UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA
Facultad de Ingeniería

Ingeniería en Sistemas Computacionales en Hardware

Proyecto Final: Detector de sirenas

Emilio Alberto Salas Ríos 235712
Oscar Alejandro Serrano Pizarro 338820
Jorge Eduardo Carrasco López 338734

Materia:
Aplicaciones Embebidas para Internet de las Cosas
9HW1

Maestro:
Alberto Pacheco González

30 de noviembre de 2023

CONTENIDO

RESUMEN

El proyecto consiste en la creación de un Sistema Avanzado de Asistencia al Conductor (ADAS) desarrollando una solución para detectar el sonido distintivo de las sirenas de vehículos de emergencia haciendo uso de un microcontrolador ESP32 y un sensor de audio (micrófono). El micrófono se encarga de capturar señales de audio analógicas, y el ESP32 procesa esta información utilizando un código implementado en MicroPython.

En términos generales el proceso consiste en capturar la señal de audio, obtener la frecuencia de los sonidos, clasificar dichas frecuencias, y comprobar si el sonido detectado pertenece a una sirena.

El proceso de detección implica clasificar las frecuencias obtenidas en altas y bajas de acuerdo a un rango especifico establecido, permitiendo así la identificación de patrones característicos de las sirenas de emergencia al analizar y comparar los datos de las muestras. Mediante este enfoque, el sistema puede discernir la presencia de una sirena y alertar al conductor, mejorando así la seguridad vial.

Para facilitar el análisis de los sonidos se utilizó la transformada rápida de Fourier, mediante una función integrada en la librería ulab de MicroPython, la cual permite convertir los valores capturados en frecuencias (Hz).

Los resultados obtenidos fueron satisfactorios al lograr la correcta detección de algunos sonidos de sirenas aunque no de todos, debido a diversos factores como la presencia de ruido, por lo que siguen siendo necesarios algunos ajustes para lograr una mayor eficiencia. Sin embargo, los resultados muestran que es viable y factible técnicamente la solución propuesta, utilizando tecnologías de bajo costo.

INTRODUCCIÓN

Motivación

Actualmente existe una creciente complejidad en el tráfico de ciudades y carreteras por el mayor volumen de vehículos presentes y algunos otros factores, por lo que es cada vez más necesario la implementación de alternativas y soluciones para aumentar la seguridad vial a través de tecnologías innovadoras.

De dicha necesidad surge este proyecto, el cual busca abordar esta problemática mediante la implementación de un sistema ADAS en los vehículos, el cual detecte el sonido de las sirenas de dos tonos, para alertar al conductor que se aproxima un vehículo de emergencia.

La capacidad de los conductores para detectar rápidamente la presencia de vehículos de emergencia, identificados comúnmente por el sonido de sus sirenas, desempeña un papel crucial en la prevención de accidentes y la reducción de tiempos de respuesta en situaciones críticas, promoviendo un entorno de conducción más seguro.

Objetivo

Desarrollar en MicroPython el prototipo de un sistema para la detección de señales acústicas, específicamente sirenas de dos tonos para vehículos de emergencia, utilizando elementos de bajo costo y que sea factible de implementar como un sistema ADAS en los vehículos. Al final, el código MicroPython deberá correr en un microcontrolador ESP32 para comprobar y verificar su funcionamiento mostrando un mensaje que indique la presencia o no presencia de una sirena.

DETECTOR DE SIRENAS

Descripción del proyecto y explicación de su funcionamiento

El proyecto consiste en diseñar un prototipo de un sistema ADAS para los vehículos que sea capaz de capturar las señales acústicas en el entorno para analizarlas y detectar la presencia de sirenas de vehículos de emergencia, alertando al conductor mediante un mensaje en caso de encontrar alguna coincidencia.

Los componentes requeridos para el proyecto fueron un microcontrolador ESP32 y un sensor de audio (micrófono). El micrófono se encarga de capturar la señal analógica de audio y se encuentra conectado al puerto GPIO 34 del ESP32, encargado de procesar la señal de acuerdo a las instrucciones del código implementado en MicroPython.

Más adelante se explicará de una manera más detallada el código utilizado, pero a continuación se brinda una descripción general del funcionamiento del mismo.

Primero se debe configurar el hardware, indicando el puerto del ESP32 al que se conectará el micrófono y configurar el convertidor analógico-digital para que los valores de la señal analógica sean convertidos a valores digitales y podamos procesarlos.

Una vez que se configuró el hardware se procede a la lectura de los datos del sensor de audio mediante la función leer_Microfono(). Los datos obtenidos se almacenan en un arreglo (datos) y se les aplica la transformada rápida de Fourier para encontrar la frecuencia dominante de los sonidos obtenidos.

La frecuencia dominante que se identificó en el paso anterior se debe clasificar en frecuencia alta o frecuencia baja según el rango asociado a las sirenas de emergencias que establecimos. A las frecuencias altas se les reconoce con una letra “A” mientras que las frecuencias bajas se reconocen con una letra “B”. Estas letras se van concatenando a una variable llamada “lecturas” definiendo el comportamiento de la señal de audio.

Luego, en esta variable “lecturas” se busca mediante expresiones regulares repeticiones de letras específicas para ver si concuerda con el patrón de comportamiento de una sirena. En caso de que exista coincidencia con el patrón predefinido en la variable “patron” se imprime un mensaje indicando que se detectó una sirena.

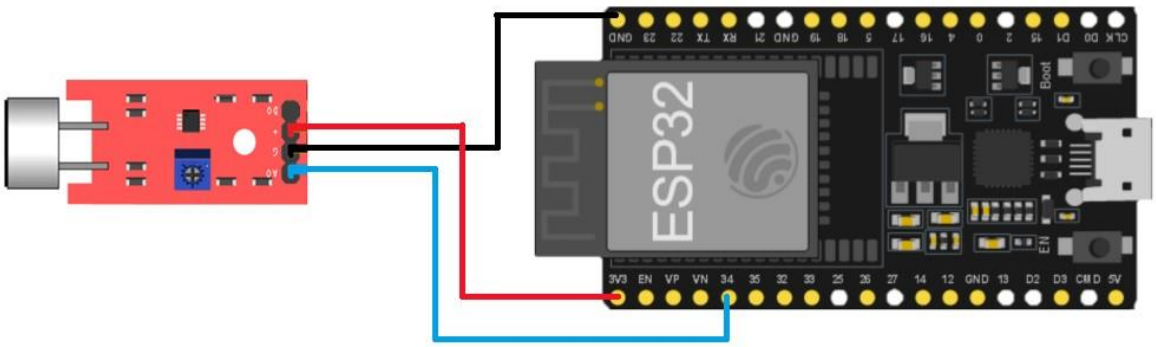
El programa se ejecuta en un bucle continuo, adquiriendo datos del sensor, procesándolos y evaluando la presencia de patrones de sirenas.

Diagrama de conexiones del circuito

El diagrama de las conexiones necesarias para armar el circuito del proyecto es bastante simple. El micrófono utilizado tiene cuatro pines correspondiente a voltaje, tierra, salida analógica y salida digital, de los cuales utilizaremos solo los primeros tres en esta ocasión.

El pin del voltaje del micrófono se conecta al pin de alimentación del ESP32, de donde recibe 3.3 V., mientras que los pines correspondientes a tierra del micrófono y el ESP32 se conectan entre sí. Por su parte el pin de salida analógica del sensor de audio se conecta al GPIO 34 del ESP32 para recibir los datos y comenzar su procesamiento.

A continuación se puede observar un diagrama para visualizar las conexiones de forma gráfica.



Código y descripción del programa

La primera parte del código se encarga de preparar el hardware especificando el puerto del ESP32 que utilizaremos (GPIO 34), y realizando la configuración del convertidor analógico-digital (ADC), integrado en el microcontrolador para la lectura de datos del sensor de audio.

```
#Configuracion de pin
pin_Microfono = Pin(34, mode=Pin.IN)
adc = ADC(pin_Microfono)

#Configuracion del ADC
adc.atten(ADC.ATTN_11DB)
```

Antes de continuar, también se deben configurar ciertos parámetros relacionados con la lectura de datos como la tasa de muestreo y cantidad de lecturas que se toman en cada muestra. En este caso la tasa de muestreo se estableció en 18250 Hz, y el número de muestras en 256, debido a que la tasa de muestro debe ser aproximadamente el doble de la frecuencia máxima que se pretende detectar.

Además, también se declaran dos variable que, en conjunto, sirven para determinar la presencia de una sirena. La primera de estas variables es “lecturas”, la cual almacena una cadena en donde se concatena una letra “A” o una letra “B” dependiendo de si la frecuencia identificada es alta o baja respectivamente. Por tanto en esta variable “lecturas” se puede apreciar el comportamiento que presenta la señal de audio capturada. La otra variable, llamada “patron”, es una expresión regular que determina ciertos comportamiento predeterminados que pueden presentar las sirenas. Al comparar estas dos variables “lecturas” y “patron” se puede identificar la señal capturada corresponde a una sirena o no.

```
#Parametros de lectura de datos
tasa_Muestreo = 18250
cantidad_Lecturas = 256

lecturas = ''   #Clasificación de frecuencias - comportamiento de la señal

#Expresión regular -> encontrar repeticiones - detectar patrones de comportamiento
patron = r"((AAAAA|AAAA|AAA|AA)(BBBBB|BBBB|BBB|BB))((AAAAA|AAAA|AAA|AA)(BBBBB|BBBB|BBB|BB))+"
#patron = r".*(A{2,4}B{2,4}){2,}.*"  -> MicroPython no admite este tipo de notación
```

El bucle principal de programa se repite de manera indefinida y la primera instrucción que realiza es leer los datos del micrófono, llamando a la función leer_Microfono(), la cual devuelve un arreglo con los valores digitales de las 256 lecturas capturadas por el sensor de audio. Los datos se almacenan en la variable “datos” y se regresan al bucle principal del programa.

```
#Toma 256 lecturas del sensor de audio y almacena los valores en un arreglo
def leer_Microfono():
    #La función adc.read() convierte los valores analógicos a digital
    datos = np.array([adc.read() for _ in range(cantidad_Lecturas)], dtype=np.float)
    return datos
```

Después de la lectura del micrófono, el bucle principal manda llamar a la función frecuencia_Dominante(), la cual se encarga de aplicar la transformada rápida de Fourier a todos los 256 valores en la variable “datos”, con la intención de identificar la frecuencia dominante (con mayor potencia) entre los datos. La transformada rápida de Fourier se aplica utilizando la función np.fft.fft() de la biblioteca “ulab” y los resultados se almacenan en la variable “resultadoFft”, la cual es un arreglo. El índice 0 cero dicho arreglo contiene otro arreglo con la parte real de los resultados y el índice 1 contiene otro arreglo con la parte imaginaria.

El primer valor de la parte real del resultado tuvimos que descartarlo debido a que siempre se generaba un valor sin sentido debido al ruido. Se obtenía un valor demasiado alto que alteraba el resultado al buscar la frecuencia dominante.

Para encontrar la frecuencia dominante fue necesario obtener un índice que representaba la magnitud completa de los sonidos, juntando la parte real e imaginaría, elevando estos valores al cuadrado y sumarlos. Entre estos índices se busca el mayor valor que corresponde al sonido de mayor potencia y se almacena en la variable “magnitudMaxima”. Por último, se obtiene la frecuencia en Hertz a partir de este índice de magnitud máxima, la tasa de muestreo y la cantidad de lecturas tomadas, y se almacena en la variable “frecuenciaDominante”, la cual se envía al bucle principal del programa.

```
#Encuentra la frecuencia dominante (mayor potencia) entre las 256 lecturas
def frecuencia_Dominante(datos):
    #Aplica la transformada de rápida de Fourier (fft) a los 256 valores leídos.
    resultadoFft = np.array(np.fft.fft(datos), dtype=np.float)

    #Descartamos el primer valor real de la fft - Valor sin sentido (altisimo) debido a ruido.
    resultadoFft[0][0] = 0

    #Elevar parte real e imaginaría al cuadrado para representar magnitudes completas mediante un indice
    magnitudesCuadradas = resultadoFft[0]**2 + resultadoFft[1]**2

    #Encuentra el mayor de los indices - mayor magnitud
    magnitudMaxima = np.argmax(magnitudesCuadradas)

    #Obtener el valor de la frecuencia en Hz
    frecuenciaDominante = magnitudMaxima * tasa_Muestreo / cantidad_Lecturas

    return frecuenciaDominante
```

Una vez que el programa identificó la frecuencia dominante, procede a clasificarla en frecuencia alta o frecuencia baja haciendo uso de condicionales “if” mediante los que se establecen ciertos rangos de frecuencia. Esta clasificación se lleva a cabo en el bucle principal del programa, como se puede observar en la siguiente imagen, sin embargo, cabe mencionar que las instrucciones de las

condicionales se tuvieron que colocar en la misma línea de código pues el IDE utilizado para compilar el programa, Thonny, arrojaba errores de otra manera. Dependiendo del rango de frecuencias, se concatena una letra “A” o “B” para frecuencias altas o bajas respectivamente, o por el contrario, si no entra en ninguno de los rangos definidos, concatena un carácter “-”.

```
#Bucle principal del programa
while True:
    datos = leer_Microfono()    #Obtención de señal
    frecuencia = frecuencia_Dominante(datos)    #Detectar frecuencia más potente
    print(frecuencia)

    #Clasificación de frecuencias
    if 300 < frecuencia < 1200: lecturas += 'B'
    elif 1500 < frecuencia < 17000: lecturas += 'A'
    else: lecturas += '-'

    #Número de lecturas necesarias para comprobar la detección de sirena
    if len(lecturas) > 20: print(detectar_Sirena())

    #Controlar el tiempo de toma de muestra
    sleep(0.2)
```

Cuando concluye la clasificación de la frecuencia, el programa comprueba que el número de lecturas realizadas sea mayor a 20 para llamar a la función encargada de buscar y detectar la sirena en la señal, detectar_Sirena(). Establecimos que la detección se realice con 20 lecturas mediante prueba y error, debido a que al usar un mayor número de lecturas el funcionamiento del programa era muy lento y decaía su rendimiento, mientras que con números menores, la velocidad era tan rápida que no detectaba nada.

La función detectar_Sirena(), recibe la cadena (“lecturas”) con los caracteres que describen el comportamiento de la señal capturada y busca si concuerda con los patrones prestablecidos de las sirenas de emergencia utilizando un patrón de búsqueda de las expresiones regulares. El resultado de dicha búsqueda se almacena en la variable “encontrada”. Luego, con base en el resultado se muestra un mensaje indicando si se detectó o no una sirena en los sonidos capturados.

Finalmente, se reestablece la variable “lecturas” estableciendo una cadena vacía y dejarla lista para la ejecución del programa con la siguiente muestra.

```
#Busca si la variable "lecturas" coincide con el patrón predeterminado
def detectar_Sirena():
    global lecturas
    print(lecturas)

    #Usa expresiónes regulares para buscar coincidencias del patrón
    encontrada = re.search(patron ,lecturas)

    #Vaciar variable "lecturas" para siguiente muestra
    lecturas = ''

    #Mostra mensaje indicando si encontro o no la sirena
    if encontrada: return 'Sirena encontrada'
    else: return 'Sirena NO encontrada'
```

Experimentos, preparativos, recursos y escenario de prueba

Durante el desarrollo de este proyecto fue necesario realizar diversos ajustes y correcciones a nuestro código para lograr el funcionamiento esperado del sistema de detección.

La primera versión de nuestro código no detectaba nada, debido a que el patrón de búsqueda de la expresión regular no estaba trabajando de la forma deseada, e incluso si el comportamiento de la señal capturada cumplía con el patrón predefinido de las sirenas no se detectaba correctamente y mostraba el mensaje “Sirena No encontrada”.

Resulto que el problema era la notación utilizada para definir el patrón de búsqueda de la expresión regular, pues MicroPython no era compatible con ella.

```
#Expresión regular -> encontrar repeticiones - detectar patrones de comportamiento
patron = r".*(A{2,4}B{2,4}){2,}.*"
```

Debido a esto fue necesario definir el patrón de búsqueda de la expresión regular, definiendo los distintos casos en que se trata de una sirena uno por uno, como se puede observar en la siguiente imagen.

```
#Expresión regular -> encontrar repeticiones - detectar patrones de comportamiento
patron = r"((AAAAA|AAAA|AAA|AA)(BBBBB|BBBB|BBB|BB))((AAAAA|AAAA|AAA|AA)(BBBBB|BBBB|BBB|BB))+"
```

En esta segunda versión del código, la detección se realizaba de manera correcta, pero aun así requirió otro ajuste. Teníamos definido que la función de detección de sirenas se ejecutara después de 30 lecturas, pero aunque detectaba correctamente la sirena, su funcionamiento era muy lento.

```
#Número de lecturas necesarias para comprobar la detección de sirena
if len(lecturas) > 30: print(detectar_Sirena())
```

Es por eso que decidimos establecerlo en 20 finalmente, al realizar pruebas la ejecución se realizaba correctamente y a una velocidad aceptable. Con números más bajos la ejecución era tan veloz que no detectaba nada.

```
#Número de lecturas necesarias para comprobar la detección de sirena
if len(lecturas) > 20: print(detectar_Sirena())
```

RESULTADOS

Equipo y software usados

Para construir e integrar el proyecto fue necesario el uso de diversas librerías para implementar algunas funciones necesarias para el análisis, procesamiento y detección de la señal acústica capturada, al trabajar en conjunto con el sensor de audio (micrófono) y el microcontrolador ESP32.

A continuación, se describen los diversos recursos empleados en el desarrollo del proyecto.

- **Sensor de audio (KY-038)**: El micrófono es un componente clave en el proyecto. Se conecta a través de los pines VCC para la alimentación, GND para tierra y la salida analógica, para capturar las señales de sonido del entorno circundante.

- **Microcontrolador ESP32:** El microcontrolador ESP32 es esencial para procesar las señales de audio y ejecutar el código implementado en MicroPython. Utiliza los pines VCC para la alimentación, GND para la tierra, y el GPIO34 para conectar el sensor de audio.
- **MicroPython:** Es una implementación de Python 3 diseñada para trabajar en microcontroladores. El código de MicroPython realiza el procesamiento de señales de audio y la lógica de detección de sirenas.
- **IDE Thonny:** El proyecto se desarrolló utilizando el IDE Thonny, debido a que proporciona herramientas para escribir, ejecutar y depurar código en MicroPython, facilitándonos el proceso de programación y carga de código en el ESP32.
- **Librería "machine":** Se utiliza para configurar el hardware, en este caso, para establecer la configuración del pin y el ADC (Convertidor Analógico-Digital) del ESP32.
- **Librería "time":** Permite usar la función "sleep" para controlar el ritmo en que se ejecuta el programa, definiendo un tiempo de pausa entre la toma de muestras.
- **Librería "ulab":** Esta librería proporciona la función para realizar la transformada rápida de Fourier (FFT) en el procesamiento de las señales de audio. Facilita la identificación de la frecuencia dominante.
- **Librería "re" (expresiones regulares):** La librería "re" se utiliza para trabajar con expresiones regulares, específicamente para detectar el patrón de sirenas en las lecturas acumuladas.

Dificultades, fallas (bugs) y limitantes

Como se mencionó anteriormente, nuestra implementación del código presento varias dificultades, debido a algunas fallas y limitantes.

- **Incompatibilidad en MicroPython:** Este lenguaje tiene una limitante y es el hecho de que no es compatible con cierta notación para encontrar repeticiones de caracteres a la hora de buscar el patrón de las sirenas, por lo que fue necesario especificar distintos escenarios uno a uno.
- **Bug en IDE Thonny:** Este IDE presentó un bug bastante molesto. El problema es que te fuerza a escribir las instrucciones de las condicionales "if" en una sola línea de código, pues en otro caso arrojaba un error de compilación.
- **Librería "ulab":** Encontrar una implementación de ulab que funcionara de manera correcta para nuestros fines fue complicado, pues la mayoría de versiones que encontramos no contaban con las funciones completas para realizar la transformada rápida de Fourier.
- **Duración de la muestra:** El código necesita una muestra con una duración de entre 5 y 6 segundos aproximadamente para detectar las sirenas de forma correcta.
- **Descarte de valor generado por ruido:** Al realizar la transformada rápida de Fourier, fue necesario establecer el primer valor real generado en cero, debido a que ese primer valor siempre era una cantidad muy alta por el ruido detectado, y si no se descartaba siempre era detectado como la frecuencia dominante, alterando los resultados.

Resultados de operación del proyecto

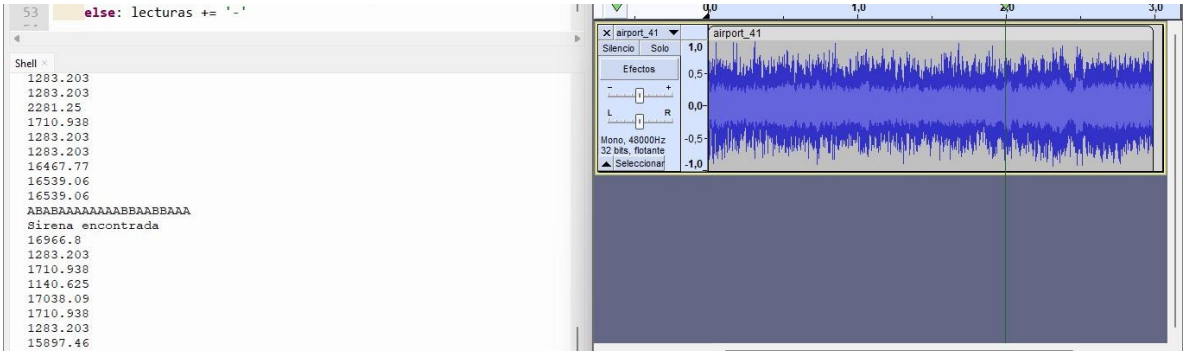
Una vez que logramos que nuestro código detectara algunos sonidos de sirenas correctamente y vimos que los valores de frecuencias que detectaba eran muy cercanos a los valores reales, comenzamos a utilizar los 54 audios que nos brindó el profesor para probar el proyecto. En términos generales los audios eran de una duración muy corta por lo que no alcanzaba a detectar las sirenas, por lo que decidimos reproducir los sonido en bucle utilizando el programa Audacity para lograr una mejor detección por parte de nuestro código. De esta forma obtuvimos los resultados en la siguiente tabla, donde se muestra en color verde los audios que si se lograron detectar.

AUDIO	DETECCIÓN DE AUDIO
75490-8-0-0	
75490-8-1-0	
90014-8-0-1	
90014-8-0-2	
90014-8-0-3	
94636-8-0-9	
105289-8-0-0	
105289-8-0-3	
105289-8-0-4	
105289-8-0-5	
105289-8-1-2	
107357-8-1-9	
107357-8-1-12	
107357-8-1-16	
107357-8-1-18	
159747-8-0-9	
airport_41	
airport_48	
airport_66	
airport_68	
ambulance31_v	
Ambulance137_v	
Ambulance141_v	
Ambulance273_v	
Ambulance295_v	
Ambulance335_v	
ambulance370_v	
Ambulance636_v	
hilo_01	
hilo_09	
hilo_13	
london_4	
nyc_9	
nz_1	
nz_2	

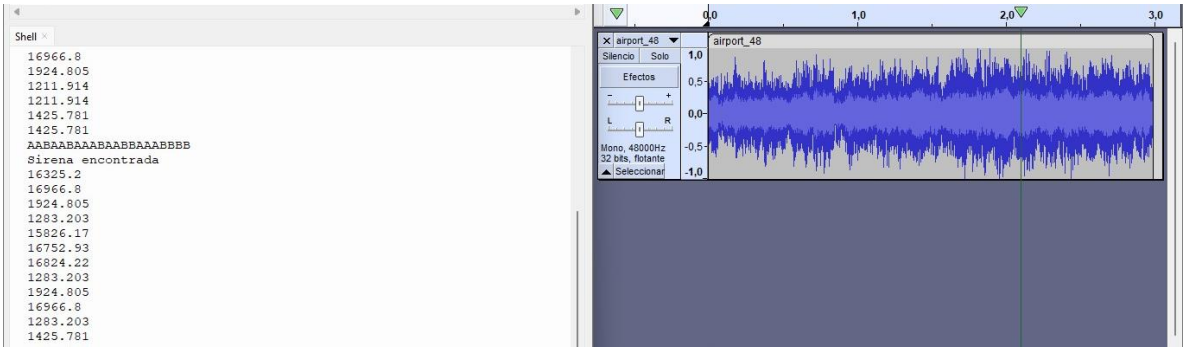
nz_3	
scdf_4	
scdf_5	
scdf_7	
siren_hilo_044	
siren_hilo_058	
siren_hilo_062	
siren_hilo_085	
siren_hilo_115	
siren_hilo_1110	
siren_twotone_02	
uk_3	
uk_4	
uk_7	
uk_8	
uk_10	
uk_16	

En seguida se muestran algunas capturas de pantalla de la ejecución y resultados del programa durante la reproducción de los audios con sirenas detectadas correctamente. A la izquierda se ve la ventana del IDE Thonny con los resultados de la detección, mientras a la derecha se encuentra el programa Audacity, reproduciendo los audios en bucle.

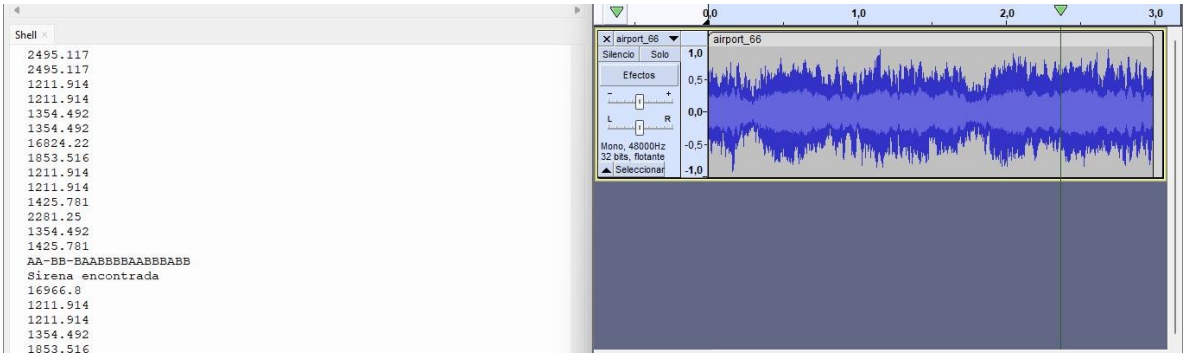
Audio airport_41



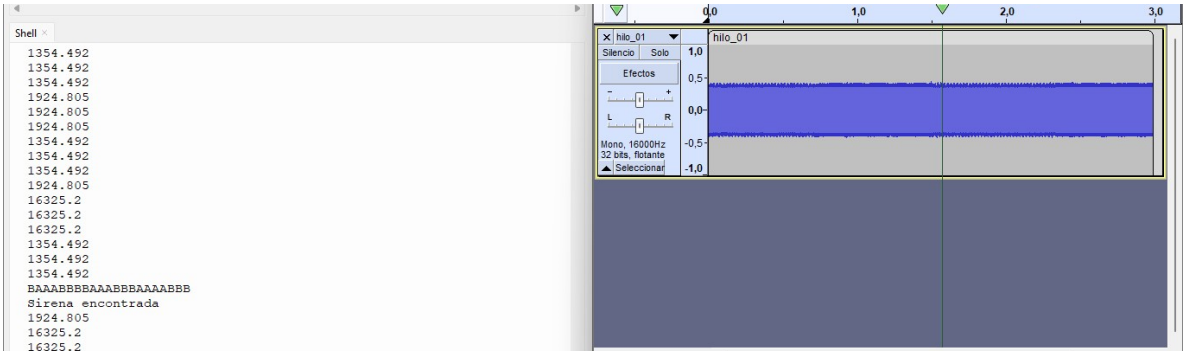
Audio airport_48



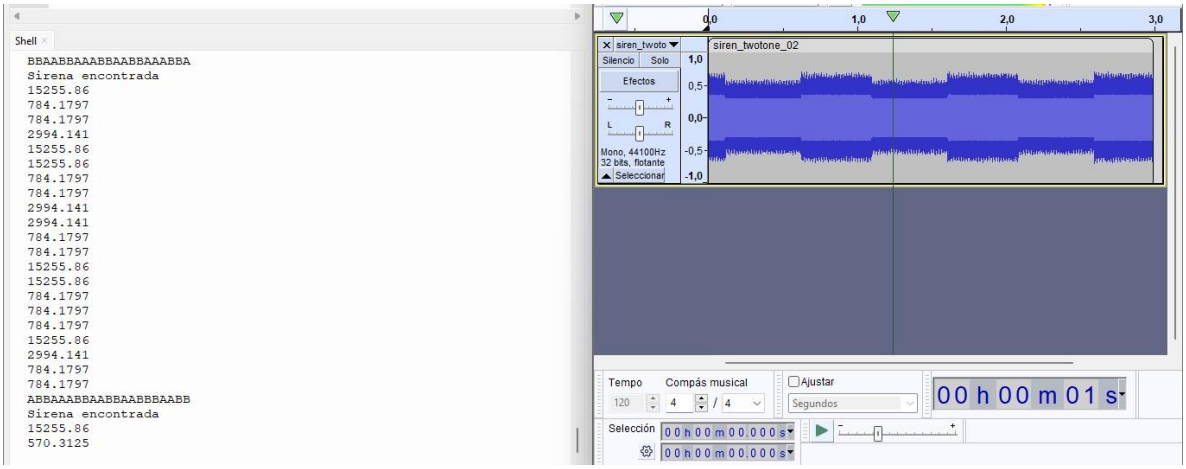
Audio airport_66



Audio hilo_01



Audio siren_twotone_02



A continuación se proporciona el link al video donde se puede observar el proyecto en funcionamiento, detectando sirenas de manera correcta.

Link: https://drive.google.com/file/d/1I7ACsc-8k_UShNSQnLwYYqsoDF_Eltar/view?usp=sharing

El código fuente se puede consultar en el siguiente repositorio de GitHub.

Link: https://github.com/OscSerrano/ProyectoFinalSirenas

CONCLUSIONES

Emilio Alberto Salas Ríos

Durante este curso obtuve muchos aprendizajes sobre varios temas como, de hecho, en general muchos de esos aprendizajes estuvieron relacionados con la programación avanzada tanto en lenguaje C++ como en Python como el uso de las expresiones regulares y el uso de funciones lambda en el paradigma de programación funcional. Pero también pienso, que uno de los mayores aspectos positivos del curso, fue el ampliar la visión respecto a que existen muchos campos distintos donde podemos aplicar nuestros conocimientos, en específico relacionadas al IOT, para mejorar o crear soluciones a diversas problemáticas como puede ser este proyecto de detección de sirenas de vehículos de emergencia, por medio del cual se puede mejorar la seguridad vial en el día a día de las personas, previniendo accidentes y mejorando tiempos de respuesta de las autoridades en situaciones críticas.

En cuanto al proyecto en sí, fue interesante comprobar cómo se pueden lograr buenos resultados e innovar incluso utilizando componentes de bajo costo. Sin embargo, se pudo observar también las limitaciones que presentaba nuestro proyecto, pues disminuía su eficiencia cuando existía una gran presencia de ruido u otros factores como la distorsión del sonido causada por el efecto Doppler, esto se puede apreciar si revisamos los audios donde si logro detectar la sirena, ya que son los que presentan un sonido más claro y uniforme.

Oscar Alejandro Serrano Pizarro

A lo largo del curso hemos podido aprender varias cuestiones acerca del desarrollo de proyectos embebidos, desde el desarrollo de prototipos basados en lenguajes de alto nivel como Python o MicroPython hasta la implementación final, más rápida en lenguajes de bajo nivel como C o C++. Para cada una de estas opciones vimos como descargar e implementar las librearías a utilizar ya sea utilizando el comando PIP para Python, buscando el firmware correcto de MicroPython o buscando en internet la librería .h para el código de C.

En general siento que el curso amplia la visión en cuanto al desarrollo de proyectos para el internet de las cosas, así como también amplia la visión en cuanto a la cantidad de proyectos que podríamos llegar a realizar, sin embargo, creo que la clase debería enfocarse más en el desarrollo con el internet de las cosas que simplemente ver cualquier proyecto utilizando embebidos, también creo que podría haber sido bastante interesante ver la implementación de pantallas para visualizar en el mismo embebido la información recopilada en tiempo real.

Por parte de este proyecto, creo que fue bastante interesante ver la implementación con MicroPython, aunque debo comentar que fue bastante problemático buscar el firmware con ulab para poder utilizar la transformada rápida de Fourier , así como lo fue hacer las modificaciones al código para que al menos pudiese reconocer las señales de una sirena Hi-Lo sin ningún ruido de fondo. Y aunque fue bastante cansado, también fue bastante interesante y un buen proyecto en general.

Jorge Eduardo Carrasco López

Por medio de este gran proyecto que estuvimos trabajando a lo largo del semestre pudimos observar el cómo funcionan los pequeños programas que se utilizan al momento de crear pequeños aparatos o aplicaciones como en este caso empezamos a trabajar con un detector de sirenas el cual por medio de conceptos que fuimos adquiriendo sobre el comportamiento de estos sonidos los cuales supimos que tenían una frecuencia especial así también el cómo podíamos detectar este tipo de frecuencias empezando con la programación del sistema el cual tendrá que detectar estas, para en un tiempo después usar un programa el cual será el encargado de hacer la distinción de estas frecuencias a las cuales trabaja por medio de un micrófono que nos entrega una señal para esta poderla graficar por lo cual utilizamos una herramienta la cual nos mostraba dicha gráfica, y todo esto mientras utilizábamos el lenguaje C++ tratando de hacer las menos líneas posibles. Ya que esto significaría un uso más extenso de la memoria como del procesamiento de nuestro ESP 32.

Por ultimo para la finalización de nuestro proyecto fue necesario cambiar todo y pasarlo a MicroPython lo cual fue un nuevo reto a la hora de programarlo ya que aunque era la misma lógica algunas funciones no están totalmente disponibles en este a comparación de C++ por lo cual es necesario cambiar algunas cosas y modificar el orden de algunas líneas pero todo esto resulto bien gracias a la ayuda de la transformada rápida de Fourier la cual solo con una búsqueda exhaustiva de este para que fuera compatible fue suficiente y ya con ello solo quedaron hacer las pruebas correspondientes lo cual resulto satisfactoriamente.

REFERENCIAS

v923z. (s.f.). *micropython-builder*. Obtenido de GitHub: https://github.com/v923z/micropython-builder/tree/master

v923z. (s.f.). *micropython-ulab*. Obtenido de GitHub: https://github.com/v923z/micropython-ulab

Campbell, S. (s.f.). *How to Use Microphones on the Arduino*. Obtenido de Circuit Basics: https://www.circuitbasics.com/how-to-use-microphones-on-the-arduino/

del Río, J. (17 de Febrero de 2023). *Práctica 6: Arduino y micrófono*. Obtenido de Librería CATEDU: https://libros.catedu.es/books/arduino-y-pure-data-ondas-color-y-sonido-v1/page/practica-6-arduino-y-microfono