Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Profesor: Dr.-Ing. Pablo Alvarado Moya
EL 5805 Procesamiento Digital de Señales
II Semestre 2023

Entrega: **15 de setiembre, 2023**

Versión: 28 de agosto de 2023

Proyecto 1: Procesado de señales en tiempo real con C++

En este proyecto se introducen conceptos de procesamiento digital de señales en el lenguaje de programación C++, utilizando el paradigma de programación orientado a objetos y principios básicos de programación genérica.

El proyecto persigue la experimentación con técnicas de manipulación de señales directamente en el dominio del tiempo, con el fin de extraer características básicas, y se pueda a partir de la información generada producir señales con características particulares.

1. Plataforma de trabajo

Código base para GNU/Linux está a disposición a través de la invitación de Github Classroom. Dicho código utiliza el sistema de manejo de sonido en GNU/Linux conocido como Jack, en una aplicación minimalista de línea de comandos. Adrede se prescinde de utilizar el sistema más reciente para manejo de sonido Pipewire, debido a que su integración en las distribuciones recientes de GNU/Linux no es aun completa.

1.1. Dependencias

Si usted utiliza las distribuciones de GNU/Linux Ubuntu o Debian, vaya al directorio del código y ejecute el script de instalación de dependencias:

```
> cd <directorio_de_código>
> sudo ./ubuntu.sh
```

El instalador de Jack le preguntará si usted desea poder usar audio en tiempo real, a lo que deberá responder que sí. Una vez finalizada la instalación, asegúrese de que su usuario pertenezca al grupo `audio`. Para hacer eso puede ejecutar:

```
> sudo usermod -aG audio $USER
```

En caso de que usted utilice otra distribución de GNU/Linux, debe instalar los paquetes que correspondan. Esto incluye el compilador de C++ (`clang` o `g++`), las bibliotecas de desarrollo de Jack, la aplicación `qjackctl` o equivalente. Además, para construir la aplicación se utiliza el sistema `meson` junto a `ninja`, que en la actualidad parecen reemplazar con cada vez mayor frecuencia a CMake o autotools.

Cerciórese con la aplicación `qjackctl` que usted puede iniciar y detener exitosamente el servidor de Jack. Utilice los foros disponibles en Internet si tiene algún problema, y en

dado caso utilice el foro "Cafetería" para solicitar ayuda a los compañeros y profesor. Usualmente con las últimas versiones de (K)Ubuntu no debería haber problema alguno con Jack.

Algunas herramientas básicas para uso del Jack, como `jack-scope` (instalado por el script anterior en el paquete `jack-tools`), permiten determinar si todo funciona.

Usted necesitará un audífono con micrófono o un *head set*. En este proyecto se procesará la entrada del micrófono directamente, para producir una señal de salida en tiempo real. Esta configuración facilita la ocurrencia de procesos de realimentación positiva, que causan tonos desagradables si se utilizan parlantes, cuya salida puede ser capturada por el micrófono. Usted puede experimentar configuraciones de espacio si lo desea, pero es más sencillo evitar esto con auriculares para cortar el lazo de realimentación.

1.2. Compilando la aplicación por primera vez

En el directorio `proyecto1` ejecute

```
> meson setup build
> ninja -C build
```

El primer comando crea todo lo necesario en el subdirectorio `build`, y el segundo inicia el proceso de compilación. Si todo marcha bien, debería existir un archivo ejecutable `build/dsp1`.

1.3. Ejecutando la aplicación

Es necesario levantar el servidor de Jack antes de ejecutar la aplicación. Esto pude hacerlo con `qjackctl` (figura 1) o, si lo desea, con las aplicaciones de línea de comando que los

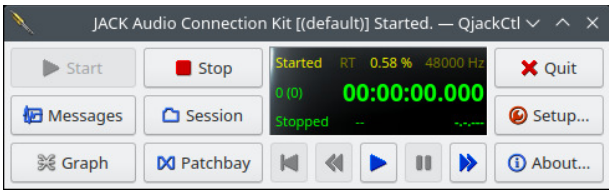

Figura 1: Ventana de inicio del controlador de Jack `qjackctl`.

paquetes de Jack ofrecen.

En principio, la aplicación intenta levantar el servidor de Jack si no encuentra uno activo, pero dependiendo de cada configuración, suele ocurrir que lo hace a destiempo, y la aplicación termina antes de que el servidor esté completamente listo, lo que produce errores.

Una vez esté el servidor de Jack activo, ejecute la aplicación con

```
> cd build
> ./dsp1
```

La entrada de audio será tomada directamente del micrófono, y la salida a uno de los dispositivos de salida configurados por defecto.

Usted puede cambiar qué dispositivos se utilizan en el diálogo de "`Setup`".

Además, usted puede cambiar la entrada de la aplicación en el grafo de conecciones ("`Graph`").

2. Tareas

Usted deberá construir una aplicación paso a paso, que permite activar varios modos de operación. Para ello, usted deberá crear una clase `dsp_client` que hereda de `jack::client`. Como punto de partida usted puede utilizar la clase `passthrough_client`.

Es esencial comprender cómo funciona todo el concepto de procesamiento de audio, para tener muy claro dónde y cuándo se tiene a disposición cuánto tiempo, para hacer qué tarea, sin poner el desempeño del sistema en riesgo. Cuando usted instaló Jack, debió recibir una advertencia de que otorga fuertes privilegios al sistema de audio. Al permitir procesamiento en tiempo real, usted puede comprometer la integridad del sistema, en particular si no retorna el control al sistema en los tiempos destinados para ello. Considere esto a la hora de diseñar sus programas.

El procesamiento de señal como tal — ¡y única- y exclusivamente dicho procesamiento!— debe ocurrir en el método `process`, que sobrecarga al método homónimo de la clase `jack::client`. Dicho método recibe un número de muestras de entrada y debe producir el mismo número de muestras de salida. La memoria para trabajar la reserva y libera el servidor de Jack, por lo que usted no debe intervenir con dicho proceso, o causará problemas serios con el sistema operativo.

En general, el sistema de audio lo administra el sistema operativo. Este pone a disposición hilos especializados de alta prioridad para que ejecuten ese método `process` en tanto el búfer de entrada esté lleno. La información la toma Jack del dispositivo asociado a ese puerto de entrada, lo que puede ser reasignado desde la aplicación `qjackctl`. Cuando `process` termina, Jack indica al dispositivo de salida asociado que tome la señal del búfer de salida. Es por ello, que dicha función no debe, bajo ninguna circunstancia, realizar ningún tipo de interacción con el usuario, pues en ese caso excedería fácilmente la ventana de tiempo disponible para reaccionar. Sin embargo, a través de su clase `dsp_client`, el método `process` puede utilizar información que el usuario deje en atributos de dicha clase para modificar la operación (por ejemplo, el volumen, la latencia, el tamaño de ventanas para estimación de energía, etc.) o `process` puede colocar allí características extraídas de la señal (como cálculos de energía o potencia, resultados de correlación, etc.), para que alguien más utilice los resultados para mostrarlos o para tomar alguna acción.

Para la parte de visualización de información, puede serle útil la diferencia entre "`\n`" y "`\r`".

2.1. Cambiemos el volumen

La primer tarea a resolver (para "romper el hielo") es permitir a la aplicación cambiar entre dos modos de operación. El modo "passthrough", que es la funcionalidad ya entregada con el código base, debe activarse cuando el usuario presione la tecla '`p`'. El modo "volume change", permite al usuario cambiar el volumen de la señal de entrada, y debe activarse con la tecla '`v`'. Si la aplicación está en modo "volume change", las teclas '`+`' y '`-`' permiten incrementar y decrementar el volumen. Restrinja el volumen máximo a 10 veces el valor de entrada y a 0 el volumen mínimo.

Determine alguna escala entre los valores máximo y mínimo que permita pasar por volumen 1 (equivalente al *pass-through*), y que sea acorde con la percepción humana para cambio de volumen.

2.2. Midamos energía

La medición de energía o potencia promedio será importante para las próximas tareas del proyecto. El usuario debe poder especificar el tamaño de la ventana para medición de energía y potencia promedio, como argumento en la línea de comando:

```
> ./dsp1 --energy 0.5  # o también ./dsp1 -e 0.5
```

donde el número indicado deberá ser el tiempo en segundos utilizado para estimar la energía. Si el usuario no provee explícitamente un valor, dicho parámetro debe ser por defecto $0,5\,\mathrm{s}$.

Para interpretar los argumentos, utilice bibliotecas como `getopt` o `program_options` de boost, porque iremos agregando más y más opciones a la línea de comandos en los siguientes puntos. Hay algunas sugerencias en Internet de cómo resolver esta tarea de administrar las opciones de línea de comando utilizando las herramientas de la STL (Standard Template Library) en C++, que también usted puede utilizar, si lo encapsula bien en sus propias clases y funciones, aisladas a su vez en archivos correspondientes.

Observe que usted debe determinar cuántos bloques y fragmentos de bloque debe considerar para el cálculo de la energía y potencia. Debe actualizar el valor de energía y potencia calculadas en cada llamada a `process`, de modo que la aplicación principal pueda mostrar valores actualizados constantemente. Esto es una estimación de "ventana deslizante" de la energía. Observe que sería muy ineficiente almacenar toda una ventana muestras, dar el mantenimiento para descartar el trozo viejo y agregar el trozo nuevo, y además hacer el cálculo para toda la ventana. La idea es realizar el cálculo lo más eficientemente posible.

Con la tecla '`e`' el usuario debe poder activar o desactivar la visualización del valor de energía o potencia y con la tecla '`E`' cambia si muestra energía o potencia.

Observe que esta opción debe funcionar independientemente si el modo es "passthrough" o "volume change".

2.3. Calculemos la autocorrelación para encontrar el periodo

Este es el punto más retador del proyecto, porque debe realizarse de forma muy eficiente, para evitar agotar el tiempo disponible para el cálculo dentro de `process`. Aquí queremos replicar en principio el trabajo realizado en clase, con la ventaja de que podemos reducir los cálculos para calcular la autocorrelación de la entrada con desplazamientos de 0 en adelante, y con algunos heurísticos podemos alcanzar funcionalidades más interesantes.

La idea es con la tecla '`n`' activar el modo de estimación de periodo. Cuando entra en este modo, desactiva otros modos, como el de visualizar energía/potencia y viceversa. En ese modo, el usuario observará información del número de muestras por periodo y la frecuencia equivalente de ese periodo, en hertz.

En la línea de comando, el usuario debe poder indicar el rango de frecuencias a ser detectadas. Esto lo hace con `--minfreq` y `--maxfreq`. Por ejemplo, si solo se desean detectar señales con frecuencias entre 100 Hz y 400 Hz:

```
> ./dsp1 --minfreq 100 --maxfreq 400
```

Con esas frecuencias y la frecuencia de muestreo utilizada, usted debe determinar los periodos discretos esperables. Esto a su vez guía qué tramos de la función de autocorrelación es necesario calcular.

Realice y documente experimentos (en GNU/Octave) para determinar con qué tamaño de señal usted puede hacer una estimación robusta del periodo. Recuerde que mientras menos periodos cubra su señal, la caída de la autocorrelación es más fuerte, porque no hay suficiente traslape que soporte la detección. Mientras mayor sea la ventana utilizada, más estable es la detección, pero más cálculo deberá realizar, y la cantidad de cálculos crece cuadráticamente con el tamaño de la ventana. Deberá encontrar un buen compromiso.

Ahora viene lo interesante. Con la tarea anterior usted puede calcular la energía y potencia de la señal en una determinada ventana de tiempo. Dicha estimación de energía/potencia tiene cierta "inercia" dada por el tamaño de la ventana. Si usted pasa de silencio a señal acústica, la ventana cubrirá un trozo con ceros por un cierto tiempo, lo que hace que la estimación de energía suba lentamente. Un efecto similar sucede cuando la señal desaparece: pasará un tiempo hasta que la ventana de estimación cubra únicamente trozos sin señal, para mostrar energía baja, observándose una caída lineal de la energía, aun cuando el silencia haya llegado abruptamente. Eso no es problemático para mostrar el nivel de energía, pero sí lo es, si queremos "segmentar" la señal.

El proceso de "segmentación" consiste en determinar a partir de cuándo aparece una señal y cuándo termina dicha señal. Esta estimación sí tiene sentido hacerla a nivel de un bloque, o incluso menos si se utiliza adecuadamente un filtro IIR de aproximación exponencial de media, utilizando además mecanismos de histéresis para colocar marcadores de dónde inicia y dónde termina un bloque activo de señal. Usted debe establecer límites del tamaño de la ventana capturada, para no excederse en tamaños de memoria reservados, ni en tiempos de procesamiento.

Note que en C (`malloc/free`) y C++ (`new/delete`) la reserva de memoria en el *heap*

es un proceso computacionalmente caro, por lo que debe asegurar la reserva previa de la memoria que se va a utilizar para guardar esos bloques, así como su correspondiente liberación. Revise el concepto RAII en C++ para realizar eso adecuadamente, por ejemplo, utilizando `std::shared_ptr`.

Su programa debe permitir establecer límites máximos de captura (por ejemplo, 5 s) con la opción `--ringsize`, y debe utilizar para ello búferes anulares. Se recomienda realizar una clase `ringbuffer` que se encargue de todo el manejo de dicho concepto: reserva y liberación de memoria, manejo de los punteros de entrada y salida, etc.

El usuario debe poder indicar un nivel de potencia mínimo que debe superarse para iniciar una detección de vocal (con "`--minlevel`"). Los niveles de histéresis usted tiene libertad de elegir cómo los puede indicar el usuario.

En el modo de estimación de periodo el usuario entonces pronuncia una vocal, y la aplicación automáticamente detecta el inicio y el final de la señal acústica. La señal debe superar una duración mínima indicada con "`--nwindow`". Cuando dicha señal es detectada, usted tiene la opción de pasar la información al hilo principal para el cálculo y despliegue del periodo / frecuencia de la señal detectada, o puede intentar hacer el cálculo del periodo dentro de `process`, y solo indicar el resultado a la aplicación principal para su despliegue. En este último caso debe estar seguro de que su método es muy eficiente.

Observe que para el cálculo del periodo no es necesario utilizar toda la señal capturada, sino únicamente una ventana de tiempo que puede tomar de algún lugar que usted considere conveniente dentro de la señal, y que tenga eso sí al menos el tamaño por usted determinado anteriormente para lograr una estimación robusta.

La localización del siguiente pico de la función de autocorrelación requerirá también la programación manual de algún heurístico. Debe investigar y documentar cómo hacer esto eficientemente.

Debe mostrar el periodo/frecuencia detectada cuando lo considere conveniente. Hay varias opciones: lo muestra continuamente conforme se va detectando, o únicamente cuando el usuario terminó de pronunciar la vocal.

2.4. Repetidor de tono senoidal

En este modo, activado con la tecla '`r`', una vez que el usuario termina de pronunciar su vocal, una señal senoidal de la misma frecuencia que el periodo detectado debe sonar, con la misma potencia promedio de la señal de entrada. El usuario en este modo también puede con las teclas '`+`' y '`-`' incrementar o bajar el volumen, para compensar las características del dispositivo físico reproductor, pero debe notarse que, si el usuario cambia el volumen promedio en la pronunciación de la vocal, también lo hace la señal de salida.

2.5. Afinador

Este modo, activado con la tecla '`t`', simplemente se complementa el modo de estimación de periodo, indicando la nota musical más cercana correspondiente, y si el usuario debe

subir o bajar su nota para entonar la nota musica estándar más cercana.

Para encontrar dicha nota, se utilizará el llamado "temperamento igual". En este, una octava musical se divide en 12 intervalos perceptualmente iguales. Así, si utilizamos la nota $\mathrm{La}_4{=}440\,\mathrm{Hz}$, entonces todas las notas vecinas se generan multiplicando o dividiendo por $2^{1/12}$, de modo que, luego de doce notas, se alcanza una octava. Así, las notas de las escalas 3, 4 y 5 están dadas las frecuencias listadas en el cuadro 1. El mismo patrón

Tabla 1: Frecuencias de tres escalas musicales bajo el temperamento igual.

$\mathrm{Do}_3{=}\mathrm{La}_4 2^{-21/12}$	$\mathrm{Do}_4{=}\mathrm{La}_4 2^{-9/12}$	$\mathrm{Do}_5{=}\mathrm{La}_4 2^{3/12}$
$\mathrm{Do}_3^{\#}{=}\mathrm{La}_4 2^{-20/12}$	$\mathrm{Do}_4^{\#}{=}\mathrm{La}_4 2^{-8/12}$	$\mathrm{Do}_5^{\#}{=}\mathrm{La}_4 2^{4/12}$
$\mathrm{Re}_3{=}\mathrm{La}_4 2^{-19/12}$	$\mathrm{Re}_4{=}\mathrm{La}_4 2^{-7/12}$	$\mathrm{Re}_5{=}\mathrm{La}_4 2^{5/12}$
$\mathrm{Re}_3^{\#}{=}\mathrm{La}_4 2^{-18/12}$	$\mathrm{Re}_4^{\#}{=}\mathrm{La}_4 2^{-6/12}$	$\mathrm{Re}_5^{\#}{=}\mathrm{La}_4 2^{6/12}$
$\mathrm{Mi}_3{=}\mathrm{La}_4 2^{-17/12}$	$\mathrm{Mi}_4{=}\mathrm{La}_4 2^{-5/12}$	$\mathrm{Mi}_5{=}\mathrm{La}_4 2^{7/12}$
$\mathrm{Fa}_3{=}\mathrm{La}_4 2^{-16/12}$	$\mathrm{Fa}_4{=}\mathrm{La}_4 2^{-4/12}$	$\mathrm{Fa}_5{=}\mathrm{La}_4 2^{8/12}$
$\mathrm{Fa}_3^{\#}{=}\mathrm{La}_4 2^{-15/12}$	$\mathrm{Fa}_4^{\#}{=}\mathrm{La}_4 2^{-3/12}$	$\mathrm{Fa}_5^{\#}{=}\mathrm{La}_4 2^{9/12}$
$\mathrm{Sol}_3{=}\mathrm{La}_4 2^{-14/12}$	$\mathrm{Sol}_4{=}\mathrm{La}_4 2^{-2/12}$	$\mathrm{Sol}_5{=}\mathrm{La}_4 2^{10/12}$
$\mathrm{Sol}_3^{\#}{=}\mathrm{La}_4 2^{-13/12}$	$\mathrm{Sol}_4^{\#}{=}\mathrm{La}_4 2^{-1/12}$	$\mathrm{Sol}_5^{\#}{=}\mathrm{La}_4 2^{11/12}$
$\mathrm{La}_3{=}\mathrm{La}_4 2^{-1}$	$\mathrm{La}_4{=}440\,\mathrm{Hz}$	$\mathrm{La}_5{=}\mathrm{La}_4 2^{1}$
$\mathrm{La}_3^{\#}{=}\mathrm{La}_4 2^{-11/12}$	$\mathrm{La}_4^{\#}{=}\mathrm{La}_4 2^{1/12}$	$\mathrm{La}_5^{\#}{=}\mathrm{La}_4 2^{13/12}$
$\mathrm{Si}_3{=}\mathrm{La}_4 2^{-10/12}$	$\mathrm{Si}_4{=}\mathrm{La}_4 2^{2/12}$	$\mathrm{Si}_5{=}\mathrm{La}_4 2^{14/12}$

continua con las frecuencias de escalas inferiores y superiores.

Recuérdese de los diagramas de Bode, que una octava corresponde a la duplicación de la frecuencia. Debido a que el oído humano tiene respuesta logarítmica a la frecuencia, las comparaciones a las notas musicales debe hacerse en alguna escala logarítmica.

2.6. Repetidor de tono senoidal con auto-tune

Continuando con la idea de los puntos anteriores, en este modo, activado con la tecla '`a`', usted en vez de ejecutar el tono senoidal en la frecuencia que el usuario pronunció, lo ejecutará en la frecuencia de la nota musical más cercana.

2.7. Resumen de opciones

El cuadro 2 resume las opciones disponibles en línea de comando. Observe que hay una opción adicional para mostrar ayuda. Si usa bibliotecas para la interpretación de la línea de comandos, posiblemente tengan la opción de generar la ayuda automáticamente.

El cuadro 3 resume la configuraciones de teclas para cambiar de modo de operación.

Tabla 2: Opciones de línea de comando.

larga	c.	tipo	descripción	def.
`--energy`	`-e`	`float`	longitud de ventana para estimación de energía	$0,5\,\mathrm{s}$
`--minfreq`		`int`	frecuencia mínima detectable	$60\,\mathrm{Hz}$
`--maxfreq`		`int`	frecuencia máxima detectable	$600\,\mathrm{Hz}$
`--minlevel`		`float`	nivel de potencia mínimo para disparar detección	
`--nwindow`	`-n`	`float`	longitud de ventana para estimar periodo	$0,5\,\mathrm{s}$
`--ringsize`	`-r`	`float`	longitud del búfer anular	$3\,\mathrm{s}$
`--help`	`-h`	`float`	muestra ayuda con las opciones disponibles	

Tabla 3: Teclas para cambiar de modos de operación.

`a`	auto-tune
`E`	conmute entre mostrar energía o potencia
`e`	active modo para mostrar energía/potencia de señal
`n`	active modo de estimación de periodo
`p`	modo pasa-todo
`r`	modo repetidor
`t`	modo de afinador
`v`	modo de ajuste de volumen
`+`	subir volumen
`-`	bajar volumen

3. Grupos de trabajo

Este trabajo deberá ser realizado en grupos de 2 o 3 personas. Sin embargo, hay un factor multiplicativo de la nota final determinado por la cantidad de trabajo en el Git de cada persona, en comparación con sus compañeros.

4. Entregables

1. Código fuente documentado, subido al tecDigital, que debe corresponder a la rama `main` o `master` de su repositorio Git.

2. Informe corto de no más de 2 páginas, donde se explique la estructura de la aplicación realizada, y las justificaciones solicitadas en el texto.

3. Archivo README con indicaciones de cómo compilar y ejecutar su aplicación.

4. Recuerde que se evalúa el avance gradual del proyecto.

5. Se realizará una pequeña presentación (no estructurada) del proyecto al profesor.