



## Bloques funcionales

### ¿Qué va a hacer?

Se va a realizar un sistema de infoentretenimiento compacto capaz de reproducir, pausar y cambiar de archivo de audio además de comunicarse e interactuar efectivamente con el usuario dando información como la hora .

### ¿Para qué?

Para demostrar todos los conocimientos que vamos a adquirir durante este bloque de 10 semanas y poder cumplir con el reto de realizar un reproductor de música. Para conocer y aplicar una metodología de planeación hasta validación de un sistema embebido en chip, como se realiza en empresas grandes como Intel.

### ¿Cómo?

- Definición conceptual y de bloques del sistema.
- Diseño del sistema y su arquitectura.
- Pruebas y simulaciones.
- Prototipo físico
- Validación y retroalimentación
- Proyecto final
- Presentación

### ¿Qué sabemos hacer?

- Sabemos utilizar VHDL para el FPGA
- Funcionamiento del display LCD
- Funcionamiento de VGA
- Programar en C++
- Máquinas de estados
- Multiplexores
- Compuertas lógicas
- Lógica del ensamblador

### ¿Qué recursos y materiales necesitamos?

- Display LCD o pantalla touch
- Memoria SD
- Microprocesador Raspberry Pi 4
- Mínimo 3 botones físicos o integrados en pantalla
- Software Proteus
- Microcontrolador Arduino Mega

### ¿Qué necesitamos aprender?

- Necesitamos saber hacer un sistema operativo para nuestro dispositivo
- Saber crear un interfaz amigable con el usuario
- Saber utilizar la Raspbian y lograr hacer la conexión de C++/Python con la tarjeta
- Simular en Proteus

### ¿Qué sería bueno incorporar?



11 de junio de 2021

- Sería bueno incorporar una pantalla touch para que la interfaz sea más amigable con el usuario
- Que se pueda modificar el volumen del reproductor usando potenciómetro.
- Podría verse la posibilidad desde software, que el usuario planee su reproducción de canciones (playlists).
- Utilizar una bocina que emita un sonido cuando se inicialice el sistema: no buzzers. Revisar si podemos usar dos bocinas para audio stereo o mono.
- Varios perfiles de usuario usando calculadora de membrana interruptor.
- Fecha y hora con módulo RTC.
- “Colorear el sonido” o estado de reproducción con LEDs o LED RGB.
- Cambio de canciones con giroscopio.

### Analogía del avión

#### **Where ?**

Diseño finalizado de un sistema de infoentretenimiento compacto. Diseño de sistema embebido cuyos requerimientos mínimos son:

- Reproducción de audio mediante microcontrolador.
- Lectura, almacenamiento y acceso de archivos de audio mediante memoria SD (Secure Digital).
- Display LCD para mostrar al usuario los datos del archivo con respectiva información de la digitalización (frecuencia de muestreo, tipo de digitalización, audio mono/stereo), y la información de la canción en reproducción (archivo, nombre, artista, álbum, año).
- Botones interfaz

#### **Flight time**

Aproximadamente 9 semanas y media

#### **Flight plan**

Establecer una planeación a seguir del proyecto	26 de marzo - 7 de abril del 2021
Conceptualización del problema y objetivo del reto	26 de marzo - 7 de abril del 2021
Definición conceptual y de bloques del	9 - 16 de abril del 2021



11 de junio de 2021

sistema	
Diseño del sistema y su arquitectura	19 - 30 de abril del 2021
Pruebas y simulaciones	10- 20 de mayo del 2021
Prototipo físico	21- 28 de mayo del 2021
Validación y retroalimentación	1 - 7 de junio del 2021
Proyecto final	7 - 10 de junio del 2021
Presentación	11 junio del 2021

## Fuel/Pilots/Air Stewards

### Recursos

- Display LCD o pantalla touch
- Memoria SD
- Microprocesador Raspberry Pi 4
- Mínimo 3 botones físicos o integrados en pantalla
- Software Proteus
- Microcontrolador Arduino Mega

### Personal:

- Manuel Agustín Diaz Vivanco A01379673
- Carlos Antonio Pazos Reyes A01378262

## Flight is tracked

Establecer una planeación a seguir del proyecto	7 de abril del 2021 Evidencia necesaria: documento “Etapa 1”
Conceptualización del problema y objetivo del reto	7 de abril del 2021 Evidencia necesaria: listado desglosado de todos los requerimientos mínimos y extras: requerimiento, materiales, recursos y pendientes.
Definición conceptual y de bloques del sistema	16 de abril del 2021 Evidencia necesaria: descripción del funcionamiento del sistema, componentes y recursos del sistema, interconexiones e interfaz del sistema, y diagrama de bloques del sistema.



Diseño del sistema y su arquitectura	30 de abril del 2021 Evidencia necesaria: descripción y modelación de la arquitectura del sistema con todas sus conexiones y componentes. Modelo en Proteus.
Pruebas y simulaciones	20 de mayo del 2021 Evidencia necesaria: validación del funcionamiento del sistema. Pruebas planeadas en Proteus. Retroalimentación de los profesores. Validación de arquitectura, componentes y conexiones. Rediseño y ajustes necesarios del sistema.
Prototipo físico	28 de mayo del 2021 Evidencia necesaria: primeras implementaciones físicas del sistema. Conexiones, interfaz y pruebas físicas iniciales. Modelo físico usando los recursos Raspberry Pi, Arduino, display/pantalla, sensores, etcétera.
Validación y retroalimentación	7 de junio del 2021 Evidencia necesaria: Prototipo físico para poder recibir la retroalimentación y poder lograr la validación. Ajustes, observaciones, mejoras y cambios de implementación finales.
Proyecto final	10 de junio del 2021 Evidencia necesaria: Sistema físico funcional finalizado ya validado y con retroalimentación de los profesores
Presentación	11 junio del 2021 Evidencia necesaria: "Entregable reto final"

## Arrival

Entrega final: 10 de junio de 2021.

Parámetro mínimo de cumplimiento: calificación aprobatoria.

Parámetros de calidad:

- Calificación mayor a 90.
- Retroalimentación de usuarios en casa positivos.
- Sin fallos ni interrupciones en su uso recurrente.
- Interfaz amigable, con mejoras muy específicas posibles.



## Etapa 2

### Arduino MEGA

#### **Avance Arduino Keypad, sensor ultrasónico, buzzer y LCD**

Se realizaron pruebas con cada uno de los sensores de manera individual para ver que funcionaran de manera correcta. Al lograr que cada uno funcionara correctamente, nos propusimos juntarnos y adaptarlos para que se comuniquen entre ellos y puedan mandar información acorde a lo que reciben. En este avance logramos que el usuario al usar el keypad pueda observar el número presionado en el LCD, además, al presionar el botón, el buzzer produce un tono corto. Finalmente, el sensor ultrasónico funciona detrás de todo el sistema, dando información de las distancias

Código Arduino presentado en Anexos (1)

#### **Video demostrativo**

<https://drive.google.com/file/d/1Mdd7u9sQzyo-fbTN3mzuJZ7r2VmT5ImC/view?usp=sharing>

#### **Avance Arduino Control remoto, sensor ultrasónico y LCD**

Otra propuesta que tuvimos fue cambiar el keypad por un control remoto, por lo cual experimentamos con él de forma individual y logramos que funcionara en sintonía con los otros componentes. Empezamos a investigar cómo mejor conectar el audio con un sistema estéreo si es posible para reproducir el sonido y conseguimos un par de bocinas, el puro cableado y transductor.

Código Arduino presentado en Anexos (2)

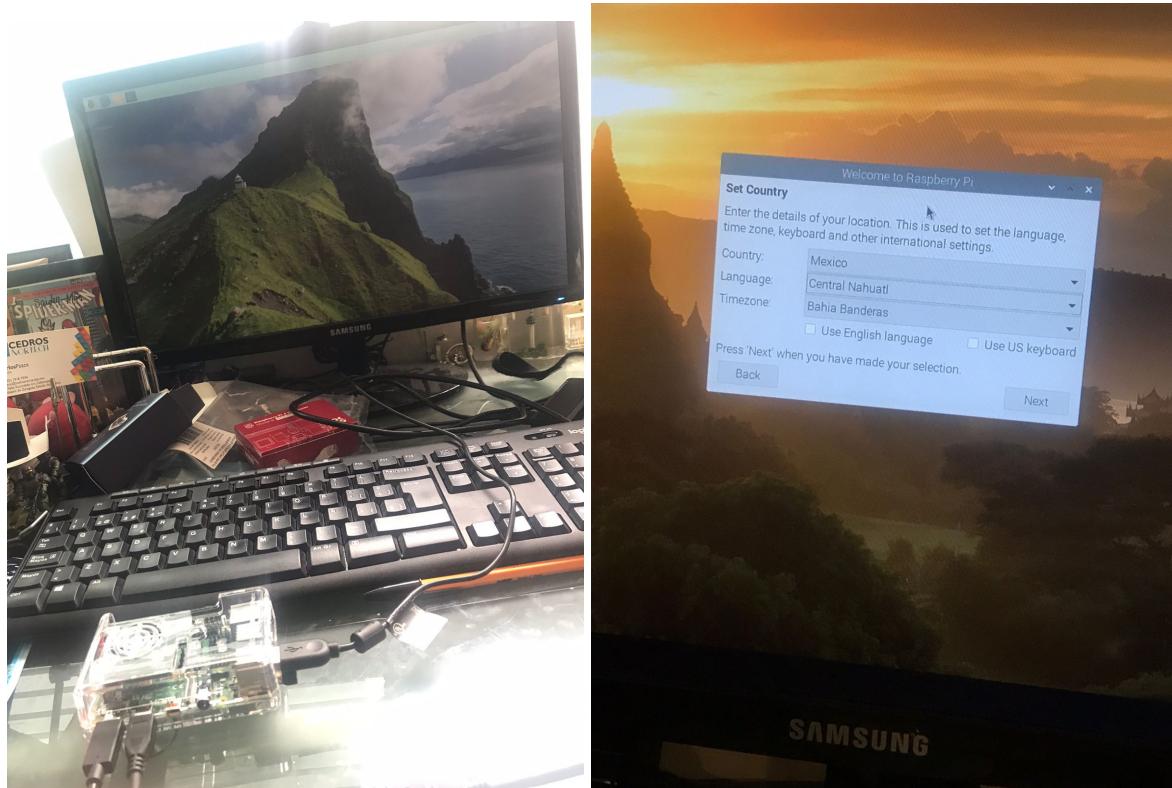
#### **Video demostrativo**

[https://drive.google.com/file/d/1qXrfTqkGPPuiLBiqEh8GG5Z5BhD6E\\_w1/view?usp=sharing](https://drive.google.com/file/d/1qXrfTqkGPPuiLBiqEh8GG5Z5BhD6E_w1/view?usp=sharing)



11 de junio de 2021

## Raspberry Pi



Pudimos también conectar y bajar el sistema mínimo de Raspberry Pi 4 y realizarle configuraciones mínimas de funcionamiento. Exploramos un poco con los comandos de consola de Linux Debian y estamos listos para la siguiente etapa.

Le interfazamos cosas muy sencillas como el mouse, teclado, cámara, y un dispositivo bluetooth. Básicamente se hace como en una laptop cualquiera. Probamos el Thonny de python y todo parece funcionar correctamente.





11 de junio de 2021

Para continuar, estamos buscando cómo se hace para interfazar estas bocinas desde el cableado puro con transductor, y empezar a hacer la conexión de Raspberry Pi con nuestro Arduino y periféricos.

Se nos ocurre que el funcionamiento extra que tenemos pensado lo hagamos con base en el control remoto, donde todas las funciones extras las programemos con el botón MODE, como inicialmente reproducir audio y mostrar la información de la canción, picar MODE y ahora muestra los datos del archivo de audio, y MODE otra vez sería para puntuar la canción con los mismos botones del control.

Sobre la reproducción queremos buscar si podríamos lograr algún tipo de reproducción de audio desde bluetooth puesto que hemos visto en clase que los sistemas embebidos son muy cortos en memoria, entonces así romper esta limitante.



## Etapa 3

### Prototipado, Retroalimentación y Diseño Final

Ya teniendo los componentes esenciales con pruebas básicas, podemos empezar a interfazarlos.

Después de conocer todos los materiales proporcionados en el kit, decidimos que la idea central del proyecto sería tener funciones útiles para todos los botones dentro del control remoto IR. La contraparte es que, debemos tener una interfaz gráfica portátil, como el display LCD. Contemplamos y encuestamos a familiares y amigos para conocer las diferentes perspectivas de lo que creen que es útil y práctico, y en definitiva hubo una opinión positiva hacia interfaces gráficas de pantalla táctil, por lo que decidimos implementar un display táctil de 3.5 pulgadas para la Raspberry Pi.



Entonces para aprovechar la variedad de hardware, podemos tener varias formas de interactuar con el reproductor. Usamos la interfaz táctil para algunas funciones, y nuestros componentes conectados como periféricos a través del Arduino para controlar otras funcionalidades.

Dentro de estas mismas investigaciones acerca de la usabilidad del sistema, los controles mediante sensores ultrasónicos son muy poco prácticos, sobretodo para el conductor, por lo que decidimos cambiar el control de volumen por un encoder rotacional, que es de mayor precisión que un potenciómetro y es fácil de usar para los usuarios ya sea conductor o copiloto.



De la misma forma, hubo consenso en la practicidad que el conductor tenga los tres botones esenciales de control de audio: canción anterior/rebobinar, play/pausa, y siguiente, lo más cercanos al volante.

Para complementar la interfaz gráfica táctil, decidimos obtener los datos básicos del ambiente para el usuario, que son el día y hora, y temperatura, humedad e índice de calor, con un módulo RTC DS1307 y sensor DHT11 respectivamente.

Este proyecto utilizó como microcontrolador el Arduino Mega 2560 y como microprocesador una Raspberry Pi 4B conectados de forma Serial. Esto por el simple hecho de que es una forma muy sencilla y eficiente de comunicación entre dos dispositivos, además de que nos permitió dedicarle más tiempo a otros detalles de la entrega. Cabe recalcar que el proyecto se puede realizar con otros microcontroladores, solo se tendrían que adaptar los pines y los códigos.

La comunicación serial es eficiente para este caso por su sencillez de uso.. Simplemente descargamos el programa de nuestra computadora a Arduino y conectamos su puerto serial USB a la Raspberry Pi. A diferencia del bloque de IoT, este sistema no necesita conexiones a internet, por lo que la comunicación serial es suficiente. Además, ambos Arduino y Python (ya instalado en Raspberry Pi 4B) cuentan con librerías de soporte para comunicación serial.

#### Ventajas de utilizar el Arduino Mega 2560 como microcontrolador

- Fácil de usar
- Gran cantidad de librerías
- Programación sencilla para los sensores
- Espacio de memoria grande para el proyecto

#### Desventajas de utilizar Arduino como microcontrolador

- Ocupa mucho espacio
- Muchos sensores pueden complicar las conexiones y distribución de energía

Entonces aprovechando las ventajas de cada uno, usamos el microprocesador Raspberry Pi para realizar el procesamiento central, es decir toda la lógica del programa incluyendo el procesamiento de datos, control del audio, funciones, interactividad y display de interfaz, datos de archivos, lecturas de datos de archivos, etcétera. Mientras el Arduino microcontrolador se encarga de mandar los datos de los periféricos para su procesamiento bajo un formato específico referenciando {sensor:valores} para el reconocimiento ordenado en Python.

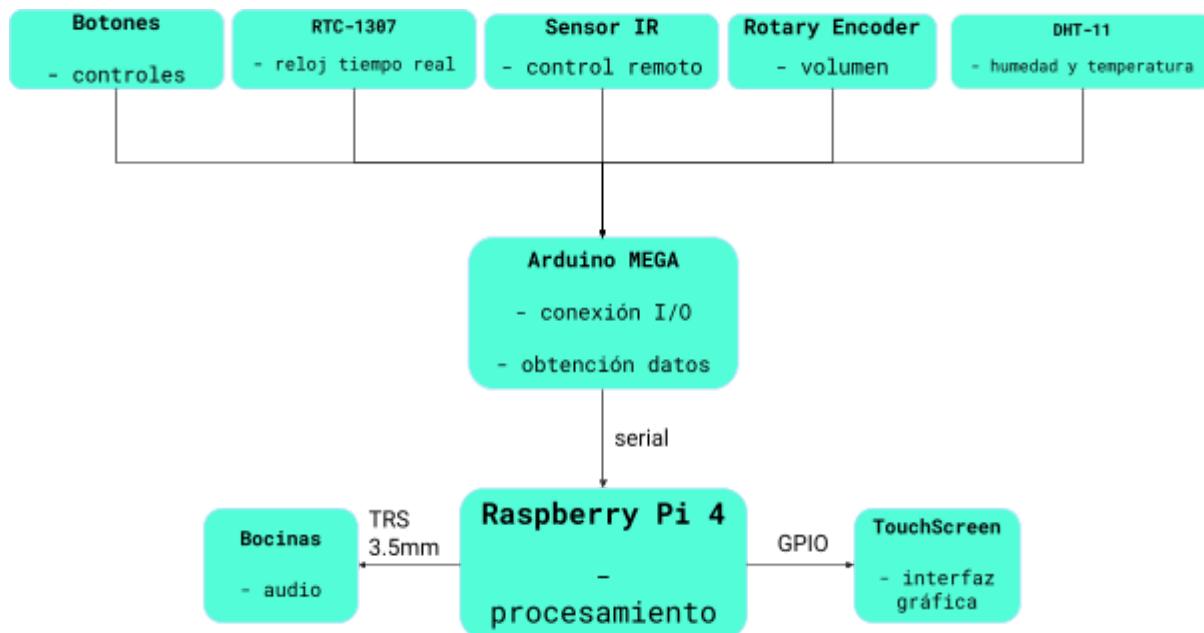


11 de junio de 2021

A grandes rasgos, nuestro proyecto consiste en la recepción de datos mediante 5 sensores y actuadores, que se interfazan como periféricos mediante el Arduino MEGA, el cual obtiene todos los datos y se encarga de mandarlos mediante conexión serial bajo el formato especificado. Entonces la Raspberry Pi funciona para el procesamiento central del sistema. Recibe la información de forma serial y procesa los datos para llevar a cabo las funciones programadas. Esencialmente, el script de Python en la RPi se encarga de procesar y llevar a cabo el control y manejo del audio al igual que el procesamiento y obtención de datos de cada canción, playlist y archivos; mientras se encarga de procesar, desplegar y recibir información en la interfaz gráfica.

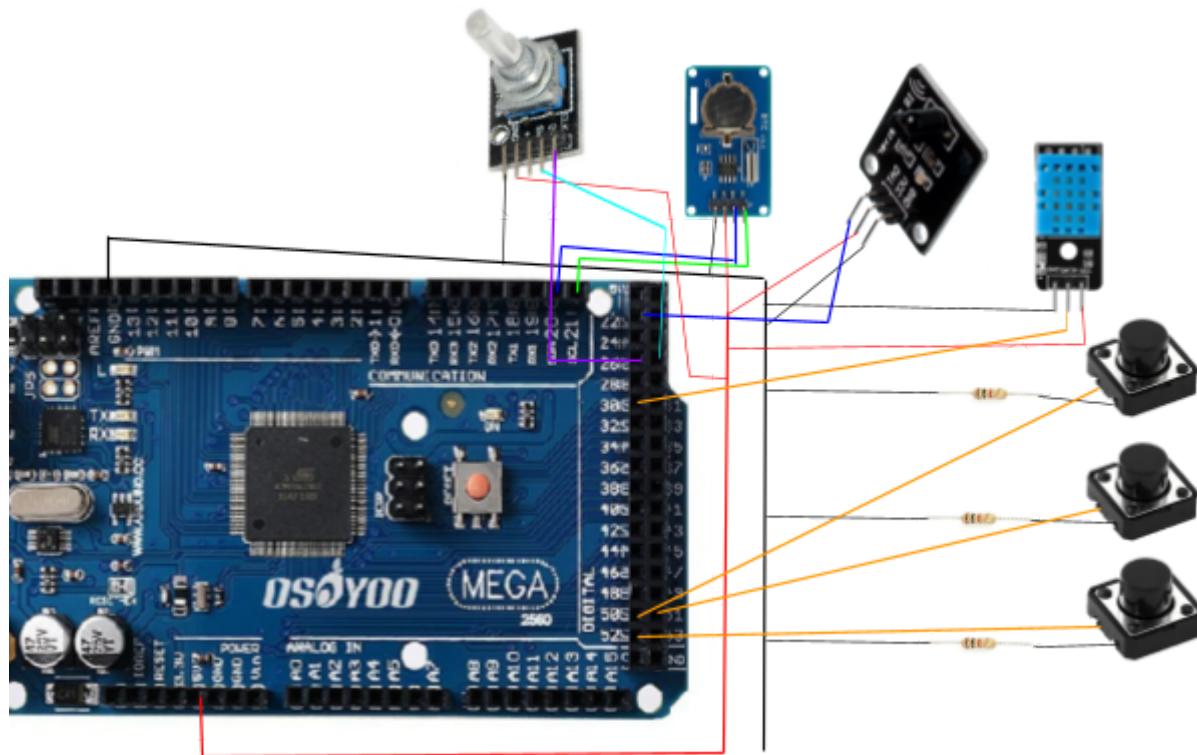
Aprovechamos los puertos de la Raspberry Pi, mediante conexión serial USB se conecta para la transmisión y recepción de información con el Arduino; mediante entrada de audio tipo TRS 3.5 mm se conectan las bocinas para salida de audio comandada en el script; y los puertos GPIO para la conexión de la interfaz táctil.

#### Diagrama de bloques proyecto completo

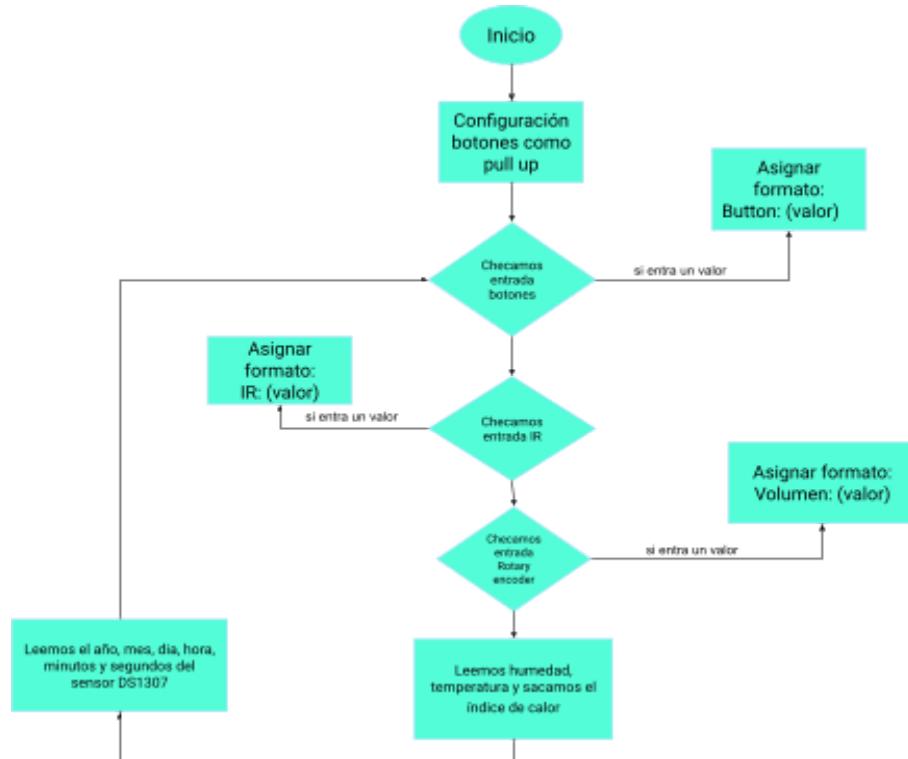




### Diagrama de conexiones sensores Arduino



### Diagrama de bloques código final arduino



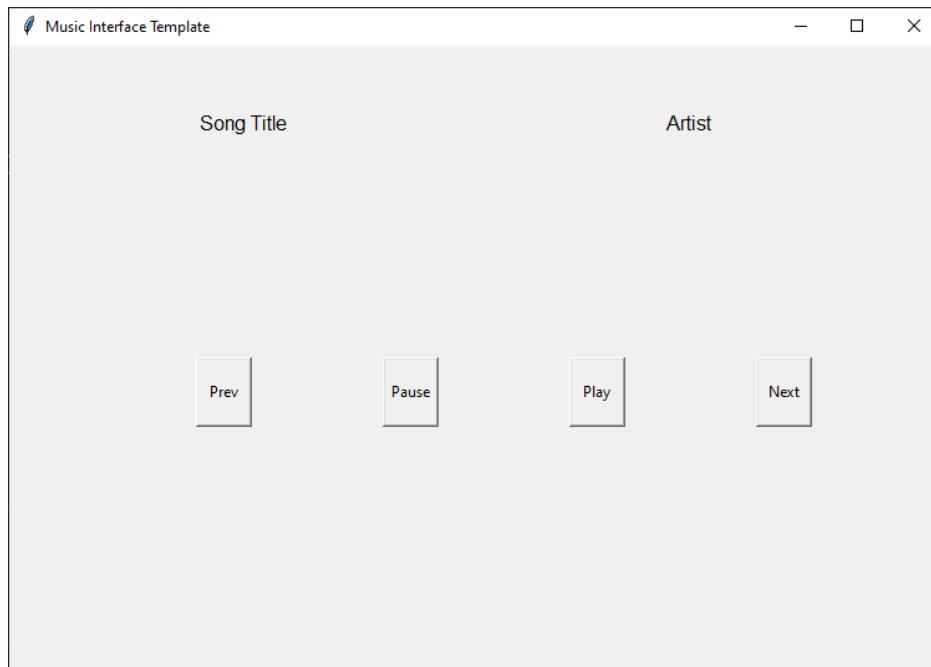


11 de junio de 2021

El código de Arduino final se muestra en anexo (3).

Probamos todos los sensores, primero uno por uno, luego todos en conjunto. Básicamente es fijar las conexiones físicas, en código importar las librerías pertinentes alrededor de cada sensor, llamar a las funciones mediante condicionales de recepción de datos, y mandar la información a través del puerto serial, bajo el formato especificado. Por ejemplo, con el módulo RTC DS1307 de tiempo real, se obtiene el día y la hora a cada minuto, y aprovechamos ese mismo minuto para actualizar también los datos del DHT11, y enviar toda la información del DS1307 en una sola línea, y toda la información del DHT11 en otra sola línea en serial. Ya teniendo toda la parte de periféricos completada, nos dedicamos a desarrollar el procesamiento de audio e interfaz táctil.

Empezamos a desarrollar las primeras pruebas del diseño básico de la interfaz gráfica. Primero completamos el sistema mínimo que solamente mostrara el nombre y artista de la canción, con tres botones básicos de control. Nosotros empezamos con cuatro botones.



Solamente se trataba de un primer acercamiento hacia el sistema mínimo: tener un template básico a seguir y aumentar en su funcionalidad. La interfaz se desarrolla con la librería tkinter en Python. En adelante, trabajamos en simultáneo desarrollando las diferentes funcionalidades. Primeramente, intentamos trabajar el control y procesamiento de audio con una librería conocida como omxplayer wrapper, que básicamente, mediante comandos de sistema de Linux, se controla el sistema de audio desde Python, pero la librería tenía muchos bugs aún y problemas para nuestro proyecto, por lo que decidimos optar por la librería pygame, que es



11 de junio de 2021

más sencilla de usar, cuenta con grandes cantidades de documentación y ejemplos, y ya viene instalada con Python en nuestra Raspberry Pi.

Decidimos no utilizar pygame para desarrollar la interfaz gráfica, porque pygame no cuenta con cosas como botones o deslizadores, entre otros varios elementos con los que ya habíamos trabajado para crear APIs simples como en Matlab. Tkinter, por su parte, sí ofrece todas estas funcionalidades y tiene grandes cantidades de documentación y ejemplos, al igual que ya viene instalada con Python, y es muy similar al desarrollo en código de Matlab en este aspecto.

Lo siguiente fue conectar el módulo mixer de pygame con nuestra interfaz gráfica, al mismo tiempo que, en vez de correr la interfaz en un sólo código directo, creamos una clase llamada MusicInterface, que recibe un Frame de tkinter para su construcción, en la que llevamos a cabo todo el procesamiento. De esta forma es más sencillo conseguir variables globales, pues tenemos variables de clase. El módulo mixer de pygame es muy sencillo de usar, pues sus funciones son muy directas, por ejemplo pygame.mixer.music.play() es la función que inicia la reproducción del archivo de audio cargado previamente. Entonces creamos funciones para cada botón en tkinter, que funcionara como comando de respuesta en el control de reproducción. Redujimos los botones a los tres básicos: previous, play/pause, next, y le programamos el control de audio con el mixer de pygame. Otra ventaja de este módulo, es que el audio lo reproduce en un segundo plano, por lo que no tuvimos problemas para compartir los recursos para el audio, y éste sonara ininterrumpidamente. Eventualmente implementamos que al finalizar cada canción, se levante un evento de finalización de reproducción de audio, para llamar a la función next() de la clase para reproducir la siguiente en la playlist.

Para obtener los archivos de audio, utilizamos la librería os para Python, que nos permite realizar comandos de Linux, como si fuera la consola. Colocamos todos los archivos organizados dentro de directorios en las rutas clave a buscar, para llevar a cabos las instrucciones ls y listdir. Eventualmente en el proyecto, decidimos ordenar esta lista por fecha de acceso, como usualmente se hace en apps como Spotify o Apple Music para crear playlists.

Las complejidades empezaron cuando comenzamos a procesar las varias interfaces en el programa para funcionar simultáneamente. Para conectar de forma serial el Arduino con la Raspberry Pi, usamos la librería pyserial, la cual conecta a partir del puerto USB de RPi. Son un par de líneas solamente para hacer la conexión, y usar un comando de lectura de lo enviado por el Arduino para crear variables de Python.

La cuestión llegó cuando tuvimos que implementar las funciones multihilo en nuestro programa. El Arduino funciona de forma independiente al script de Python, pero



11 de junio de 2021

deben tener respuestas síncronas, por lo que con la librería threading en Python, abrimos un nuevo hilo para hacer polling (función que se llama cada 100ms) con los datos enviados en el puerto serial, para que si llega alguna línea bajo el formato especificado, se procese la información correspondientemente. Al principio solamente hicimos los botones para cambiar canción, que funcionaban correctamente, pero empezó a haber fallos cuando actualizábamos la información en la interfaz gráfica, porque tkinter no es amigable con multihilo, por lo que nos arrojaba errores que el hilo de Arduino no puede realizar cambios en el hilo centrar en el cual corre la interfaz táctil.

Para resolver el problema, con la recomendación del Profesor Francisco de usar colas, retomamos los conocimientos aprendidos en el módulo de Sistemas Operativos en Tiempo Real para implementar colas y semáforos. En el módulo, usamos freeRTOS en Arduino, pero no en Python, pero para obtener resultados semejantes usamos colas y banderas como semáforos. Para hacer la conexión con la interfaz gráfica táctil, desarrollamos otra clase llamada GUI, en la cual solamente procesamos información para actualizar la interfaz visual, sin que haya conflicto entre hilos. Dentro de nuestra clase MusicInterface, realizamos el procesamiento central, el backend, y en la clase GUI, el frontend. Dentro de MusicInterface, instanciamos una cola que luego servirá como atributo para la construcción del objeto GUI que se instancia en el constructor de MusicInterface. Al instanciar el GUI, también mandamos todas las funciones que corresponden a la funcionalidad de los botones dentro de la interfaz gráfica para que tanto el hardware Arduino como la interfaz táctil tengan las mismas funciones y se tenga una respuesta asíncrona. Al tener en ambas clases la misma cola, el Arduino, manda desde su propio hilo, la información que recibe de los periféricos, se procesa como strings mediante indexado y delimitadores, y con una gran cantidad de condicionales se decide qué datos corresponden a qué sensor, y finalmente, o llaman directamente las funciones en backend (como lo son los botones), o se mandan a la cola para que se actualice la interfaz gráfica, como lo son la gran variedad de funcionalidades implementadas con el control remoto de IR. Al igual que las colas, implementamos muchas variables de estado para que asemejen el funcionamiento RTOS como semáforos e interrupciones, que básicamente las funciones del Arduino, muchas se considerarían como interrupciones, como lo son oprimir los botones de control de las canciones.

Por ejemplo, tenemos una función backend para cerrar todas las instancias del mixer, de todos los hilos, de la raíz tkinter, etc (para finalizar todos los procesos correctamente), que se llama cuando gráficamente se cierra la ventana, pero también mediante un botón en el control remoto. Al oprimirse, manda su señal en el sensor IR del Arduino, bajo el formato IR:OFF, que el hilo del script de Python lo recibe al hacer polling cada 100ms, recibe el string desde el puerto serial y con sus



11 de junio de 2021

respectivos condicionales llega a determinar que se trata de llamar a la función de `Quit()`, pero llamarla directamente genera errores de hilo, por lo que se implementó una variable de clase que funciona como bandera, declarada `False` en el constructor, pero que cuando se oprime el botón OFF del control remoto, se coloca en `True`, y dicha variable tiene una condicional en la misma función de polling, para que cuando sea `True`, se llame a la función `Quit()` y se termine el programa.

Simultáneamente desarrollamos una función que utiliza la librería `subprocess`, para poder acceder a los comandos de consola `mediainfo`, la cual nos otorga toda la información del archivo en reproducción, que sirve para actualizar los datos en la interfaz gráfica y presentar la información completa del archivo, tanto de la canción como del archivo. Obtuimos todos los datos básicos como artista, álbum, año de lanzamiento, etcétera, pero también obtuvimos los datos del archivo como el número de canales (estéreo o mono), el bit rate, sample rate, etcétera. Esta función también genera problemas en el hilo al interactuar con el hilo de Arduino (hilo no central). Entonces, por ejemplo, cuando presionamos `next` con el control remoto, el script recibe la información de forma serial, procesa el string recibido, y los condicionales filtran la información para determinar que se trata de llamar a la función de clase `next()`, la cual cambia de canción el audio y actualiza las variables de estado como pausado, y carga la siguiente canción en la playlist y la reproduce en el mixer, pero también llama a esta función `info()` que obtiene todos los datos del archivo de audio, que generaría un error de hilo si no se llama mediante el hilo central, por lo que toda esa información que debe presentarse en el GUI, se manda a la cola, también bajo un formato específico, para procesarse, y actualizar la información de los widgets `tkinter`.

Otro ejemplo, también es hacer la conexión entonces, de los botones gráficos y eventos del GUI con el backend. Para eso, al crear el objeto GUI en el constructor de `MusicInterface`, se mandan como parámetros todas las funciones que se deben realizar cuando un botón virtual se oprime correspondientemente, ya solamente se crean los objetos widgets y se le mandan las funciones como parámetros.

Implementamos simultáneamente las funciones de volumen, puntuación de canciones y finalmente agregar canciones mediante USB. El funcionamiento de volumen, corresponde a recibir datos mediante el encoder rotacional desde Arduino para sumar o restar 1 punto al volumen, o el mismo funcionamiento pero desde los botones + y - del control remoto, pero también el botón mute del control remoto para silenciar o volver al volumen anterior, mismo que se puede hacer desde el botón gráfico en la interfaz táctil. Inicialmente, el volumen es del 50%, en una variable que puede ir de 0 a 100.



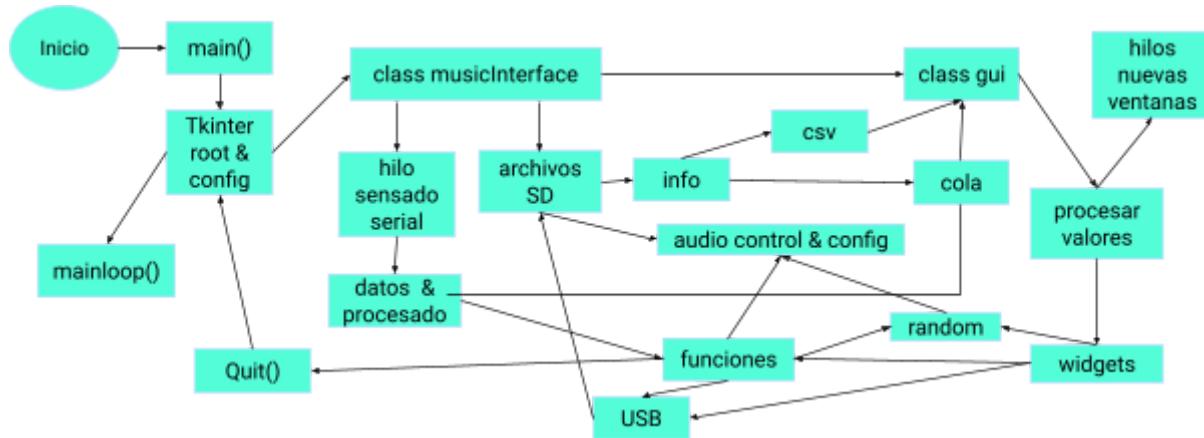
11 de junio de 2021

La función de puntuar canciones, es la única función exclusiva del control remoto puesto que es el hardware que cuenta con 10 botones de números, que pueden dar una puntuación de 0 a 9 a cada canción. Requiere de la librería csv para Python. Para la implementación, en el backend, se recibe el número del control IR, se procesa y se llama a una función que abre (o crea si aún no existe) un archivo .csv, para leer los datos y contrastarlos con los que ya se tienen en la playlist en curso, al igual que contrastar las puntuaciones y si existe algún cambio por parte del usuario, actualizar el archivo, y finalmente al terminar de usar la aplicación, tener todos los datos guardados a permanencia en el csv para ocasiones futuras. Esta misma puntuación se manda a la función que obtiene la información para mandarse a la cola y poder desplegarse actualizada en el GUI. La cuestión más difícil ocurrió cuando el archivo csv no se actualizaba o mandaba error debido a que agregábamos o quitábamos canciones de la playlist, situación que resolvimos con otras dos funciones que se dedican a comparar las playlist en caso de agregarse, u otra playlist para el caso de canciones eliminadas.

Esta misma función, se complejizó un poco cuando implementamos los botones tanto en la interfaz gráfica como con el control remoto de poner la playlist en modo aleatorio (usando la librería random), o regresar a modo ordenado, pues en ese momento el csv puntuaba la canción errónea. Lo pudimos resolver mediante condicionales e indexado.

Finalmente para agregar canciones de forma sencilla, implementamos un botón en la interfaz gráfica pero también botón en el control remoto, que cuando se conecta una USB a la Raspberry Pi (si no hay USB despliega el aviso de USB faltante), se oprime el botón y se agreguen canciones a la playlist en automático para reproducirse sin necesidad de dejar de pausar la reproducción ni algún otro proceso. Cuando las canciones se agregan correctamente, se despliega el mensaje. Para implementar esta función, utilizamos la librería pyudev, y ajustamos un script de Python que nos ayuda a identificar cuando un dispositivo del tipo memoria se detecta en los puertos USB, y obtener los datos del dispositivo y la ruta en la cual se encuentra para poder copiar los archivos a la playlist. Para facilidad de la identificación de ruta y copia de archivos, el script monta el dispositivo en la misma ruta del directorio donde se está corriendo el programa, y se desmonta al cerrar la aplicación para poder extraer la memoria sin problemas.

## Diagrama de bloques script final Python



El script de Python fue la parte de procesamiento para centrar nuestro proyecto en uno que sea multiusuario, en el cual tenga varias interfaces, física y virtual para que todos puedan acceder al audio y modificar cosas simultáneamente.

Ya obteniendo el funcionamiento que deseamos, al final solamente nos dedicamos a generar una interfaz gráfica amigable y fácil de usar. Estábamos conscientes que tkinter no es la librería más famosa por sus interfaces llamativas y coloridas, en comparación con kivy u otras similares, pero no se trata de un proyecto de diseño gráfico, y nosotros nos centramos en un aplicación que destaque en hardware mezclando un poco con software. Tkinter es muy buena para procesamiento, y tomamos la decisión de continuar con tkinter, por las razones antes expuestas, pero también porque nuestro display táctil es de tan sólo 3.5 pulgadas, que en realidad no es mucho como para explotar con un interfaz gráfica de punta.

Decidimos meterle entonces, una interfaz gráfica centrada en la facilidad de uso, un poco minimalista, que despliegue la información de la mejor manera. En realidad llegamos a tener mucha información. Decidimos que para los botones, todos salvo el botón de ver el csv con las puntuaciones, todos deberían tener un pequeño ícono y texto explicativo que intuitivamente pueda entenderse la funcionalidad. Lo bueno, es que los íconos de un reproductor de audio son casi universales y fácilmente reconocibles, como el botón de play/pausa o el de random. Decidimos que debemos desplegar el nombre de la canción y artista en negritas para resaltar, y que toda la información que obtuvimos con la función de info, como el sample rate, el álbum o el puntaje, se despliegan mediante un botón en forma de pop up, al igual que el botón de puntajes despliega el csv con la playlist completa y sus respectivos puntajes, también en un pop up. El botón de usb, debe desplegar un aviso de error cuando se oprima sin detectar una memoria USB conectada, y un mensaje de éxito cuando las canciones se hayan agregado correctamente. Agregamos un display de volumen, que indica el volumen actual, y un botón de mute y unmute al lado. Los tradicionales



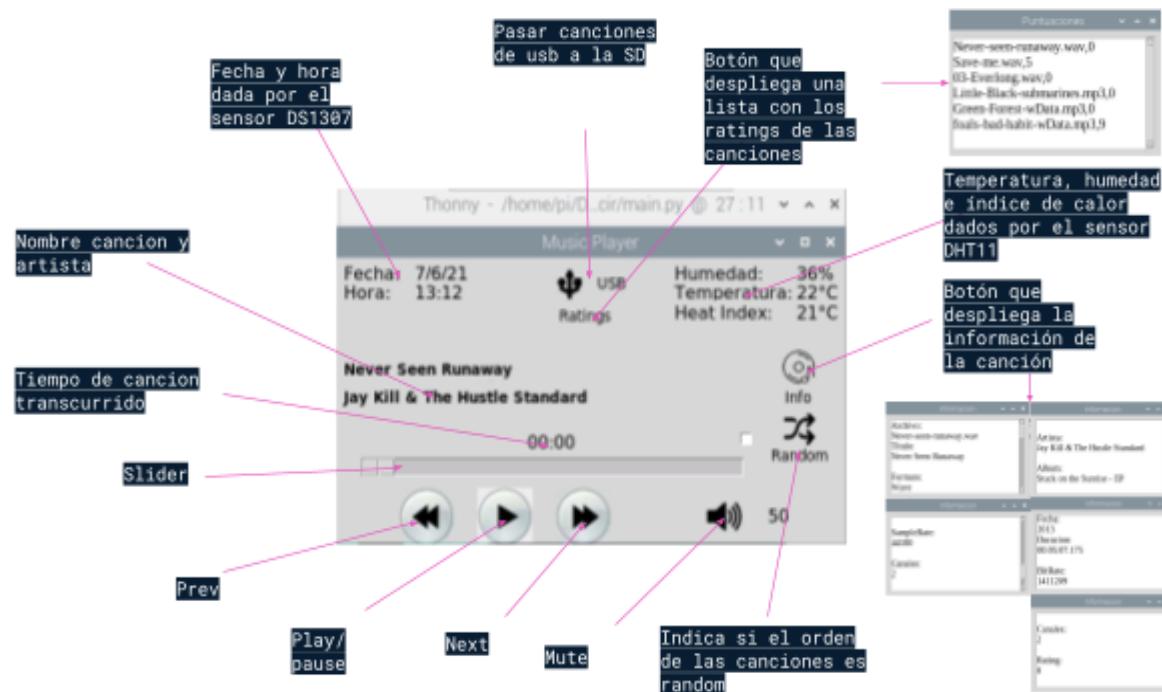
11 de junio de 2021

botones de prev, play y next en serie como íconos, y un slider visual con el tiempo de corrido de la canción que avance a la par del transcurso de la canción. Finalmente, despegamos hasta arriba la fecha, hora y datos del exterior.

Ordenamos los widgets de la siguiente forma:



### Funcionalidades de interfaz gráfica





## Funcionalidades del control remoto IR



## Resumen de uso de librerías

Librería	Uso
tkinter	implementación interfaz gráfica
pygame	módulo mixer para audio
os	comandos Debian con python
random	randomizar la playlist
serial	comunicación serial con Arduino
time	procesamiento tiempo y control de flujo
subprocess	obtención metadata de archivos
threading	manejo hilos sensado y popups
queue	sincronía GUI y procesos
csv	lectura/escritura datos csv
pyudev	procesamiento datos de memoria USB
Librería	Uso
IRremote.h	datos de control remoto IR
Wire.h	lectura datos módulo RTC
DHT.h	lectura datos módulo DHT-11



11 de junio de 2021

Los scripts de Python se presentan en el anexo (4): script Python main.py, (5): script Python musicInterface.py [contiene las clases MusicInterface y GUI], (6): script Python soporte para función de agregar canciones por USB.

Video demostrativo del sistema:

<https://youtu.be/0I5TJeVoINs>

Para el Arrival, tuvimos dos videos para evaluación:

- video técnico con demostración:  
<https://www.youtube.com/watch?v=0tdKPMAL0zY>
- video pitch con demostración:  
<https://www.youtube.com/watch?v=zWv9Se78EYc>

Adicionalmente, la presentación virtual en vivo tuvo como fecha 10 de junio de 2021.

Con base en los criterios que habíamos establecido en la etapa 1, aún no recibimos una calificación al respecto, pero los criterios de retroalimentación y efectividad de implementación de adicionales ha sido muy buena. El sistema no nos presentó errores para grabar los videos ni para presentarlo en directo virtual. El criterio de retroalimentación ha sido el más importante, pues en nuestras casas, usuarios y personas a quienes les enseñamos el sistema, nos dieron su opinión y han sido muy buenas las críticas. La retroalimentación de nuestros Profesores fue muy buena en la presentación virtual, pues los tres nos comentaron que muy buen trabajo y buena y eficiente presentación. Pero sobre todo, notamos que dos o tres equipos, decidieron implementar también funcionalidades del control IR, cosa que no tenían en la presentación grupal pasada, lo cual nos da buen presentimiento porque éramos los únicos quienes lo teníamos implementado, ahora expandido en funcionalidad. Parafraseando la retroalimentación de uno de nuestros Profesores, “la funcionalidad de control remoto, solamente espero a que salga al mercado para adquirirlo”.

El único criterio que pudo quedar mejor fue el diseño gráfico de la interfaz, porque intuitiva nos retroalimentaron que sí es. Teníamos conciencia de que debíamos mejorar el aspecto, y en este documento teníamos planeado esto para la semana 5, sin embargo no sabíamos que el reto se entregaría el lunes en vez de jueves o viernes.

Cerramos la materia con un proyecto que realmente creemos que es muy bueno, quedamos satisfechos con el desempeño, trabajo en equipo, y grandes aprendizajes que integramos en estas semanas.



## Anexos

### Anexo (1): código Arduino keypad, sensor ultrasónico, buzzer, LCD

```
//librerias
#include <Keypad.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//
//Crear el objeto lcd dirección 0x3F y 16 columnas x 2 filas
LiquidCrystal_I2C lcd(0x27,16,2); //
//
// soporte keypad
const byte filas = 3, columnas = 3;
byte pinsFilas [filas] = {22, 23, 24};
byte pinsColumnas[columnas] = {25, 26, 27};

char teclas[filas][columnas] = {
    {'1', '2', '3'},
    {'4', '5', '6'},
    {'7', '8', '9'}
};
// inicializacion de Keypad
Keypad teclado = Keypad(makeKeymap(teclas), pinsFilas, pinsColumnas, filas,
columnas);
char tecla; // tecla oprimida
//Ultrasonico
const int pinEcho = 41;
const int pinTrigger = 40;

int sensor(int pinTrigger, int pinEcho)
{
    long duracionPulso, distancia;

    // Iniciamos el sensor enviando un pulso alto
    // al pin trigger por un lapso de 10uS

    // Prepara el sensor enviando un pulso bajo al trigger
    digitalWrite(pinTrigger, LOW);
    delayMicroseconds(5);

    // Envia el pulso de trigger por 10uS
    digitalWrite(pinTrigger, HIGH);
    delayMicroseconds(10);

    // Finaliza el pulso de trigger
    digitalWrite(pinTrigger, LOW);
```



```
// Tomamos medida del ancho del pulso en el pin echo
duracionPulso = pulseIn(pinEcho, HIGH);

// La lectura del pulso esta en microsegundos
// Obtenemos lectura de distancia en centimetros
distancia = duracionPulso / 58;

return distancia;
}

void setup()
{
    //Ultrasonico
    // Inicia la consola
    Serial.begin(9600);

    // Configura los pines de control del sensor
    pinMode(pinEcho, INPUT);
    pinMode(pinTrigger, OUTPUT);
    //keypad
    //buzzer
    pinMode(28, OUTPUT);
    // Inicializar el LCD
    lcd.init();
    //Encender la luz de fondo.
    lcd.backlight();
}

void loop()
{
    // obtener tecla
    tecla = teclado.getKey();
    if (tecla != NO_KEY) {
        lcd.print(tecla);
    }
    // Iniciamos lectura de la distancia
    int distancia = sensor(pinTrigger, pinEcho);

    // Presenta los resultados
    Serial.print("Lectura de Distancia: ");
    Serial.println(distancia);

    // Espera un segundo y vuelta a empezar
    delay(10);

    if (tecla == '1') {
        tone(28, 440, 100); // (A4 = 440 Hz)
    }
    if (tecla == '2') {
```



11 de junio de 2021

```
tone(28, 340, 100); // (A4 = 440 Hz)
}
if (tecla == '3') {
tone(28, 240, 100); // (A4 = 440 Hz)
}
if (tecla == '4') {
tone(28, 140, 100); // (A4 = 440 Hz)
}
if (tecla == '5') {
tone(28, 40, 100); // (A4 = 440 Hz)
}
delay(100);
lcd.setCursor(0, 1);
```



## Anexo (2): código Arduino control remoto, sensor ultrasónico y LCD

```
#include <IRremote.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//Crear el objeto lcd dirección 0x3F y 16 columnas x 2 filas
LiquidCrystal_I2C lcd(0x27,16,2);
//
const int RECV_PIN = 22;
IRrecv irrecv(RECV_PIN);
decode_results results;
unsigned long key_value = 0;
//Ultrasonico
const int pinEcho = 41;
const int pinTrigger = 40;
int sensor(int pinTrigger, int pinEcho)
{
    long duracionPulso, distancia;

    // Iniciamos el sensor enviando un pulso alto
    // al pin trigger por un lapso de 10us

    // Prepara el sensor enviando un pulso bajo al trigger
    digitalWrite(pinTrigger, LOW);
    delayMicroseconds(5);

    // Envia el pulso de trigger por 10uS
    digitalWrite(pinTrigger, HIGH);
    delayMicroseconds(10);

    // Finaliza el pulso de trigger
    digitalWrite(pinTrigger, LOW);

    // Tomamos medida del ancho del pulso en el pin echo
    duracionPulso = pulseIn(pinEcho, HIGH);

    // La lectura del pulso esta en microsegundos
    // Obtenemos lectura de distancia en centimetros
    distancia = duracionPulso / 58;

    return distancia;
}

void setup(){
    Serial.begin(9600);
    irrecv.enableIRIn();
```



11 de junio de 2021

```
// Configura los pines de control del sensor
pinMode(pinEcho, INPUT);
pinMode(pinTrigger, OUTPUT);
// Inicializar el LCD
lcd.init();
//Encender la luz de fondo.
lcd.backlight();
}
```

```
void loop(){
```

```
// Iniciamos lectura de la distancia
int distancia = sensor(pinTrigger, pinEcho);
```

```
// Presenta los resultados
Serial.print("Lectura de Distancia: ");
Serial.println(distancia);
if (irrecv.decode(&results)){
```

```
    lcd.clear();
```

```
    if (results.value == 0xFFFFFFFF)
```

```
        results.value = key_value;
        lcd.setCursor(0, 1);
```

```
    switch(results.value){
```

```
        case 0xFFA25D:
```

```
            lcd.print("OFF");
```

```
            break;
```

```
        case 0xFF629D:
```

```
            lcd.print("Mode");
```

```
            break;
```

```
        case 0FFE21D:
```

```
            lcd.print("Mute");
```

```
            break;
```

```
        case 0xFF22DD:
```

```
            lcd.print(">=");
```

```
            break;
```

```
        case 0xFF02FD:
```

```
            lcd.print("|<<");
```

```
            break ;
```

```
        case 0xFFC23D:
```

```
            lcd.print(">>|");
```

```
            break ;
```

```
        case 0FFE01F:
```

```
            lcd.print("EQ");
```

```
            break ;
```



11 de junio de 2021

```
case 0xFFA857:  
lcd.print("-");  
break ;  
case 0xFF906F:  
lcd.print("+");  
break ;  
case 0xFF6897:  
lcd.print("0");  
break ;  
case 0xFF9867:  
lcd.print("100+");  
break ;  
case 0xFFB04F:  
lcd.print("200+");  
break ;  
case 0xFF30CF:  
lcd.print("1");  
break ;  
case 0xFF18E7:  
lcd.print("2");  
break ;  
case 0xFF7A85:  
lcd.print("3");  
break ;  
case 0xFF10EF:  
lcd.print("4");  
break ;  
case 0xFF38C7:  
lcd.print("5");  
break ;  
case 0xFF5AA5:  
lcd.print("6");  
break ;  
case 0xFF42BD:  
lcd.print("7");  
break ;  
case 0xFF4AB5:  
lcd.print("8");  
break ;  
case 0xFF52AD:  
lcd.print("9");  
break ;  
}  
key_value = results.value;  
delay(250);  
irrecv.resume();  
}  
}
```



### Anexo (3): código final de Arduino

```
// Autores:  
// Manuel Agustin Diaz Vivanco  
// Carlos Antonio Pazos Reyes  
  
#include <IRremote.h>  
#include <Wire.h>  
#include <DHT.h>  
  
//infra  
const int RECV_PIN = 22;  
  
IRrecv irrecv(RECV_PIN);  
//IRrecv irrecv;  
decode_results results;  
unsigned long key_value = 0;  
  
//Botones play/pause, next, prev  
const int play = 51;  
const int next = 50;  
const int prev = 52;  
  
//sensor DHT11 temp y hum  
const int DHTPIN = 32;  
//Inicializamos el sensor DHTT11  
DHT dht(DHTPIN, DHT11);  
  
//Rotary encoder  
const int outputA = 26; //CLK  
const int outputB = 27; //DT  
const int swch = 28;  
int volumen = 0;  
int aState;  
int aLastState;  
  
//Calendario con horas,min y seg  
// Declaracion de las variables para almacenar informacion de tiempo leida desde  
RTC  
uint8_t second, minute, hour, wday, day, month, year, ctrl;  
int min_ant;  
  
void setup(){  
    Serial.begin(9600);  
    irrecv.enableIRIn();  
    //irrecv.begin(RECV_PIN, ENABLE_LED_FEEDBACK);
```



```
//Botones en pullup para evitar fallos al iniciar
pinMode(play, INPUT_PULLUP);
pinMode(next, INPUT_PULLUP);
pinMode(prev, INPUT_PULLUP);

//sens temp y humedad
dht.begin();

//Rotary encoder
pinMode(outputA,INPUT);
pinMode(outputB,INPUT);
pinMode(swch, INPUT_PULLUP);
//leemos el estado inicial del encoder
aLastState = digitalRead(outputA);

//Setup Calendario con horas,min y seg
// Preparar la librería Wire (I2C)
Wire.begin();

}

void loop(){
    // Iniciamos lectura de la distancia

    // lectura de botones
    if (digitalRead(play) == LOW){
        Serial.println("Button:play");
        //Serial.println("\n");
        delay(500);
    }
    else if (digitalRead(next) == LOW){
        Serial.println("Button:next");
        //Serial.println("\n");
        delay(500);
    }
    else if (digitalRead(prev) == LOW){
        Serial.println("Button:prev");
        //Serial.println("\n");
        delay(500);
    }

    if (irrecv.decode(&results)){
        if (results.value == 0xFFFFFFFF){
            results.value = key_value;
        }
    }
}
```



11 de junio de 2021

}

```
switch(results.value){  
case 0xFFA25D:  
Serial.println("IR:OFF");  
break;  
case 0xFF629D:  
Serial.println("IR:Mode");  
break;  
case 0xFFE21D:  
Serial.println("IR:Mute");  
break;  
case 0xFF22DD:  
Serial.println("IR:>=");  
break;  
case 0xFF02FD:  
Serial.println("IR:|<<");  
break ;  
case 0xFFC23D:  
Serial.println("IR:>>|");  
break ;  
case 0FFE01F:  
Serial.println("IR:EQ");  
break ;  
case 0xFFA857:  
Serial.println("IR:-");  
break ;  
case 0xFF906F:  
Serial.println("IR:+");  
break ;  
case 0xFF6897:  
Serial.println("IR:0");  
break ;  
case 0xFF9867:  
Serial.println("IR:100+");  
break ;  
case 0xFFB04F:  
Serial.println("IR:200+");  
break ;  
case 0xFF30CF:  
Serial.println("IR:1");  
break ;  
case 0xFF18E7:  
Serial.println("IR:2");  
break ;  
case 0xFF7A85:  
Serial.println("IR:3");  
break ;
```



11 de junio de 2021

```
case 0xFF10EF:  
    Serial.println("IR:4");  
    break ;  
case 0xFF38C7:  
    Serial.println("IR:5");  
    break ;  
case 0xFF5AA5:  
    Serial.println("IR:6");  
    break ;  
case 0xFF42BD:  
    Serial.println("IR:7");  
    break ;  
case 0xFF4AB5:  
    Serial.println("IR:8");  
    break ;  
case 0xFF52AD:  
    Serial.println("IR:9");  
    break ;  
default:  
NULL;  
break;  
}  
key_value = results.value;  
delay(250);  
irrecv.resume();  
  
}  
  
//Rotary encoder  
// leemos la posicion actual  
aState = digitalRead(outputA);  
// si el estado anterior y el actual son diferentes, ocurrio un pulso  
if (aState != aLastState){  
    // si el outputB es distinto al outputA, se gira a la derecha  
    if (digitalRead(outputB) != aState){  
        volumen = 1;  
    } else{  
        volumen = -1;  
    }  
    Serial.print("Volumen:");  
    Serial.print(volumen);  
    Serial.println("");  
}  
aLastState = aState;  
  
//Calendario  
// Leer los registros del RTC
```



```
if (read_ds1307()) {
    // Mostrar la fecha y hora
    if ( minute != min_ant){
        print_time();
        min_ant = minute;

        //Leemos la humedad
        int hum = dht.readHumidity();
        int temp = dht.readTemperature();
        //checamos errores en la lectura
        if (isnan(hum) || isnan(temp)) {
            Serial.println("Error obteniendo los datos del sensor DHT11");
            return;
        }
        //sens temperatura y hum
        // Calcular el índice de calor en grados centígrados
        int hic = dht.computeHeatIndex(temp, hum, false);

        Serial.print("DHT:");
        //Serial.print("Humedad:");
        Serial.print(hum);
        Serial.print("%,");
        //Serial.print("Temperatura:");
        Serial.print(temp);
        Serial.print("°C,");
        //Serial.print("Indice de calor:");
        Serial.print(hic);
        Serial.println("°C");
        //Serial.println();

    }
}
else {
    // No se puede leer desde le DS1307 (NACK en I2C)
    Serial.println("No se detecta el DS1307, revisar conexiones");
}

bool read_ds1307()
{
    // Iniciar el intercambio de información con el DS1307 (0xD0)
    Wire.beginTransmission(0x68);

    // Escribir la dirección del segundero
    Wire.write(0x00);

    // Terminamos la escritura y verificamos si el DS1307 respondio
```



```
// Si la escritura se llevo a cabo el metodo endTransmission retorna 0
if (Wire.endTransmission() != 0)
    return false;

// Si el DS1307 esta presente, comenzar la lectura de 8 bytes
Wire.requestFrom(0x68, 8);

// Recibimos el byte del registro 0x00 y lo convertimos a binario
second = bcd2bin(Wire.read());
minute = bcd2bin(Wire.read()); // Continuamos recibiendo cada uno de los registros
hour = bcd2bin(Wire.read());
wday = bcd2bin(Wire.read());
day = bcd2bin(Wire.read());
month = bcd2bin(Wire.read());
year = bcd2bin(Wire.read());

// Recibir los datos del registro de control en la dirección 0x07
ctrl = Wire.read();

// Operacion satisfactoria, retornamos verdadero
return true;
}

/**
 * Esta función convierte un número BCD a binario. Al dividir el número guardado
 * en el parametro BCD entre 16 y multiplicar por 10 se convierten las decenas
 * y al obtener el módulo 16 obtenemos las unidades. Ambas cantidades se suman
 * para obtener el valor binario.
 */
uint8_t bcd2bin(uint8_t bcd)
{
    // Convertir decenas y luego unidades a un numero binario
    return (bcd / 16 * 10) + (bcd % 16);
}

/**
 * Imprime la fecha y hora al monitor serial de arduino
 */
void print_time()
{
    Serial.print("UTC:");
    //Serial.print("Fecha:");
    Serial.print(day);
    Serial.print('/');
    Serial.print(month);
    Serial.print('/');
    Serial.print(year);
}
```



ITESM TE2003B.501  
Manuel Agustin Diaz Vivanco A01379673  
Carlos Antonio Pazos Reyes A01378262

Profesor: Dr. Francisco Javier Ortiz Cerecedo  
Profesor: Dr. Rafael San Vicente Cisneros  
Profesor: Mtro. Luis Humberto Paniagua Sabines

11 de junio de 2021

```
//Serial.print(" Hora:");
Serial.print(",");
Serial.print(hour);
Serial.print(':');
Serial.println(minute);

//Serial.println();
}
```



### Anexo (4): script Python main.py

```
#!/usr/bin/env python3

#Manuel Agustin Diaz Vivanco
#Carlos Antonio Pazos Reyes
#Diseno de sistemas en chip

from musicInterface import MusicInterface
import tkinter as tk

def main():
    root = tk.Tk()
    playlist_path = "/home/pi/Music/"
    root.title("Music Player")
    #root.minsize(1000,750)
    root.minsize(480,320)
    #root.attributes('-zoomed', True)
    #root.resizable(False, False)

    #display App al centro de la pantalla
    #ancho y alto del mainframe de App
    windowHeight = root.winfo_reqheight()
    windowWidth = root.winfo_reqwidth()
    #centro de ancho y alto de ambas pantalla y App
    positionRight = int(root.winfo_screenwidth()/2 - windowWidth*1.2)
    positionDown = int(root.winfo_screenheight()/2 - windowHeight*0.8)
    #centrar el App
    root.geometry("{}x{}+{}+{}".format(windowWidth, windowHeight, positionRight, positionDown))

    app = MusicInterface(playlist_path, root)
    app.mainloop()

if __name__ == "__main__":
    main()
```



## Anexo (5): script Python musicInterface.py [contiene clases MusicInterface y GUI]

```
#!/usr/bin/env python3

#Manuel Agustin Diaz Vivanco
#Carlos Antonio Pazos Reyes
#Diseno de sistemas en chip

#librerias
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import scrolledtext
import pygame as pg
import os, random, serial, time, subprocess, threading, queue, csv
import usbdev

#clase para multi hilo grafico de la interfaz
class GUI:

    #constructor
    def __init__(self, master, queue, prevFunc, playPauseFunc, nextFunc,
randFunc, usbFunc, muteFunc):

        #cola de comunicacion asincrona
        self._queue = queue

        #variables display
        self._valueTitle = tk.StringVar()
        self._valueArtist = tk.StringVar()
        self._valueDate = tk.StringVar()
        self._valueHour = tk.StringVar()
        self._valueHum = tk.StringVar()
        self._valueTemp = tk.StringVar()
        self._valueHeatIdx = tk.StringVar()
        self._random = tk.IntVar(value=0)
        self._volume = tk.StringVar(value="50")
        self._tiempo = tk.StringVar(value="00:00")

        #variable support
        self._pausa = True
        self._contador = 0
        self._val_prev = 0

        #variables de datos
```



```
self._info = {"Archivo":' ', "Titulo":' ', "Formato":' ', "Artista":' ', "Album":' ',  
"Fecha":' ', "Duracion":' ', "BitRate":' ', "SampleRate":' ', "Canales":' ', "Rating":' ',  
"Milis":' '}  
self._dia = {"Fecha":' ', "Hora":' '}  
self._clima = {"Hum":' ', "Temp":' ', "HeatIdx":' '}  
  
#crear widgets  
#labels: datos de cancion  
titleLabel = tk.Label(master, textvariable=self._valueTitle, justify="center",  
font=(("Verdana"),10, "bold"))  
artistLabel = tk.Label(master, textvariable=self._valueArtist, justify="center",  
font=(("Verdana"), 10, "bold"))  
#labels: datos del exterior  
dateLabel = tk.Label(master, text="Fecha:", justify="center",  
font=(("Verdana"),12))  
dateValueLabel = tk.Label(master, textvariable=self._valueDate,  
justify="center", font=(("Verdana"),12))  
hourLabel = tk.Label(master, text="Hora:", justify="center",  
font=(("Verdana"),12))  
hourValueLabel = tk.Label(master, textvariable=self._valueHour,  
justify="center", font=(("Verdana"),12))  
humLabel = tk.Label(master, text="Humedad:", justify="center",  
font=(("Verdana"),12))  
humValueLabel = tk.Label(master, textvariable=self._valueHum,  
justify="center", font=(("Verdana"),12))  
tempLabel = tk.Label(master, text="Temperatura:", justify="center",  
font=(("Verdana"),12))  
tempValueLabel = tk.Label(master, textvariable=self._valueTemp,  
justify="center", font=(("Verdana"),12))  
heatIdxLabel = tk.Label(master, text="Heat Index:", justify="center",  
font=(("Verdana"),12))  
heatIdxValueLabel = tk.Label(master, textvariable=self._valueHeatIdx,  
justify="center", font=(("Verdana"),12))  
#labels: volumen  
volumeValueLabel = tk.Label(master, textvariable=self._volume,  
justify="center", font=(("Verdana"),12))  
#labels: tiempo  
TiempoValueLabel = tk.Label(master, textvariable=self._tiempo,  
justify="center", font=(("Verdana"),12))  
  
#imagenes de los botones  
self._prevBtnImage = tk.PhotoImage(file="images/prevBtn_50x50alpha.png")  
self._pauseBtnImage =  
tk.PhotoImage(file="images/pauseBtn_50x50alpha.png")  
self._playBtnImage = tk.PhotoImage(file="images/playBtn_50x50alpha.png")  
self._nextBtnImage = tk.PhotoImage(file="images/nextBTn_50x50alpha.png")  
self._usbBtnImage = tk.PhotoImage(file="images/25-USB-512_30x30.png")
```



```
self._randomBtnImage =
tk.PhotoImage(file="images/random_30x30alpha.png")
self._vollImage = tk.PhotoImage(file="images/volume_35x35.png")
self._mutelImage = tk.PhotoImage(file="images/mute_35x35.png")
self._songInfoImage =
tk.PhotoImage(file="images/songinfo_30x30alpha.png")

#controles menu
controls = tk.Frame()
controls.place(relx = 0.1, rely=0.8)

#volumen frame
volFrame = tk.Frame()
volFrame.place(relx=0.7, rely=0.83)
self._volumeBtnLabel = tk.Button(volFrame, image=self._vollImage, bd=0,
command=muteFunc)
volumeValueLabel = tk.Label(volFrame, textvariable=self._volume,
justify="center", font=(("Verdana"),12))

#checkbox
randomCheckbox = tk.Checkbutton(master, image=self._randomBtnImage,
borderwidth=0, text="Random", compound="top", variable=self._random, onvalue=1,
offvalue=0, command=randFunc)

#botones
btnPrev = tk.Button(controls, image=self._prevBtnImage, bd=0,
command=prevFunc)
self._btnPlayPause = tk.Button(controls, image=self._playBtnImage, bd=0,
command=playPauseFunc)
btnNext = tk.Button(controls, image=self._nextBtnImage, bd=0,
command=nextFunc)
btnRatings = tk.Button(master, justify="center", bd=0, text="Ratings",
command=self.showRatings, width=5, height=1)
btnInfo = tk.Button(master, image=self._songInfoImage, bd=0, text="Info",
compound="top", command=self.showInfo)
btnUSB = tk.Button(master, image=self._usbBtnImage, bd=0, text="USB",
compound="left", command=usbFunc)

#mostrar items
titleLabel.place(relx=0.01, rely=0.35)
artistLabel.place(relx=0.01, rely=0.45)

dateLabel.place(relx=0.01, rely=0.01)
dateValueLabel.place(relx=0.15, rely=0.01)
hourLabel.place(relx=0.01, rely=0.08)
hourValueLabel.place(relx=0.15, rely=0.08)

humLabel.place(relx=0.66, rely=0.01)
```



```
humValueLabel.place(relx=0.9, rely=0.01)
tempLabel.place(relx=0.66, rely=0.08)
tempValueLabel.place(relx=0.9, rely=0.08)
heatIdxLabel.place(relx=0.66, rely=0.15)
heatIdxValueLabel.place(relx=0.9, rely=0.15)

TiempoValueLabel.place(relx=0.37, rely=0.6)

btnPrev.grid(row=0, column=0, padx=10)
self._btnPlayPause.grid(row=0, column=1, padx=10)
btnNext.grid(row=0, column=2, padx=10)

self._volumeBtnLabel.grid(row=0, column=0, padx=10)
volumeValueLabel.grid(row=0, column=1, padx=10)

btnRatings.place(relx=0.42, rely=0.15)
btnInfo.place(relx=0.85, rely=0.3)
btnUSB.place(relx=0.4, rely=0.01)

randomCheckbox.place(relx=0.75, rely=0.53)

self._song_slider = ttk.Scale(master, from_ = 0, to = 100, orient =
tk.HORIZONTAL,length = 360, value = 0, state="disabled")
self._song_slider.place(relx=0.05, rely=0.7)

#process I/O
def processInputOutput(self):
while self._queue.qsize():
try:
    msg = self._queue.get(0)
    delimiter = msg.find(':')
    if delimiter != -1:
        key = msg[0:delimiter]
        value = msg[delimiter+1:]
        if key == "Info":
            values = value.split("^#^")
            self._info["Archivo"] = values[0]
            self._info["Titulo"] = values[1]
            self._info["Formato"] = values[2]
            self._info["Artista"] = values[3]
            self._info["Album"] = values[4]
            self._info["Fecha"] = values[5]
            self._info["Duracion"] = values[6]
            self._info["BitRate"] = values[7]
            self._info["SampleRate"] = values[8]
            self._info["Canales"] = values[9]
            self._info["Rating"] = values[10]
            self._info["Milis"] = values[11]
```



```
self._valueTitle.set(self._info["Titulo"])
self._valueArtist.set(self._info["Artista"])
elif key == "UTC":
    values = value.split("^#^")
    values = value.split(",")
    self._dia["Fecha"] = values[0]
    times = values[1].split(":")
    if len(times[1]) == 1:
        times[1] = "0"+times[1]
    self._dia["Hora"] = times[0]+":"+times[1]

    self._valueDate.set(self._dia["Fecha"])
    self._valueHour.set(self._dia["Hora"])
elif key == "DHT":
    values = value.split ","
    self._clima["Hum"] = values[0]
    self._clima["Temp"] = values[1]
    self._clima["HeatIdx"] = values[2]

    self._valueHum.set(self._clima["Hum"])
    self._valueTemp.set(self._clima["Temp"])
    self._valueHeatIdx.set(self._clima["HeatIdx"])
elif key == "Error":
    tk.messagebox.showerror("No hay archivos", message="No hay
canciones")
elif key == "IR":
    if value == "100+":
        if self._random.get() == 1:
            self._random.set(0)
        else:
            self._random.set(1)
    elif value == "EQ":
        self.showRatings()
    elif value == "Mode":
        self.showInfo()
    elif value == "USBerror":
        tk.messagebox.showerror("USB Faltante", message="No USB
detectada")
    elif value == "USBcorrect":
        tk.messagebox.showinfo("USB Exitosa", message="Nuevas
canciones agregadas")
    elif key == "BTN":
        if value == "PAUSE":
            self._pausa = True
            self._btnPlayPause.config(image=self._playBtnImage)
        elif value == "PLAY":
            self._pausa = False
```



```
        self._btnPlayPause.config(image=self._pauseBtnImage)
    elif key == "VOL":
        self._volume.set(value)
        if int(self._volume.get()) == 0:
            self._volumeBtnLabel.config(image=self._muteImage)
        else:
            self._volumeBtnLabel.config(image=self._vollImage)
    elif key == "Pos":
        current_time = time.strftime('%M:%S',time.gmtime(int(value)))
        self._tiempo.set(current_time)
        self._song_slider.config(to = int(self._info["Milis"]))
    if self._pausa == False:
        self._contador += 1
        if self._contador == 10:
            next_time = int(self._song_slider.get())+ 1
            self._song_slider.config(value = next_time)
            self._contador = 0
        if self._val_prev > int(value):
            self._song_slider.config(value = 0)
        self._val_prev = int(value)
    except queue.Empty:
        print("Empty Queue")

#display de las puntuaciones
def showRatings(self):
    createPopUp = threading.Thread(target=self.ratingsWindow, daemon=True)
    createPopUp.start()
    createPopUp.join(0.1)

#pop up puntuaciones
def ratingsWindow(self):
    popup = tk.Tk()
    popup.title("Puntuaciones")

    windowHeight = popup.winfo_reqheight()
    windowWidth = popup.winfo_reqwidth()
    positionRight = int(popup.winfo_screenwidth()/2 - windowWidth*2.3)
    positionDown = int(popup.winfo_screenheight()/2 - windowHeight*1.8)
    popup.geometry("+{}+{}".format(positionRight, positionDown))

    with open("Rating.csv") as f:
        # reading file
        Rating = str(f.read())
        # Display whole message
        display = scrolledtext.ScrolledText(popup, width=30, height=7, font=("Times New Roman",15))
        display.pack(padx=10, pady=10)
        display.insert(tk.INSERT, Rating)
```



```
display.configure(state="disabled")
popup.mainloop()

#informacion de la cancion
def showInfo(self):
    createPopUp = threading.Thread(target=self.infoWindow, daemon=True)
    createPopUp.start()
    createPopUp.join(0.1)

#pop up informacion
def infoWindow(self):
    popup = tk.Tk()
    popup.title("Informacion")

    windowHeight = popup.winfo_reqheight()
    windowWidth = popup.winfo_reqwidth()
    positionRight = int(popup.winfo_screenwidth()/2 - windowWidth*2.3)
    positionDown = int(popup.winfo_screenheight()/2 - windowHeight*1.8)
    popup.geometry("{}x{}+{}+{}".format(positionRight, positionDown))

    k = list(self._info.keys())
    v = list(self._info.values())
    info =
        k[0]+":\n"+v[0]+\n+k[1]+:\n"+v[1]+\n+k[2]+:\n"+v[2]+\n+k[3]+:\n"+v[3]+\n+k[4]
        +:\n"+v[4]+\n+k[5]+:\n"+v[5]+\n+k[6]+:\n"+v[6]+\n+k[7]+:\n"+v[7]+\n+k[8]+:
        n+v[8]+\n+k[9]+:\n"+v[9]+\n+k[10]+:\n"+v[10]+\n"
    display = scrolledtext.ScrolledText(popup, width=30, height=7, font=("Times
New Roman",15), background="white")
    display.pack(padx=10, pady=10)
    display.insert(tk.INSERT, info)
    display.configure(state="disabled")
    popup.mainloop()

class MusicInterface(tk.Frame):

    #constructor
    def __init__(self, path, master=None):

        #playlist path
        self._playlist_path = path

        #propiedades de la interfaz
        super().__init__(master)
        self._master = master
        self._master.protocol('WM_DELETE_WINDOW',self.Quit)
        self.pack()

    #comunicacion con arduino
```



11 de junio de 2021

```
self._sensing = threading.Thread(target=self.sensors, daemon=True)
self._arduino = serial.Serial('/dev/ttyACM0',9600,timeout=1)
time.sleep(1)
self._sensorsReading = True

#crear playlist
self._playlist = [file for file in os.listdir(self._playlist_path) if
os.path.isfile(os.path.join(self._playlist_path, file))] #excluye directorios
self._playlist.sort(key = lambda file:
os.path.getmtime(os.path.join(self._playlist_path, file))) #ordena por ultima
modificacion (fecha agregada a SD)

#cola a mandar a GUI
self._queue = queue.Queue()
self._gui = GUI(self._master, self._queue, self.prev, self.playPause, self.next,
self.randomSelect, self.copyUSBfiles, self.mute)

#guardar copia de playlist ordenada
self._playlist_orden = self._playlist

#longitud de playlist
self._playlist_length = len(self._playlist)

#playlist vacia
if self._playlist_length == 0:
    self._queue.put("Error:Empty")
    self._sensorsReading = False
    self._arduino.close()
    self._sensing.join(0.1)
    self._master.quit()
    self._master.destroy()
    print("No songs in directory")

#ordenamiento de reproduccion
self._randomCheck = False

#finalizar asincrono
self._terminate = False

#path csv
self._csvPath = "/home/pi/Documents/Reto/Rating.csv"

#inicializamos un csv con valores default de 0
self.Datos_Rating()

#cancion apuntada
self._current = 0
```



11 de junio de 2021

```
#inicializar mixer
pg.init()
pg.mixer.init()

#tipo de evento al terminar la cancion
self._SONG_END = pg.USEREVENT + 1

#cargar current y pausar y enviar evento de fin de cancion
pg.mixer.music.load(self._playlist_path + self._playlist[self._current])
pg.mixer.music.play()
pg.mixer.music.pause()
pg.mixer.music.set_endevent(self._SONG_END)

#estado de reproduccion
self._playing = False

#estado de volumen
self._unmuted = True

#senales de hardware por recibir y procesar
self._hardware = {"Button": None, "IR": None, "Volumen":50, "UTC":None,
"DHT":None}

#volumen medio
self._volume = 50
pg.mixer.music.set_volume(self._volume/100)
self._queue.put("VOL:"+str(self._volume))

#info de la primera cancion
self.info(self._playlist, self._current)

#usb listener y path support
self._observer = usbdev.startListener()
self._devpath = None

#empezar a sensar asincrono
self._sensing.start()

#empezar a ver en la cola
self.checkQueue()

#llamado periodico a la cola para checar contenidos
def checkQueue(self):
    #llamar a procesar los datos en cola
    self._gui.processInputOutput()

#tiempo actual de la cancion
current_time = int(pg.mixer.music.get_pos())/1000
```



```
self._queue.put("Pos:"+str(current_time))

#manejo de fin de cancion
for event in pg.event.get():
    if event.type == self._SONG_END:
        self.next()

#cierre de programa asincrono
if self._terminate:
    self.Quit()

if self._sensorsReading:
    #cada 100ms
    self._master.after(100, self.checkQueue)

def Datos_Rating(self):
    #abrimos un csv con los titulos de las columnas
    with open('Rating.csv', 'a', newline='') as Rating:
        R = csv.reader(open(self._csvPath))
        #sacamos los valores que ya estan en el csv guardados
        self.Rating_list = list(R)
        #si la lista esta vacia, la inicializamos
        if len(self.Rating_list) == 0:
            for i in range (self._playlist_length):
                self.Rating_list.insert(i,[self._playlist[i],0])
        #chequemos si en la direccion de memoria hay mas archivos que los
        #guardados en la lista
        if self._playlist_length > (len(self.Rating_list)):
            for i in range(self._playlist_length - (len(self.Rating_list))):
                self.agregar()
        #chequemos si en la direccion de memoria hay menos archivos que los
        #guardados en la lista
        if self._playlist_length < (len(self.Rating_list)):
            for i in range((len(self.Rating_list)) - self._playlist_length):
                self.eliminar()
        #corregimos los numeros de la primera variable del csv
        writer = csv.writer(open(self._csvPath, 'w'))
        writer.writerows(self.Rating_list)

    #agregar cancion a csv
    def agregar(self):
        #Ya chequemos y hay mas archivos en la direccion que en la lista
        for i in range(self._playlist_length):
            #insertamos el valor que no coincide
            if i >= len(self.Rating_list):
                self.Rating_list.insert(i,[self._playlist[i],0])
                break
            elif (self._playlist[i] != self.Rating_list[i][0]):
```



```
#insertamos el valor que no coincide
self.Rating_list.insert(i,[self._playlist[i],0])
#hacemos un break para que se agregue y se escriba, para evitar
errores (se agregan uno por uno los archivos faltantes)
break

#eliminar canciones del csv que no esten en el path
def eliminar(self):
#Ya checamos y hay mas archivos en la direccion que en la lista
for i in range(len(self.Rating_list)):
    #insertamos el valor que no coincide
    if i >= self._playlist_length:
        del self.Rating_list[i]
        break
    elif (self._playlist[i] != self.Rating_list[i][0]) :
        #insertamos el valor que no coincide
        del self.Rating_list[i]
        #hacemos un break para que se agregue y se escriba, para evitar
        errores (se agregan uno por uno los archivos faltantes)
        break

#funcion de Rating
def Rating(self,value):
for i in range(len(self.Rating_list)):
if self._playlist[self._current] == self.Rating_list[i][0]:
    self.Rating_list[i][1] = value
writer = csv.writer(open(self._csvPath, 'w'))
writer.writerows(self.Rating_list)

#funcion hilo sensado activo desde arduino
def sensors(self):
while self._sensorsReading:
line = self._arduino.readline().decode('utf-8').rstrip()
if line != "":
    delimiter = line.find(":")
    if delimiter != -1:
        key = line[0:delimiter]
        value = line[delimiter+1:]
        if key in self._hardware.keys():
            self._hardware[key] = value
        if key == "Button":
            if value != None and value == "prev":
                self.prev()
            elif value != None and value == "play":
                self.playPause()
            elif value != None and value == "next":
                self.next()
        elif key == "IR":
```



11 de junio de 2021

```
if value != None and value == "OFF":  
    self._terminate = True  
elif value != None and value == "Mode":  
    self._queue.put(line)  
elif value != None and value == "Mute":  
    self.mute()  
elif value != None and value == ">=":  
    self.playPause()  
elif value != None and value == "|<<":  
    self.prev()  
elif value != None and value == ">>|":  
    self.next()  
elif value != None and value == "EQ":  
    self._queue.put(line)  
elif value != None and value == "-":  
    self._volume -= 1  
    if self._volume < 0:  
        self._volume = 0  
    pg.mixer.music.set_volume(self._volume/100)  
    self._queue.put("VOL:"+str(self._volume))  
elif value != None and value == "+":  
    self._volume += 1  
    if self._volume > 100:  
        self._volume = 100  
    pg.mixer.music.set_volume(self._volume/100)  
    self._queue.put("VOL:"+str(self._volume))  
elif value != None and value == "100+":  
    self._queue.put(line)  
    self.randomSelect()  
elif value != None and value == "200+":  
    self.copyUSBfiles()  
elif value != None:  
    self.Rating(value)  
    self.info(self._playlist, self._current)  
elif key == "Volumen":  
    if value != None:  
        self._volume += int(value)  
    if self._volume > 100:  
        self._volume = 100  
    elif self._volume < 0:  
        self._volume = 0  
    pg.mixer.music.set_volume(self._volume/100)  
    self._queue.put("VOL:"+str(self._volume))  
elif key == "UTC":  
    if value != None:  
        self._queue.put(line)  
elif key == "DHT":  
    if value != None:
```



```
        self._queue.put(line)

#mute de volumen
def mute(self):
    if self._unmuted:
        self._unmuted = False
        pg.mixer.music.set_volume(0)
        self._queue.put("VOL:0")
    else:
        self._unmuted = True
        pg.mixer.music.set_volume(self._volume/100)
        self._queue.put("VOL:"+str(self._volume))

#archivos de la USB
def copyUSBfiles(self):
    status = usbdev.isDeviceConnected()
    if status:
        device = usbdev.getDevData()
        if device != None:
            self._devpath = device["DEVPATH"]
            path = usbdev.getMountPathUsbDevice()
            if path != None:
                os.system("cp " + path + "/*.mp3 " + self._playlist_path)
                os.system("cp " + path + "/*.wav " + self._playlist_path)
                #recrear playlist
                self._playlist = [file for file in os.listdir(self._playlist_path) if
os.path.isfile(os.path.join(self._playlist_path, file))] #excluye directorios
                self._playlist.sort(key = lambda file:
os.path.getmtime(os.path.join(self._playlist_path, file))) #ordena por ultima
modificacion (fecha agregada a SD)
                #reset guardar copia de playlist ordenada
                self._playlist_orden = self._playlist
                #reset longitud de playlist
                self._playlist_length = len(self._playlist)
                #reset current
                self._current = 0
                self._queue.put("IR:USBcorrect")
                self.Datos_Rating()
            else:
                self._queue.put("IR:USBerror")

#funcion de boton prev
def prev(self):
    if pg.mixer.music.get_pos() > 10000:
        pg.mixer.music.load(self._playlist_path + self._playlist[self._current])
        pg.mixer.music.set_volume(self._volume/100)
        pg.mixer.music.play()
    else:
```



```
if self._current == 0:  
    self.info(self._playlist, self._playlist_length - 1)  
    pg.mixer.music.load(self._playlist_path +  
self._playlist[self._playlist_length - 1])  
    pg.mixer.music.set_volume(self._volume/100)  
    pg.mixer.music.play()  
    self._current = self._playlist_length - 1  
else:  
    self.info(self._playlist, self._current - 1)  
    pg.mixer.music.load(self._playlist_path + self._playlist[self._current - 1])  
    pg.mixer.music.set_volume(self._volume/100)  
    pg.mixer.music.play()  
    self._current -= 1  
self._queue.put("BTN:PLAY")  
self._playing = True  
time.sleep(0.5)  
  
#funcion de boton pausa  
def playPause(self):  
if self._playing:  
    pg.mixer.music.pause()  
    self._queue.put("BTN:PAUSE")  
    self._playing = False  
else:  
    pg.mixer.music.unpause()  
    self._queue.put("BTN:PLAY")  
    self._playing = True  
time.sleep(0.5)  
  
#funcion de boton next  
def next(self):  
if self._current == self._playlist_length - 1:  
    self.info(self._playlist, 0)  
    pg.mixer.music.load(self._playlist_path + self._playlist[0])  
    pg.mixer.music.set_volume(self._volume/100)  
    pg.mixer.music.play()  
    self._current = 0  
else:  
    self.info(self._playlist, self._current + 1)  
    pg.mixer.music.load(self._playlist_path + self._playlist[self._current + 1])  
    pg.mixer.music.set_volume(self._volume/100)  
    pg.mixer.music.play()  
    self._current += 1  
self._queue.put("BTN:PLAY")  
self._playing = True  
time.sleep(0.5)  
  
#info de la cancion
```



```
def info(self,playlist,numero):
    nombre_archivo = playlist[numero]
    nombre_cancion = subprocess.check_output ("mediainfo
--Inform=""General;%Title%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
        formato = subprocess.check_output("mediainfo
--Inform=""General;%Format%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
        if "Wave" in formato:
            artista = subprocess.check_output ("mediainfo
--Inform=""General;%Director%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            album = subprocess.check_output ("mediainfo
--Inform=""General;%OriginalSourceForm/Name%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            elif "MPEG Audio" in formato:
                artista = subprocess.check_output ("mediainfo
--Inform=""General;%Performer%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            album = subprocess.check_output ("mediainfo
--Inform=""General;%Album%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            fecha = subprocess.check_output("mediainfo
--Inform=""General;%Recorded_Date%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            duracion = subprocess.check_output("mediainfo
--Inform=""General;%Duration/String3%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            duracion_milis = subprocess.check_output("mediainfo
--Inform=""General;%Duration%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            bit_rate = subprocess.check_output("mediainfo
--Inform=""General;%BitRate%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            sample_rate = subprocess.check_output("mediainfo
--Inform=""Audio;%SamplingRate%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)
            canales = subprocess.check_output("mediainfo
--Inform=""Audio;%Channel(s)%
'+self._playlist_path+playlist[numero],shell=True,universal_newlines=True)

            #procesamiento pre envio
            duracion_milis = int(duracion_milis)/1000
            duracion_milis = int(duracion_milis)
            if " " in fecha[0:4]:
                pos = fecha.find(" ")
                fecha = fecha[pos+1:pos+5]
            else:
```



```
fecha = fecha[0:4]
for i in range(len(self.Rating_list)):
    if self._playlist[numero] == self.Rating_list[i][0]:
        rating = str(self.Rating_list[i][1])

information =
"Info:"+nombre_archivo+"^#^"+nombre_cancion+"^#^"+formato+"^#^"+artista+"^#^"+
album+"^#^"+fecha+"^#^"+duracion+"^#^"+bit_rate+"^#^"+sample_rate+"^#^"+canal
est+"^#^"+rating+"^#^"+str(duracion_milis)

#enviar a cola para procesar I/O
self._queue.put(information)

#modificar variables de random
def randomSelect(self):
    if self._randomCheck == False:
        new_playlist = random.sample(self._playlist, len(self._playlist))
        self._playlist = new_playlist
        self._randomCheck = True
    else:
        self._playlist = self._playlist_orden
        self._randomCheck = False

#funcion al cerrar la ventana de la interfaz
def Quit(self):
    self._sensorsReading = False
    pg.mixer.music.stop()
    pg.mixer.quit()
    pg.quit()
    time.sleep(1)
    usbdev.stopListener(self._observer)
    if self._devpath != None:
        os.system("sudo umount " + self._devpath)
    self._arduino.close()
    self._sensing.join(0.1)
    self._master.quit()
    self._master.destroy()
    print("closed")
```



## Anexo (6): script Python soporte USB usbdev.py

```
from pyudev import Context, Monitor, MonitorObserver
import os
```

```
#some globals for the device details
USBDEV_UUID = None
USBDEV_VENDOR = None
USBDEV_SERID = None
USBDEV_FSTYPE = None
USBDEV_MODEL = None
USBDEV_DEVPATH = None

USBDEV_HAVEDATA = False
```

```
#callback when a usb device is plugged in
def usbEventCallback(action, device):
```

```
    global USBDEV_UUID
    global USBDEV_VENDOR
    global USBDEV_SERID
    global USBDEV_FSTYPE
    global USBDEV_MODEL
    global USBDEV_DEVPATH

    global USBDEV_HAVEDATA

    if action == 'add':
        #store the device values
        USBDEV_VENDOR = device.get('ID_VENDOR')
        USBDEV_SERID = device.get('ID_SERIAL')
        USBDEV_UUID = device.get('ID_FS_UUID')
        USBDEV_FSTYPE = device.get('ID_FS_TYPE')
        USBDEV_MODEL = device.get('ID_MODEL')
        USBDEV_DEVPATH = device.get('DEVNAME')

        USBDEV_HAVEDATA = True

    elif action == 'remove':
        #clear the device data
        USBDEV_VENDOR = None
        USBDEV_SERID = None
        USBDEV_UUID = None
        USBDEV_FSTYPE = None
        USBDEV_MODEL = None
        USBDEV_DEVPATH = None
        USBDEV_HAVEDATA = False
```



```
def startListener():
    # create a context, create monitor at kernel level, select devices
    context = Context()
    monitor = Monitor.from_netlink(context)
    monitor.filter_by(subsystem='block')

    observer = MonitorObserver(monitor, usbEventCallback, name="usbdev")
    #set this as the main thread
    observer.setDaemon(False)
    observer.start()

    return observer

def isDeviceConnected():
    global USBDEV_HAVEADATA
    return USBDEV_HAVEADATA

def getDevData():
    if isDeviceConnected():
        global USBDEV_UUID
        global USBDEV_VENDOR
        global USBDEV_SERID
        global USBDEV_FSTYPE
        global USBDEV_MODEL
        global USBDEV_DEVPATH
        return {'UUID': USBDEV_UUID,
                'SERID': USBDEV_SERID,
                'VENDOR': USBDEV_VENDOR,
                'FSTYPE': USBDEV_FSTYPE,
                'MODEL': USBDEV_MODEL,
                'DEVPATH': USBDEV_DEVPATH}
    return None

def stopListener(observer):
    observer.stop()

#returns the accessible path of the device on the Raspberry pi
#you can change how the path gets calculated.
def getMountPathUsbDevice():
    global USBDEV_DEVPATH
    if not isDeviceConnected() or USBDEV_DEVPATH == None:
        return None

    #check if the dev path exists
    if os.path.exists(USBDEV_DEVPATH):

        #create a mount directory
        if not os.path.exists('mp'):
```



11 de junio de 2021

```
os.makedirs('mp')

#mount the dev path to the folder
os.system("sudo mount " + USBDEV_DEVPATH + " mp")

#return the path to the folder from root
truePath = os.getcwd() + '/mp'

return truePath

return None
```