de energía de la plataforma al estar apagada.

3.5. Otros Elementos auxiliares

Además de los elementos nombrados anteriormente, a continuación se van a explicar una serie de dispositivos que han sido necesarios para el correcto funcionamiento de la plataforma.

Para la correcta alineación de la cadena se han utilizado dos tacos a la altura de la misma, consiguiendo que esta no se combe y que el movimiento sea lo más rectiline posible.

Debido a la gran cantidad de cable y a la dispersa posición que este tiene, se han utilizado una serie de anclajes de goma así como de encapsulados de plástico que permiten mejorar la disposicón de estos sobre la plataforma.

En cuanto a la fijación de los elementos a la plataforma, esta se ha realizado mediante tirafondos para el caso del microprocesador y del servomotor del pájaro, y tornillos para los soportes 3D.

Dentro de la plataforma se encuentra un transformador de tensión que permite conectar el teatro a una fuente de alimentación externa. Además se encuentra el cajetín de las pilas para el caso en el que se quiera alimentar la plataforma con ellas.

3.6. Elementos artísticos

La artista Marina Anaya ha sido la inspiradora de la idea original de este teatro móvil, y ha diseñado y realizado los distintos elementos artísticos que intervienen en la plataforma.

El resultado es una pieza que respira de todo el universo de trabajo de esta artirsta, siempre inspirado por los encuentros, los abrazos y los momentos más amorosos de la vida. Situados en un entorno natural, los personajes principales de la obra se encuentran, activando una ilusión de la que se hace partícipe del entorno. El pájaro vuela movido por esa energía que irradian los personajes.

Todos los elementos artísticos (figura 3.16) están realizados en papel grabado con las técnicas de aguafuerte y acetato, partiendo de pruebas de color y de autor empleadas por la artista para la realización de las tiradas definitivas de algunas de sus obras. De esta manera, a modo de collage con estos retales, se va conformando esta nueva obra.

Figura 3.16: Detalle de los personajes principales del teatro

Capítulo

4. Diseño software de la plataforma

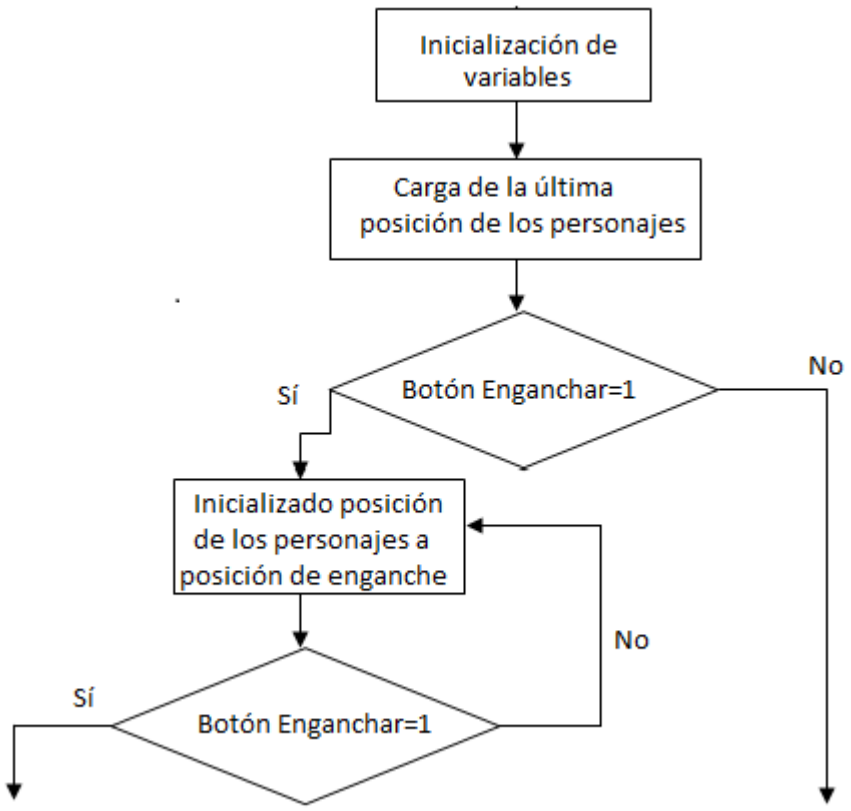
Gran parte de este proyecto está basado en dos tipos de programación. Una es la programación del microprocesador Arduino para conseguir que el teatro cobre vida a través de movimientos y luces, y el otro es la programación de las piezas que posteriormente van a ser impresas con la impresora 3D.

Ambas partes se detallarán a lo largo de éste capítulo.

4.1. Programación de Arduino

Como se ha comentado anteriormente, la programación del Arduino permitirá dotar de movimiento a los personajes, encender y apagar el led, activar o desactivar el conjunto, etc. Para entender todas las funcionalidades que este microprocesador lleva a cabo, lo primero es ver la secuencia de funcionamiento que se realiza en la representación teatral.

En el momento en el que se enciende la plataforma se realiza una inicialización de las variables para cargar la última situación en la que se encontraba esta. En este momento se pueden realizar dos acciones: enganchar o desenganchar los personajes principales, pulsando el botón *Enganchar,* o activar el movimiento de la plataforma pulsando el botón *Activar.* En el caso de pulsar el botón *Activar* comienza el movimiento de los personajes, el cuales se realiza en tres fases: aproximación de los personajes principales, encuentro y alejamiento. Durante la fase de aproximación se realiza un acercamiento de los personajes desde el punto en el que se encuentren. En la fase de encuentro, un personaje realiza una caricia al otro, mientras que a este le late cada vez más fuerte el corazón. Además, puede verse el vuelo de un pájaro durante esta fase. Por último, en la fase de alejamiento, se realiza un distanciamiento de los personajes, empezando de nuevo la fase de aproximación.



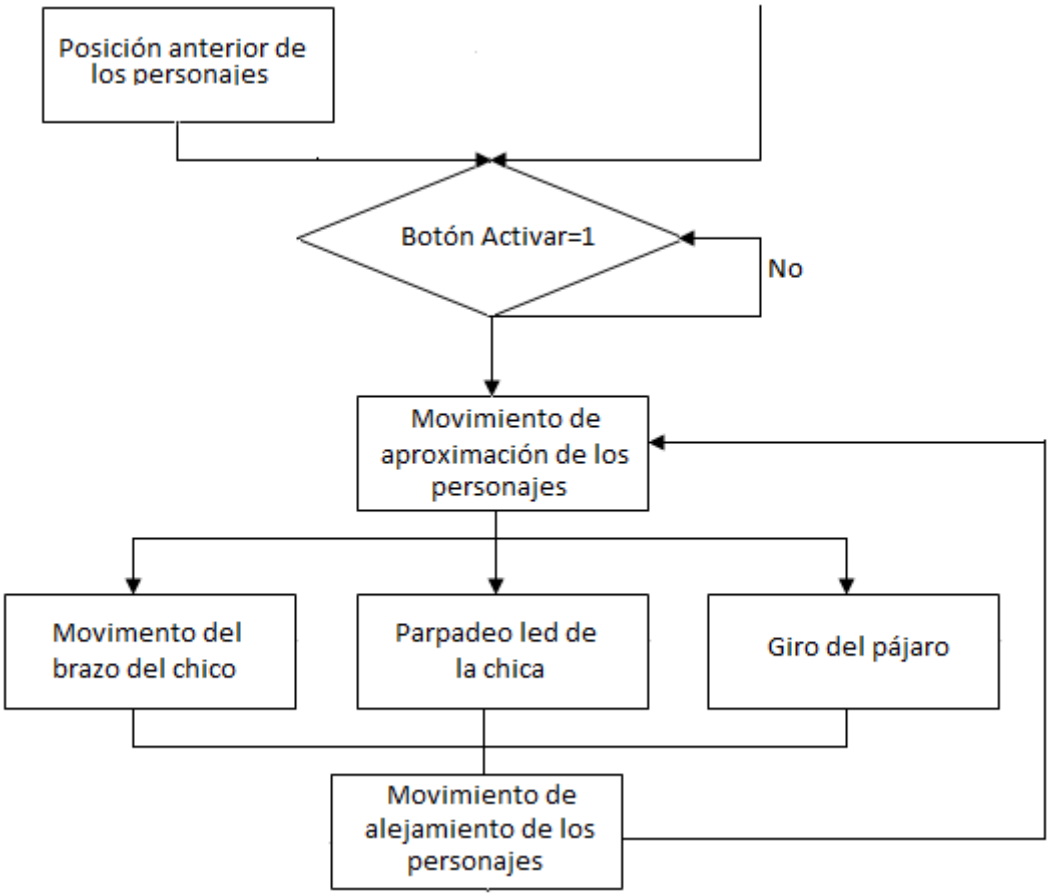


Figura 4.1: Flujograma de la representación teatral

Cabe destacar que las acciones paralelas *de Movimiento del brazo del chico, Parpadeo led de la chica* y *Giro del pájaro*, son las subacciones de una acción principal llamada *Encuentro*.

Por lo tanto, y como se puede ver de manera resumida en el flujograma de la figura 4.1, el funcionamiento de esta plataforma está basado en una máquina de estados. A modo de resumen cabe recordar que una máquina de estados es un modelo de comportamiento de un sistema formado por varias entradas y salidas, en el que las salidas pueden depender no sólo de los estados actuales sino también de las señales de entrada [9].

El ciclo de funcionamiento se va a analizar dividiéndolo en cuatro partes:

- Inicialización de variables.
- Enganche de personajes.
- Activar movimiento.
- Movimiento de la plataforma
  - Aproximación de los personajes.
  - Encuentro de los personajes (Movimiento del brazo del chico, parpadeo del led y giro del pájaro).
  - Alejamiento de los personajes.

4.1.1. Inicialización de variables.

Se describen a continuación las variables que intervienen en el funcionamiento de la plataforma.

- **analoInPin1 = A0:** variable asignada al puerto A0. Este puerto, es el que lee la tensión de entrada al pulsar el botón *Activar,* como se puede ver en la figura 2.5 del capítulo 2.2.
- **analoInPin2 = A1:** variable asignada al puerto A1. Este puerto, es el que lee la tensión de entrada al pulsar el botón *Enganchar,* como se puede ver en la figura 2.5 del capítulo 2.2.
- **encender = 13:** esta variable sirve para asignar el puerto 13 del microprocesador al led.
- **sensorValue1:** esta variable almacena la conversión analógica-digital realizada en el microprocesador al pulsar el botón *Activar*. Dicha conversión se realiza a través de la función *analogRead* tal y como se puede ver en el ejemplo siguiente:

```
sensorValue1 =  analogRead(analogInPin1)
```

Cuando la tensión de entrada sea 0V, dicha variable valdrá 0, y cuando la tensión sea de 5V, la variable valdrá 1024. Como el funcionamiento de la plataforma es por nivel, nivel alto o nivel bajo, esta variable deberá estar siempre a 1024 (nivel alto) para que el mecanismo esté en movimiento, tal y como se ve en la figura 4.2.

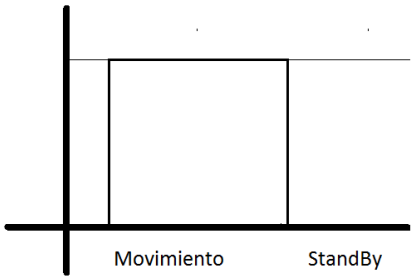


Figura 4.2: SensorValue1

- **sensorValue2:** esta variable almacena la conversión analógica-digital realizada en el microprocesador al pulsar el botón *Enganchar*. Cuando la tensión de entrada sea 0V dicha variable valdrá 0 y cuando la tensión sea de 5V la variable valdrá 1024. Como el botón para enganchar los personajes funciona por pulsos a nivel alto, para poder engancharlos/desengancharlos, la variable sensorValue2 deberá leer un flanco de subida, y para volver a su posición inicial deberá leer otro flanco de subida, tal y como se ve en la figura 4.3.

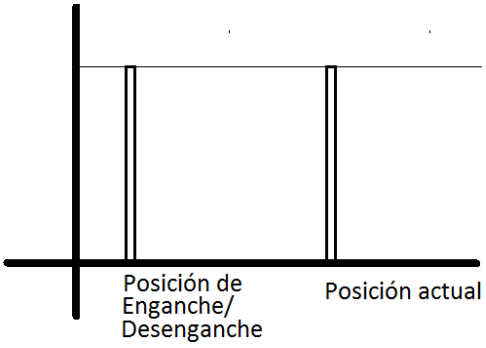


Figura 4.3: SensorValue2

- **colocar_munecos:** Con esta variable se consiguen 2 efectos:

  1.- Si el botón *Enganchar* se pulsa y esta variable vale 0, la cadena se coloca en la posición inicial del recorrido para que se coloquen o se quiten los personajes.

  2.- Si el botón *Enganchar* se pulsa y esta variable vale 1, la cadena vuelve a su posición original.
- **myservo:** esta función permite controlar el servo. Dicha función está formada a su vez por otras subfunciones de las cuales se han usado dos para la realización de la plataforma:
  - **myservo.attach(Puerto):** permite asignar un servomotor a un pin de salida del microprocesador, tal y como se ve en el ejemplo de abajo.

    myservo.attach(9);

  - **myservo.write(posición):** permite indicar al servo en qué posición se ha de ubicar, tal y como se ve en el ejemplo de abajo. Las posibles posiciones van desde 0˚ a 180˚.

    Myservo.write(pos);

- **pos:** variable que almacena la posición de los dos servomotores que mueven los personajes. Las posibles posiciones del servo van desde 0˚ a 180˚.
- **pos_brazo:** variable que almacena la posición del servomotor que mueve el brazo del personaje. Las posibles posiciones del servo en este caso van desde 0˚ a 115˚.
- **recorrido:** esta variable indica el sentido de movimiento de los personajes, los valores que puede tomar son:
  - **0:** cuando las posiciones del servo van desde 0˚ hasta 180˚.
  - **1:** cuando las posiciones del servo van desde la última posición de los personajes, en caso de que se haya dejado de pulsar el botón *Activar,* hasta 180˚.
  - **2:** cuando las posiciones del servo van desde 180˚ hasta 0˚.
  - **3:** cuando las posiciones del servo van desde la última posición de los personajes, en caso de que se haya dejado de pulsar el botón *Activar,* hasta 180˚.

  Es importante indicar que la secuencia del ciclo empieza cuando el servo está en la posición 180˚ y termina cuando llega a la 0˚.
- **addr0 = 0**: esta constante guarda la dirección 0 de la memoria EEPROM.
- **addr1 = 1**: esta constante guarda la dirección 1 de la memoria EEPROM.
- **primera_vez:** gracias a esta variable, si se pulsa el botón *Activar* tras cambiar la alimentación o alimentar la plataforma de nuevo, el microprocesador lee las posiciones almacenadas en la memoria EEPROM para que la plataforma siga su funcionamiento en las mismas condiciones en las que esta estaba cuando se quitó la alimentación o se cambió.
- **estado_abrazo:** esta variable valdrá:
  - **1**: el brazo va de 0˚a 115˚.
  - **2**: el brazo va desde la última posición guardada, en caso de que se haya dejado de pulsar el botón *Activar*, hasta 115˚.
  - **3**: el brazo va de 115˚ a 0˚.

- **4**: el brazo va desde la última posición guardada, en caso de que se haya dejado de pulsar el botón *Activar*, hasta 0˚.

- **brazo_inicio:** variable que almacena la posición inicial del brazo, valiendo 0 si el brazo sube ó 115 si este baja.
- **brazo_final:** variable que almacena la posición final del brazo, valiendo 115 si el brazo sube ó 0 si este baja.
- **brazo_desactivado:** Esta variable es analizada cuando se pulsa el botón *Activar*. Si vale 0 significa que el brazo no estaba siendo utilizado, si vale 1 significa que el brazo estaba en movimiento cuando se dejó de pulsar el botón *Activar* y que, por lo tanto, se debe realizar la secuencia de movimiento del brazo.
- **contador:** variable que almacena la duración del ciclo de trabajo de la PWM del led, es decir, el tiempo que parpadea el led del personaje.
- **led_encendido:** si esta variable vale 1, el led está encendido, de lo contrario, está apagado.
- **tiempo_encendido = 18:** tiempo que permanece el led encendido, está iniciado a 18 segundos, pero al variar la PWM este tiempo va creciendo y decreciendo según la frecuencia del led.
- **pos_pajaro = 115:** variable que almacena la posición del pájaro.
- **movimiento:** Como con un mismo microprocesador se deben controlar a la vez diversos servos con distintas velocidades, esta variable junto con la variable *Par* permite que por cada dos incrementos de giro del brazo del personaje se mueva un incremento el pájaro.
- **par:** Como con un mismo microprocesador se deben controlar a la vez diversos servos con distintas velocidades, esta variable junto con la variable *Movimiento* permite que por cada dos incrementos de giro del brazo del personaje se mueva un incremento el pájaro.

A lo largo de este capítulo se va a hacer referencia a las variables inicio y fin, las cuales son las posiciones de inicio y de fin de los servomotores dependiendo del valor que tenga la variable *recorrido.*

Las variables se encuentran declaradas al inicio del programa, tal y como se muestra en la figura 4.4.

```
const int analogInPin2 = A1;  // Señal analogica del boton ENGANCHAR
const int Encender =  13;      // Puerto del micro configurado para el led
int sensorValue1 = 0;         // Lee el valor del boton ACTIVAR (0V ó 5V)
int sensorValue2 = 0;         // Lee el valor del boton ENGANCHAR (0V ó 5V)
int activar = 0;
int Colocar_munecos = 0; //esta variable sirve para que si el pin A1 está a nivel alto
                         //y esta variable vale 0 los servos giran para colocar los muñeco y
                        // si A1 está a nivel alto y la variable a 1 vuelvo a poner los servos en su sitio paravolver a la posicion
Servo myservo;  // variable para controlar el servo
int pos = 0;    // variable para almacenar el valor la posición de los servos.
int pos_brazo = 0;    // variable para almacinar la posición de del brazo del muñeco
int recorrido = 0;// sentido de giro(de 0 a 180:1)(de 180 a 0:2)
int inicio = 0;
// the current address in the EEPROM (i.e. which byte
// we're going to write to next)
int addr0 = 0;//Direccion 0 de la memoria EEPROM
int addr1 = 1;//Dirección 1 de la memoria EEPROM
int primera_vez=1;//variable para activar recogida de material.
int brazo_abajo; // si brazo_abajo = 1 está bajando, por defecto se considerará subiendo
int estado_brazo = 1;// si el brazo va desde 0 a 115, inicio a 115, 115 a 0 ó inicio 0
int brazo_inicio;// posición inicial del brazo
int brazo_final;// posición final brazzo
int brazo_desactivado = 0;// si vale 1 el brazo está parado en carrera
int contador = 0;
int led_encendido = 0; // si vale 1 el led está encendido
int tiempo_encendido = 18; // tiempo que el led está encendido
int pos_pajaro = 113; //Variable para almacenar el movimiento del pájaro
int movimiento = 2;// para que por cada dos movimientos del pajaro haya uno del pajaro
```

Figura 4.4: Declaración de variables

4.1.2. Enganche de personajes.

Esta parte del código, como puede verse en el flujograma de la figura 4.5, permite colocar la cadena en la posición de arranque para colocar o descolocar los personajes y después ponerlos en su posición anterior.

Al pulsar el botón *Enganchar* se analiza la variable colocar_munecos. Si ésta vale 0, significa que la cadena está en cualquier posición de su recorrido, por lo que ésta se situará en la posición 180˚ de los servomotores para colocar los personajes. Si ésta vale 1, significa que los personajes están en la posición 180˚, es decir, con la cadena recogida, por lo que la cadena se situará en la posición que ésta tenía antes de pulsar por primera vez el botón *Enganchar*.

Cabe destacar el retraso de 200 ms que se produce en cada una de las dos posibles situaciones para que si se pulsa el botón más tiempo del normal no se den flancos indeseados.

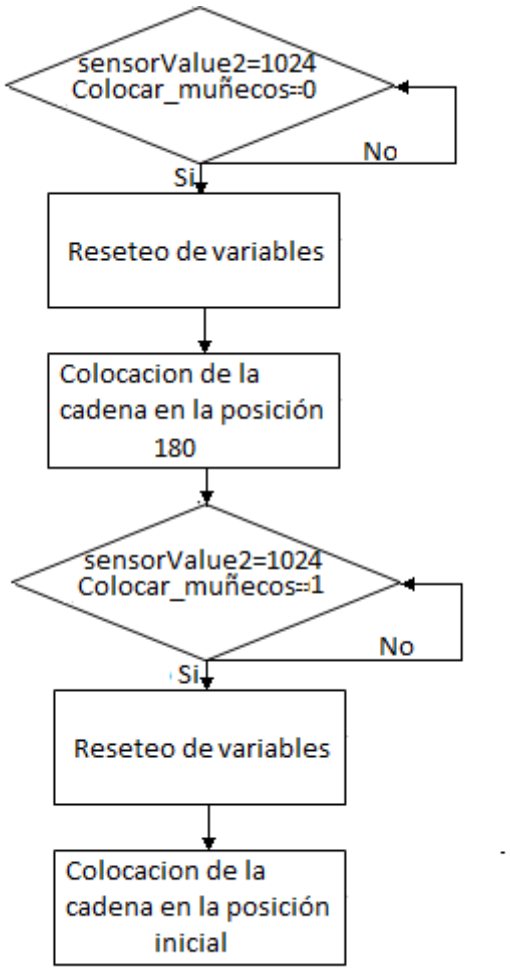


Figura 4.5: Flujograma botón *Enganchar*

4.1.3. Activar movimiento.

Esta parte del código, cuya funcionalidad se puede ver en el flujograma de la figura 4.6, es la que permite el movimiento de los personajes. En ella, el microprocesador está en un estado de StandBy, con todos los motores parados y el led apagado hasta que la variable sensorValue1 vale 1024. En ese momento empieza el movimiento de la plataforma.

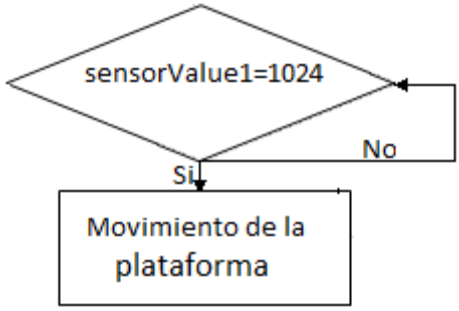


Figura 4.6: Activar movimiento

Para la realización de esta parte del programa han sido piezas clave las dos líneas que se muestran en el código de abajo. La primera permite leer valor de la tensión que le llega al puerto A0; la segunda permite ejecutar el movimiento del mecanismo si el valor leído anteriormente es 1024, es decir, si la tensión es 5V.

```
sensorValue1 = analogRead(analogInPin1);

if (sensorValue1 == 1024)

{

…

}
```

Estas dos líneas se van a ejecutar bastantes veces a lo largo del código, ya que es necesario actualizar constantemente el valor de la variable sensorValue1 para que cuando se deje de pulsar el botón *Activar* se desactive el movimiento de la plataforma al instante.

4.1.4. Movimiento de la plataforma

A continuación, en la figura 4.7 se muestra un flujograma para poder comprender mejor las fases que componen el movimiento de la plataforma (*Aproximación de los personajes, Encuentro de los personajes* y *Alejamiento de los personajes*).

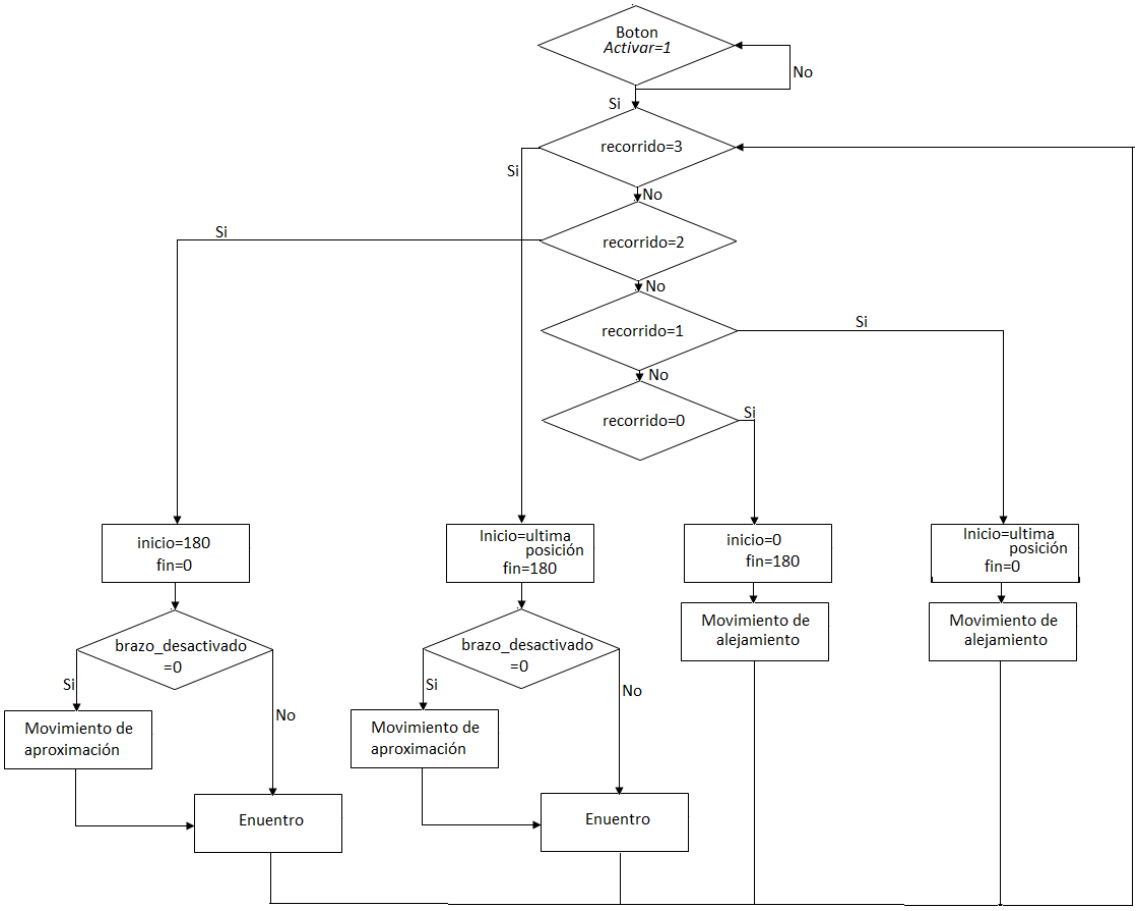


Figura 4.7: Flujograma ciclo de funcionamiento

Como se ha comentado anteriormente, cada ciclo de funcionamiento tiene tres movimientos o fases principales, *Aproximación de los personajes, Encuentro de los personajes, Alejamiento de los personajes*. Estos movimientos dependerán del valor que tenga la variable recorrido: si ésta vale 2 ó 3, se dará la fase de *Aproximación* y la de *Encuentro;* en cambio, cuando la variable recorrido valga 0 ó 1, sólo se dará la fase de *Alejamiento*.

A continuación se explican cada una de estas situaciones.

4.1.4.1. Aproximación de los personajes

Esta situación se da cuando la variable *recorrido* vale 2 ó 3.

Variable recorrido=2

Si se está en la situación en la que la variable *recorrido=2*, se comprueba si la fase de funcionamiento es la *de Aproximación de personajes* o la de *Encuentro* (fase que se analizará en la sección siguiente 4.1.4.2). Para ello se analiza la variable *brazo_desactivado,* si esta variable vale 0, se activarán los servomotores para que se dé el movimiento de aproximación. En cambio, si la variable *brazo_desactivado* vale 1, se pasará a la fase de *Encuentro*.

Para realizar el movimiento de los personajes se ha realizado un bucle *for* que recorra las 180 posiciones de los servos de grado en grado realizando una parada de 15 ms entre cada posicionamiento, regulando así la velocidad con la que se acercan los personajes principales. Esto se ha realizado con las líneas de código siguiente:

```
For (pos = 180; pos >0; pos +=1)
    {
        …
        Myservo.write(pos);
        Delay(15);
        EEPROM.write(addr0, pos+1);
        …
    }
```

Como se puede ver en el código, tenemos una sentencia que recorre la variable *pos* desde 180 a 0, decrementándose en 1 en cada ciclo. Dentro del bucle están los comandos *myservo.write* y *delay.* Con el primero movemos el servo hasta la posición almacenada en la variable *pos,* y con la segunda se realiza un retraso de 15 ms entre cada movimiento. Además, dentro del bucle se encuentran la instrucción *EEPROM.write,* para tener guardado en memoria la posición en la que se encuentra el motor.

Cabe destacar que en esta secuencia de movimiento pueden pasar dos situaciones: la primera es que el movimiento se realice con normalidad y finalice dando paso a la fase de encuentro de los personajes, y la segunda opción es que el movimiento se corte en cualquier punto del recorrido al desactivar el botón *Activar*. Si se está en este segundo caso se analiza la secuencia de la figura 4.8, la cual tiene por nombre *Guardado de posiciones,* es de vital importancia y se ejecuta constantemente en el código.

```
//Guardado de posiciones
sensorValue1 = analogRead(analogInPin1);
if (sensorValue1 == 0) {
  inicio = pos;
  recorrido = 3;
  break;
}
```

Figura 4.8: Guardado de posiciones

Como se puede ver en el código de la figura 4.8, al principio se lee si el botón *Activado* está pulsado o no, tal y como se explicó anteriormente. Si este no está apretado (sensorValue1 == 0), se guarda la posición actual en la que se encuentran los personajes y se indica que la próxima vez que se entre en el estado *Activar movimiento* se debe ir a la secuencia que marca la variable *recorrido=3.* Esto se debe a que la próxima vez que se deban mover los personajes, este movimiento no va desde 180˚ a 0˚, como antes, sino que ahora se reanuda el movimiento desde el punto en que se encuentran los personajes actualmente hasta 0˚, siendo el punto actual la posición almacenada en la variable *inicio.*

Es importante distinguir la diferencia entre cuando se guarda la posición en la variable *inicio* y cuando se guarda en la memoria EEPROM. La primera sólo se hace cuando el botón *Activar* deja de estar pulsado para que la próxima vez que se deban mover los personajes se empiece el movimiento desde la posición guardada en la variable *inicio,* mientras que la memoria EEPROM se está actualizando constantemente con la posición de los personajes para que si se quita la alimentación y se apaga el microprocesador se tenga guardada en la memoria la posición de los personajes.

Variable recorrido=3

La fase de movimiento en la que la variable *recorrido* vale 3, tiene la misma funcionalidad que cuando *recorrido* vale 2, y su estructura es igual con la única diferencia de que en esta etapa el movimiento no empieza en 180˚, sino que empieza en el punto que le indique la variable *pos*. También es importante indicar que en caso de que se desactive el botón *Activar*, la próxima vez que se pulse se volverá a ir a *recorrido=3* sólo que ahora la variable *inicio* tendrá por posición la posición de esta última etapa en que se paró el movimiento.

4.1.4.2. Encuentro de los personajes

La fase de Encuentro está formada por tres acciones simultáneas:

- Movimiento del brazo de un personaje.
- Latido del corazón del otro personaje.
- Movimiento de un pájaro del decorado.

Como se ha comentado anteriormente, esta fase también depende del valor de la variable *recorrido,* como se detalla a continuación.

Variable recorrido=2

En esta fase de la representación uno de los personajes realiza una elevación del brazo hasta alcanzar la mejilla del otro, punto en el cual realiza una acaricia para después bajar hasta la posición inicial. Aunque estos parecen movimientos muy parecidos, su programación es un poco diferente, por lo que se va a estudiar diviendola en tres subfases: subida del brazo, caricia y bajada del brazo.

Estas tres secuencias se ejecutan de manera secuencial una vez terminada la fase de *Aproximación de los personajes*, y tiene una estructura igual a la del movimiento de la cadena, ya que las fases del movimiento se dividen en 4 posibles opciones:

- **estado_brazo=1**: el brazo va de 0˚ a 115˚.
- **estado_brazo=2**: el brazo va desde la última posición guardada, en caso de que se haya dejado de pulsar el botón *Activar*, hasta 115˚.
- **estado_brazo=3**: el brazo va de 115 a 0˚.
- **estado_brazo=4**: el brazo va desde la última posición guardada, en caso de que se haya dejado de pulsar el botón *Activar*, hasta 0˚.

estado_brazo=1 y estado_brazo=2 realizan las subfases de subida del brazo y caricia, mientras que estado_brazo=3 y estado_brazo=4 realizan la subfase de bajada del brazo. Es importante indicar que, aunque la variable se llama *estado_brazo,* también se realizan las acciones latido del corazón y movimiento del pájaro.

A continuación se pasa a describir cada uno de las situaciones que se dan dependiendo del valor de la variable *estado_brazo.*

estado_brazo=1

Subida del brazo

Al igual que para el movimiento de la cadena, el movimiento del servomotor que mueve el brazo se consigue gracias a un bucle *for* que recorre todos los grados comprendidos entre 0˚ y 115˚.

Dentro de este bucle *for,* lo primero que se hace es comprobar el estado del led para su correcto parpadeo: si este está encendido, deberá apagarse, y si por el contrario está apagado, deberá encenderse. Como este parpadeo tiene una frecuencia que va aumentando con el tiempo, éste va creciendo según el brazo se acerca a la cara del personaje.

En esta secuencia se deben mover dos motores (el del brazo y el del pájaro) a la vez y simulando movimientos distintos, con una única señal PWM. Para ello, se ha establecido un movimiento de tipo continuo para el pájaro, el cual se mueve siempre, y otro de tipo discreto para el brazo, el cual se mueve un incremento sí y otro no. Para conseguir esto, se han creado dos variables, *movimiento* y *par,* la variable *movimiento* está inicializada a 2 y se incrementa en 1 en cada ciclo. En cambio, la variable *par* es el resto de dividir *movimiento* entre 2 y por lo tanto va a ir alternando su valor en cada ciclo entre 0 ó 1. El resultado que se consigue es que cuando *par* valga 0 el brazo se moverá y cuando valga 1, no se moverá.

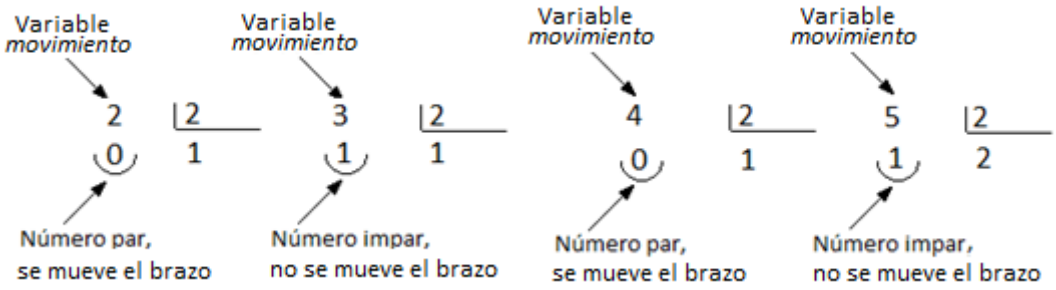


Figura 4.9: Movimiento del brazo y del pájaro (I)

Como podemos ver en la figura 4.9, este es el cálculo que en cada ciclo realiza el microprocesador para saber cuándo ha de moverse el brazo.

No obstante, con una simple división no basta para conseguir las dos velocidades, sino que es necesario ejecutar el código mostrado en la figura 4.10 para conseguir los movimientos deseados en el brazo y pájaro.

```
par = movimiento % 2;
myservo.attach(7);
myservo.write(pos_brazo);
if (par == 0 )
  {myservo.attach(8);
  myservo.write(pos_brazo);
  contador++;}
else
  {pos_brazo = pos_brazo - 1;}
  movimiento = movimiento + 1;
```

Figura 4.10: Movimiento del brazo y del brazo (II)

En esta secuencia además, al igual que ocurría con el movimiento de la cadena, pueden darse dos situaciones:

- La primera es que el código se ejecute sin ninguna alteración, por lo que al final de esta fase de movimiento la variable *estado_brazo* pasa a valer 3 para indicar que la siguiente secuencia que se debe ejecutar es la de *estado_brazo=3*.
- La segunda es que se deje de pulsar el botón *Activar*, ejecutándose en este caso la secuencia *Guardado de posiciones*.

```
sensorValue1 = analogRead(analogInPin1);
if (sensorValue1 == 0)
{
   digitalWrite(13, LOW);
   brazo_desactivado = 1;
   recorrido = 2;
   estado_brazo = 2;
   break;
}
```

Figura 4.11: Guardado de variables (Movimiento del brazo)

Como se puede ver en el código de la figura 4.11, la función *Guardado de posiciones* es similar a la que se ejecuta en la fase de aproximación pero con las siguientes diferencias. Es

importante que cada vez que se deje de pulsar el botón *Activar* se apague el led porque puede darse la situación que en cierto punto del movimiento se apague dicho botón cuando el led está en estado de encendido. En este caso lo primero que se debe hacer es la puesta a cero de la variable que lo mantiene encendido. Esto se realiza con la instrucción que se ve en la abajo.

```
digitalWrite(13,LOW)
```

Además, se asigna a las variables *brazo_desactivado, recorrido* y *estado_brazo* los valores 1, 2 y 2, respectivamente. Esto se hace para que cuando se active el movimiento se vaya a la fase Recorrido=2, dentro de Recorrido=2 se vaya a la subfase *Encuentro (*marcada por la variable *brazo_desactivado=1)* y dentro de la subfase *Encuentro* se vaya a la secuencia de subir el brazo (marcada por la variable *estado_brazo=2*).

Cabe recordar que también en esta situación se ha de guardar constantemente la posición del brazo en la memoria EEPROM.

Caricia

Cuando la posición del brazo llega a 115˚ se pasa a la fase de *Caricia*. En esta fase el brazo realiza un movimiento de bajada y de subida mientras que el pájaro sigue su movimiento en el mismo sentido que lo empezó.

El movimiento de bajada va desde los 114˚ hasta los 95˚ y su estructura es exactamente igual que la de *Subida del brazo* con las dos siguientes diferencias:

- La posición de inicio es 113˚ y la de fin es 95˚.
- Como el movimiento del brazo cambia de sentido y el pájaro no, para mover este segundo se ha creado una variable que se llama *pos_pajaro*, la cual por cada decremento de *pos_brazo* tiene un incremento para mantener la misma velocidad y sentido que tenía en la fase *Subida del brazo.*

El movimiento de subida va desde los 95˚ hasta los 114˚ y es igual que el explicado en el párrafo anterior con la diferencia de la posición inicio es 96˚ y la de fin es 115˚.

Es importante indicar que el movimiento de caricia es el punto de máxima intensidad ya que es el momento en el que los dos personajes se encuentran, y el led está encendido en su máxima intensidad.

Estado_brazo=2

Esta fase tiene por objeto continuar el movimiento del brazo y del pájaro en la posición en la que sus servos correspondientes estaban, así como de seguir la secuencia de parpadeo del led en caso del botón *Activar* se dejase de pulsar. Su estructura es igual que la de Estado_brazo=1 con la diferencia de que la posición de inicio es la ultima que tuviera el brazo en el momento que se dejó de pulsar dicho botón.

Estado_brazo=3

Bajada del brazo

Este estado es igual que el de *Estado_brazo=1* con las siguientes diferencias:

- El único movimiento que el brazo realiza es el de descenso, el cual va desde 115˚ a 0˚.
- La variable *estado_brazo* pasa a valer 1 si el movimiento es correcto ó 4 si se apaga el botón *Activar* a mitad del funcionamiento.
- Cuando termina el movimiento la variable *brazo_desactivado* se pone a 0 para indicar que la fase de encuentro ha terminado.

Estado_brazo=4

Estado_brazo=4 es igual que *Estado_brazo=3* con la única diferencia de que el movimiento se inicia desde el punto en el que se encontraban los personajes cuando se dejó de pulsar el botón *Activar*.

4.1.4.3. Alejamiento de los personajes

El movimiento de alejamiento es aquel en el que la variable *recorrido* vale 0 ó 1.

Variable recorrido=0

En esta fase de separación se realiza el movimiento de los servos que mueven los personajes, desde la posición 0˚ hasta la posición 180˚.

Esta fase tiene una estructura similar a la de Recorrido=2 con las siguientes diferencias:

- Se cambia el bucle

  for(pos = 180; pos >0; pos -=1)

  por el siguiente:

  for (pos = 0; pos <180; pos +=1)

- Si el movimiento se ejecuta sin dejar de pulsar el botón *Activar,* se pasa a la fase Recorrido=2 para iniciar de nuevo el ciclo de movimiento de los personajes.
- Si el movimiento se ve interrumpido porque se deja de pulsar el botón *Activar,* se ejecuta secuencia *Guardado de posiciones*, pasando la variable *recorrido* a valer 1.

Variable recorrido=1

Esta fase de funcionamiento tiene la misma funcionalidad que la de Recorrido=0 con la diferencia de que ahora el movimiento se empieza en la variable *inicio,* es decir, en la última posición en la que se encontraban los motores en la fase de Recorrido=0 cuando se dejó de pulsar el botón *Activar*.

4.2. Programación de las piezas 3D

En este apartado se presenta el diseño de las piezas 3D que integran la plataforma. Estas son: rueda motriz, rueda arrastrada, cadena, soporte del servomotor, soporte de la rueda arrastrada y enganches.

4.2.1. Rueda motriz

Como se ha comentado anteriormente, las ruedas motrices y arrastradas ya están diseñadas y, debido a su diseño parametrizable, sólo se debe escribir su diámetro interno, externo, número de dientes, separación ente estos y la altura de la rueda. No obstante se va a analizar el código de esta para comprender su funcionamiento.

Tanto la rueda motriz como la arrastrada tienen la misma forma y dimensiones, con la diferencia de que los diámetros internos son diferentes ya que la primera debe ajustarse al eje del servo mientras que la segunda se ajusta al tornillo sobre el cual gira. Por lo tanto el código va a estar basado en dos formas geométricas: la rueda base, la cual es igual para ambas ruedas y el cilindro interno, que depende del tipo de rueda.

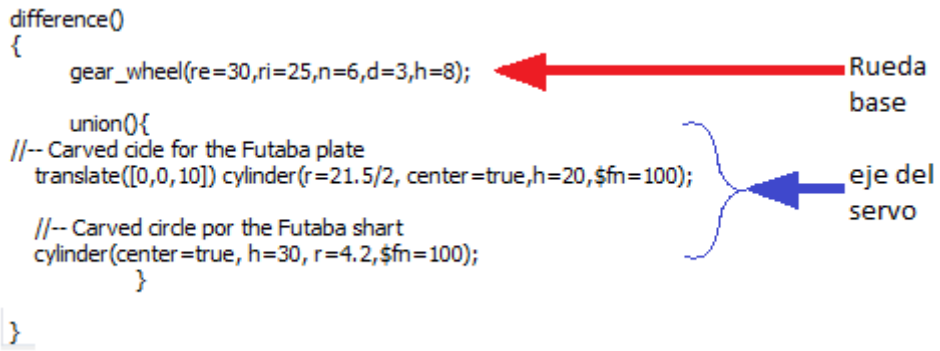


Figura 4.12: Código OpenScad de la rueda motriz

Como se puede ver en el código de la figura 4.12, la rueda motriz se obtiene por la resta del eje del servo a la rueda base.

El código completo de esta pieza puede verse en el anexo A.2.1.

4.2.2. Rueda arrastrada

La rueda arrastrada se consigue, al igual que la rueda motriz, mediante una resta. Sólo que ahora a la rueda base se le resta el cilindro del tornillo, tal y como puede verse en el código de la figura 4.13.

```
difference()
{
      gear_wheel(re=30,ri=25,n=6,d=3,h=8);
      cylinder(center=true, h=30, r=4.2,$fn=100);
}
```

Figura 4.13: Código OpenScad de la rueda arrastrada

El código completo de esta pieza puede verse en el anexo A.2.2.

4.2.3. Cadena

En el caso de la cadena, esta está formada por una serie de eslabones que enganchan cada uno con el siguiente. La singularidad que presentan estos eslabones es que el hueco interior que estos tienen, deben poseer las mismas dimensiones que los dientes de las ruedas en los que estos engranan para que el movimiento de la cadena sea suave y preciso. Además deben de llevar unos cilindros interiores en la parte delantera y trasera para poder engranar con los eslabones anteriores y posteriores, tal y como se ve en la figura 4.14.

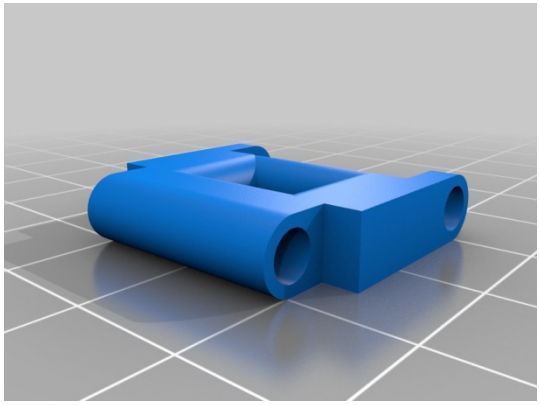


Figura 4.14: Eslabón de la cadena

El código de esta pieza puede verse en el anexo A.2.5.

4.2.4 Soporte del servomotor

Como puede verse en la figura 4.15, esta pieza se ha formado principalmente a partir de las siguientes formas geométricas: una base rectangular, un soporte vertical rectangular también y un tercer rectángulo hueco que sobresale del soporte vertical y en el que se aloja el servomotor. Los cuatro cilindros que se abren en la base están destinados a alojar los tornillos que unirán la pieza con la caja.

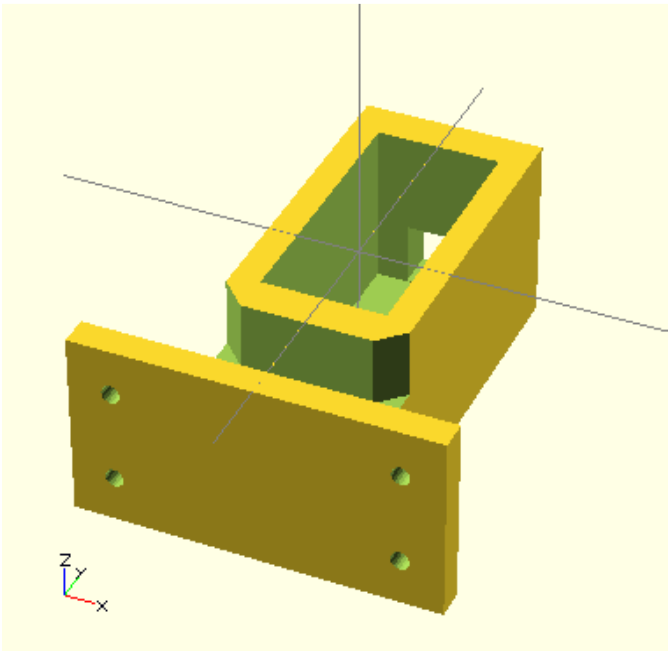


Figura 4.15: Soporte servomotor

Cabe destacar que la pieza se ha diseñado a partir de un cubo vertical, al cual se le han ido quitando partes para darle la forma necesaria gracias a la función *difference*, tal y como puede verse en el código de la figura 4.16.

```
difference(){
translate([0,0,-15])
cube([30,75,30],true);

translate([-34,-3,-11])
rotate([0,0,-45])
cube([35,5,12]);

translate([13,-24.5,-11])
rotate([0,0,45])
cube([35,5,12]);

translate([0,9,-13.2])
cube([20.5,44,26.5],true);

translate([0,30,-21.2])
rotate([90],0,0)
cube([10,10,28],true);

translate([0,-28,6.5])
cube([33,17,35],center=true);
}
```

Hueco con las dimensiones del servomotor

Figura 4.16: Código OpenScad del soporte servomotor

El código completo de esta pieza puede verse en el anexo A.2.3.

4.2.5. Soporte de la rueda arrastrada

Como puede verse en la figura 4.17, esta pieza se parece bastante a la anterior con la diferencia de que tiene un cilindro en el eje Z en vez del rectángulo para la inserción del servomotor. Por lo tanto, esta pieza se ha formado principalmente a partir de las siguientes formas geométricas: una base rectangular, un soporte vertical rectangular también y un cilindro horizontal en el que se atornillará la rueda arrastrada. Al igual que en el caso anterior, los cuatro cilindros que se abren en la base están destinados a alojar los tornillos que unirán la pieza con la caja.

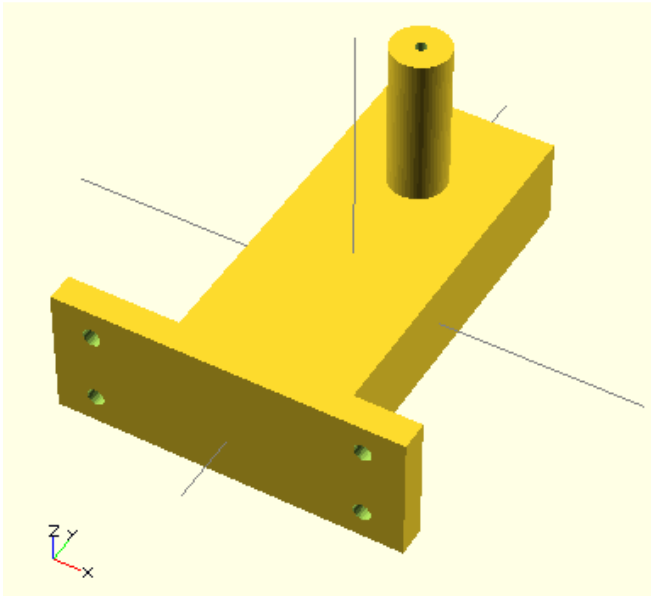


Figura 4.17: Soporte rueda arrastrada

El eje horizontal se ha formado mediante la resta a un cilindro de radio 5 mm otro de radio 1 mm, ya que el tornillo sobre el que gira la rueda es de 2 mm de diámetro, tal y como se ve en el código de la figura 4.18.

```
difference(){

rotate(90,0,0)
translate([20,0,7])
cylinder(r=5,h=27,center=false,$fn=50);

rotate(90,0,0)
translate([20,0,20])
cylinder(r=1,h=15,center=false,$fn=50);
}
```

Figura 4.18: Código rueda arrastrada

El código completo de esta pieza puede verse en el anexo A.2.4.

4.2.6. Enganche

Como puede verse en la figura 4.19, el enganche está formado por tres formas geométricas: una base con las dimensiones exactas para adaptarse al eslabón de la cadena, un cilindro vertical y un rectángulo interior en el que se alojan las varillas cuadradas que soportan los personajes.

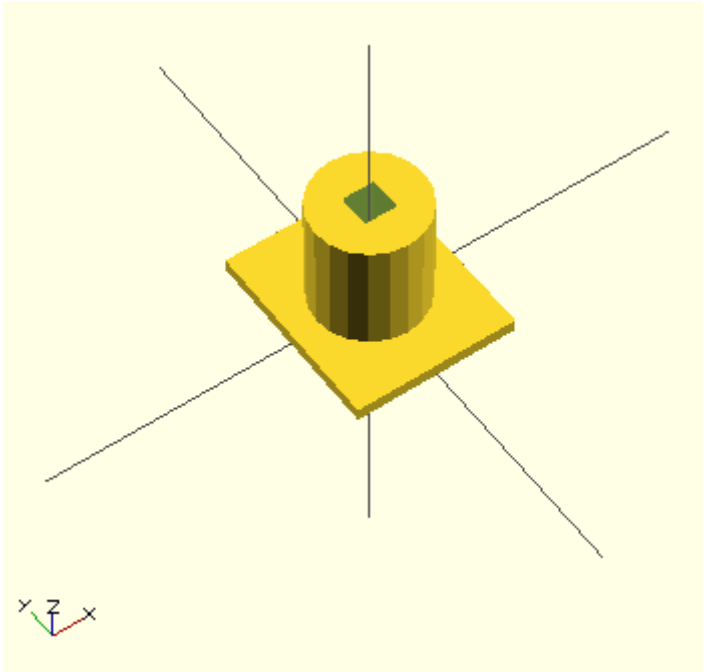


Figura 4.19: Soporte enganches

El código completo de esta pieza puede verse en el anexo A.2.6.

4.3. Diseño de la placa de circuito impreso

El circuito electrónico estará realizado en una placa de circuito impreso y tendrá como función conectar el microprocesador con el resto de componentes electrónicos y eléctricos que lleva la plataforma (motores, led, pilas y transformador de tensión para la alimentación a una fuente externa).

4.3.1. Diseño del circuito electrónico

El diseño del esquema del circuito se ha realizado con el programa Orcad Capture CIS, ya que es una gran herramienta integral con gran cantidad de librerías que incluyen, todos los componentes electrónicos necesarios y que permite el análisis de los circuitos de distintos puntos.

Dicho circuito está pensado para llevar las señales del microprocesador Arduino a los motores y led así como permitir la alimentación a pilas y red. Para ello se va a realizar una placa de circuito impreso que se base en las dimensiones y puertos del microprocesador, pero se utilizarán sólo los puertos necesarios para este proyecto fin de carrera.

A continuación se comentan los puntos más importantes del diseño.

4.3.1.1. Pin 5V

Este pin es de los mas importantes, ya que es necesario para poder alimentar los servomotores y los circuitos para la activación del movimiento o de los enganches. La alimentación de los motores se ha realizado mediante un conexionado en paralelo para cada uno de los motores que integran la plataforma, tal y como se ve en la figura 4.20. Al implementar cuatro servos la intensidad que circula por ellos se va a dividir entre 4, no obstante esta intensidad es suficiente para poder accionar cada uno de dichos motores.

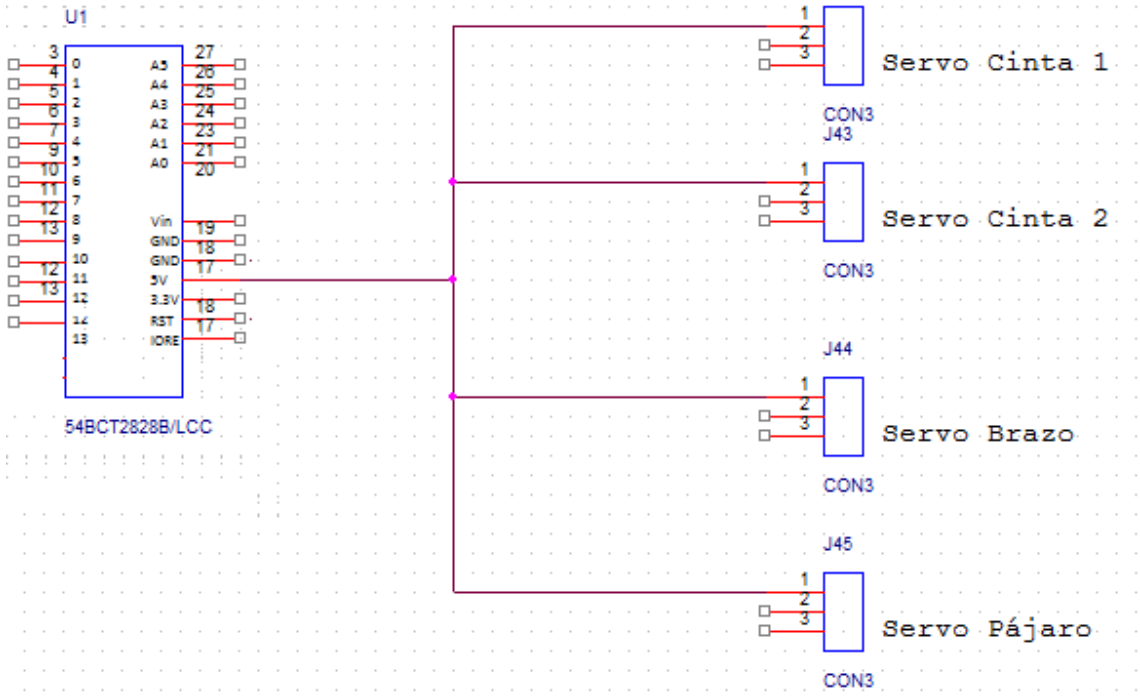


Figura 4.20: Alimentación de los motores

En cuanto al conexionado de los botones, éste se podría haber hecho conectando el puerto de 5V con el puerto de entrada del micro a través de un switch, tal y como se muestra en la figura 4.21.

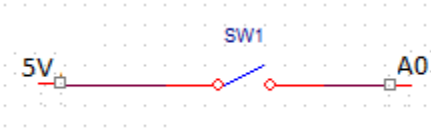


Figura 4.21: Conexionado de los botones

Este circuito funciona perfectamente cuando el el switch está cerrado y al puerto de entrada del microprocesador le llegan los 5V. No obstante, si el circuito está abierto, el puerto de entrada lee ruido, por lo que puede estar leyendo cualquier valor, incluido 5V. Por lo tanto, esta solución no es precisa.

Para solucionar este inconveniente se ha recurrido al esquema de la figura 4.22. Este tipo de conexionado permite que cuando el interruptor está pulsado, se establezca un circuito cerrado en el que se puede apreciar cómo una intensidad circula por la resistencia, produciendose una diferencia de potencial, la cual es leída por el puerto de entrada.

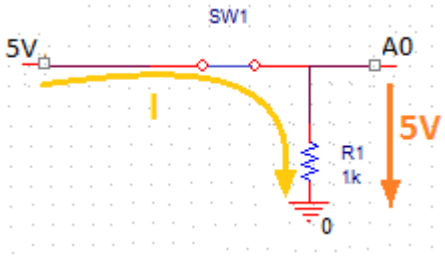


Figura 4.22: Circuito cerrado

Sin embargo, cuando el switch está abierto, la única tensión que el puerto lee en la resistencia es la de 0V ya que ninguna intensidad circula por ella, tal y como se ve en la figura 4.23.

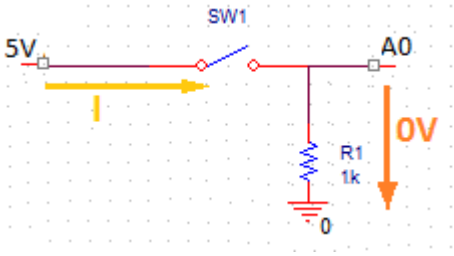


Figura 4.23: Circuito abierto

4.3.1.2. Pin GND

Como puede verse en la figura 4.24, todos los componentes están conectados a tierra. La conexión a tierra se realiza mediante el plano de masa de la PCB, la cual se une al microprocesador a través de una de las tres tierras que posee.

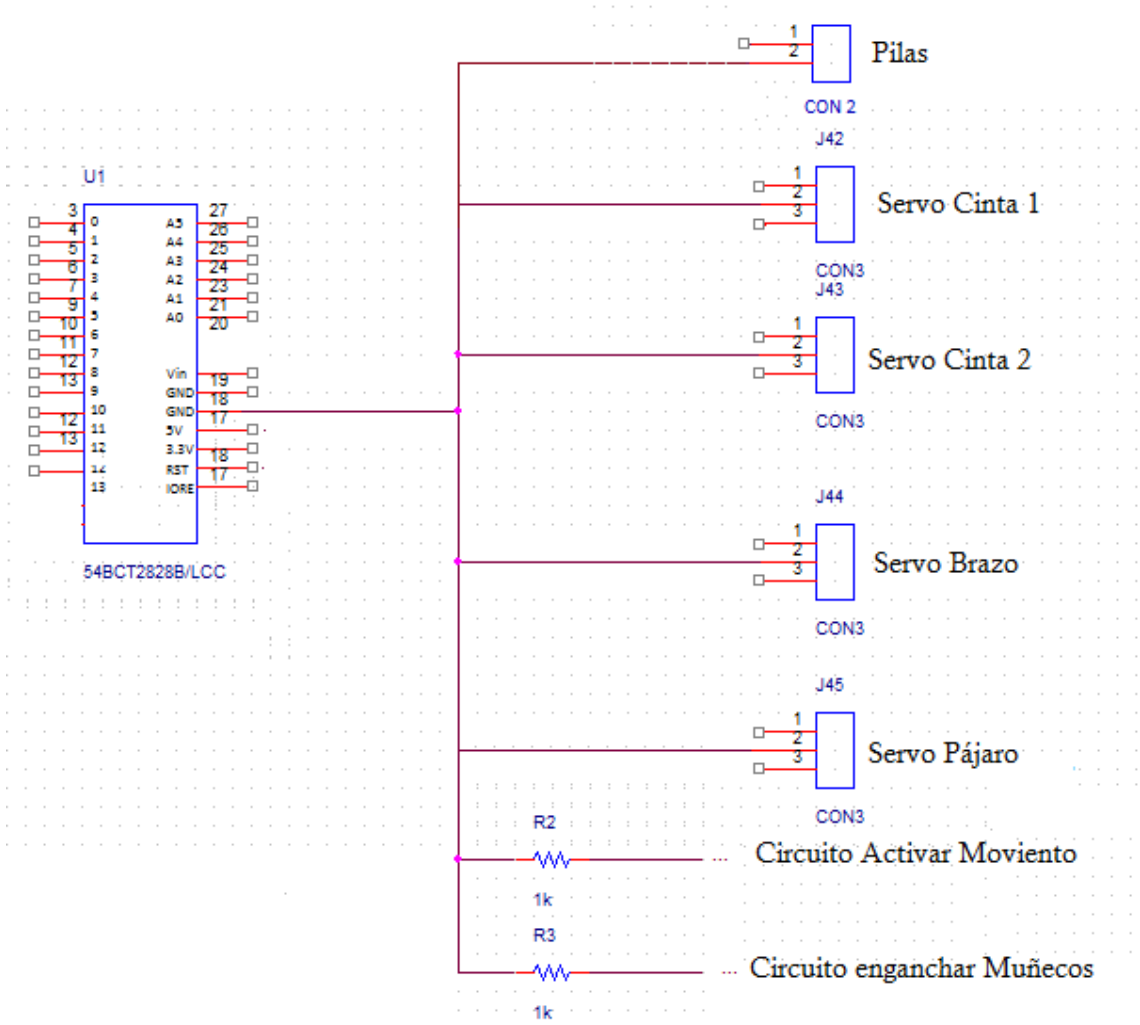


Figura 4.24: Tierra

4.3.1.3. Señal PWM

La PWM que mueve las dos cadenas sobre la que se montan los personajes principales tiene que llegar a dos motores, por lo que es necesario sacar del pin 9 dos cables para que lleven la misma señal, tal y como se ve en la figura 4.25.

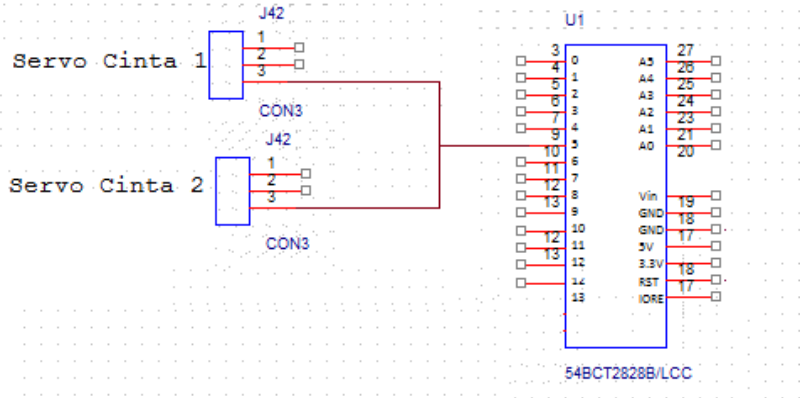


Figura 4.25: PWM cintas

4.3.2. Diseño de la placa

El diseño de la placa se ha realizado con herramienta informática Orcad PCB editor, ya que permite exportar fácilmente el circuito realizado previamente en Orcad Capture CIS y diseñar todos los footprints a las medidas necesarias, lo cual es muy importante porque las pistas de la placa deben adaptarse perfectamente al microprocesador.

Como puede verse en la figura 4.26, la placa está basada en un plano de masa al que se le han añadido a los laterales los pines del microprocesador para poder encajarlo con el mismo. El resto de componentes se han distribuidos en la placa de tal manera que los caminos y elementos de la placa estén lo más claro posible. A continuación se explican los distintos componentes que aparecen en el layout:

- **Conectores CN1, CN2, CN3:** como se ha indicado anteriormente, los conectores están dispuestos en los laterales. Cabe destacar que en el lado derecho, la placa consta de los conectores CN2 y CN3. Esto se debe a que, como la precisión para juntar placa y microprocesador es milimétrica, se han incluido dos conectores para quedarnos con el que mejor se adapte al Arduino.
- **PWM_MU1** y **PWM_MUW2:** estos terminales son los que van a los dos motores que accionan la cadena. Como puede verse en la figura 4.26, los puntos 1, 2 y 3 son para señal, tierra y alimentación respectivamente.
- **PWM_PAJ:** este terminal es el que va al motor que acciona el pájaro. Los puntos 1, 2 y 3 son para señal, tierra y alimentación, respectivamente.
- **PWM_BZO:** este terminal es el que va al motor que acciona el brazo del personaje. Los puntos 1, 2 y 3 son para señal, tierra y alimentación respectivamente.
- **P_ACTIV:** este conector es el que va al botón que permite activar el movimiento de los personajes. Como puede verse en la figura 4.26, lleva asociada una resistencia por la razón explicada en el punto 4.3.1.1.
- **P_ENGAN:** este conector es el que va al botón que permite enganchar los personajes. Como puede verse en la figura 4.26, lleva asociada una resistencia por la razón explicada en el punto 4.3.1.1.
- **BAT_CON:** este conector sirve para conectar las pilas. Como puede verse en la figura 4.26, el conector 1 une la tierra del portapilas con la tierra de la PCB mientras que el conector 2 une la tensión del portapilas con la tensión de entrada de la PCB.
- **ON-OFF:** asociado al conector BAT_CON está el ON-OFF, el cual va al switch de 3 posiciones y que permite seleccionar alimentación de la pilas o de un punto de red cualquiera.

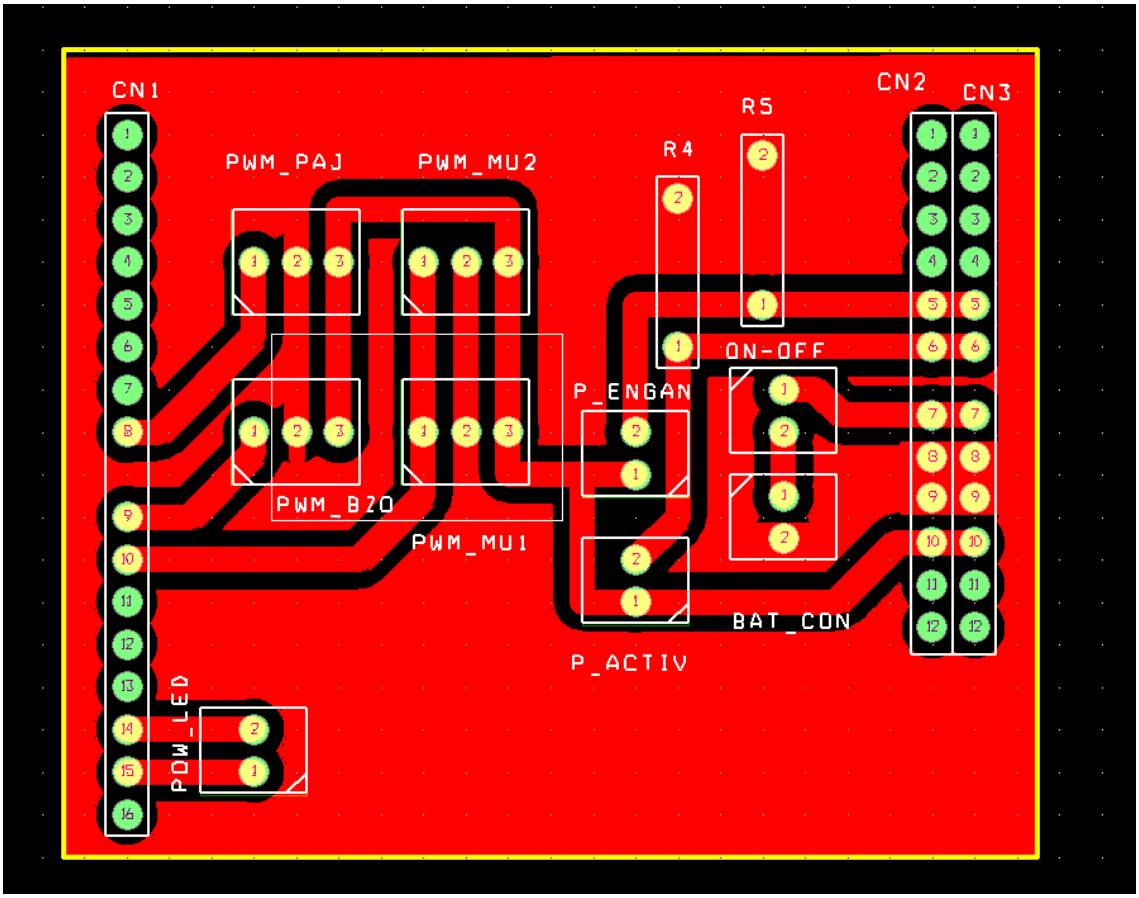


Figura 4.26: Layout de la PCB

En la figura 3.15 de la sección 3.3 se puede ver una imagen de la PCB realizada.

Capítulo

5. Puesta en marcha

A continuación se hace una pequeña descripción del funcionamiento de la maqueta así como de su uso.

5.1. Encendido de la plataforma

La puesta en marcha de la plataforma comienza en el momento en el que se selecciona con el botón negro del lateral la opción de alimentación (pilas o punto de red). En este momento la plataforma está encendida y se pueden seleccionar las opciones de activar el movimiento o enganchar los personajes, a través de los botones amarillo y verde respectivamente.

5.2. Colocación de los personajes

Teniendo la plataforma encendida y pulsando el botón amarillo, se puede observar cómo la cadena se desplaza hacia atrás para posicionarse en un punto en el que los personajes están colocados sobre las ruedas motrices. Esta posición permite enganchar o desenganchar los personajes más fácilmente al tener un apoyo.

5.3. Activar el movimiento

Teniendo la plataforma encendida y pulsando el botón verde, se puede observar cómo se activa el movimiento lineal de los personajes.

La función comienza con los personajes aproximándose el uno al otro, hasta que se encuentran. En este punto, uno de los personajes levanta el brazo para acariciar la cara del otro mientras se puede ver cómo el corazón de este último empieza a latir, latidos que serán más rápido según el brazo avanza. Además, en este punto puede verse el vuelo de un pájaro que gira sobre sí mismo mientras los personajes están juntos. Una vez terminada la caricia, los personajes se separan hasta llegar a una posición en la que empieza el ciclo de nuevo.

Puede visualizarse el video de la plataforma funcionando en el siguiente enlace:

http://www.marinaanaya.com/es/category/otras

Capítulo:

6. Conclusiones y trabajos futuros

Una vez desarrollado el proyecto y visualizado los resultados, se ha llegado a la conclusión de que se han conseguido llevar a cabo todos los objetivos.

- Se ha conseguido realizar un diseño de la plataforma optimizando tanto el tamaño de la misma como del número de elementos electrónicos que la componen.
- Se ha realizado un prototipado de todas las piezas que eran necesarias para conseguir un movimiento lineal a partir del circular, ya sea realizando los diseños desde cero o aprovechando los ya existentes.
- Se ha conseguido controlar todos los dispositivos electrónicos gracias al microprocesador Arduino y la placa PCB.

Al comienzo del proyecto no se tenía apenas conocimiento en profundidad sobre las herramientas a emplear: impresora 3D y Arduino. La lectura de manuales, foros y documentos ayudaron a comprender la problemática e importancia del software libre (Open Source), y más profundamente la ventaja de poder crear tus propios diseños y controlarlos mediante un microprocesador de bajo coste.

Esto ha permitido desarrollar el proyecto progresivamente, adaptándose en cada caso a los diferentes cambios que han surgido en la realización del mismo.

En cuanto a las herramientas en sí, la impresora 3D de la empresa Makerbot ha demostrado que la edición e impresión de piezas 3D es mucho más que un instrumento para la creación de figuras, sino que es un medio que permite el prototipado rápido de elementos. Así mismo, la plataforma Arduino ha demostrado ser una herramienta que permite fácilmente la programación para el control de diversos elementos electrónicos.

Como trabajos futuros pueden incluirse las siguientes líneas de estudio:

- Inclusión de audio en la plataforma para que ésta esté dotada de sonido mientras se realiza la representación de la función.
- Mayor número de elementos móviles en la plataforma, como más pájaros en movimiento y con distintos sentidos de giro o velocidades.
- Inclusión de otros elementos luminosos a parte del led, que se vayan encendiendo y/o apagando según las distintas fases en las que se encuentre el ciclo de funcionamiento.
- Proponer otro sistema para pasar de movimiento circular a lineal, reduciendo así la cantidad de piezas impresas por la impresa 3D y asegurando mayor robustez del sistema.
- Sensorización de la plataforma de modo que permita su integración con el entorno, permitiendo que el flujo de funcionamiento se vea alterado o definido por eventos externos a la propia plataforma.

7. Bibliografía

[1] Página oficial de la artista Marina Anaya: http://www.marinaanaya.com/ , última visita 29/12/2012.

[2] Página web de la impresora Replicator, de Makerbot Industries: http://store.makerbot.com/, última visita 01/12/2012.

[3] Página web de la comunidad Thingiverse: http://www.thingiverse.com/, última visita 15/11/2012.

[4] Página web con toda la información sobre el diseño Orugator que incluye los códigos para la impresión del mismo: http://www.thingiverse.com/thing:7752, última visita 23/11/2012.

[5] Página web de Arduino: http://www.arduino.cc/, última visita 1/12/2012.

[6] Página web con toda la información necesaria para la programación con Arduino: http://arduino.cc/en/Reference/HomePage, última visita 20/11/2012.

[7] Página web de la casa Futaba: http://www.futaba-rc.com/, última visita 25/1/2012.

[8] Página web del profesor Juan Gonzalez Gómez con información sobre el funcionamiento del servomotor Futaba S30003: http://www.iearobotics.com/proyectos/cuadernos/ct3/ct3.html, última visita 20/11/2012.

[9] Electrónica digital en la práctica, Rafael Reina Acedo, Ra-Ma, 2010.

Anexos

A.1. Arduino

A.1.1. Especificaciones Arduino

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

A.1.2. Código Arduino

```
/*----------------------------Proyecto fin de carrera----------------------*/

#include <Servo.h>

#include <EEPROM.h>

const int analogInPin1 = A0; // Señal analogica del boton Activar

const int analogInPin2 = A1; // Señal analogica del boton Enganchar

const int Encender =  13;    // Puerto del micro configurado para el led

int sensorValue1 = 0;        // Lee el valor del boton ACTIVAR (0V ó 5V)

int sensorValue2 = 0;        // Lee el valor del boton ENGANCHAR (0V ó 5V)

int activar = 0;

int Colocar_munecos = 0; //esta variable sirve para que si el pin A1 está

                  // a nivel alto y esta variable vale 0 los servos

                  // giran para colocar los muñeco ysi A1 está a nivel

                  //alto y la variable a 1 vuelvo a poner los

                  //servos en su sitio paravolver a la posicion

Servo myservo;          // variable para controlar el servo

int pos = 0;           // variable para almacenar el valor la posición de los servos.

int pos_brazo = 0;      // variable para almacinar la posición de del brazo del muñeco

int recorrido = 0;      // sentido de giro(de 0 a 180:1)(de 180 a 0:2)

int inicio = 0;

int addr0 = 0;          //Direccion 0 de la memoria EEPROM

int addr1 = 1;          //Dirección 1 de la memoria EEPROM

int primera_vez=1;      //variable inicializar las variables.

int brazo_abajo;        // si brazo_abajo = 1 está bajando,

                  //por defecto se considerará subiendo

int estado_brazo = 1;    // variable para indicar las posiciones del brazo:

                  //si el brazo va desde 0 a 10, inicio a 10,

                  //10 a 0 ó inicio 0

int brazo_inicio;       // posición inicial del brazo
```

```
int brazo_final;        // posición final brazzo
int brazo_desactivado = 0;// si vale 1 el brazo está parado en carrera
int contador = 0;
int led_encendido = 0;     // si vale 1 el led está encendido
int tiempo_encendido = 18; // tiempo que el led está encendido
int pos_pajaro = 113;      //Variable para almacenar el movimiento del pájaro
int movimiento = 2;        // para que por cada dos movimientos del pajaro haya
                           // uno del brazo
int par = 0;



void setup()
{
  // comunicación inicializada a  9600 bps:
  Serial.begin(9600);
  myservo.attach(9);  // asignamos el servo del pin 9 (personajes principales)
  pinMode(13, OUTPUT); // Asignación del led al pin 13
}



void loop()
{
  digitalWrite(13, LOW);    // ponemos el led a cero porque alguna
                            //vez se queda encendido tras el movimiento del brazo
  if (primera_vez ==1){     //Si encendemos la plataforma se inicializan las variables
                            //a su último valor
    inicio = EEPROM.read(addr0);    //carga variable 'pos',
                                    //última posición de los personajes principales
    recorrido = EEPROM.read(addr1); //carga la variable recorrido para ver si se está
```

```
                    // en la fase 0,1,2,3
  primera_vez = 0;
 }
 sensorValue2 = analogRead(analogInPin2);// se lee la pulsación del botón Enganchar
 if ((sensorValue2 >1000) && (Colocar_munecos == 0))//Si se ha pulsado el botón Enganchar
                                    //por primera vez
 {
   sensorValue2 = 0;
   Colocar_munecos = 1;
   pos = 180;
   myservo.attach(9);  // asignamos el servo del pin 9 (personajes principales)
   myservo.write(pos);
   delay(200);     // espero 200 ms para que no entre en el if de abajo
 }
 sensorValue2 = analogRead(analogInPin2); // se lee la pulsación del botón Enganchar
 if ((sensorValue2 >1000) && (Colocar_munecos == 1))//Si se ha pulsado el botón Enganchar
                                    //por segunda vez
 {
   sensorValue2 = 0;
   Colocar_munecos = 0;
   pos = inicio;
   myservo.attach(9);  // asignamos el servo del pin 9 (personajes principales)r ;
   myservo.write(pos);
   delay(200);   // espero 200 ms para que no entre en el if de arriba
}


 sensorValue1 = analogRead(analogInPin1); // se lee la pulsación del botón Activar
 if (sensorValue1 >1000)//Si se ha pùlsado el botón Activar
 {
```

```
  /*----------------------------------recorrido==0-----------------------------------------------*/
  /*-----------------------------Recorrido de 0 a 180-----------------------------------------*/
   if ((recorrido == 0) && (brazo_desactivado == 0)){
    for(pos = 0; pos < 180; pos += 1)  // va desde la posición 0 a la 180 con incrementos de 1
    {
      myservo.attach(9);              // asignamos el servo del pin 9 (personajes principales)r
      myservo.write(pos);             // dile al servo que vaya a la poisición 'pos'
      delay(0);                       // espera 0 ms tras alcanzar la posición 'pos'
      EEPROM.write(addr0, pos+1);     //guardo la posición actual de los personajes principales
      EEPROM.write(addr1, 1);         // guardamos el valor de la variable recorrido a la que debe
ir
                                //en caso de que se deja de pulsar el botón Activar
      recorrido = 2;
      sensorValue1 = analogRead(analogInPin1);
      if (sensorValue1 < 1000) { // Pulsación del boton Desactivar
       inicio = pos;
       recorrido = 1;
       break;
      }
    }
   }
   /*----------------------------------recorrido==1-----------------------------------------------*/
   /*----------------------------------Recorrido desde inicio a 180-----------------------------*/
   if  ((recorrido == 1) && (brazo_desactivado == 0)){
    //Serial.print("2o (desde inicio a 180)\n");
    for(pos = inicio; pos < 180; pos += 1)     // va desde la posición actual hasta 180
    {
      myservo.attach(9);        // asignamos el servo del pin 9 (personajes principales)
      myservo.write(pos);       // dile al servo que vaya a la poisición 'pos'
```

```
    delay(0);              // espera 0 ms tras alcanzar la posición 'pos'
    EEPROM.write(addr0, pos+1);
    EEPROM.write(addr1, 1);
    recorrido = 2;
    sensorValue1 = analogRead(analogInPin1);
    if (sensorValue1 < 1000) {// Si se deja de pulsar el botón Activar
      inicio = pos;
      recorrido = 1;
      break;
    }
  }
}
/*---------------------------------------recorrido==2---------------------------------------*/
/*-------------------------------------Recorrido de 180 a 0-------------------------------------*/
if (recorrido == 2){
  if (brazo_desactivado == 0){       // si no se está en la fase encuentro
  for(pos = 180; pos>=0; pos-=1)     // va desde 180 a 0
  {
    myservo.attach(9);              // asignamos el servo del pin 9 (personajes principales);
    myservo.write(pos);              // dile al servo que vaya a la poisición 'pos'
    delay(0);                       // espera 0 ms tras alcanzar la posición 'pos'
    EEPROM.write(addr0, pos-1);
    EEPROM.write(addr1, 3);
    recorrido = 0;
    sensorValue1 = analogRead(analogInPin1);
    if (sensorValue1 < 1000) {    // Si se deja de pulsar el botón Activar
      inicio = pos;
      recorrido = 3;
      break;
```

```
    }
  }
  }
  if (sensorValue1 > 1000){
   if ( brazo_desactivado == 1 )  //si se está en la fase encuentro
    {brazo_desactivado = 0;
     recorrido = 0; }
   myservo.attach(8);                    // asignamos el servo del pin 8 (pajaro)
   if (estado_brazo == 1 || estado_brazo == 2){  //si el estado del brazo está entre 1 y 2
    switch (estado_brazo){
     case 1:
      brazo_inicio = 0;
      brazo_final = 10;
      break;
     case 2:
      brazo_inicio = pos_brazo;
      brazo_final = 10;
      break;
    }
    contador = 0; // contador para controlar el tiempo encendido
    tiempo_encendido = 0;
    for(pos_brazo = brazo_inicio; pos_brazo < brazo_final;pos_brazo += 1)//el brazo va desde
0 a 10
       {
        if (contador == tiempo_encendido){ // si el contador ha llegado al tiempo
                                  //en el que el led tiene que estar encedido.
          contador = 0; //reseteo del contador para empezar de nuevo
          switch(led_encendido){
          case 0://Si el led está apagado lo encendemos
```

```
  led_encendido = 1;
  digitalWrite(13, HIGH);  // Encendemos el led
 case 1://Si el led está encendido lo apagamos
  led_encendido = 0;
  digitalWrite(13, LOW);    // Apaga el led
  if (tiempo_encendido > 3)// mientras el tiempo de encendido sea mayor de 3,
                    //el tiempo de encendido se
                    //reduce de 3 en 3 para cambiar la frecuencia de parpadeo
   {tiempo_encendido = tiempo_encendido - 3;}
  else //si el tiempo de encedido es menor o igual de 3 el led se queda encendido.
  {tiempo_encendido = 1;}
  break;
 }
}
par = movimiento % 2; //calculamos el resto de la división
myservo.attach(7); //asignamos el servo del pin 8 (brazo)
myservo.write(pos_brazo);
if (par == 0 )
{myservo.attach(8);
myservo.write(pos_brazo);//asignamos el servo del pin 7 (pájaro)
contador++;}
delay(0);             // esperamos 0 ms
EEPROM.write(addr0, pos+1);
EEPROM.write(addr1, 3);
estado_brazo = 3;
sensorValue1 = analogRead(analogInPin1);
if (sensorValue1 <1000) { // Pulsación del botón Activar
  brazo_desactivado = 1;
  recorrido = 2; // recorrido vale 2 para que entre en este estado
```

```
    estado_brazo = 2;
    break;
  }
  if(pos_brazo == 114){// En la posición se produce el rebote del brazo
    digitalWrite(13, HIGH);  // Encendemos el led;
    for (pos_brazo= 113; pos_brazo>=95; pos_brazo -=1){
      myservo.attach(7); // asignamos el servo del pin 7 (pajaro)
      myservo.write(pos_pajaro);
      pos_pajaro = pos_pajaro + 1;
      par = movimiento % 2; //calculamos el resto de la división
       if (par == 0 )
        {myservo.attach(8);// asignamos el servo del pin 8 (brazo)
        myservo.write(pos_brazo);}
      movimiento = movimiento + 1;
      delay(0);
      EEPROM.write(addr0, pos+1);
      EEPROM.write(addr1, 3);
      estado_brazo = 3;
      sensorValue1 = analogRead(analogInPin1);
      if (sensorValue1 < 1000) { // Pulsación del botón Desactivar
        digitalWrite(13,LOW); // se apaga el led
        brazo_desactivado = 1;
        recorrido = 2; // recorrido vale 2 para que entre en este estado
        estado_brazo = 2;
        break;
       }
    }
    for (pos_brazo= 96; pos_brazo<114; pos_brazo +=1){ //el brazo va desde 96 a 114
      myservo.attach(7);//asignamos el servo del pin 7 (pajaro)
```

```
            myservo.write(pos_pajaro);
            pos_pajaro = pos_pajaro - 1;
            par = movimiento % 2; //calculamos el resto de la división
            if (par == 0 )
             {myservo.attach(8);
             myservo.write(pos_brazo);}
            movimiento = movimiento + 1;
            delay(0);
            EEPROM.write(addr0, pos+1);
            EEPROM.write(addr1, 3);
            estado_brazo = 3;
            sensorValue1 = analogRead(analogInPin1);
            if (sensorValue1 < 1000) { // Pulsación del botón Desactivar
             digitalWrite(13,LOW);
              brazo_desactivado = 1;
              recorrido = 2; // recorrido vale 2 para que entre en este estado
              estado_brazo = 2;
              break;
            }
          }
        }
      }
    }
  if (estado_brazo == 3 || estado_brazo == 4){//si el estado del brazo está entre 3 y 4
    switch (estado_brazo){
    case 3:
     brazo_inicio = 10;
     brazo_final = 12;
     break;
```

```
case 4:
 brazo_inicio = pos_brazo;
 brazo_final = 12;
 break;
}
  contador = 0;//inicializamos de nuevo el valor del contador
  tiempo_encendido = 1; //inicializamos de nuevo el valor de la PWM
  for(pos_brazo = brazo_inicio; pos_brazo > brazo_final;pos_brazo -= 1)
  {
   if (contador == tiempo_encendido){
      contador = 0;
      switch(led_encendido){
      case 0://Si el led está apagado lo encendemos
        led_encendido = 1;
        digitalWrite(13, HIGH);  // Encendemos el led
        break;
      case 1://Si el led está encendido lo apagamos
        led_encendido = 0;
        digitalWrite(13, LOW);    // Apagamos el led
        tiempo_encendido = tiempo_encendido + 3;
        break;
      }
   }
   myservo.attach(7);
   myservo.write(pos_brazo);
   par = movimiento % 2; //calculamos el resto de la división
   if (par == 0 )
    { myservo.attach(8);
      myservo.write(pos_brazo);
```

```
            contador++;}
          movimiento = movimiento + 1;
          delay(0);
          EEPROM.write(addr0, pos+1);
          EEPROM.write(addr1, 3);
          estado_brazo = 1;
          sensorValue1 = analogRead(analogInPin1);
          if (sensorValue1 < 1000) { // Pulsación del botón descativar
           digitalWrite(13,LOW);
            recorrido = 2; // recorrido vale 2 para que entre en este estado
            brazo_desactivado = 1;
            estado_brazo = 4;
            break;
         }
       }
     }
  }
}
/*-------------------------------------recorrido=3-----------------------------------------*/
/*--------------------------------Recorrido de inicio a 0-----------------------------------*/
if (recorrido == 3){
 for(pos = inicio; pos>0; pos-=1)  // va desde la última posición hasta la posición 0
                          //con incrementos de 1
 {
  myservo.write(pos);            // dile al servo quye vaya a la posición 'pos'
  delay(0);                    // espera 0 ms
  EEPROM.write(addr0, pos-1);
  EEPROM.write(addr1, 3);
  recorrido = 0;
```

```
  sensorValue1 = analogRead(analogInPin1);
  if (sensorValue1 < 1000) {
    inicio = pos;
    recorrido = 3;
    break;
   }
}
if (sensorValue1 > 1000){
   myservo.attach(8); if ( brazo_desactivado == 1 )
   {brazo_desactivado = 0;
    recorrido = 0; }
    myservo.attach(8);
if (estado_brazo == 1 || estado_brazo == 2){//si el estado del brazo está entre 1 y 2
  switch (estado_brazo){
  case 1:
    brazo_inicio = 0;
    brazo_final = 10;
    break;
  case 2:
    brazo_inicio = pos_brazo;
    brazo_final = 10;
    break;
  }
 contador = 0;
 tiempo_encendido = 0;
 for(pos_brazo = brazo_inicio; pos_brazo < brazo_final;pos_brazo += 1)
    {
      if (contador == tiempo_encendido){
        contador = 0;
```

```
 switch(led_encendido){
 case 0://Si el led está apagado lo encendemos
  led_encendido = 1;
  digitalWrite(13, HIGH);  // Encendemos el led;
  break;
 case 1://Si el led está encendido lo apagamos
  led_encendido = 0;
  digitalWrite(13, LOW);    // set the LED off
  if (tiempo_encendido > 3)
   {tiempo_encendido = tiempo_encendido - 3;}
  else
   {tiempo_encendido = 1;}
  break;
 }
}
par = movimiento % 2; //calculamos el resto de la división
myservo.attach(7);
myservo.write(pos_brazo);
if (par == 0 )
 {myservo.attach(8);
  myservo.write(pos_brazo);
  contador++;}
movimiento = movimiento + 1;
delay(0);
EEPROM.write(addr0, pos+1);
EEPROM.write(addr1, 3);
estado_brazo = 3;
sensorValue1 = analogRead(analogInPin1);
if (sensorValue1 < 1000) {
```

```
 digitalWrite(13,LOW);
  brazo_desactivado = 1;
  recorrido = 2; // recorrido vale 2 para que entre en este estado
  estado_brazo = 2;
  break;
 }
if(pos_brazo == 114){
 digitalWrite(13, HIGH);  // Encendemos el led
 for (pos_brazo= 113; pos_brazo>=95; pos_brazo -=1){//2
   myservo.attach(7);
   myservo.write(pos_pajaro);
   pos_pajaro = pos_pajaro + 1;
   par = movimiento % 2; //calculamos el resto de la división
   if (par == 0 )
    {myservo.attach(8);
     myservo.write(pos_brazo);}
   movimiento = movimiento + 1;
   delay(0);
   EEPROM.write(addr0, pos+1);
   EEPROM.write(addr1, 3);
   estado_brazo = 3;
   sensorValue1 = analogRead(analogInPin1);
   if (sensorValue1 < 1000) {
    digitalWrite(13,LOW);
     brazo_desactivado = 1;
     recorrido = 2; // recorrido vale 2 para que entre en este estado
     estado_brazo = 2;
     break;
    }
```

```
      }
     for (pos_brazo= 96; pos_brazo<114; pos_brazo +=1){
       myservo.attach(7);
       myservo.write(pos_pajaro);
       pos_pajaro = pos_pajaro - 1;
       par = movimiento % 2; //calculamos el resto de la división
       if (par == 0 )
        {myservo.attach(8);
         myservo.write(pos_brazo);}
       movimiento = movimiento + 1;
       delay(0);
       EEPROM.write(addr0, pos+1);
       EEPROM.write(addr1, 3);
       estado_brazo = 3;
       sensorValue1 = analogRead(analogInPin1);
       if (sensorValue1 < 1000) { // Pulso el boton DESACTIVAR
       digitalWrite(13,LOW);
         brazo_desactivado = 1;
         recorrido = 2; // recorrido vale 2 para que entre en este estado
         estado_brazo = 2;
         break;
      }
     }
   }
  }
}
if (estado_brazo == 3 || estado_brazo == 4){//si el estado del brazo está entre 3 y 4
  switch (estado_brazo){
  case 3:
```

```
  brazo_inicio = 10;
  brazo_final = 0;
  break;
case 4:
  brazo_inicio = pos_brazo;
  brazo_final = 0;
  break;
}
   for(pos_brazo = brazo_inicio; pos_brazo > brazo_final;pos_brazo -= 1)
   {
     if (contador == tiempo_encendido){
        contador = 0;
      switch(led_encendido){
      case 0://Si el led está apagado lo encendemos
        led_encendido = 1;
        digitalWrite(13, HIGH);  // Encendemos el led;
        break;
      case 1://Si el led está encendido lo apagamos
        led_encendido = 0;
        digitalWrite(13, LOW);    // Apagamos el led
        tiempo_encendido = tiempo_encendido + 3;
        break;
       }
      }
     myservo.attach(7);
     myservo.write(pos_brazo);
     par = movimiento % 2; //calculamos el resto de la división
     if (par == 0 )
     {myservo.attach(8);
```

```
            myservo.write(pos_brazo);
            contador++;}
            movimiento = movimiento + 1;
            delay(0);
            EEPROM.write(addr0, pos+1);
            EEPROM.write(addr1, 3);
            estado_brazo = 1;
            sensorValue1 = analogRead(analogInPin1);
            if (sensorValue1 < 1000) { // Pulso el boton DESACTIVAR
              digitalWrite(13,LOW);
               recorrido = 2; // recorrido vale 2 para que entre en este estado
               brazo_desactivado = 1;
               estado_brazo = 4;
               break;
           }
         }
       }
     }
   }
 }
```

}A.2. Código Piezas 3D

A.2.1. Rueda motriz

```
// PARAMETRIC GEAR WHEELS
// Parameters
//
// re= external radius
// ri= internal radius
// n=number of gears the wheel has
// d=parameter to control the separation between gears
// h=height of the wheel
module gear_wheel(re,ri,n,d)
{
// Angle difference between middle points of gears
```

```
am=360/n;
// Angle difference between middle and ending points of gears
ad=360/((d+1)*n*4);
// Drawing gears
for (i=[0:n-1])
linear_extrude(height = h, center = true, convexity = 10, twist = 0)
polygon(
points=[
[0,0],
[ri*cos(am*i-2*ad),ri*sin(am*i-2*ad)],
[re*cos(am*i-ad),re*sin(am*i-ad)],
[re*cos(am*i),re*sin(am*i)],
[re*cos(am*i+ad),re*sin(am*i+ad)],
[ri*cos(am*i+2*ad),ri*sin(am*i+2*ad)]
],
paths=[[0,1,2,3,4,5]]);
// Drawing internal circle
cylinder(r=ri,h=h,center=true,$fn=100);
}
// example of use
difference()
{
gear_wheel(re=30,ri=25,n=6,d=3,h=8);
union(){
//-- Carved cicle for the Futaba plate
translate([0,0,10]) cylinder(r=21.5/2, center=true,h=20,$fn=100);
//-- Carved circle por the Futaba shart
cylinder(center=true, h=30, r=4.2,$fn=100);
}
}
```

A.2.2. Rueda arrastrada

```
// PARAMETRIC GEAR WHEELS
// Parameters
//
// re= external radius
// ri= internal radius
// n=number of gears the wheel has
// d=parameter to control the separation between gears
// h=height of the wheel
module gear_wheel(re,ri,n,d)
{
// Angle difference between middle points of gears
am=360/n;
// Angle difference between middle and ending points of gears
ad=360/((d+1)*n*4);
// Drawing gears
for (i=[0:n-1])
linear_extrude(height = h, center = true, convexity = 10, twist = 0)
polygon(
points=[
[0,0],
```

```
[ri*cos(am*i-2*ad),ri*sin(am*i-2*ad)],
[re*cos(am*i-ad),re*sin(am*i-ad)],
[re*cos(am*i),re*sin(am*i)],
[re*cos(am*i+ad),re*sin(am*i+ad)],
[ri*cos(am*i+2*ad),ri*sin(am*i+2*ad)]
],
paths=[[0,1,2,3,4,5]]);
// Drawing internal circle
cylinder(r=ri,h=h,center=true,$fn=100);
}
// example of use
difference()
{
gear_wheel(re=30,ri=25,n=6,d=3,h=8);
//-- Carved circle por the screw shart
cylinder(center=true, h=30, r=4.2,$fn=100);
}
```

A.2.3. Soporte servomotor

```
difference(){
translate([0,0,-15])
cube([30,75,30],true);
translate([-34,-3,-11])
rotate([0,0,-45])
cube([35,5,12]);
translate([13,-24.5,-11])
rotate([0,0,45])
cube([35,5,12]);
translate([0,9,-13.2])
cube([20.5,44,26.5],true);
translate([0,30,-21.2])
rotate([90],0,0)
cube([10,10,28],true);
translate([0,-28,6.5])
cube([33,17,35],center=true);
}
difference(){
translate([0,-35,-15])
cube([60,5,30],true);
translate([24,-35,10])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([-24,-35,10])
rotate([90,0,0])
cylinder(r=1.25,h=50, center=true, $fn =20);
translate([23,-35,-22.5])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([-23,-35,-7.5])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([23,-35,-7.5])
```

```
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([-23,-35,-22.5])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
}
```

A.2.4. Soporte rueda arrastrada

```
difference(){
rotate(90,0,0)
translate([20,0,7])
cylinder(r=5,h=27,center=false,$fn=50);
rotate(90,0,0)
translate([20,0,20])
cylinder(r=1,h=15,center=false,$fn=50);
}
cube([30,75,15],true);
difference(){
translate([0,-35,3.75])
cube([60,5,22.5],center=true);
translate([23,-35,10])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([-23,-35,10])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([23,-35,-2.5])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
translate([-23,-35,-2.5])
rotate([90,0,0])
cylinder(r=1.5,h=50, center=true, $fn =20);
}
```

A.2.5. Eslabones de la cadena

```
// ******************** PARAMETRIZED LINK ************************
//
// (c) Daniel GÃ³mez LendÃnez & Olalla Bravo Conde
// www.thingiverse.com/dannynoc www.thingiverse.com/olalla
// GPL Licence Non commercial
// ****** PARAMETERS *************
//
// dtd: distance between axis
// d: drills diameter
// lh: length of the teeth hole
// wh: width of the teeth hole
// e: it controls the total width of the link
// tl: tolerance for the own hole of the link so it engages well with the following link
// th: tolerance of the teeth hole
// w: width of the whole link
// ********* MAIN BODY **********
//
```

```
// It builds the basic rectangle with the hole for the tooth of the gear wheel
module main_body(dtd,lh,wh,a,w)
{
difference()
{
union()
{
cube([dtd,a,w],center=true);
rotate (a=[90,0,0]) translate ([-0.5*dtd,0,0]) cylinder (r=w/2, h=a,$fn=100, center=true);
rotate (a=[90,0,0]) translate ([0.5*dtd,0,0]) cylinder (r=w/2, h=a,$fn=100, center=true);
}
cube ([wh+w,lh+w,2*w], center=true);
}
}
// ******* DEFINITION OF THE LINK ***********
module link(dtd,d,lh,wh,e,tl,th)
{
// ********* EXTRA PARAMETERS NEEDED TO DRAW THE LINK
//Hole without tolerances (it refers to the indentation needed so that two consecutive
links engage
u=lh+2*w;
// Hole with tolerances
ut=u+tl;
//Teeth hole with tolerances
lh=lh+2*th;
wh=wh+2*th;
// Total width of the piece
a=2*lh+4*e+tl+w;
difference()
{
union()
{
main_body(dtd=dtd,w=w,a=a,lh=lh,wh=wh);
// - - - - - - - - - - - - - - - - - - - -
// Making the round parts of the teeth hole
// Lateral cylinders
rotate (a=[90,0,0]) translate ([(w+wh)/2,0,0]) cylinder (r=w/2, h=lh+w,$fn=100,
center=true);
rotate (a=[90,0,0]) translate ([-(w+wh)/2,0,0]) cylinder (r=w/2, h=lh+w,$fn=100,
center=true);
// Front cylinders
rotate (a=[0,90,0]) translate([0,-(w+lh)/2,0])
cylinder(r=w/2,h=wh+w,$fn=100,center=true);
rotate (a=[0,90,0]) translate([0,(w+lh)/2,0])
cylinder(r=w/2,h=wh+w,$fn=100,center=true);
// Giving body under the round part
translate([-(wh/2+w/4),0,-w/4]) cube([w/2,a,w/2],center=true);
translate([(wh/2+w/4),0,-w/4]) cube([w/2,a,w/2],center=true);
translate([0,(lh/2+w/4),-w/4]) cube([wh+w,w/2,w/2],center=true);
translate([0,-(lh/2+w/4),-w/4]) cube([wh+w,w/2,w/2],center=true);
}
union()
{
```

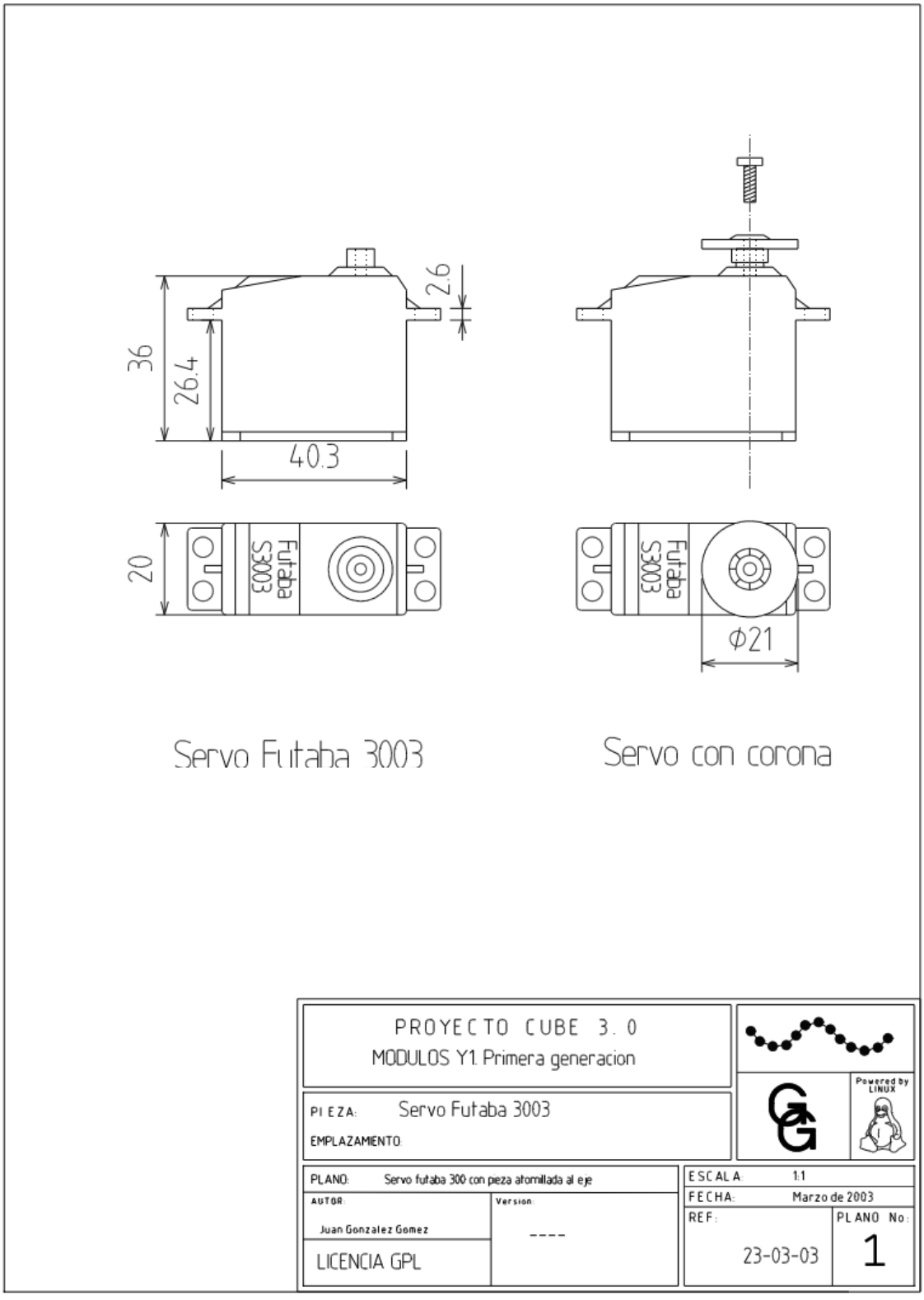
```
// - - - - - - - - - - - - - - - - - - - -
// Perforating the main body
// Lateral holes -- in order to make front indentation
translate ([-(0.5*dtd),(a+u)/4+a/2,0]) cube ([w+4*tl,(a-u)/2+a,2*w], center=true);
translate ([-(0.5*dtd),-(a+u)/4-a/2,0]) cube ([w+4*tl,(a-u)/2+a,2*w], center=true);
// Back hole
translate ([0.5*dtd,0,0]) cube ([w+2*tl,ut,2*w], center=true);
// Drills for axis
rotate (a=[90,0,0]) translate ([-0.5*dtd,0,0]) cylinder (r=d/2+tl/2, h=4*lh,$fn=100,
center=true);
rotate (a=[90,0,0]) translate ([0.5*dtd,0,0]) cylinder (r=d/2+tl/2, h=4*lh,$fn=100,
center=true);
//Difference to acotate the link laterally
translate([0,a,0]) cube([dtd,a,w],center=true);
translate([0,-a,0]) cube([dtd,a,w],center=true);
//Difference to acotate the link in the upper and lower part
translate([0,0,w/2+a/2]) cube([dtd+2*w,a,a],center=true);
translate([0,0,-w/2-a/2]) cube([dtd+2*w,a,a],center=true);
//Diference to acotate the link in the front and bottom part
translate([dtd/2+w,0,0]) cube([w,a,a],center=true);
translate([-(dtd/2+w),0,0]) cube([w,a,a],center=true);
}
}
}
//for (i=[0:2])
//{
//translate([i*19.29,0,0]) link(dtd=19.29,d=3,lh=8,wh=3.9,e=2,tl=1,w=6,th=0.3);
//}
//main_body(dtd=16,w=6,a=10,lh=2,wh=3);
```

A.2.6. Enganches

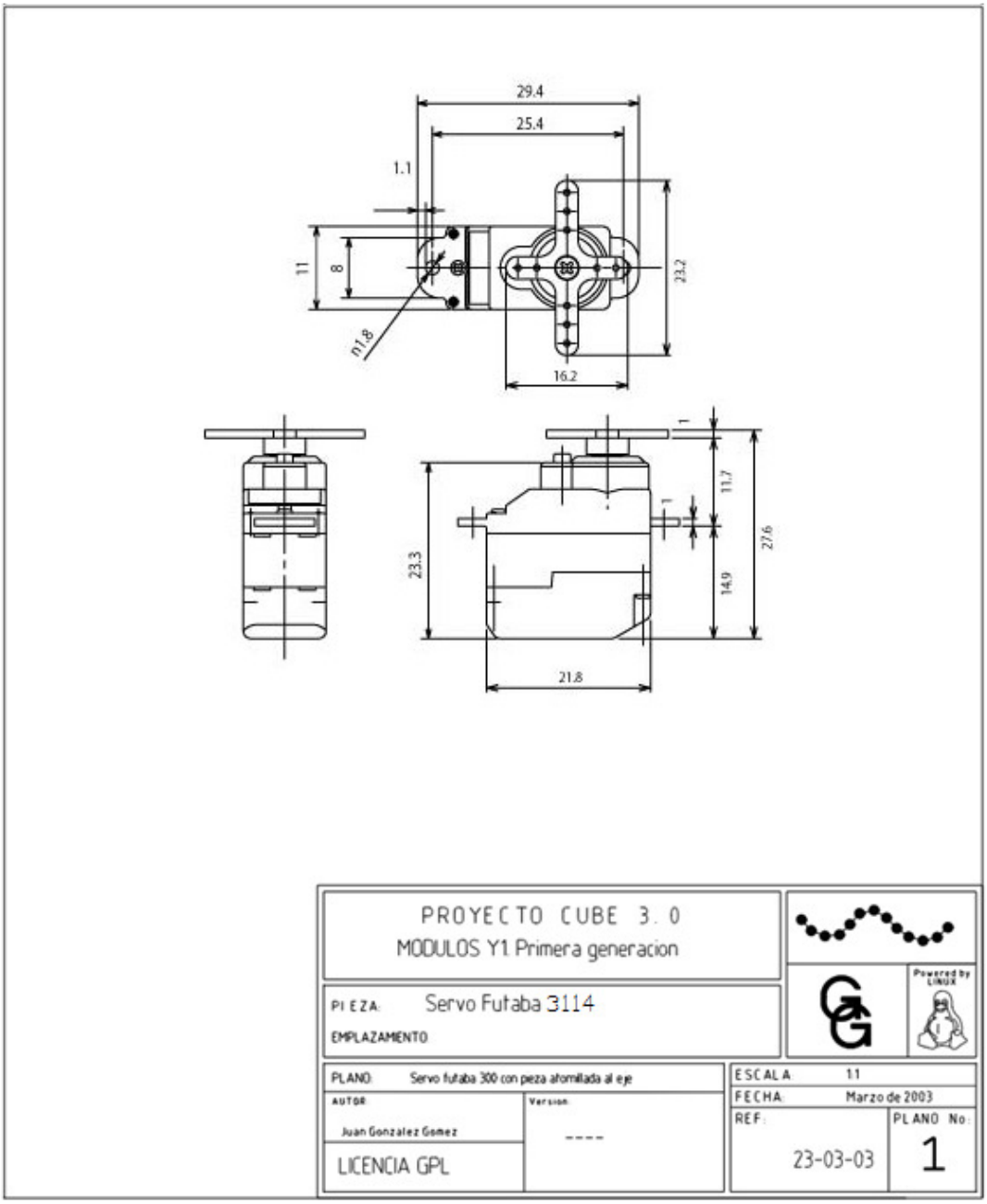
```
difference(){

cylinder(r=8.5,h=20, center=false, $fn =20);
translate([-2.5,-2.5,10])
cube([5, 5, 20], center = false);
};
translate([-12.5,-15,0])
cube([25, 30, 2], center = false);
```

A.3. Motor Futaba

A.3.1. Plano Futaba S3003



A.3.2. Plano Futaba S3114



A.3.3. Especificaciones Futaba S3003

Velocidad: 0.23 seg/60 grados (260 grados/seg)

Par de salida mínimo: 3.2 Kg-cm (0.314 N.m)

Dimensiones: 40.4 x 19.8 x 36 mm

Peso: 37.2 gr

Frec. PWM: 50Hz (20ms)

Rango de giro: 180 grados

A.3.4. Especificaciones Futaba S3114

Velocidad: 0.10 seg/60 grados (260 grados/seg)

Par de salida mínimo: 1.5 Kg-cm

Dimensiones: 22 x 11 x 20 mm

Peso: 7.8 gr

Frec. PWM: 50Hz (20ms)

Rango de giro: 180 grados