PROYECTO CONCEPTUAL

GENERADOR DE NOTAS ALEATORIO

→ Sarah del Carmen González Molina (B21AP)
→ Marina España Florido (B21AP)
→ Susana Soler Cazalla (B22AP)
→ Francisco Javier Comino Sáez (GS21ME)
→ Fernando Chito Solís (GS21ME)

INDICE DE CONTENIDOS

- INTRODUCCIÓN…….3
- DESCRIPCIÓN GENERAL DEL PROYECTO……...4
- PROYECTO. PRIMERAS IDEAS Y CONCEPCIONES…….5
- PROYECTO. PRIMERA Y SEGUNDA FASE: ESCRITURA MANUAL Y DETECCIÓN IR ……...6
- PRINCIPIOS DE FUNCIONAMIENTO. PROGRAMACIÓN……...7
- PRINCIPIOS DE FUNCIONAMIENTO. AJUSTES PARA CALIBRACIÓN…….9
- PRINCIPIOS DE FUNCIONAMIENTO. MONTAJE DE LOS COMPONENTES…….11
- PROYECTO. TERCERA FASE: MODULO DE VOZ…….12
- BIBLIOGRAFÍA Y CONTENIDOS……..13

INTRODUCCIÓN

Enmarcado dentro del módulo “Mantenimiento Electrónico de Equipos Industriales” fue propuesto al alumnado la realización de un proyecto conjunto entre el grupo de alumnos de Mantenimiento Electrónico en conjunto con el alumnado perteneciente al grupo de Arte. El objetivo de este trabajo colaborativo era poder llevar a cabo una combinación entre ambos ámbitos a primera vista tan lejanos, el arte y la electrónica, adaptándose principalmente el ámbito de la electrónica (de una naturaleza técnica y objetiva) a las posibles demandas y/o requeridas por el arte (de una naturaleza más subjetiva).

Tras varias sesiones donde ambas partes se encontraron, se realizó una tormenta de ideas proponiendo distintos proyectos conceptuales y evaluando su viabilidad por ambas partes, dando par resultado el siguiente proyecto.

DESCRIPCIÓN GENERAL DEL PROYECTO

Durante la realización de las reuniones y las tormentas de ideas fue propuesto el presente proyecto, un “profesor generador de notas aleatorio”, alguna vez vivido y sopesado por toda persona que se haya encontrada en el ambito educativo en algún momento de su vida o que haya acatado el rol de “alumno”. Cabe mencionar que este proyecto cuenta con un trasfondo irónico pero a su vez mordaz sobre los hipotéticos métodos evaluativos de cierto sector del profesorado. Dado que el fin del proyecto es su exhibición en una feria cuyo principal sector de público está relacionado con el ámbito educativo, Malakabot 2019, pareció adecuado llevar a cabo este proyecto para la misma, a la espera tras la feria de la aceptación del mismo.

Referente al proyecto se marcaron varios aspectos importantes:

- Se llevaría a cabo la modelación de un títere, con aspecto físico de señor mayor, perfil típico de profesor cuya jubilación se encuentra cerca y descuida ciertos aspectos de su profesión como los métodos evaluativos.
- El títere debería tener capacidad para escribir números del 0 al 9, valores típicos de cualquier evaluación. Nótese que se descartó la posibilidad de escribir la nota “10”, como reflejo de aquellos profesores que defienden a capa y espada que nunca ponen esta nota, ya que implicaría el conocimiento total de la materia, por encima incluso de su nivel.
- El títere debería tener capacidad de detectar cuando ha sido colocado frente a él documentos, a representación de los exámenes a evaluar.
- Si el documento no fuese retirado, esto era síntoma de la disconformidad relativa a la nota, por lo que sería generada otra nota.
- Que el títere tuviese la capacidad de hablar e comunicar la nota que ha sido puesto, así como algunas frases para interactuar con la persona frente a él.

PROYECTO. PRIMERAS IDEAS Y CONCEPCIONES

A la hora de evaluar la viabilidad del proyecto, se llevo a cabo una tarea de investigación sobre las posibilidades para realizar el presente proyecto de manera electrónica.

Relativo a la escritura de caracteres numéricos existe un proyecto conocido y difundido, de naturaleza libre, consistente en un robot que escribe la hora sobre un panel y va realizando cambios en la misma conforme la progresión de la misma. Este proyecto es conocido como “Plot Clock”

Dada la naturaleza libre de este proyecto, pudo estudiarse su funcionamiento.

Una vez estudiado el mismo, pudo partirse de de este proyecto para la realización del presente que será descrito, haciendo uso de varias partes y modificando ó incluyendo partes nuevas.

A la hora de sopesar la construcción del títere con las funciones electrónicas incorporadas, se decidió que éste siguiera el siguiente orden de construcción, así como de prioridades.

1) Dotar al títere de la capacidad de escribir los números
2) Dotar al títere de la capacidad de detectar los documentos a evaluar y a partir de la detección realizar a cabo la evaluación
3) Dotar al títere de la capacidad de hablar y reproducir sonidos, como comentarios hacia el alumnado o decir en voz alta la nota puesta.

Partiendo del proyecto base antes mencionado, “Plot Clock”, se decidió partir del mismo a la hora de seleccionar el entorno de desarrollo sobre el cual iba a realizarse el proyecto completo. Este entorno era “Arduino” conocido por el alumnado, idóneo para la realización del proyecto.

Tras sopesar la viabilidad de las siguientes partes adaptadas al entorno “Arduino” y sopesar la misma, decidió llevarse a cabo la realización de las mismas.

PROYECTO. PRIMERA Y SEGUNDA FASE: ESCRITURA MANUAL Y DETECCIÓN IR

Como fue mencionado con anterioridad, el proyecto iba a partir de otro proyecto de naturaleza libre implementado con anterioridad, el “PlotClock”

El “PlotClock” se trata de un pequeño robor que esribe lahora sobre una pizarra y que, tras cada minuto, borra la hora y la actualiza.

Este proyecto se compone de las siguientes elementos:

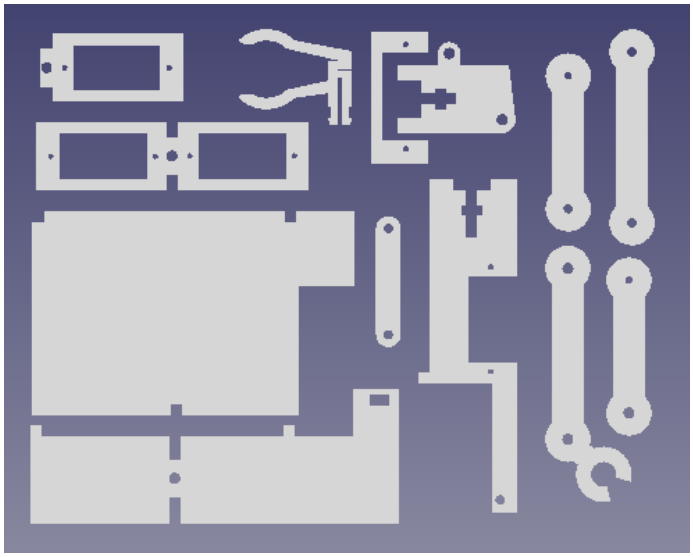
- 1 juego de piezas (ya puede ser piezas en madera MDF cortadas a laser, realizadas a mano, impresas en 3D, etc)
- 3 Servos Tower Pro SG90 y sus correspondientes coronas.
- Arduino Uno
- Sensor IR
- Led y resistencia de valor óhmico 330
- Cableado variado
- Tornillería variada

JUEGO DE PIEZAS

Dada la naturaleza libre del proyecto se decidió partir de éstas y realizar las posible adaptaciones que pudiesen ser necesarias.

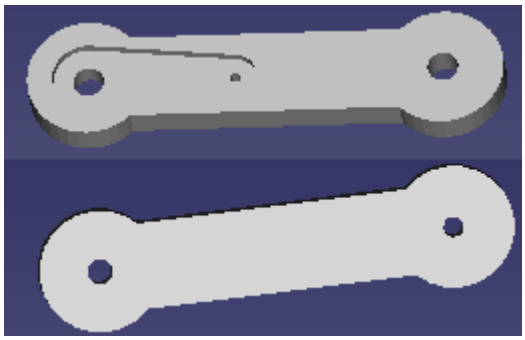
Ya que se disponía del acceso a la tecnología de impresión en 3D y la existencia de unas piezas base, decidió partirse de unas disponibles al público de forma abierta. Las piezas se encontraban en los siguientes repositorios:

- Plot Clock STL Files → juego de piezas general que incluía todas las posibles partes necesarias, así como otras que no iban a ser necesarias para nuestro proyecto. Incluye toda la estructura de soporte de los servomotores, así como los ejes que brindarán los desplazamientos del elemento de escritura.



→ Detalle de las piezas

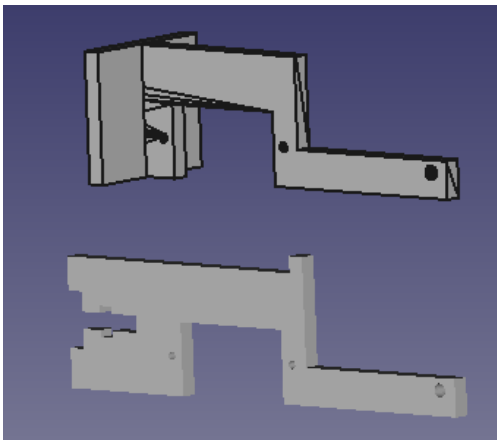
- Plotclock with more detailed instructions →este archivo cuenta con una pieza modificada respecto a la del archivo anterior. Este archivo incluye los ejes para los servomotores adaptada a la cora de los servos, implementando una mayor facilidad en el desplazamiento de esta.



→ Detalle de comparativa entre piezas

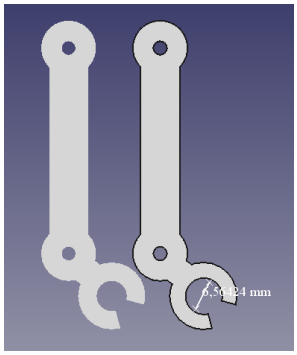
Posteriormente se llevó a cabo la adaptación de varias de las piezas a las necesidad específicas.

- ***Soporte para servo elevador***: en el diseño original, se juntan dos piezas mediante tornillería, pero se decidió modificar esta pieza para incluir una pequeña base en los laterales, menor que la pieza a anclar, para brindar de estabilidad y reducir el tamaño de las piezas que formarán parte de la futura mano del títere. Modificación mediante el software Freecad.



→ Detalle de adaptación de pieza para soporte se servo elevador

- ***Soporte de bolígrafo***: Se requería una pieza con mayor anchura para poder albergar el rotulador, por lo que se modificó mediante el software Freecad.



→ Detalle de pieza que soporta el rotulador.

PRINCIPIOS DE FUNCIONAMIENTO. PROGRAMACIÓN.

El código proporcionado por el proyecto contaba con los siguientes bloques definibles:

- ***Defines e inclusión de librerías***

En este primer bloque se encuadran las distintas librerías y parámetros que tendrán un valor constante dentro del proyecto, sirviéndose de esta definición para posteriormente poder variar su valor y que no afecte al resto del código.

- ***Void setup***

Bloque relativo al entorno "Arduino", donde se establece la definición y configuración de los distintos pines que van a ser ejecutados posteriormente.

- ***Void loop***

Bloque principal de la programación, el cual incluye las instrucciones principales a realizar por el microcontrolador, las cuales serán ejecutadas de manera repetida en bucle.

- ***void number***

Bloque el cual contiene la información y las instrucciones necesarias para realizar la escritura de un número, como pueden ser la posición bx y by donde va a iniciarse la escritura el número, los movimientos mecánicos los cuales componen la trayectoria, la selección del número que va a realizarse o la escala del mismo.

- ***void lift***

Bloque el cual contiene la información necesaria para realizar los movimiento de elevación y/o descenso del servo que cumple dicho propósito, en función del caso de elevación seleccionado.

- ***void bogen UZS && void bogen GZS***

Bloque que genera los desplazamientos diagonales y en forma de arco sincronizados de los servomotores, tanto de manera ascendente como descendente, necesarios para realizar la trayectoria de escritura en ciertos números

- ***void drawTo***

Bloque que genera los movimientos necesarios para llevar el punto de escritura a un punto concreto del plano X e Y para la escritura.

- ***Double return_angle***

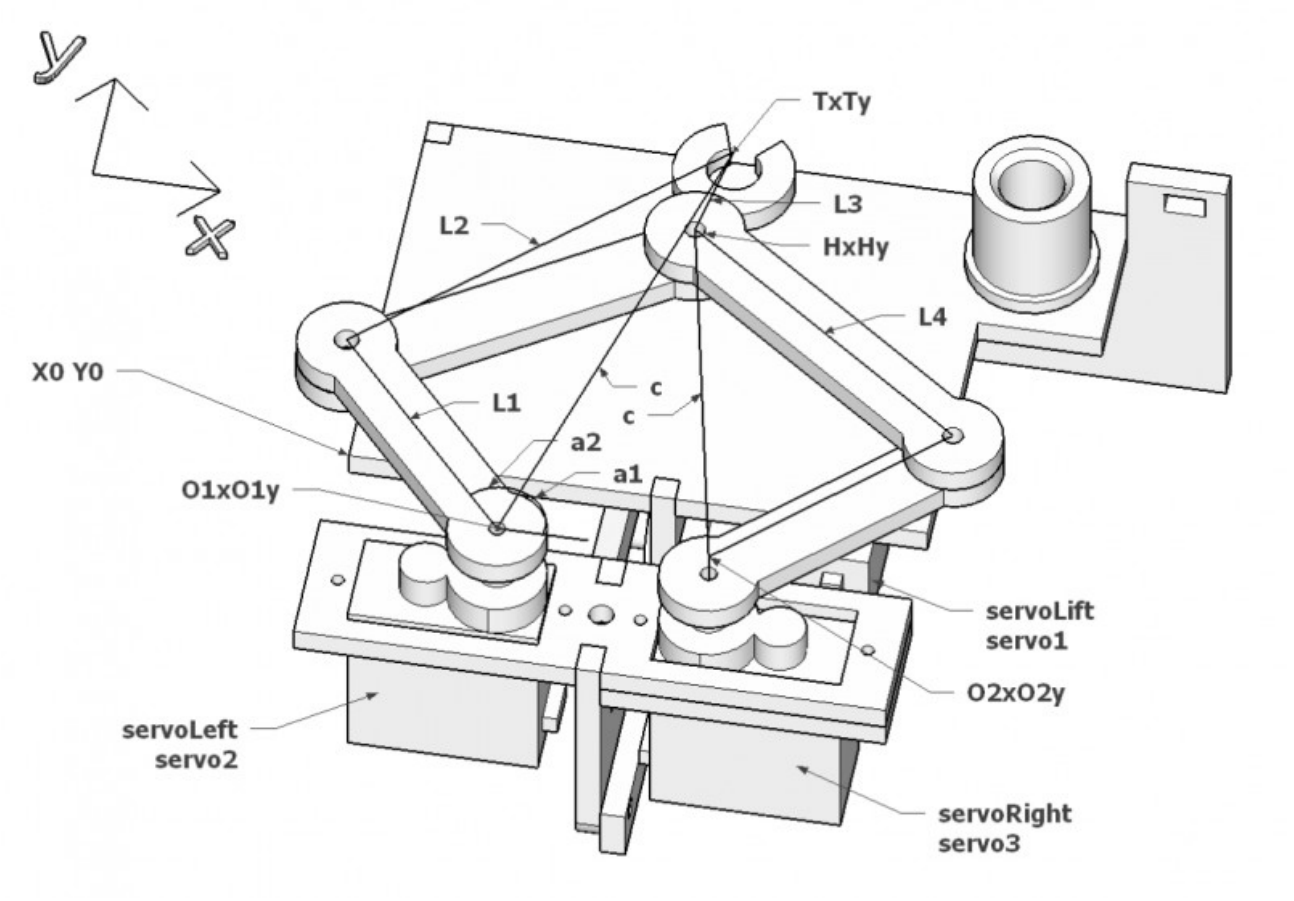
Función que realiza el teorema del coseno, necesaria para la generación de movimientos y el posicionamiento correcto.

- ***Void set_XY***

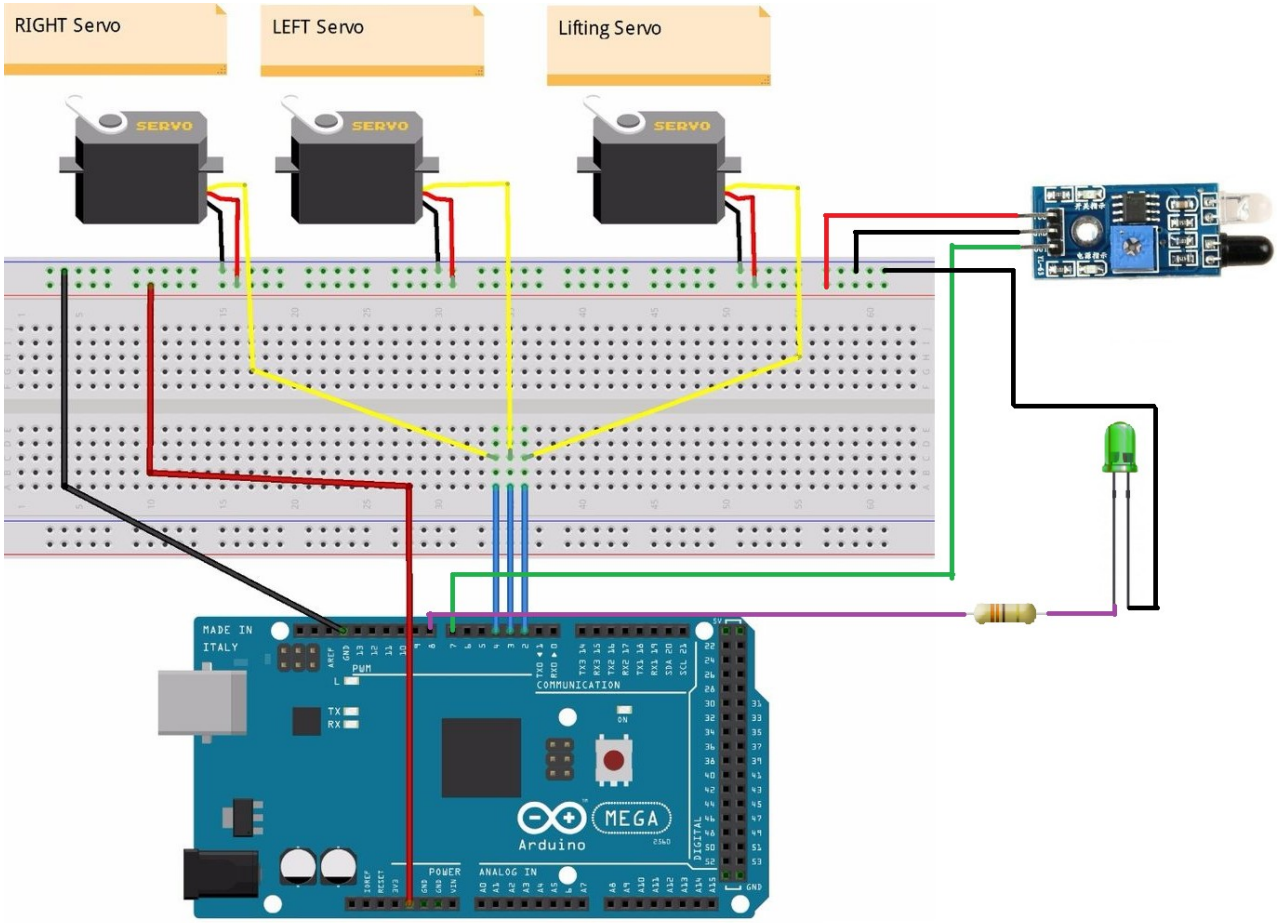
Bloque encargado de sincronizar y generar la triangulación correcta entre los servos.

PRINCIPIOS DE FUNCIONAMIENTO. AJUSTES PARA CALIBRACIÓN

Dentro de los bloques de programación mencionados con anterioridad, deben de tenerse en cuenta ciertos parámetros necesarios para la correcta calibración del dispositivo. Algunos de ellos están referidos a la parte mecánica. Para ello, se proporciona este dibujo aclaratorio donde se aprecian de manera gráfica los puntos exactos de ubicación de estas variables o con qué elementos o posiciones se corresponden respecto al bloque de variables a programar.



PRINCIPIOS DE FUNCIONAMIENTO. MONTAJE DE LOS COMPONENTES



→ Detalle de configuración del conexionado.

Nota: durante el montaje práctica fue utilizado un Arduino UNO en lugar de uno MEGA como se muestra en la figura, pero ambos modelos son compatibles mientras se respete el conexionado.

PROYECTO. TERCERA FASE: MODULO DE VOZ

Esta tercera fase será realizada por el alumno Francisco Javier Comino Sáez, como parte del proyecto a realizar durante el tercer trimestre, correspondiente con el módulo de Proyecto Integrado, para posteriormente ser integrado en el mismo proyecto.

BIBLIOGRAFÍA Y CONTENIDOS

- https://wiki.fablab-nuernberg.de/w/Ding:Plotclock
- https://www.luisllamas.es/detectar-obstaculos-con-sensor-infrarrojo-y-arduino/
- https://www.arduino.cc/reference/en/language/functions/random-numbers/random/

GENERADOR DE NOTAS ALEATORIO

AMPLIACIÓN

PROYECTO INTEGRADO

CFGS MANTENIMIENTO ELECTRÓNICO

→ Francisco Javier Comino Sáez

→ Curso 2018/2019

ÍNDICE DE CONTENIDOS

- INTRODUCCIÓN…….….3
- IDEAS CONCEPTUALES……....4
- CONCEPCIÓN. VIABILIDAD TÉCNICA …….….5
- PROYECTO. PRIMERA Y SEGUNDA FASE: ESCRITURA MANUAL Y DETECCIÓN IR…...6
  - JUEGO DE PIEZAS…….…….7
  - PRIMERA FASE: BLOQUE DE ESCRITURA MANUAL…….9
  - SEGUNDA FASE: BLOQUE DE DETECCIÓN DE FOLIO…….….12
- PROYECTO. TERCERA FASE: MÓDULO REPRODUCTOR MP3……..13
  - MODOS DE CONTROL……...……...16
  - COMUNICACIÓN ENTRADA/SALIDA → SPI…….…...21
  - REQUISITOS PREVIOS SD……...….23
- CONEXIONADO DE BLOQUES…….....24
- PRINCIPIOS DE FUNCIONAMIENTO. PROGRAMACIÓN…….…25
  - DIAGRAMA DE FLUJO DE BLOQUE PRINCIPAL…….25
  - PROGRAMA. DESCRIPCIÓN DE LOS BLOQUES……..26
  - DESCARGA DE LIBRERÍA……...….27
- ENSAMBLAJE…….…….28
- CONTENIDO WEB…….…..29
- ANEXO: CÓDIGO DE PROGRAMACIÓN…….……...30

INTRODUCCIÓN

Durante el curso 2018/2019, enmarcado dentro del módulo “Mantenimiento Electrónico de Equipos Industriales”, fue propuesto al alumnado la realización de un proyecto conjunto entre el grupo de alumnos del CFGS en Mantenimiento Electrónico en colaboración con el alumnado perteneciente al grupo de Bachillerato de Artes Plásticas. El objetivo de este trabajo colaborativo era poder llevar a cabo una combinación entre ambos ámbitos a primera vista tan lejanos, el arte y la electrónica, adaptándose principalmente el ámbito de la electrónica (de una naturaleza técnica y objetiva) a las posibles demandas y/o requeridas por el arte (de una naturaleza más subjetiva).

Tras varias sesiones donde ambas partes se encontraron, se realizó una tormenta de ideas proponiendo distintos proyectos conceptuales y evaluando su viabilidad por ambas partes.

El presente trabajo corresponde una ampliación del trabajo previamente realizado en el módulo “Mantenimiento Electrónico de Equipos Industriales”, dando por finalizado y cerrado así el proyecto.

IDEAS CONCEPTUALES

Durante la realización de las reuniones y las tormentas de ideas fue propuesto el presente proyecto conceptual, un “profesor generador de notas aleatorio”, alguna vez vivido y sopesado por toda persona que se haya encontrado en el ámbito educativo en algún momento de su vida o que haya acatado el rol de “alumno”. Cabe mencionar que este proyecto cuenta con un trasfondo irónico pero a su vez mordaz sobre los hipotéticos métodos evaluativos de cierto sector del profesorado.

Referente al proyecto se marcaron varios aspectos importantes:

- Se llevaría a cabo la modelación de un títere, con aspecto físico de señor mayor, perfil típico de profesor cuya jubilación se encuentra cerca y descuida ciertos aspectos de su profesión como los métodos evaluativos.
- El títere debería tener capacidad para escribir números del 0 al 9, valores típicos de cualquier evaluación. Nótese que se descartó la posibilidad de escribir la nota “10”, como reflejo de aquellos profesores que defienden a capa y espada que nunca ponen esta nota, ya que implicaría el conocimiento total de la materia, por encima incluso de su nivel.
- El títere debería tener capacidad de detectar cuando ha sido colocado frente a él documentos, a representación de los exámenes a evaluar.
- Si el documento no fuese retirado, esto era síntoma de la disconformidad relativa a la nota, por lo que sería generada otra nota.
- Que el títere tuviese la capacidad de hablar e comunicar la nota que ha sido puesto, así como algunas frases para interactuar con la persona frente a él.

CONCEPCIÓN. VIABILIDAD TÉCNICA

A la hora de evaluar la viabilidad del proyecto, se llevo a cabo una tarea de investigación sobre las posibilidades para realizar el presente proyecto de manera electrónica.

Relativo a la escritura de caracteres numéricos existe un proyecto conocido y difundido, de naturaleza libre, consistente en un robot que escribe la hora sobre un panel y va realizando cambios en la misma conforme la progresión de la misma. Este proyecto es conocido como “Plot Clock”

Dada la naturaleza libre de este proyecto, pudo estudiarse su funcionamiento.

Una vez estudiado el mismo, pudo partirse de este proyecto para la realización del presente que será descrito, haciendo uso de varias partes y modificando ó incluyendo partes nuevas.

A la hora de sopesar la construcción del títere con las funciones electrónicas incorporadas, se decidió que éste siguiera el siguiente orden de construcción, así como de prioridades.

1) Dotar al títere de la capacidad de escribir los números
2) Dotar al títere de la capacidad de detectar los documentos a evaluar y a partir de la detección realizar a cabo la evaluación
3) Dotar al títere de la capacidad de hablar y reproducir sonidos, como comentarios hacia el alumnado o decir en voz alta la nota puesta.

Partiendo del proyecto base antes mencionado, “Plot Clock”, se decidió partir del mismo a la hora de seleccionar el entorno de desarrollo sobre el cual iba a realizarse el proyecto completo. Este entorno era “Arduino” conocido por el alumnado, idóneo para la realización del proyecto.

Tras sopesar la viabilidad de las siguientes partes adaptadas al entorno “Arduino” y sopesar la misma, decidió llevarse a cabo la realización de las mismas.

PROYECTO. PRIMERA Y SEGUNDA FASE: ESCRITURA MANUAL Y DETECCIÓN IR

Como fue mencionado con anterioridad, el proyecto iba a partir de otro proyecto de naturaleza libre implementado con anterioridad, el “PlotClock”

El proyecto“PlotClock” se trata de un pequeño robot que escribe la hora sobre una pizarra y que, tras cada minuto, borra la hora y la actualiza. Para llevar a cabo esta tarea, es buscado llevar a cabo la representación escrita de números. Partiéndose de los fundamentos de la geometría, es posible mediante un sistema de dos coordenadas en el espacio representar la posición de la sucesión de puntos que conforman cualquier elemento mediante una trayecto que lo describa. Haciendo uso de 2 actuadores en consonancia, como dos servomotores, es posible realizar una coordinación de los mismo para obtener ubicado en el plano un punto concreto haciendo uso de coordenadas polares y cartesianas. Estos fundamentos geométricos, en combinación de fundamentos cinemáticos, son los que hacen posible la sincronización de los actuadores para poder realizar la función de escritura en el plano X e Y, en combinación con otro actuador que brinde el movimiento correspondiente a un eje Z.

En el siguiente enlace, puede encontrarse la información relativa al proyecto:
→ Plotclock: https://wiki.fablab-nuernberg.de/w/Ding:Plotclock

Este proyecto se compone de las siguientes elementos:

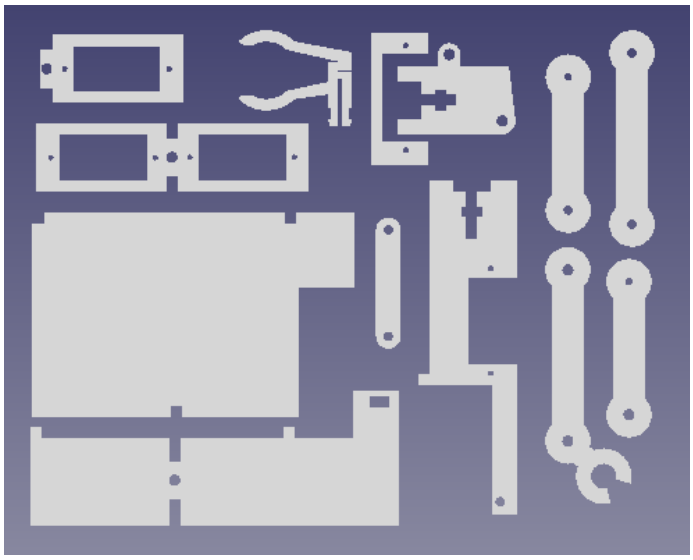
- 1 juego de piezas (ya puede ser piezas en madera MDF cortadas a laser, realizadas a mano, impresas en 3D, etc)
- 3 Servos Tower Pro SG90 y sus correspondientes coronas.
- Arduino Uno o equivalente
- Sensor IR TCRT5000
- Led y resistencia de valor óhmico 330
- Cableado variado
- Tornillería variada

JUEGO DE PIEZAS

Dada la naturaleza libre del proyecto se decidió partir de éstas y realizar las posibles adaptaciones que pudiesen ser necesarias.

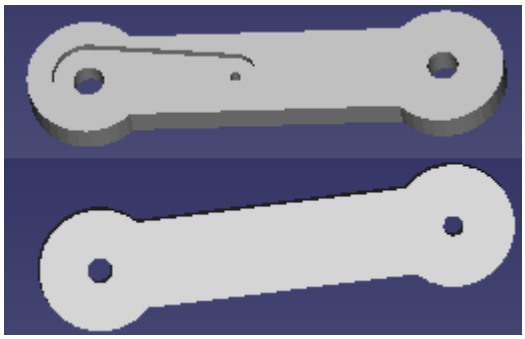
Ya que se disponía del acceso a la tecnología de impresión en 3D y la existencia de unas piezas base, decidió partirse de unas disponibles al público de forma abierta. Las piezas se encontraban en los siguientes repositorios:

- Plot Clock STL Files → juego de piezas general que incluía todas las posibles partes necesarias, así como otras que no iban a ser necesarias para nuestro proyecto. Incluye toda la estructura de soporte de los servomotores, así como los ejes que brindarán los desplazamientos del elemento de escritura.



→ Detalle de las piezas

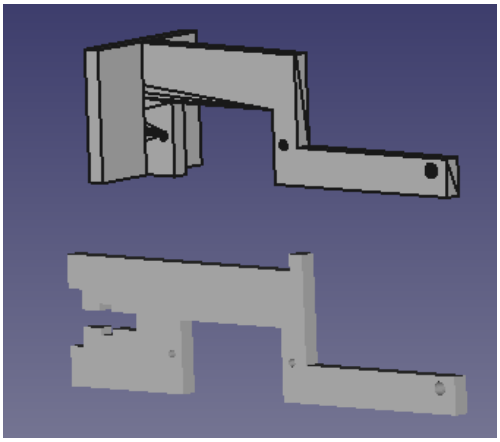
- Plotclock with more detailed instructions →este archivo cuenta con una pieza modificada respecto a la del archivo anterior. Este archivo incluye los ejes para los servomotores adaptada a la cora de los servos, implementando una mayor facilidad en el desplazamiento de esta.



→ Detalle de comparativa entre piezas

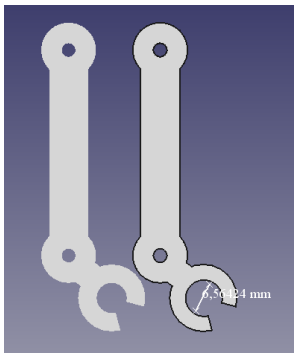
Posteriormente se llevó a cabo la adaptación de varias de las piezas a las necesidad específicas.

- ***Soporte para servo elevador***: en el diseño original, se juntan dos piezas mediante tornillería, pero se decidió modificar esta pieza para incluir una pequeña base en los laterales, menor que la pieza a anclar, para brindar de estabilidad y reducir el tamaño de las piezas que formarán parte de la futura mano del títere. Modificación mediante el software Freecad.



→ Detalle de adaptación de pieza para soporte se servo elevador

- ***Soporte de bolígrafo***: Se requería una pieza con mayor anchura para poder albergar el rotulador, por lo que se modificó mediante el software Freecad.



→ Detalle de pieza que soporta el rotulador.

PRIMERA FASE: BLOQUE DE ESCRITURA MANUAL

Como fue comentado durante la concepción del proyecto, este bloque es la parte matriz del proyecto, la cual dará pie a la ampliación del mismo.

Los componentes electrónicos que conforman este bloque son 3 servomotores tipo SG90, y un microcontrolador, como el Arduino Uno.

La información relativa a programación y a la calibración de este módulo, se llevará a cabo posteriormente, en un apéndice dedicado en exclusiva.

Las características principales de estos componentes electrónicos son:

→ ***Servomotor SG90***

(Datasheet: http://akizukidenshi.com/download/ds/towerpro/SG90.pdf)

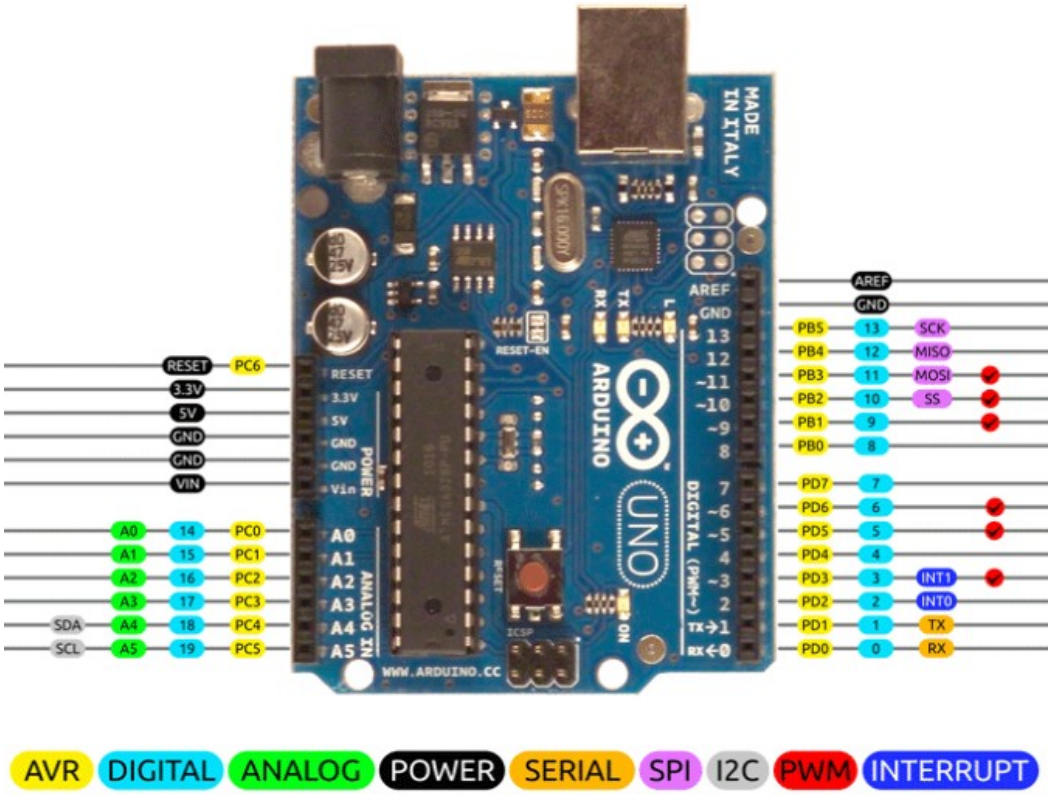
- Velocidad: 0.10 sec/60° @ 4.8V
- Torque: 1.8 Kg-cm @ 4.8V
- Voltaje de funcionamiento: 3.0-7.2V
- Temperatura de funcionamiento: -30 ℃ ~ 60 ℃
- Ángulo de rotación: 180°
- Ancho de pulso: 500-2400 µs
- Longitud de cable de conector: 24.5cm

→ Detalle de servomotor SG90 y conexionado

→ ***Arduino UNO***

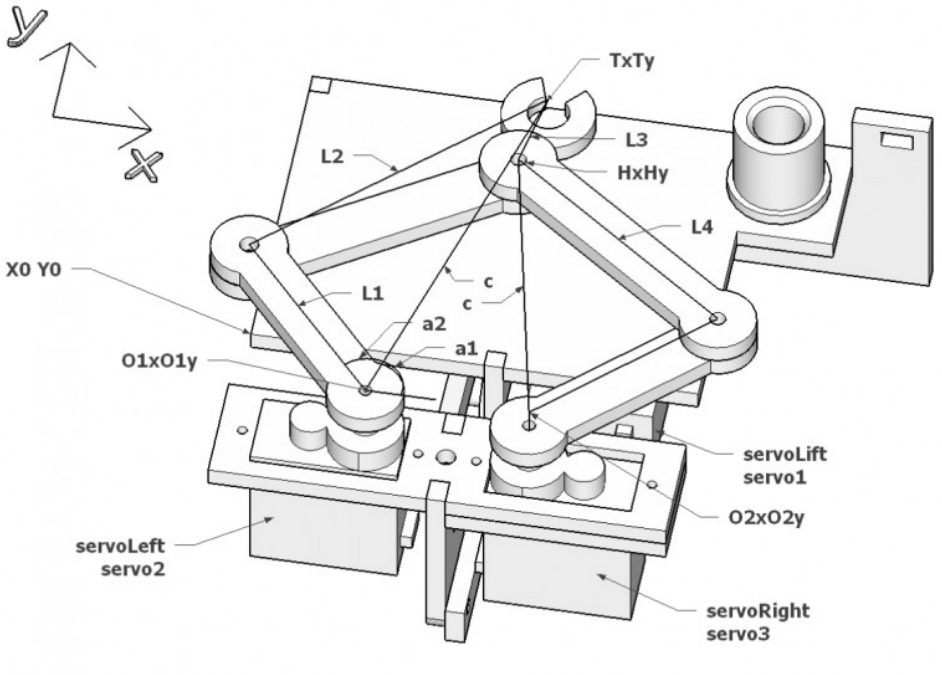
(Datasheet: https://www.farnell.com/datasheets/1682209.pdf)

- Microcontrolador: ATmega328
- Voltage: 5V
- Voltage entrada (recomendado): 7-12V
- Voltage entrada (limites): 6-20V
- Digital I/O Pins: 14 (de los cuales 6 son salida PWM)
- Entradas Analogicas: 6
- DC Current per I/O Pin: 40 mA
- DC Current parar 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328) de los cuales 0.5 KB son utilizados para el arranque
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

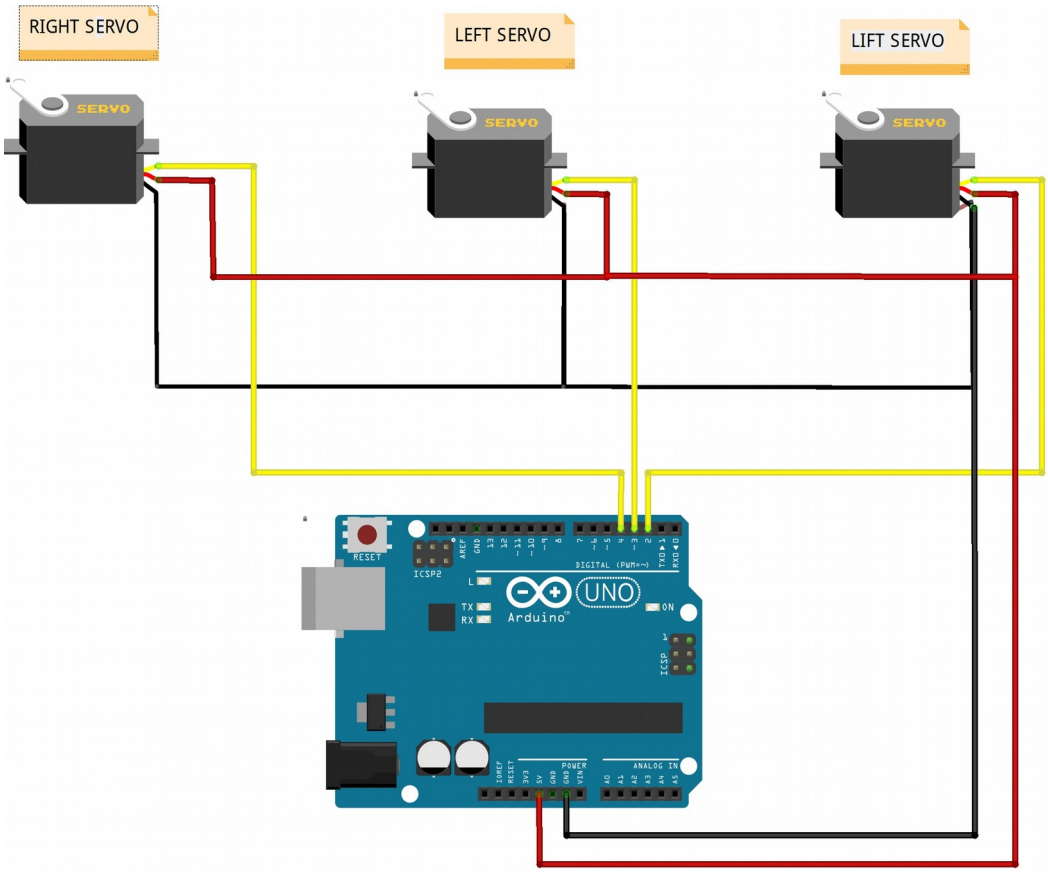


→ Detalle de Arduino UNO y conexionado

Será necesario realizar el ensamblado de las piezas impresas junto con a los servomotores. El ensamblado deberá tener el siguiente aspecto y el siguiente conexionado



→ Detalle de ensamblado mecánico



→ Detalle del conexionado electrónico

SEGUNDA FASE: BLOQUE DE DETECCIÓN DE FOLIO

Para poder llevar a cabo un detonador de la escritura, se decidió llevar a cabo la implementación de un actuador el cual por su naturaleza pudiese detectar la presencia de un folio.

Para ello, se hace uso de un detector de presencia de obstáculos IR. El funcionamiento de este sensor se basa en el siguiente principio: se dispone de un LED emisor de luz infrarroja y de un fotodiodo que recibe la luz reflejada por un posible obstáculo. Atendiendo a los principios básicas de la óptica y colorimetría, los rallos incidente sobre una superficie blanca reflejarán completamente la luz incidida en ella, mientras que de forma contraria una superficie de color negro absorbe completamente la luz, no reflejando luz alguna. A partir de este principio, puede realizarse la estimulación oportuna para proceder a ordenar al microcontrolador a que realice la escritura de la nota aleatoria o no. Además, el sensor cuenta con un potenciómetro, el cual regula la distancia a la cual se considera como detectado el objeto.

→ ***Servo IR Detector de obstáculos con TCRT5000***

(Datasheet: https://arduino.ua/docs/TCRT5000L-infrared-reflector-specs.pdf)

- Sensor Para Arduino LM39.
- Tipo: TCRT50.
- Chip: LM393
- Potenciómetro de control de sensibilidad.
- Soporte 5 V/3.5 V voltaje.
- TCRT5000 sensor de reflexión de infrarrojo.
- Distancia de prueba: 1mm a 25 m
- Corriente de funcionamiento: > 15 mA.

→ Detalle del módulo IR y especificación de pines de conexionado

PROYECTO. TERCERA FASE: MÓDULO REPRODUCTOR MP3

El presente bloque pretende profundizar en el funcionamiento del reproductor DFPlayer mini, el cual va a ser utilizado para llevar a cabo la reproducción de los sonidos, correspondientes a las notas que el generador irá escribiendo.

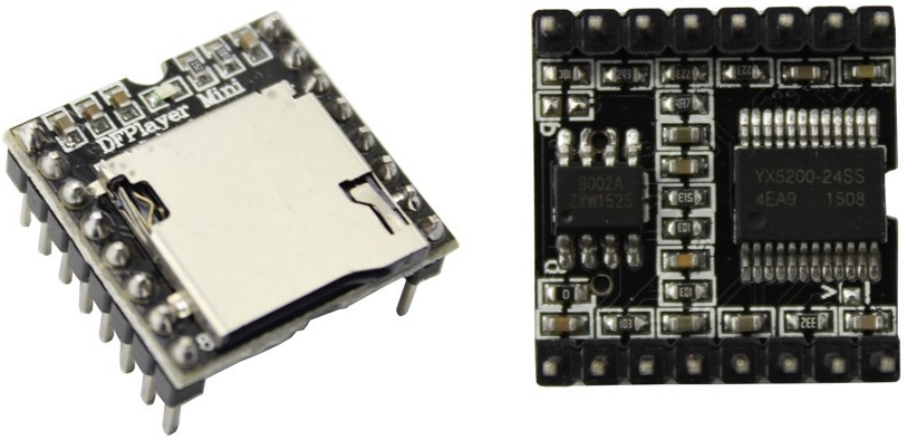
El reproductor de MP3 DFPlayer Mini para Arduino es un módulo de MP3 de bajo precio y con una salida simplificada directamente al altavoz. El módulo se puede usar como un módulo independiente con batería, altavoz y pulsadores adjuntos, o en combinación con un Arduino UNO o cualquier otro con capacidades RX / TX.

Consta de las siguientes características:
(Datasheet: https://cdn.instructables.com/ORIG/FGT/H5J3/IUKEZCYF/FGTH5J3IUKEZCYF.pdf)

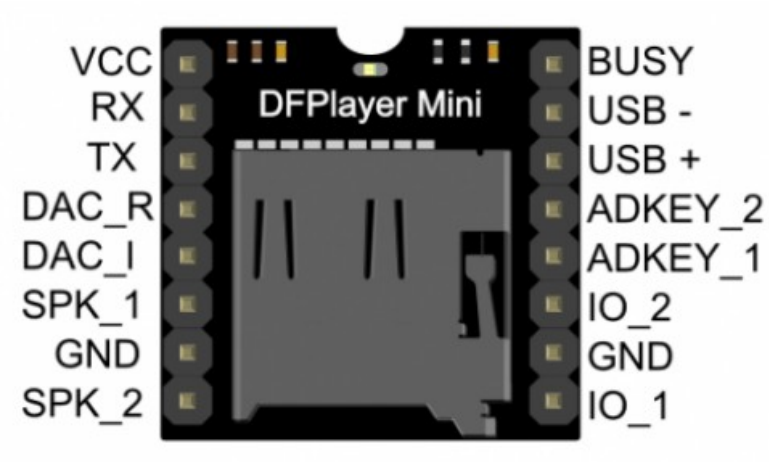
- Frecuencias de muestreo admitidas (kHz): 8 / 11.025 / 12/16 / 22.05 / 24/32 / 44.1 / 48.
- Salida DAC de 24 bits, soporte para rango dinámico 90dB, soporte SNR 85dB.
- Totalmente compatible con FAT16 y sistema de archivos FAT32.
- Capacidad máxima microSD de 32GB.
- Variedad de modos de control, modo de control de E / S, modo de serie, modo de control de botón AD.
- Soporta datos de audio ordenados por carpeta, admitiendo hasta 100 carpetas y cada carpeta puede contener hasta 255 canciones.
- Volumen ajustable de 30 niveles, ecualizador de 6 niveles ajustable.

Las posibles aplicaciones que pueden darse a este módulo pueden ser muy variadas:

- Transmisión de voz para sistemas de navegación.
- Indicadores de voz en estaciones de tren, autobús, etc.
- Sistemas de sonido para alarma.
- Efectos sonoros para maquetas o juguetes



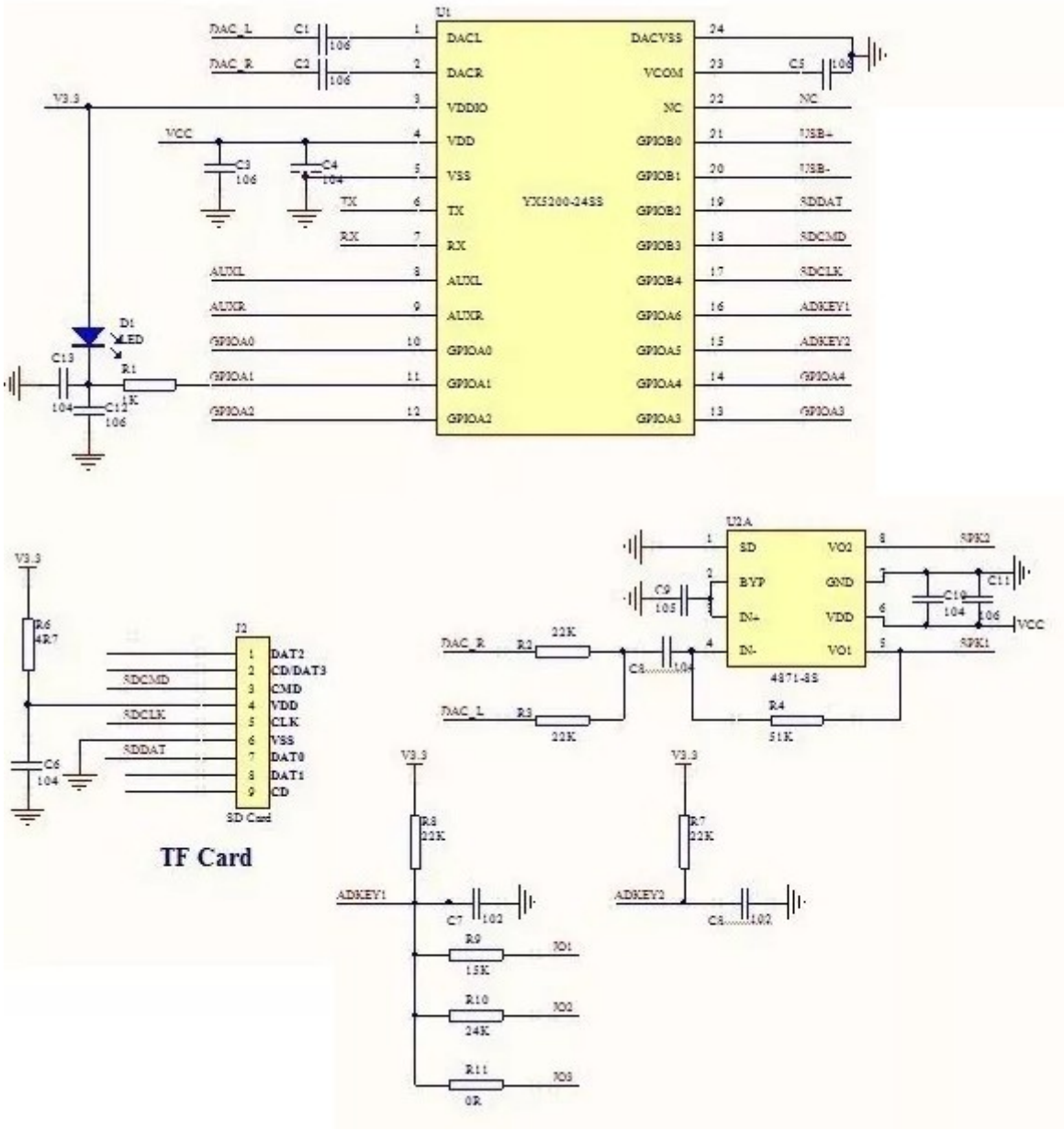
→ Detalle del módulo reproductor DFPlayer mini. Visión frontal y trasera.



→ Detalle de pines del módulo DFPlayer mini

Pin	Description	Note
VCC	Input Voltage	DC3.2~5.0V;Type: DC4.2V
RX	UART serial input	
TX	UART serial output	
DAC_R	Audio output right channel	Drive earphone and amplifier
DAC_L	Audio output left channel	Drive earphone and amplifier
SPK2	Speaker-	Drive speaker less than 3W
GND	Ground	Power GND
SPK1	Speaker+	Drive speaker less than 3W
IO1	Trigger port 1	Short press to play previous（long press to decrease volume）
GND	Ground	Power GND
IO2	Trigger port 2	Short press to play next（long press to increase volume）
ADKEY1	AD Port 1	Trigger play first segment
ADKEY2	AD Port 2	Trigger play fifth segment
USB+	USB+ DP	USB Port
USB-	USB- DM	USB Port
BUSY	Playing Status	Low means playing \High means no

→ Breve descripción de cada uno de los pines del DFPlayer mini



Detalle del esquemático del reproductor, basado en yx5200-24ss

MODOS DE CONTROL

Este módulo se caracteriza por poder ser utilizado de manera muy versátil, pudiendo ser controlado y usado de distintos modos.

- **Control mediante comunicación serial**

Las distintas funciones del reproductor pueden ser controladas mediante un PC haciendo uso de comunicación serial. Deberá tenerse en cuenta que para esto la tasa de transmisión de información deberá de ser 9600 baudios.

A continuación se muestran los comandos y las instrucciones para el control

Format: $S VER Len CMD Feedback para1 para2 checksum $O		
$S	Start bit 0x7E	Each command feedback begin with $, that is 0x7E
VER	Version	Version Information
Len	the number of bytes after "Len"	Checksums are not counted
CMD	Commands	Indicate the specific operations, such as play / pause, etc.
Feedback	Command feedback	If need for feedback, 1: feedback, 0: no feedback
para1	Parameter 1	Query high data byte
para2	Parameter 2	Query low data byte
checksum	Checksum	Accumulation and verification [not include start bit $]
$O	End bit	End bit 0xEF

For example, if we specify play NORFLASH, you need to send: 7E FF 06 09 00 00 04 FF DD EF
Data length is 6, which are 6 bytes [FF 06 09 00 00 04]. Not counting the start, end, and verification.

→ Descripción del formato de las instrucciones

CMD	Function Description	Parameters(16 bit)
0x01	Next	
0x02	Previous	
0x03	Specify tracking(NUM)	0-2999
0x04	Increase volume	
0x05	Decrease volume	
0x06	Specify volume	0-30
0x07	Specify EQ(0/1/2/3/4/5)	Normal/Pop/Rock/Jazz/Classic/Base
0x08	Specify playback mode (0/1/2/3)	Repeat/folder repeat/single repeat/ random
0x09	Specify playback source(0/1/2/3/4)	U/TF/AUX/SLEEP/FLASH
0x0A	Enter into standby – low power loss	
0x0B	Normal working	
0x0C	Reset module	
0x0D	Playback	
0x0E	Pause	
0x0F	Specify folder to playback	1~10(need to set by user)
0x10	Volume adjust set	{DH＝1:Open volume adjust }{DL: set volume gain 0~31}
0x11	Repeat play	{1:start repeat play}{0:stop play}

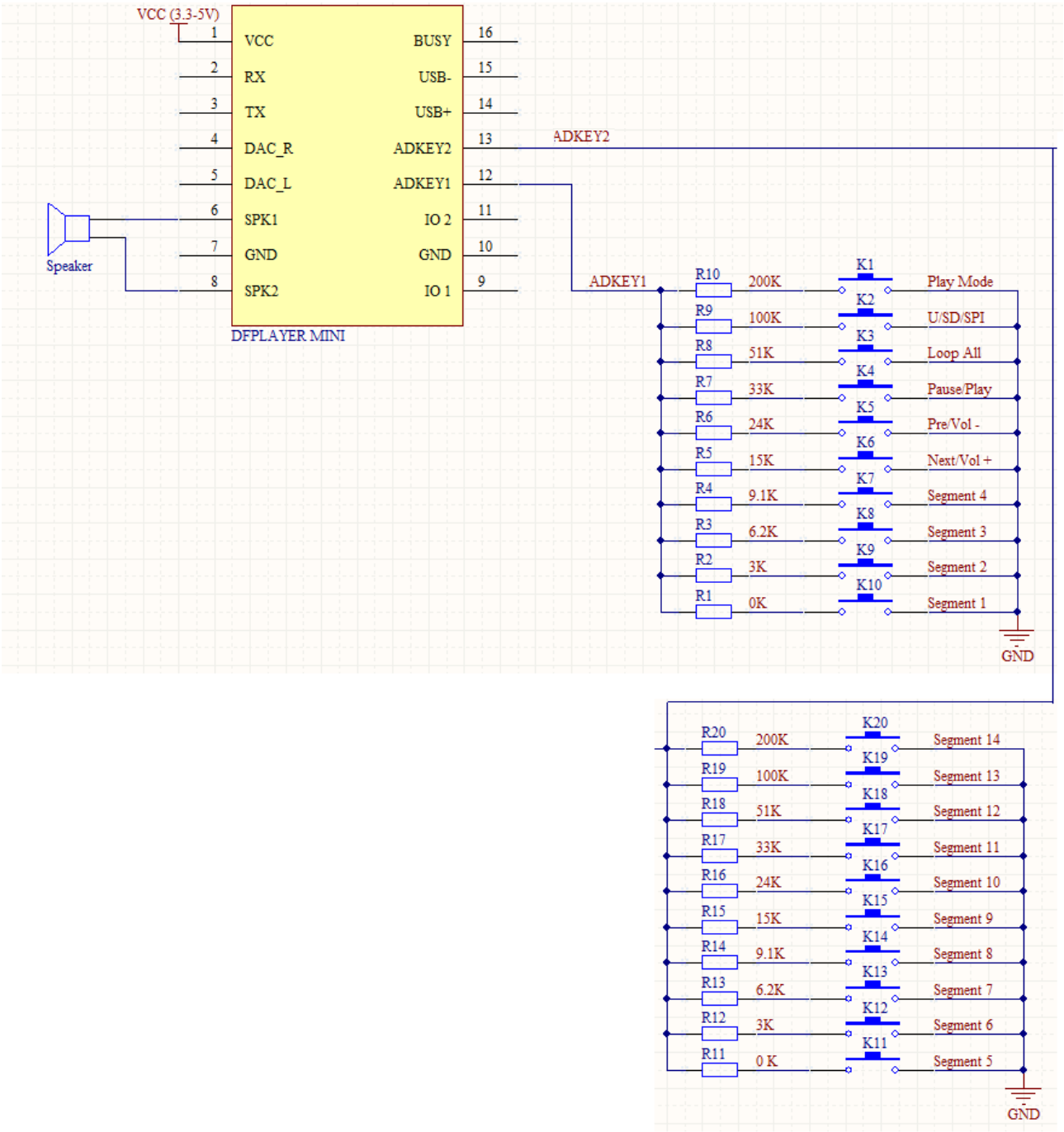
→Descripción de los comandos de control

Commands	Function Description	Parameters(16 bit)
0x3C	STAY	
0x3D	STAY	
0x3E	STAY	
0x3F	Send initialization parameters	0 - 0x0F(each bit represent one device of the low-four bits)
0x40	Returns an error, request retransmission	
0x41	Reply	
0x42	Query the current status	
0x43	Query the current volume	
0x44	Query the current EQ	
0x45	Query the current playback mode	
0x46	Query the current software version	
0x47	Query the total number of TF card files	
0x48	Query the total number of U-disk files	
0x49	Query the total number of flash files	
0x4A	Keep on	
0x4B	Queries the current track of TF card	
0x4C	Queries the current track of U-Disk	
0x4D	Queries the current track of Flash	

→ Descripción de los comandos de consulta Query

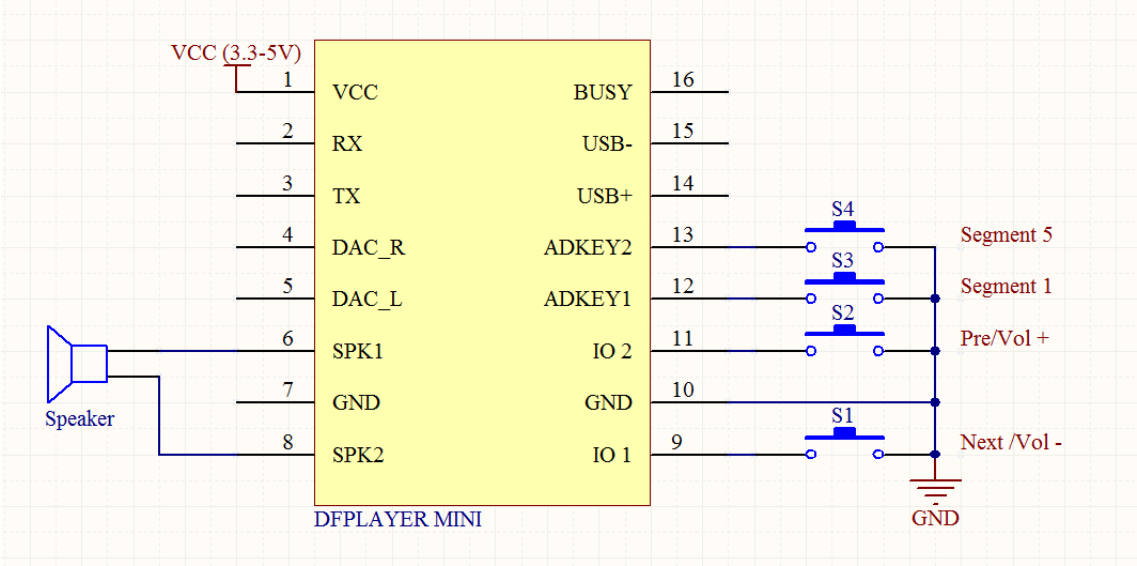
➢ ***Control mediante teclado.***

Haciendo uso de dos puertos AD, es posible realizarse un teclado haciendo uso de una posible distribución de hasta 20 resistencias para elegir la opción que se desea controlar.

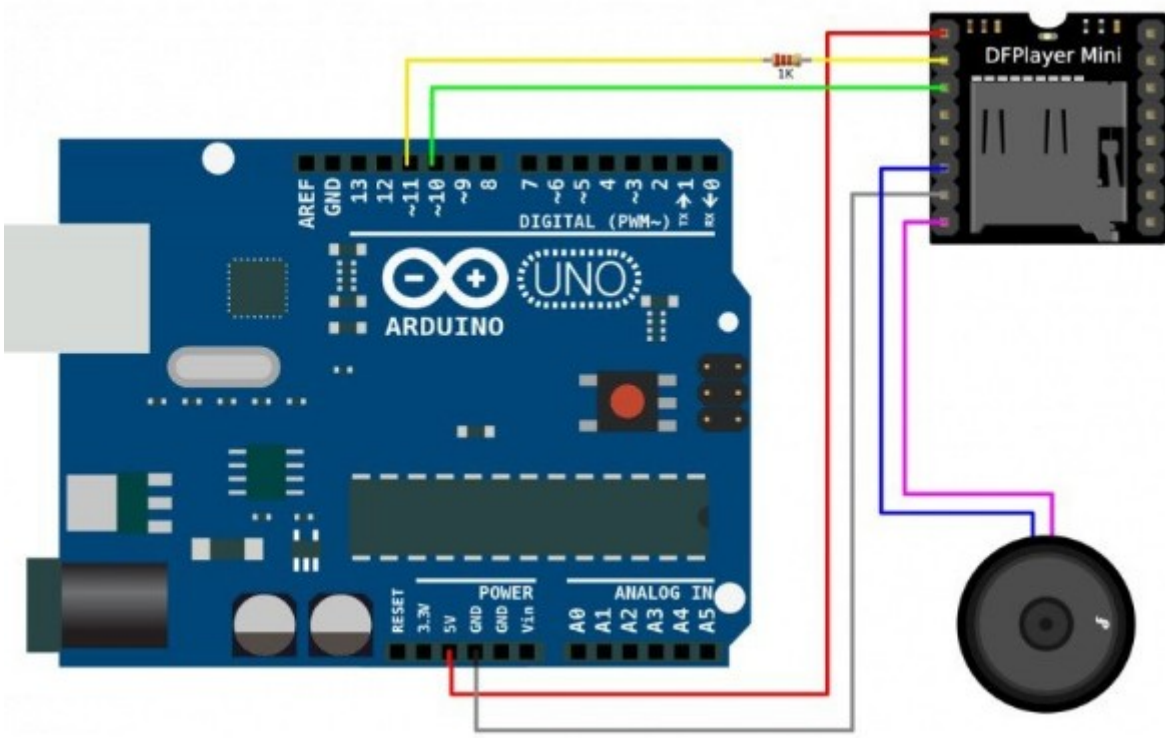


➢ ***Control híbrido Entrada/Salida***

Este modo es una mezcla de ambos modos descritos previamente, acorde a las posibles especificaciones del diseñador. Mediante el uso de librerías se puede lograr la interconexión con otros componentes electrónicos haciendo uso de entradas y salidas.



→ Ejemplo de configuración Entrada/Salida



→ Ejemplo de conexionado simple del módulo a ARDUINO UNO

COMUNICACION ENTRADA/SALIDA → SPI

Para llevar a cabo la transmisión de información entre el módulo y el microcontrolador, debe de utilizarse alguna forma de comunicación entre ambos dispositivos. Esta comunicación puede realizarse mediante Bus SPI

El Bus SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).

Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

Muchos sistemas digitales no necesitan una conexión rápida con sus periféricos. La ventajas de un bus serie es que minimiza el número de conductores, pines y el tamaño del circuito integrado. Esto reduce el coste de fabricar, montar y probar la electrónica. Un bus de periféricos serie es la opción más flexible cuando se tiene tipos diferentes de periféricos serie. El hardware consiste en señales de reloj, data in, data out y chip select para cada circuito integrado que tiene que ser controlado. Casi cualquier dispositivo digital puede ser controlado con esta combinación de señales. Los dispositivos se diferencian en un número predecible de formas. Unos leen el dato cuando el reloj sube, otros cuando el reloj baja. Algunos lo leen en el flanco de subida del reloj y otros en el flanco de bajada. Escribir es casi siempre en la dirección opuesta de la dirección de movimiento del reloj. Algunos dispositivos tienen dos relojes. Uno para capturar o mostrar los datos y el otro para el dispositivo interno.

El SPI es un protocolo síncrono. La sincronización y la transmisión de datos se realiza por medio de 4 señales:

- SCLK (Clock): Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit. También llamado TAKT (en alemán).
- MOSI (Master Output Slave Input): Salida de datos del Master y entrada de datos al Esclavo. También llamada SIMO.
- MISO (Master Input Slave Output): Salida de datos del Esclavo y entrada al Master. También conocida por SOMI.
- SS/Select: Para seleccionar un Esclavo, o para que el Master le diga al Esclavo que se active. También llamada SSTE.

Ventajas del bus SPI frente a otros protocolos de comunicación

Ventajas

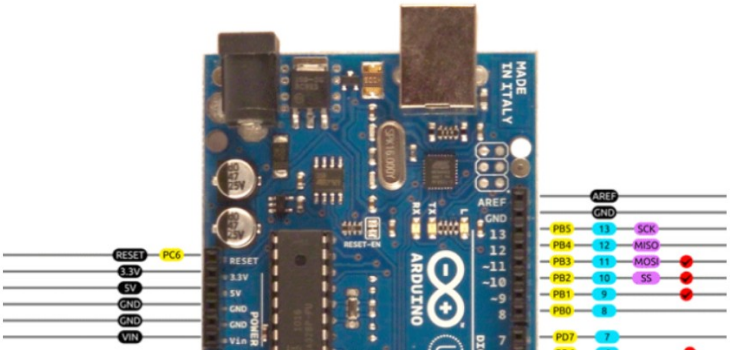
- Comunicación Full Duplex
- Mayor velocidad de transmisión que con I²C o SMBus
- Protocolo flexible en que se puede tener un control absoluto sobre los bits transmitidos
  - No está limitado a la transferencia de bloques de 8 bits
  - Elección del tamaño de la trama de bits, de su significado y propósito
- Su implementación en hardware es extremadamente simple
  - Consume menos energía que I²C o que SMBus debido que posee menos circuitos (incluyendo las resistencias pull-up) y estos son más simples
  - No es necesario arbitraje o mecanismo de respuesta ante fallos
  - Los dispositivos clientes usan el reloj que envía el servidor, no necesitan por tanto su propio reloj
  - No es obligatorio implementar un transceptor (emisor y receptor), un dispositivo conectado puede configurarse para que solo envíe, sólo reciba o ambas cosas a la vez
- Usa mucho menos terminales en cada chip/conector que una interfaz paralelo equivalente
- Como mucho una única señal específica para cada cliente (señal SS), las demás señales pueden ser compartidas

Desventajas

- Consume más pines de cada chip que I²C, incluso en la variante de 3 hilos
- El direccionamiento se hace mediante líneas específicas (señalización fuera de banda) a diferencia de lo que ocurre en I²C que se selecciona cada chip mediante una dirección de 7 bits que se envía por las mismas líneas del bus
- No hay control de flujo por hardware
- No hay señal de asentimiento. El servidor podría estar enviando información sin que estuviese conectado ningún cliente y no se daría cuenta de nada
- No permite fácilmente tener varios servidores conectados al bus
- Sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232 o RS-485

ARDUINO y del bus SPI

El entorno de programación ARDUINO es compatible con el protocolo de comunicación SPI, debiéndose hacer uso de una librería específica para poder llevar a cabo la correcta comunicación. Los pines correspondientes pueden encontrarse en los pines 10, 11, 12 y 13

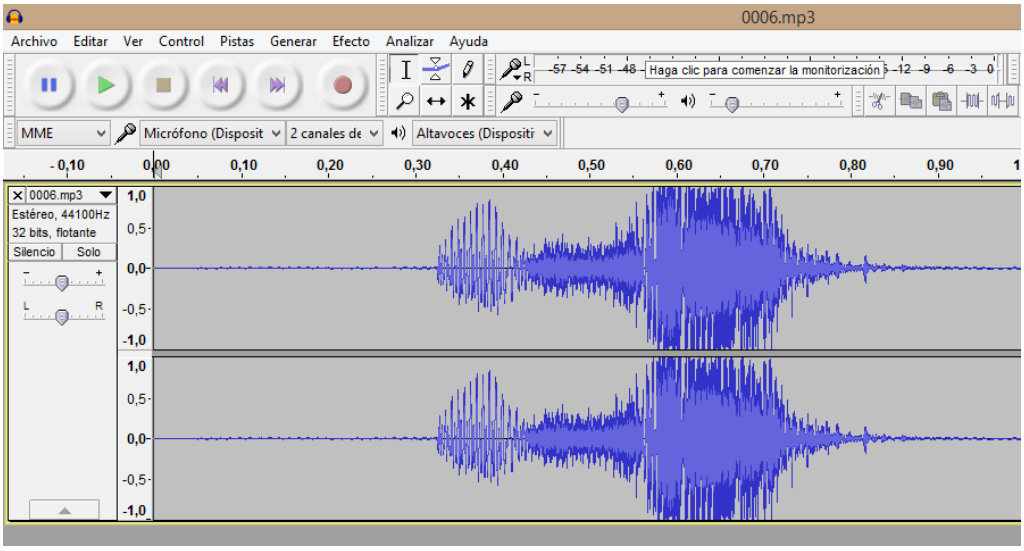


→ Detalle de pines SPI en ARDUINO

REQUISITOS PREVIOS SD

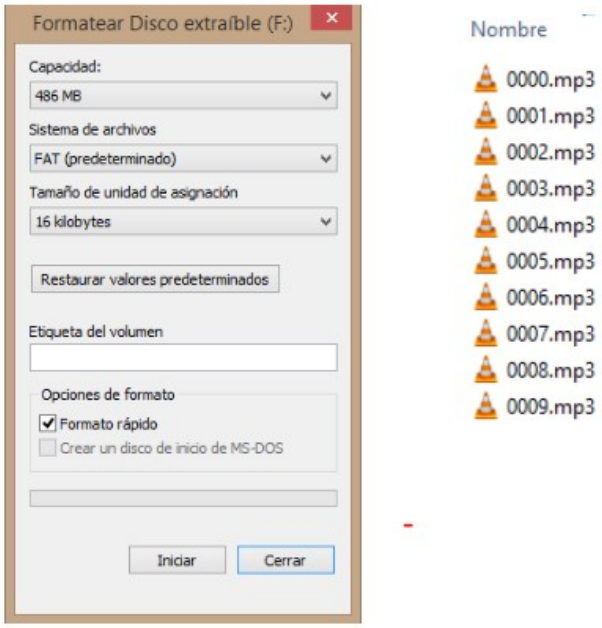
Previo a usar el módulo, es necesario llevar a cabo una serie de pasos, relativos al tratamiento de la tarjeta SD.

Una vez tenidos los sonidos (los cuales pueden ser grabados mediante cualquier DAW, como puede ser Audacity) es importante mencionar que éstos tienen que tener como nombre en formato XXXX.mp3, con el fin de una administración correcta de los distintos archivos.



→ Ejemplo de grabación de sonido haciendo uso del DAW Audacity

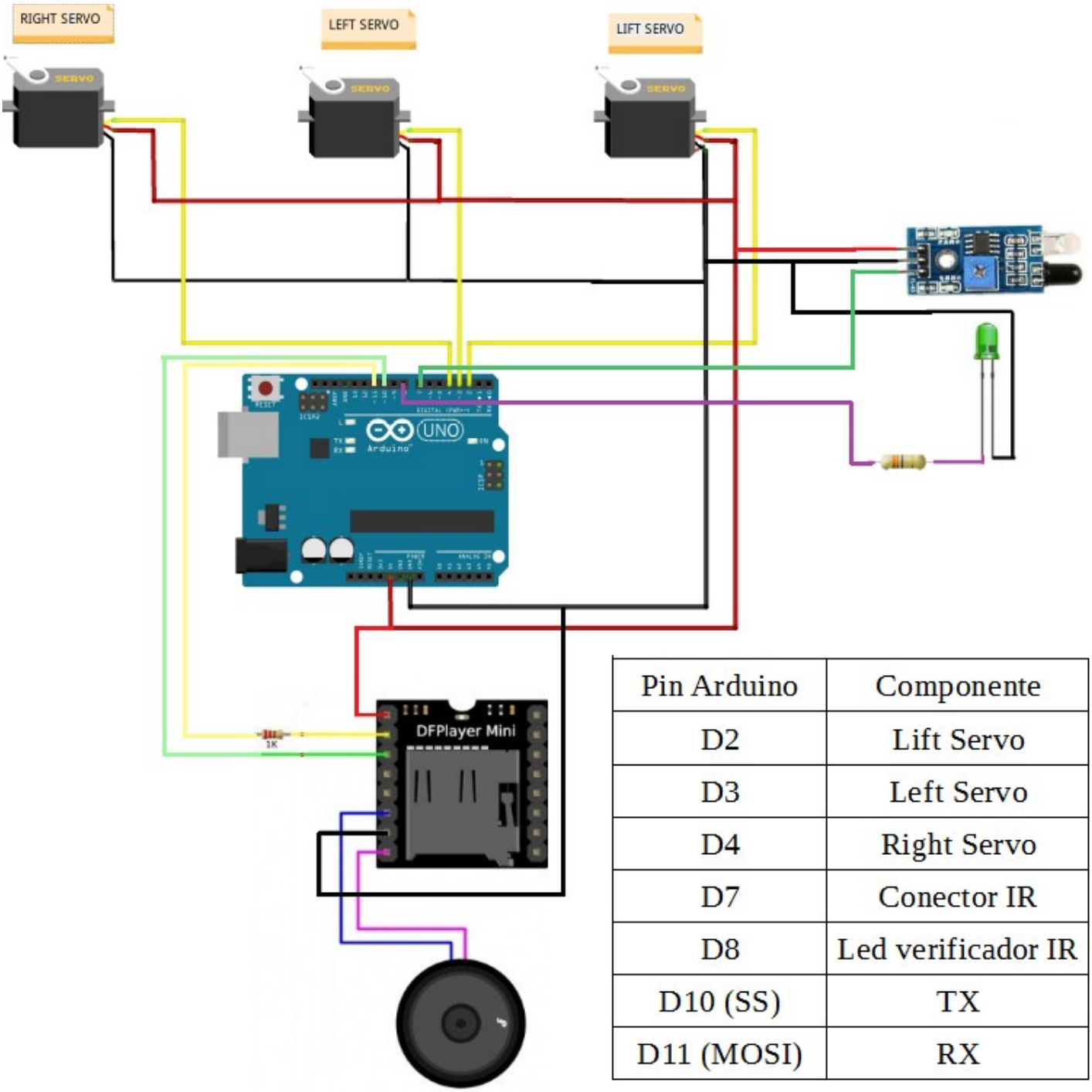
Previa a la introducción de los archivos, es importante formatear la tarjeta de memoria, acorte al Sistema de archivos FAT, y agregando un tamaño de unidad de asignación de 16 kilobytes.



→ Formateo de Disco Extraible y archivos a ingresar

CONEXIONADO DE BLOQUES

Una vez descritos de manera independiente los 3 bloques que componen el proyecto, puede llevarse a cabo el ensamblado de los mismos. Importante mencionar que es necesario el uso de una fuente de tensión externa, de al menos 1A de intensidad, ya que los aproximados 450mA que puede brindar un conector USB hacen que el sistema no funcione de forma correcta.

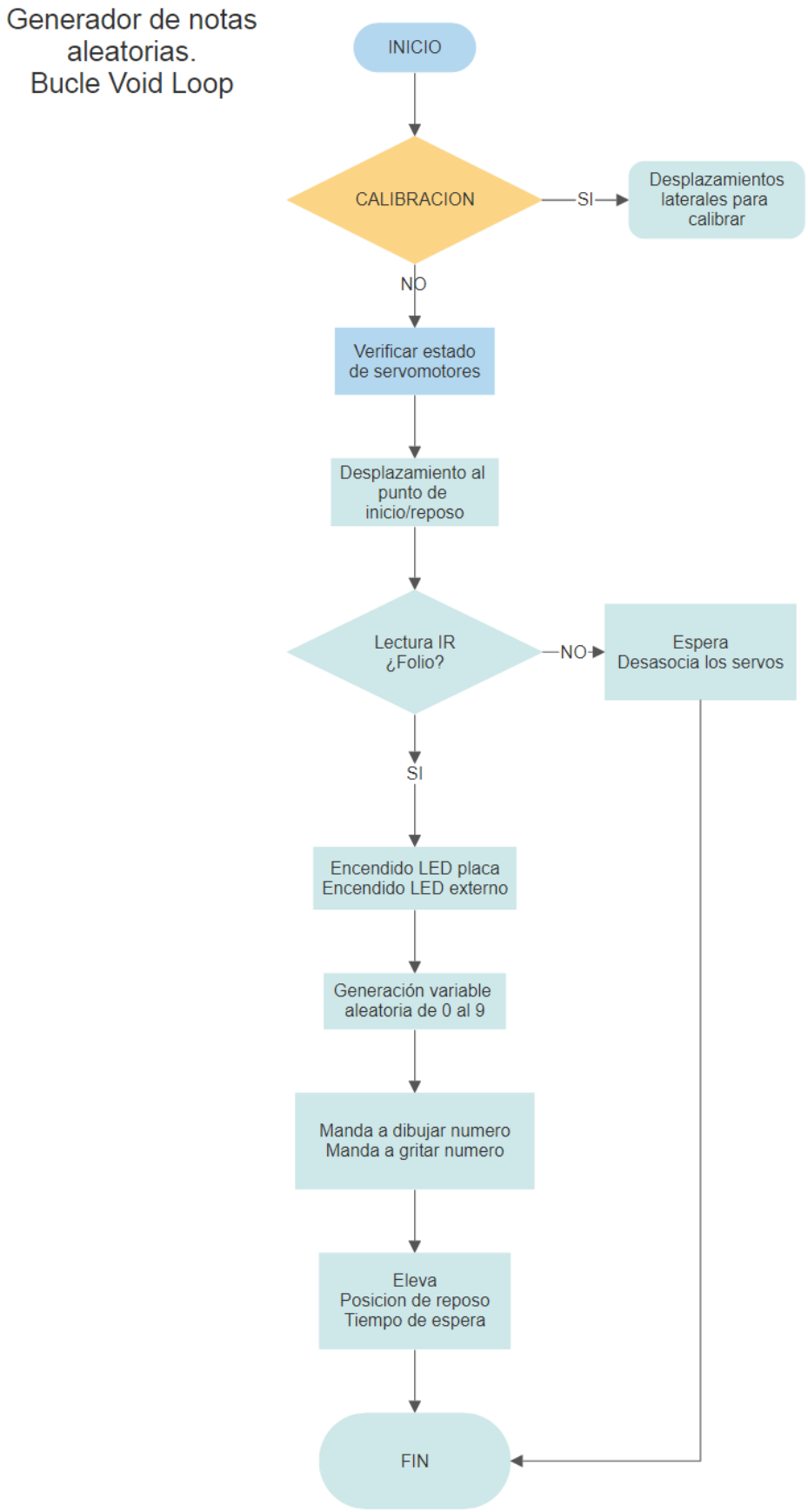


Pin Arduino	Componente
D2	Lift Servo
D3	Left Servo
D4	Right Servo
D7	Conector IR
D8	Led verificador IR
D10 (SS)	TX
D11 (MOSI)	RX

→ Detalle del conexionado de los componente con el microcontrolador

PRINCIPIOS DE FUNCIONAMIENTO. PROGRAMACIÓN

DIAGRAMA DE FLUJO DE BLOQUE PRINCIPAL



PROGRAMACIÓN. DESCRIPCIÓN DE LOS BLOQUES

El código proporcionado por el proyecto contaba con los siguientes bloques definibles:

- ***Defines e inclusión de librerías***

En este primer bloque se encuadran las distintas librerías y parámetros que tendrán un valor constante dentro del proyecto, sirviéndose de esta definición para posteriormente poder variar su valor y que no afecte al resto del código.

- ***Void setup***

Bloque relativo al entorno “Arduino”, donde se establece la definición y configuración de los distintos pines que van a ser ejecutados posteriormente.

- ***Void loop***

Bloque principal de la programación, el cual incluye las instrucciones principales a realizar por el microcontrolador, las cuales serán ejecutadas de manera repetida en bucle.

- ***void number***

Bloque el cual contiene la información y las instrucciones necesarias para realizar la escritura de un número, como pueden ser la posición bx y by donde va a iniciarse la escritura el número, los movimientos mecánicos los cuales componen la trayectoria, la selección del número que va a realizarse o la escala del mismo.

- ***void lift***

Bloque el cual contiene la información necesaria para realizar los movimiento de elevación y/o descenso del servo que cumple dicho propósito, en función del caso de elevación seleccionado.

- ***void bogen UZS && void bogen GZS***

Bloque que genera los desplazamientos diagonales y en forma de arco sincronizados de los servomotores, tanto de manera ascendente como descendente, necesarios para realizar la trayectoria de escritura en ciertos números

- ***void drawTo***

Bloque que genera los movimientos necesarios para llevar el punto de escritura a un punto concreto del plano X e Y para la escritura.

- ***Double return_angle***

Función que realiza el teorema del coseno, necesaria para la generación de movimientos y el posicionamiento correcto.

- ***Void set_XY***

Bloque encargado de sincronizar y generar la triangulación correcta entre los servos.

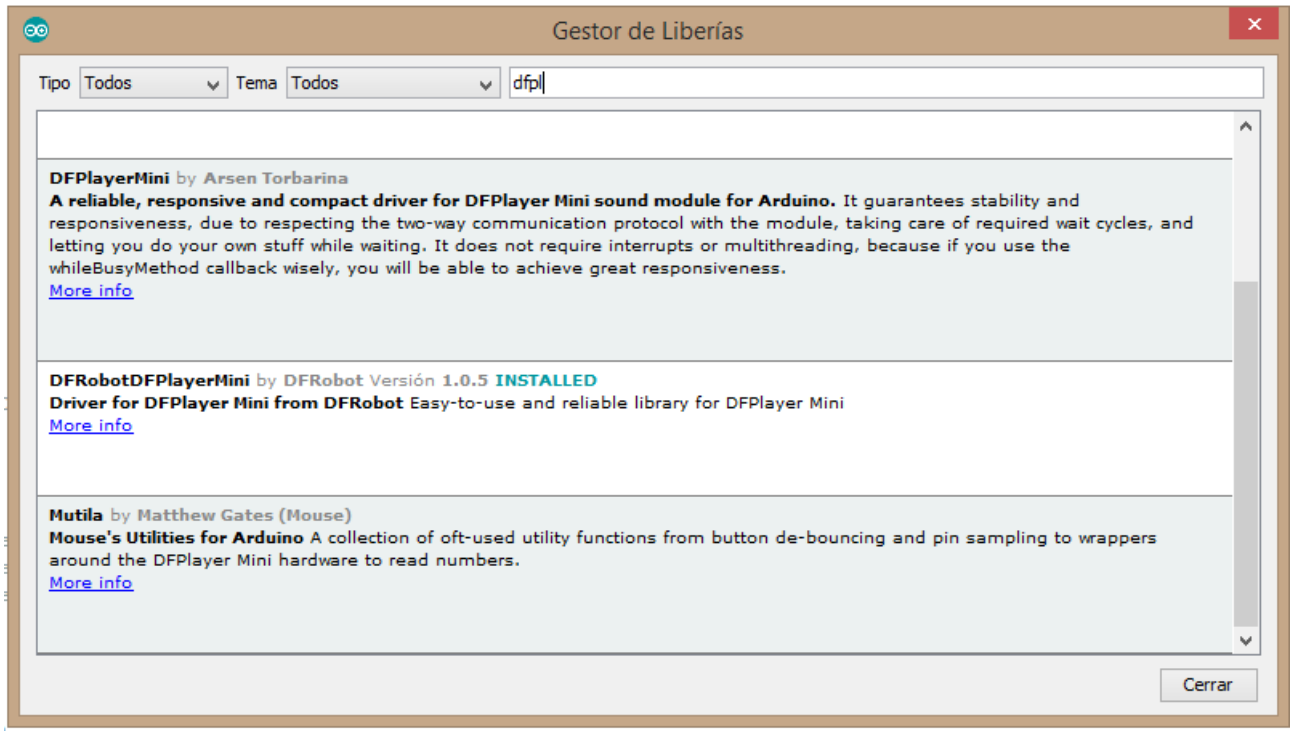
DESCARGA DE LIBRERÍA

Para hacer uso del módulo DFPlayer es necesario incluir una biblioteca espacial llamada “DFRobotDFPlayerMini”, la cual es derivada a la librería SPI para poder llevar a cabo la correcta comunicación con del microcontrolador con el módulo DFPlayer. Además, esta librería incluye ya implementadas una serie de instrucciones reservadas para hacer uso del módulo DFPlayer

Dentro del IDE de Arduino puede buscarse la librería. Los pasos son los siguientes:

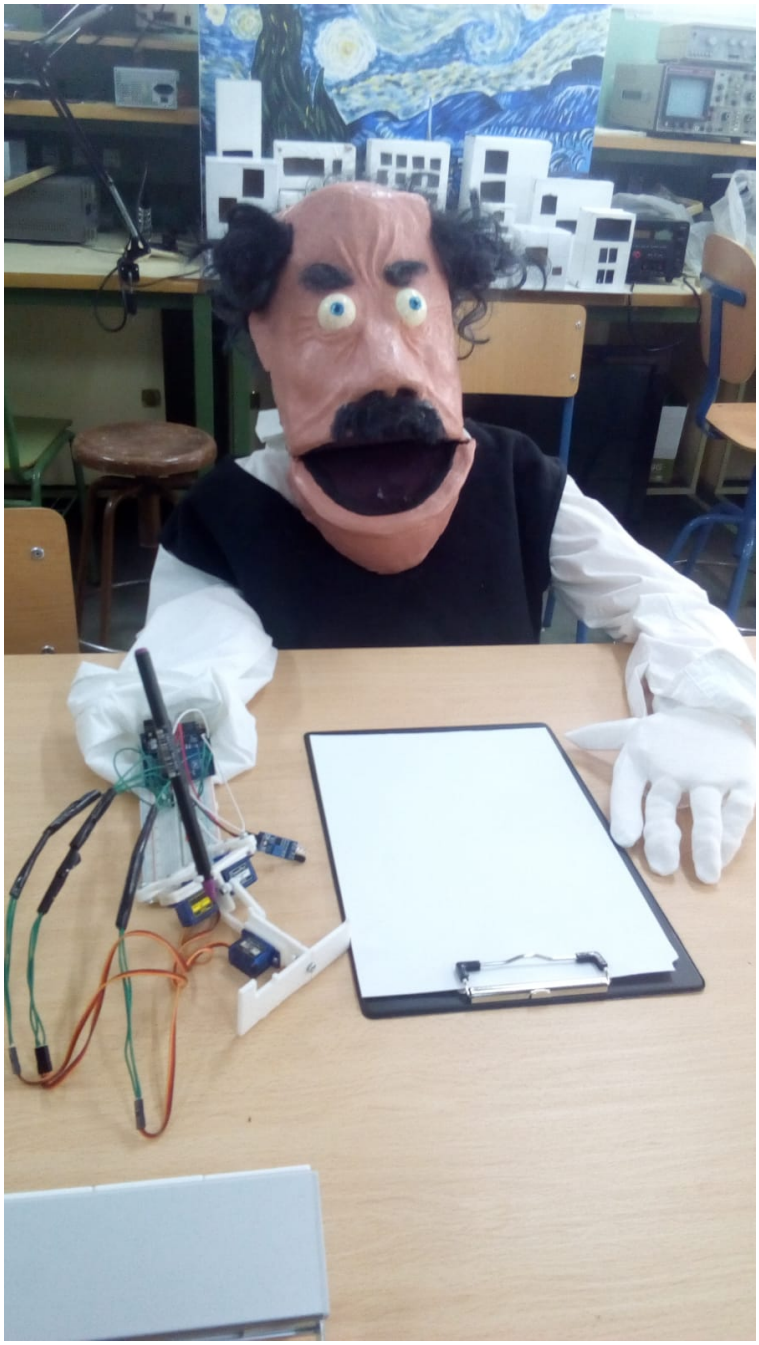
Programa >> Incluir Librería >> Gestor de librerías

En la ventana emergente buscar la librería “DFRobotDFPlayerMini” y descargarla.



ENSAMBLAJE

Una vez probada la circuitería y su correcto funcionamiento, se debe tomar el títere y proceder al ensamblado.



Acá se muestran unos vídeos donde pueden apreciarse el correcto funcionamiento de los bloques por separado.

→ Bloque de Lectura IR y escritura: https://youtu.be/56YSxQxI9zI
→ Bloque reproductor de voz: https://youtu.be/tv_PomkquXI

CONTENIDO WEB

→ Plotclock: https://wiki.fablab-nuernberg.de/w/Ding:Plotclock

→ Plotclock Files: https://www.thingiverse.com/thing:250204

→ Plotclock with more details instructions: https://www.thingiverse.com/thing:931498

→ Datasheet Servomotor SG90: http://akizukidenshi.com/download/ds/towerpro/SG90.pdf

→ Datasheet Arduino UNO: https://www.farnell.com/datasheets/1682209.pdf

→ Datasheet TCRT5000: https://arduino.ua/docs/TCRT5000L-infrared-reflector-specs.pdf

→ Sensor IR web: https://www.luisllamas.es/detectar-obstaculos-con-sensor-infrarrojo-y-arduino/

→ Datasheet DFPlayer mini:
https://cdn.instructables.com/ORIG/FGT/H5J3/IUKEZCYF/FGTH5J3IUKEZCYF.pdf

→ DFPlayer website: https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299

→ SPI: https://es.wikipedia.org/wiki/Serial_Peripheral_Interface

→ Robot Reloj, cómo se programa y calibra: https://www.youtube.com/watch?v=a_HCdVSI-lU

→ Proyecto plotclock bloque 3 https://youtu.be/tv_PomkquXI

→ Proyecto plotclock bloques 1 y 2 https://youtu.be/56YSxQxI9zI

ANEXO: CÓDIGO DE PROGRAMACIÓN

```
// CODIGO MANO_IR_VOZ DE GENERADOR DE NOTA ALEATORIA

/* Primero se debe calibrar los brazos para que varíen en Angulo de 90 grados exactos y
de forma cruzada entre el servo derecho e izquierdo.

Después de Calibrar, borre o marque como comentario la linea siguiente  */

// #define CALIBRACION // ---- PONER EN COMENTARIO TRAS CALIBRAR----

// En modo de calibracion, ajuste el factor siguiente hasta que los servos se mueven
exactamente 90 grados
#define SERVOFAKTOR 640  //#define SERVOFAKTOR 620
/* En modo de calibracion, ajuste los valores NULL para que los brazos de los servos
esten en todo momento paralelos Ya sea al eje X o Y */
#define SERVOLEFTNULL 1800 //#define SERVOLEFTNULL 1900
#define SERVORIGHTNULL 1000 //#define SERVORIGHTNULL 984

#define SERVOPINELEVADOR  2      //  Pin 2 conexión del servo elevador
#define SERVOPINIZQUIERDO  3     //  Pin 3 conexión del servo izquierdo
#define SERVOPINDERECHA 4    //  Pin 4 conexion del servo derecho


// definir posiciones del servo elevador
#define LIFT0 1700 // en la superficie de dibujo Original 1050
#define LIFT1 1400  // elevación entre escritura de numeros Original 925
 #define LIFT2 825    //#define LIFT2 725  // yendo con el borrador Original 825

// velocidad de brazo elevador, un numero mayor es mas lento
#define VELOCIDADELEVADOR 1500

// longitud de los brazos
#define L1 35
#define L2 60    //#define L2 55.1
#define L3 13.2


// puntos de origen de servo izquierda y derecha
#define O1X 22
#define O1Y -25
#define O2X 47
#define O2Y -25

//---INCLUSION LIBRERIAS-----

// Libreria para control de servo
#include <Servo.h>

// Librerias para comunicacion con DFPlayer
#include "Arduino.h"
#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"

SoftwareSerial mySoftwareSerial(10, 11); // RX, TX
DFRobotDFPlayerMini myDFPlayer;


int servoLift = 1500; // altura para servo elevador

Servo servo1;  // declaracion servo 1
Servo servo2;  // declaracion servo 2
Servo servo3;  //  declaracion servo 3

volatile double lastX = 75;
volatile double lastY = 47.5;
```

```
// VARIABLES SENSOR IR

int LED = 13; // Use the onboard Uno LED
int LED_ACCION_PIN = 8;
int isObstaclePin = 7; // This is our input pin
int isObstacle = HIGH; // HIGH MEANS NO OBSTACLE


void setup()
{
  mySoftwareSerial.begin(9600); // Bit_rate comunicacion DFPlayer
  Serial.begin(9600); //bit_rate para visualización de nota en monitor serial


// --- Asignacion de pines

  servo1.attach(SERVOPINELEVADOR);  //  asocia servo elevador
  servo2.attach(SERVOPINIZQUIERDO);  //  asocia servo izquierdo
  servo3.attach(SERVOPINDERECHA);  //  asocia servo derecho


  pinMode(LED, OUTPUT); // LED 13 en placa
  pinMode(LED_ACCION_PIN, OUTPUT); // LED 8 para visualizar lectura IR
  pinMode(isObstaclePin, INPUT); // Pin 7 detector de folio

  Serial.println();
  Serial.println(F("DFRobot DFPlayer Mini Demo"));
  Serial.println(F("Initializing DFPlayer ... (May take 3~5 seconds)"));


  // -- bloque de instrucciones necesario para comunicacion con DFPlayer, verificando
su funcionamiento

  if (!myDFPlayer.begin(mySoftwareSerial)) {  //Use softwareSerial to communicate with
mp3.
    Serial.println(F("Unable to begin:"));
    Serial.println(F("1.Please recheck the connection!"));
    Serial.println(F("2.Please insert the SD card!"));
    while(true){
      delay(0); // Code to compatible with ESP8266 watch dog.
    }
  }
  Serial.println(F("DFPlayer Mini online."));

  myDFPlayer.volume(10);  //AJUSTE DEL VOLUMEN. DESDE 0 a 30 (20 OK)


}

void loop()
{

  #ifdef CALIBRACION // si no esta omitida esta linea, continuamente mueve para
calibrar

    // los brazos de los servos tendran 90 8 entre los movimientos, paralelo al eje X e
Y
    drawTo(-3, 29.2);
    delay(500);
    drawTo(74.1, 28);
    delay(500);
```

```
  #else // en el caso de estar la calibracion en comentario, pasa a ejecutar el
programa principal


    if (!servo1.attached()) servo1.attach(SERVOPINELEVADOR); // verifica el servo
    if (!servo2.attached()) servo2.attach(SERVOPINIZQUIERDO); // verifica el servo
    if (!servo3.attached()) servo3.attach(SERVOPINDERECHA); // verifica el servo

   // Desplazamiento a punto de inicio/reposo

    delay(500);
    drawTo(74.1, 28);
    delay(500);
    lift(1);

// se evaluca si el detector detecta folio y enciente leds para verificar

  isObstacle = digitalRead(isObstaclePin);
  if (isObstacle == LOW) {
    Serial.println("SALIDA LOW 0 --> FOLIO (REBOTA LUZ)  OBSTACLE!!");
    digitalWrite(LED, HIGH);
    digitalWrite(LED_ACCION_PIN, HIGH);


    int VAR_ALE; // creacion de variable que albergará la variable aleatoria
    VAR_ALE = random(0,9); //Asocia valor aleatorio a la variable (entre 0 y 9)

    Serial.print("El numero aleatorio es = "); // monitor serial para mostrar la nota
    Serial.println(VAR_ALE);

    delay (6000);

    number(25, 25, VAR_ALE, 1.1); // number(posicion bx, posición by, NUMERO, escala)

    myDFPlayer.play((VAR_ALE+1)); // EMPIEZA REPRODUCIENDO EL 1, vamos desfasados +1,
por eso MANDAMOS REPRODUCIR VAR_ALE+1


    lift(2); // ELEVAR HASTA ARRIBA

    drawTo(74.1, 28); // se vuelve a la posición de reposo

    delay (10000); // espera durante acción;

    }else { // instrucciones para cuando no hay folio

          Serial.println("SALIDA HIGH 1 --> NEGRO, SIN FOLIO (ABSORBE LUZ)");
          digitalWrite(LED, LOW);
          digitalWrite(LED_ACCION_PIN, LOW);
      }
      delay(2000); // vuelta a hacer lectura
      servo1.detach(); // si no se encuentra se desasocia el servo
      servo2.detach(); // si no se encuentra se desasocia el servo
      servo3.detach(); // si no se encuentra se desasocia el servo


  #endif

}
```

```
/*
 * VOID NUMBER -> Bloque el cual contiene la información y las instrucciones necesarias
para realizar la escritura de un número,
 * como pueden ser la posición bx y by donde va a iniciarse la escritura el número, los
movimientos mecánicos los cuales
 * componen la trayectoria, la selección del número que va a realizarse o la escala del
mismo.
 */

// La estructura sigue a este principio: mover al primer punto de inicio del numeral,
levante hacia abajo, dibuje numeral, levante


void number(float bx, float by, int num, float scale) {

  switch (num) {

  case 0:
/* CODIGO ORIGINAL
    drawTo(bx + 12 * scale, by + 6 * scale);
    lift(0);
    bogenGZS(bx + 7 * scale, by + 10 * scale, 10 * scale, -0.8, 6.7, 0.5);
    lift(1);
*/


// Movimiento simil display
    drawTo(45, 50);
    lift(0);
    delay (500);
    drawTo(60, 50);
    delay (500);
    drawTo(60, 20);
    delay (500);
    drawTo(45, 20);
    delay (500);
    drawTo(45, 50);
    delay (500);
    break;

  case 1:


    drawTo(bx + 3 * scale, by + 15 * scale);
    lift(0);
    drawTo(bx + 10 * scale, by + 20 * scale);
    drawTo(bx + 10 * scale, by + 0 * scale);
    lift(1);
    break;

  case 2:


    drawTo(bx + 2 * scale, by + 12 * scale);
    lift(0);
    bogenUZS(bx + 8 * scale, by + 14 * scale, 6 * scale, 3, -0.8, 1);
    drawTo(bx + 1 * scale, by + 0 * scale);
    drawTo(bx + 12 * scale, by + 0 * scale);
    lift(1);
    break;
```

```
  case 3:

    drawTo(bx + 2 * scale, by + 17 * scale);
    lift(0);
    bogenUZS(bx + 5 * scale, by + 15 * scale, 5 * scale, 3, -2, 1);
    bogenUZS(bx + 5 * scale, by + 5 * scale, 5 * scale, 1.57, -3, 1);
    lift(1);
    break;

    case 4:

/*
// CODIGO ORIGINAL
    drawTo(bx + 10 * scale, by + 0 * scale);
    lift(0);
    drawTo(bx + 10 * scale, by + 20 * scale);
    drawTo(bx + 2 * scale, by + 6 * scale);
    drawTo(bx + 12 * scale, by + 6 * scale);
    lift(1);
        break;
*/


// Movimientos simil display


    drawTo(45,60);
    lift(0);
    delay (500);
    drawTo(45,35);
    delay (500);
    drawTo(60,35);
    delay (500);
    drawTo(60,50);
    delay (500);
    drawTo(60,10);
    delay (500);
    break;

  case 5:

/*
// CODIGO ORIGINAL
    drawTo(bx + 2 * scale, by + 5 * scale);
    lift(0);
    bogenGZS(bx + 5 * scale, by + 6 * scale, 6 * scale, -2.5, 2, 1);
    drawTo(bx + 5 * scale, by + 20 * scale);
    drawTo(bx + 12 * scale, by + 20 * scale);
    lift(1);
        break;
*/


    drawTo(65,50);
    lift(0);
    delay (500);
    drawTo(45,50);
    delay (500);
    drawTo(45,35);
    delay (500);
    drawTo(65,35);
    delay (500);
    drawTo(65,20);
    delay (500);
    drawTo(45,20);
    delay (500);
    break;
```

```
  case 6:

    drawTo(bx + 2 * scale, by + 10 * scale);
    lift(0);
    bogenUZS(bx + 7 * scale, by + 6 * scale, 6 * scale, 2, -4.4, 1);
    drawTo(bx + 11 * scale, by + 20 * scale);
    lift(1);
    break;

  case 7:
// CODIGO ORIGINAL

/*
    drawTo(bx + 2 * scale, by + 20 * scale);
    lift(0);
    drawTo(bx + 12 * scale, by + 20 * scale);
    drawTo(bx + 2 * scale, by + 0);
    lift(1);
        break;
*/

    drawTo(40, 50);
    lift(0);
    delay (500);
    drawTo(60, 50);
    delay (500);
    drawTo(40, 20);
    delay (2000);
    lift(1);
// elevar y descender
    drawTo (40,35);
    lift (0);
    delay (500);
    drawTo (60,35);
    delay (500);
    break;

  case 8:

/*
    drawTo(bx + 5 * scale, by + 10 * scale);
    lift(0);
    bogenUZS(bx + 5 * scale, by + 15 * scale, 5 * scale, 4.7, -1.6, 1);
    bogenGZS(bx + 5 * scale, by + 5 * scale, 5 * scale, -4.7, 2, 1);
    lift(1);
        break;
*/

    drawTo(45, 50);
    lift(0);
    delay (500);
    drawTo(60, 50);
    delay (500);
    drawTo(60, 20);
    delay (500);
    drawTo(45, 20);
    delay (500);
    drawTo(45, 50);
    delay (500);
//Elevar y descender (no necesario)
    drawTo(45,35);
    delay (500);
    drawTo (60,35);
    delay (50);
    break;
```

```
  case 9:

    drawTo(bx + 9 * scale, by + 11 * scale);
    lift(0);
    bogenUZS(bx + 7 * scale, by + 15 * scale, 5 * scale, 4, -0.5, 1);
    drawTo(bx + 5 * scale, by + 0);
    lift(1);
    break;

  }
}

/*
 * VOID LIFT Bloque el cual contiene la información necesaria para realizar los
movimiento de elevación y/o descenso
 * del servo que cumple dicho propósito, en función del caso de elevación seleccionado.
 */

void lift(char lift) {
  switch (lift) {

  case 0: //850

      if (servoLift >= LIFT0) {
      while (servoLift >= LIFT0)
      {
        servoLift--;
        servo1.writeMicroseconds(servoLift);
        delayMicroseconds(VELOCIDADELEVADOR);
      }
    }
    else {
      while (servoLift <= LIFT0) {
        servoLift++;
        servo1.writeMicroseconds(servoLift);
        delayMicroseconds(VELOCIDADELEVADOR);

      }

    }

    break;

  case 1: //150

    if (servoLift >= LIFT1) {
      while (servoLift >= LIFT1) {
        servoLift--;
        servo1.writeMicroseconds(servoLift);
        delayMicroseconds(VELOCIDADELEVADOR);

      }
    }
    else {
      while (servoLift <= LIFT1) {
        servoLift++;
        servo1.writeMicroseconds(servoLift);
        delayMicroseconds(VELOCIDADELEVADOR);
      }

    }

    break;
```

```
  case 2:

    if (servoLift >= LIFT2) {
      while (servoLift >= LIFT2) {
        servoLift--;
        servo1.writeMicroseconds(servoLift);
        delayMicroseconds(VELOCIDADELEVADOR);
      }
    }
    else {
      while (servoLift <= LIFT2) {
        servoLift++;
        servo1.writeMicroseconds(servoLift);
        delayMicroseconds(VELOCIDADELEVADOR);
      }
    }
    break;
  }
}

/*
 * Bloque que genera los desplazamientos diagonales y en forma de arco sincronizados de
los servomotores, tanto de manera
 * ascendente como descendente, necesarios para realizar la trayectoria de escritura en
ciertos números
 */

void bogenUZS(float bx, float by, float radius, int start, int ende, float sqee) {
  float inkr = -0.05;
  float count = 0;

  do {
    drawTo(sqee * radius * cos(start + count) + bx,
    radius * sin(start + count) + by);
    count += inkr;
  }
  while ((start + count) > ende);

}

void bogenGZS(float bx, float by, float radius, int start, int ende, float sqee) {
  float inkr = 0.05;
  float count = 0;

  do {
    drawTo(sqee * radius * cos(start + count) + bx,
    radius * sin(start + count) + by);
    count += inkr;
  }
  while ((start + count) <= ende);
}
```

```
/*
 * Bloque que genera los movimientos necesarios para llevar el punto de escritura a un
 * punto concreto del plano X e Y para la escritura.
 */

void drawTo(double pX, double pY) {
  double dx, dy, c;
  int i;

  // dx dy of new point
  dx = pX - lastX;
  dy = pY - lastY;
  // Longitud de la trayectoria en mm, 4 veces equivale a 4 pasos por mm
  c = floor(4 * sqrt(dx * dx + dy * dy));

  if (c < 1) c = 1;

  for (i = 0; i <= c; i++) {
    // dibuja la lÌnea punto por punto
    set_XY(lastX + (i * dx / c), lastY + (i * dy / c));

  }

  lastX = pX;
  lastY = pY;
}

/*
 * Función que realiza el teorema del coseno, necesaria para la generación de
 * movimientos y el posicionamiento correcto.
 */

double return_angle(double a, double b, double c) {
  // regla del coseno para el angulo entre c y a
  return acos((a * a + c * c - b * b) / (2 * a * c));
}
```

```
/*
 * Bloque encargado de sincronizar y generar la triangulación correcta entre los
servos.
 */

void set_XY(double Tx, double Ty)
{
  delay(1);
  double dx, dy, c, a1, a2, Hx, Hy;

  //calcular triangulo entre marcador, servo izquierdo y brazo
  // cartesiano dx/dy
  dx = Tx - O1X;
  dy = Ty - O1Y;

  // longitud polar (c)  y el angulo (a1)
  c = sqrt(dx * dx + dy * dy);
  a1 = atan2(dy, dx);
  a2 = return_angle(L1, L2, c);

  servo2.writeMicroseconds(floor(((a2 + a1 - M_PI) * SERVOFAKTOR) + SERVOLEFTNULL));

  // calcular el punto del brazo para formar el triangulo del  servo derecho y el brazo
  a2 = return_angle(L2, L1, c);
  Hx = Tx + L3 * cos((a1 - a2 + 0.621) + M_PI); //36,58
  Hy = Ty + L3 * sin((a1 - a2 + 0.621) + M_PI);

  // calcular el triangulo entre el marcador, servo derecho y el brazo
  dx = Hx - O2X;
  dy = Hy - O2Y;

  c = sqrt(dx * dx + dy * dy);
  a1 = atan2(dy, dx);
  a2 = return_angle(L1, (L2 - L3), c);

  servo3.writeMicroseconds(floor(((a1 - a2) * SERVOFAKTOR) + SERVORIGHTNULL));

}
```