UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

AUDIO BINAURAL 3D EN BELA

GRADO EN INGENIERÍA DE
SISTEMAS ELECTRÓNICOS

SERGIO FLORIDO BECERRA

MÁLAGA, 2023

E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN, UNIVERSIDAD DE MÁLAGA

Audio binaural 3D en Bela

Autor: Sergio Florido Becerra

Tutor: Arcadio Reyes Lecuona

Eduardo Javier Pérez Rodríguez

Departamento: Departamento de Tecnología Electrónica

Titulación: Grado en Ingeniería de Sistemas Electrónicos

Palabras clave: Sistemas basados en microcontrolador, sensores inerciales, audio 3D, compilación cruzada, 3DTI Toolkit, C++, diseño de circuitos, Bela.

Resumen

El objetivo de este Trabajo de Fin de Grado es el de desarrollar un sistema de renderizado de audio 3D en tiempo real sobre la placa BeagleBone Black con el interfaz de audio Bela, caracterizado por su ultra baja latencia y alta calidad en el procesado de audio. Mediante la incorporación de la librería 3D Tune-In Toolkit (desarrollada por el grupo de investigación DIANA de la Universidad de Málaga [18]) se sitúan fuentes de audio virtuales en el espacio relativo al oyente. El uso de esta librería permite simular un entorno auditivo y actuar sobre él de tal forma que un usuario, con unos auriculares, podría identificar la posición desde la que proviene el audio. Es decir, se podría modificar la posición y orientación del usuario o la posición de las fuentes de audio en tiempo real y el oyente sería consciente de este movimiento. Se ha desarrollado una aplicación que usa esta librería sobre Bela y permite la incorporación al sistema de un sensor inercial que, una vez colocado sobre la cabeza del usuario que lleva los auriculares, lea la orientación de la cabeza y adapte la posición de las fuentes de audio a dicha lectura.

En este trabajo se muestra todo el procedimiento necesario para usar la librería 3DTI Toolkit con Bela, el funcionamiento de la aplicación creada y las optimizaciones realizadas para mejorar su rendimiento. Este proyecto se ha realizado junto al TFG “Diseño de un *tracker* cefálico para audio 3D” de Juan

José Navarrete Gálvez, encargado del desarrollo del hardware y la impresión 3D para obtener un prototipo funcional con la aplicación descrita.

E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN, UNIVERSIDAD DE MÁLAGA

3D binaural audio on Bela

Author: Sergio Florido Becerra

Supervisor: Arcadio Reyes Lecuona

Eduardo Javier Pérez Rodríguez

Department: Departamento de Tecnología Electrónica

Degree: Grado en Ingeniería de Sistemas Electrónicos

Keywords: Microcontroller system based, inertial sensors, 3D audio, cross compiling, 3DTI Toolkit, C++, circuit design, Bela.

Abstract

The objective of this Final Degree Project is to develop a rendering audio 3D system in real time on the BeagleBone Black board with the audio interface Bela, which is known for its ultra-low latency and high quality in audio processing. By including the 3D Tune-In Toolkit (developed by the DIANA research team of the University of Málaga [18]) the system can place several virtual audio sources in the space relative to the listener. The use of this library allows the system to simulate an aural environment and to act on it in a way that a user, wearing headphones, will be able to identify the position where the audio is coming from. In other words, the position and the orientation of the user or the position of the audio sources could be changed in real time and the listener would always be aware of this movement. The developed application uses this library on Bela and allows the inclusion to the system of an inertial sensor that, once it is set over the head of the user who is wearing headphones, it reads the head orientation and adapts the audio source position to this reading.

This essay shows the necessary method to follow to use the 3DTI Toolkit library on Bela, the functioning of the created application and the optimizations made to improve its performance. This project is made together with the FDP “Diseño de un *tracker* cefálico para audio 3D” of Juan José Navarrete Gálvez,

responsible of the develop of the hardware and the 3D printings to get a functional prototype with the described application.

Contenido

Lista de Acrónimos

TFG	Trabajo de Fin de Grado
FDP	Final Degree Project
ITD	Interaural Time Difference
ILD	Interaural Level Difference
HRTF	Head Related Transfer Function
3DTI Toolkit	3D Tune-In Toolkit
OSC	Open Sound Control
VR	Virtual Reality
LED	Light Emitting Diode
USB	Universal Serial Bus
IDE	Integrated Development Environment
Tarjeta SD	Secure Digital card
eMMC	Embedded MultiMediaCard
IP	Internet Protocol
SSH	Secure SHell
GDB	Gnu DeBugger
GCC	Gnu Compiler Collection
FFT	Fast Fourier Transform

IFFT	Inverse Fast Fourier Transform
DSP	Digital Signal Processors
ARM	Advanced RISC Machine
I2C	Inter-Integrated Circuit
WAV	Waveform Audio File Format

Capítulo 1. Introducción

1.1. Contexto tecnológico

El audio binaural es todo aquel que está adaptado específicamente para que su captación se lleve a cabo mediante los dos oídos de una persona. Un ejemplo muy común en la actualidad es el audio reproducido mediante auriculares, ya que se divide la señal original en dos distintas pero complementarias, una para cada oreja. El audio 3D es aquel que se sitúa en el espacio, es decir, se puede ubicar coordenadas en las tres dimensiones. Todos los sonidos que nos rodean están situados en una posición y las personas somos capaces de percibirlos y saber desde dónde provienen. En este proyecto se ha desarrollado una unión de ambos, audio binaural 3D. Esto quiere decir que se desea reproducir una fuente de audio para ser captado por los dos oídos y que, a su vez, el oyente sea capaz de situar desde donde proviene la fuente, sin embargo, existen más factores que intervienen en esta operación dificultando el proceso.

Para poder entender todo el sistema en su conjunto, hay que tener en cuenta cómo es el cuerpo de una persona y qué partes de este influyen en la ubicación de una fuente de sonido.

La existencia de dos oídos permite situar la fuente de audio en el eje horizontal muy fácilmente ya que, si un sonido alcanza antes el oído izquierdo que el derecho, el cerebro interpreta que ese sonido proviene de la izquierda y viceversa [1]. La cabeza, el cuello, el torso y la parte exterior del oído intervienen en la percepción de la elevación y profundidad de un sonido mediante rebotes o atenuaciones de la señal inicial.

El cerebro es capaz de localizar la posición de sonidos en un entorno cercano mediante tres índices de localización [1]. Los dos índices binaurales son:

- **ITD** (Interaural Time Difference). Es la diferencia de tiempo entre la llegada de la misma señal de audio a los dos oídos.
- **ILD** (Interaural Level Difference). Es la diferencia de intensidad de una determinada señal de audio que percibe un oído respecto al otro.

El cerebro interpreta estos dos índices para identificar la posición de la fuente de audio respecto a los dos oídos en el plano horizontal. Sin embargo, hay zonas en las que el cerebro percibe los mismos ITD e ILD para varios casos. Estas zonas son llamadas conos de confusión y el cerebro no es capaz de situar la fuente en ellos, necesitando de otro índice de localización.

El tercer índice, **índice espectral monaural**, corresponde a cada oído específicamente y depende de las modificaciones de este y del resto del cuerpo en la señal de audio dependiendo de la dirección de la que provenga.

El oído externo destaca sobre el resto del cuerpo en cómo influye sobre la señal de audio percibida por el oyente [1]. Está formado por el pabellón y el conducto auditivo y modifica el espectro de la señal de audio de forma distinta atendiendo a la dirección desde la que provenga el sonido. Es decir, dependiendo de la dirección desde la que provenga el sonido, la señal sufrirá modificaciones características a esa dirección y específicas a cada oído externo como aparece representado en la Figura 1.1. A partir de estas, el cerebro es capaz de distinguir, en la mayoría de los casos, la posición desde la que proviene el sonido.

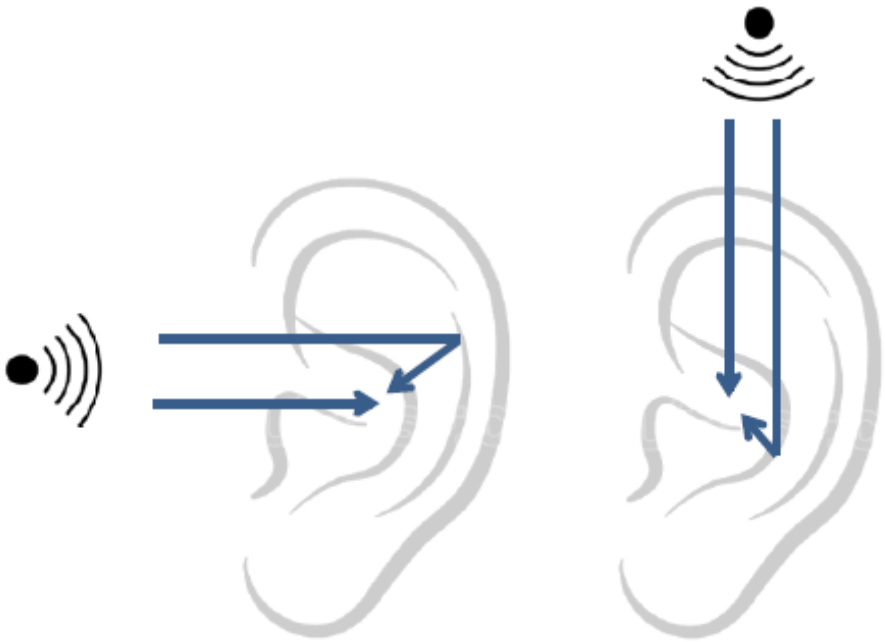

Figura 1.1. Representación gráfica de las distintas modificaciones que realiza el oído externo sobre una señal de audio dependiendo de la dirección desde la que provenga. [1]

En este TFG se propone poder escuchar el audio binaural 3D mediante unos auriculares. Si se reprodujese un audio con ellos, el cerebro interpretaría correctamente que el sonido surge desde el mismo oído, luego habría que modificar la señal reproducida de la misma forma que lo hace el cuerpo del usuario si se quiere que este sea capaz de situar una fuente de audio en el espacio. La suma de todas estas modificaciones es un filtro específico de esa persona aplicado a una señal de audio. Este filtro se puede representar como un conjunto de valores en el dominio de la frecuencia que, aplicado a la señal de audio en el mismo dominio, se obtendría un resultado muy parecido al que obtiene el cerebro en una situación real. A este filtro en el dominio de la frecuencia se le conoce como Función de Transferencia Relacionada con la Cabeza, *Head Related Transfer Function* o HRTF y cada persona tiene uno propio [3].

1.2. Mercado

El sonido 3D es un ámbito que aún está en desarrollo y, por lo tanto, no existen muchos sistemas centrados en su reproducción. Comercialmente se está expandiendo en los videojuegos que usan realidad virtual, ya que, debido a su carácter inmersivo contribuye a la mejora de la experiencia del jugador. La interacción social de jugadores en un entorno simulado se ve especialmente amplificada por la calidad del audio 3D, es decir, si el usuario es capaz de percibir de forma natural la distancia a la que está un objeto o individuo por el sonido, este se verá mucho más inmerso en esa realidad virtual [3].

Tras la pandemia originada por el COVID-19, ha proliferado el uso de herramientas de llamadas o videollamadas *online* tanto para ocio como para el ámbito profesional. En estos casos, la señal de audio que se transmite entre los participantes es única. Esto supone que, si varias personas hablan simultáneamente, se solapan sus discursos y complica el entendimiento al oyente. Aquí es donde aparece otra rama de desarrollo del audio 3D. La audición humana diferencia los sonidos que percibe mediante la posición desde la que provienen entre otras cosas, luego, si se implementa el audio binaural 3D en las llamadas y se sitúan a los interlocutores en el espacio, facilitaría la comunicación y mejoraría la experiencia general del discurso [4].

El prototipo desarrollado en este proyecto integra audio binaural 3D en un sistema empotrado donde se mide el movimiento de la cabeza del usuario y se actúa al respecto. Al realizarse esto sobre un sistema empotrado, se obtiene un sistema de muy baja latencia con el movimiento de la cabeza, algo útil en la investigación dentro de este ámbito. Como este proyecto está destinado a la investigación, no se espera su comercialización, sino que continúe desarrollándose y se use dentro de la Universidad de Málaga [2].

1.3. Descripción del Proyecto

El grupo de investigación DIANA de la Universidad de Málaga [18] ha desarrollado una librería que, basándose en los filtros HRTF, es capaz de simular un entorno auditivo virtual en tres dimensiones en tiempo real para que un oyente

en concreto pueda situar fuentes de audio en el espacio con unos auriculares, también conocido como espacialización binaural.

Mediante el uso de esta librería, llamada 3D Tune-In Toolkit [2], se pueden generar, caracterizar y mover fuentes de audio virtuales en el espacio en tiempo real. También se puede caracterizar y cambiar la posición y orientación del oyente en este espacio de tal forma que el usuario experimente una interacción auditiva con el entorno y las fuentes simuladas. Para poder adaptar el oyente simulado al usuario, es necesario conocer la posición y orientación real de la persona que usa el prototipo, ya que, si no se tiene en cuenta y el usuario se mueve, este entorno permanece siendo el mismo que en la posición inicial y, por lo tanto, acompañará al oyente. Por ejemplo, no sabemos la orientación de la cabeza del usuario, este tiene una fuente delante y gira la cabeza 90º hacia la izquierda, la fuente debería quedarse en la misma posición, es decir, el oyente debería poder situarla a la derecha, sin embargo, la fuente acompaña su giro situándose enfrente suya.

Para suplir este problema en tiempo real, es necesario algún tipo de sensor que sea capaz de leer el movimiento de la cabeza de un usuario para que, mediante las funciones de la librería, se pueda adaptar la fuente de audio a este.

Una posible solución es el uso de un casco de realidad virtual (Figura 1.2). Mediante mensajes OSC (Open Sound Control), que es un protocolo de comunicación centrado en transmitir características de síntesis de sonido, enviaría valores sobre la posición de la cabeza al ordenador que procesa la señal de audio. La librería 3DTI Toolkit adaptaría la o las fuentes de audio a esta posición u orientación de la cabeza y reproduciría el audio modificado en tiempo real. Esta configuración es problemática porque las gafas de realidad virtual realizan muchas más funciones aparte de la de obtener la orientación y posición de la cabeza, suponiendo un coste de procesamiento y de energía adicional para un proceso que es relativamente sencillo. Además, en este caso, la latencia entre la medida del movimiento de la cabeza y la reproducción de audio 3D es alta para esta funcionalidad. En esta necesidad es donde surge y se desarrolla la idea de este TFG.

Figura 1.2. Uso de un casco de VR para procesado de audio con 3DTI Toolkit. Sistema de medida de latencia.

El prototipo desarrollado está formado por la unión entre la placa PocketBeagle [16], el interfaz de audio *Bela Mini* [15] y una placa diseñada apilada sobre esta con un sensor inercial BNO550 [17], **dos potenciómetros** de 10k, una resistencia de 1k, una de 220, un **botón** y un diodo LED (Figura 1.3).

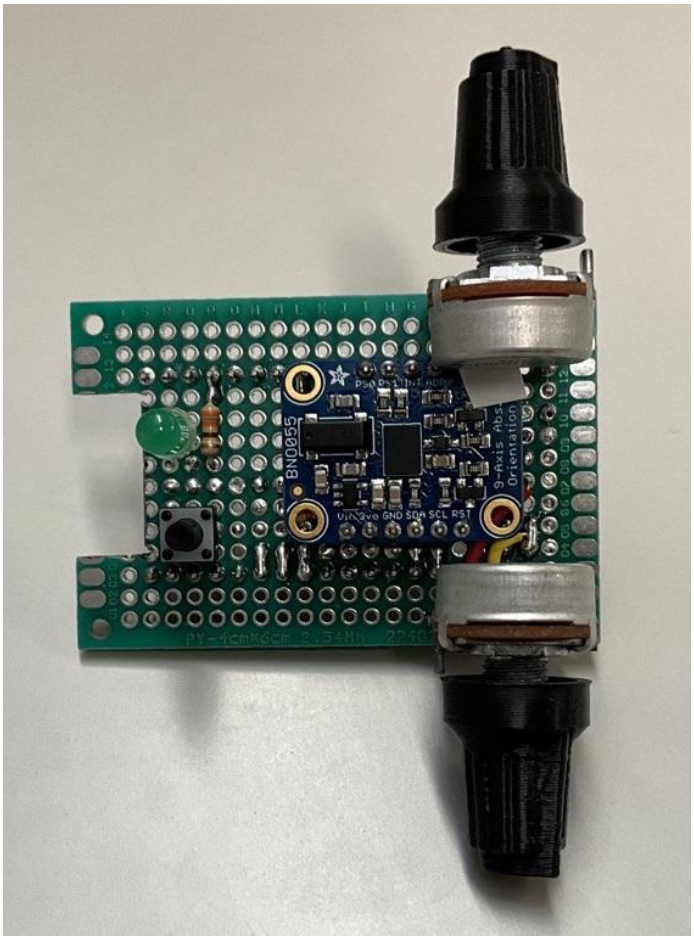

Figura 1.3. Hardware del prototipo desarrollado.

El sistema cargará varias pistas de audio y procesará una de tal forma que, al reproducirla, el usuario sea capaz de situarla en el espacio. Además, incluso si esta persona mueve la cabeza, percibirá que la fuente de audio permanece en la misma posición. La Bela Mini detecta mediante el sensor inercial el movimiento de la cabeza y procesa en tiempo real la pista de audio para adaptarse a este cambio de posición.

En caso de que el usuario haya comenzado el programa con la cabeza en una posición distinta a la natural, puede pulsar el **botón** para que el programa tome la posición actual de la cabeza como punto de referencia y, por lo tanto, colocar las fuentes de audio respecto a esta posición.

Cada pista de audio que ha cargado la Bela Mini se sitúa en una posición distinta. Si el usuario desea cambiar de pista de audio, puede usar uno de los **potenciómetros** para elegir entre seis pistas. El otro **potenciómetro** varía el volumen del sonido reproducido.

Figura 1.4. Sistema final obtenido en el desarrollo del proyecto.

Este sistema está encapsulado en una caja de impresión 3D e irá fijado a unos auriculares de tal forma que cuando una persona los use, quedará encima de su cabeza (Figura 1.4). Se alimentará mediante una *powerbank* de 5V y 3A conectada a la entrada USB de la *Bela Mini* y se conectará la salida de audio de la placa al conector *mini-jack* de los auriculares. Toda esta parte hardware y la impresión 3D se encuentra mejor detallado en el TFG asociado.

1.4. Objetivos del Trabajo Fin de Grado

Este documento describe el desarrollo de un prototipo que se fija sobre la diadema de unos auriculares de forma que quede paralelo al suelo cuando lo lleve una persona. Este sistema reproduce una fuente de audio por unos auriculares de tal forma que el oyente pueda situarla en las tres dimensiones del espacio a su alrededor. Además, actuará frente a cambios de orientación de la cabeza del usuario actualizando mediante un filtro la fuente de audio para que permanezca en la misma posición virtual independientemente del movimiento de la cabeza del oyente. Esta lectura se realiza con un sensor inercial y se usa la placa Bela Mini [15] para procesar tanto la lectura del sensor como el procesamiento de las fuentes como la reproducción del audio.

En este trabajo proponemos realizar dos aplicaciones finales para el uso del sistema descrito y se desarrolla el proceso llevado a cabo para optimizar el sistema y asegurar su funcionamiento.

Una aplicación realiza la función especificada en el párrafo anterior con varios actuadores para modificar el volumen, escoger entre distintas fuentes de audio o establecer la posición que tenga la cabeza en ese instante como posición de referencia. La otra aplicación genera un sistema para poder tomar medidas de latencia del proyecto final en un banco de ensayos.

1.5. Requisitos

Id.	Nombre	Descripción	Prioridad
R1	Audio	El sistema debe poder reproducir audio.	1
R2	Sensor	El sistema debe medir el movimiento de la cabeza del usuario mediante un sensor inercial.	1
R3	Controlable	Se puede interactuar con el sistema durante su funcionamiento.	3
R4	Portátil	Es sencillo desplazar el sistema.	2
R5	Espacialización de audio	El sistema debe espacializar una fuente de sonido en una posición estable del entorno adaptando la reproducción mediante renderizado binaural a la orientación de la cabeza del oyente.	1

Tabla 1.1. Tabla de descripción de requisitos

Id.	Nombre	Descripción	Prioridad
R1.1	Tiempo real	Debe procesar el audio en tiempo real.	1
R1.2	3DTI Toolkit	El procesado de audio del sistema debe realizarse mediante la librería 3DTI Toolkit.	1
R1.3	Bela	El sistema debe implementarse sobre la plataforma Bela.	1
R1.4	Calidad	El audio reproducido no debe tener errores o ruido.	2
R1.5	Fuentes	El sistema debe poder reproducir más de una fuente de audio.	2
R2.1	Lectura	Debe proporcionar variables tipo Quaternion.	2
R2.2	Constante	No debe tener glitches y, si tiene derivas, que sean controlables.	2
R2.3	Rápido	Debe hacer una lectura para cada muestra de audio.	1
R3.1	Volumen	Debe tener un control de volumen manual.	3
R3.2	Variedad	Debe tener un selector de pista manual.	3
R3.3	Reposicionamiento	Debe tener un botón para reestablecer la posición de inicial.	2
R3.4	LED	Debe tener un LED habilitado y sin función inicial.	4

Tabla 1.2. Tabla de descripción de requisitos detallados

1.5.1. Criterios de comprobación

R1. Audio. Encender el sistema, ejecutar la aplicación básica, conectar los auriculares y escuchar por ellos.

R1.1. Tiempo real. Comprobar mediante la depuración de la aplicación básica en Eclipse que la función de procesado de audio se ejecuta dentro de la función render() de la aplicación que se encarga de reproducir el audio.

R1.2. 3DTI Toolkit. Comprobar mediante la depuración de la aplicación básica en Eclipse que se ejecutan las funciones de la librería 3DTI Toolkit.

R1.3. Bela. Comprobar mediante la depuración de la aplicación básica en Eclipse que se ejecutan las funciones de Bela con su compilador específico. Ejecutar la aplicación en Bela y se debe escuchar el resultado.

R1.4. Calidad. Reproducir una pista de audio desde un ordenador. Usar la misma pista de audio en la aplicación básica y comprobar que el contenido es el mismo.

R1.5. Fuentes. Colocar dos fuentes de audio y ejecutar la aplicación básica. Comprobar que ambas pistas cumplen el requisito de calidad.

R2. Sensor. Ejecutar la aplicación de la incorporación del sensor y preparar físicamente el sistema. Se debe detectar un cambio en la señal de audio relacionado con el movimiento de la cabeza.

R2.1. Lectura. Comprobar mediante la depuración de la aplicación básica en Eclipse que las funciones del controlador del sensor devuelven Quaternion.

R2.2. Constante. Realizar una prueba donde se coloca el sensor en todas las posiciones posible y se representan los valores obtenidos en una función donde se puede ver la evolución de los valores obtenidos y analizarlos estadísticamente.

R2.3. Rápido. Crear un programa donde se realice la lectura del sensor en la función render() y comprobar que no surgen errores.

R3. Controlable. Llevar el prototipo, ejecutar la aplicación final y accionar todos los actuadores del sistema. Debe tener algún tipo de repercusión en el funcionamiento.

R3.1. Volumen. Llevar el prototipo, ejecutar la aplicación final y mover el potenciómetro encargado de variar el volumen de un extremo a otro. La intensidad del sonido debe variar en la misma proporción.

R3.2. Variedad. Llevar el prototipo, ejecutar la aplicación final y mover el potenciómetro encargado de seleccionar la pista de audio de un extremo a otro. Debe cambiar la pista de audio reproducida en el proceso.

R3.3. Reposicionamiento Llevar el prototipo, ejecutar la aplicación final y mover la cabeza y pulsar el botón. La fuente de audio debe escucharse en la misma posición que cuando comenzó a sonar el audio tras pulsarlo.

R.3.4. LED. Encender el LED mediante una aplicación simple en Bela.

R4. Portátil. Moverse con el prototipo con libertad y comprobar que sigue funcionando.

R5. Espacialización del audio. Ejecutar la aplicación final y comprobar que funciona correctamente.

1.6. Estructura de la memoria

El presente documento se estructura como se indica a continuación:

- **Capítulo 1**. Introducción.

En este capítulo se desarrolla el contexto en el que se ha llevado a cabo este proyecto. En él, se definen varios conceptos necesarios para entender el desarrollo del prototipo y su necesidad. Se valoran las distintas opciones que existen en el mercado que podrían realizar la misma función que realiza el prototipo y se hace una breve descripción del proyecto, su objetivo y los requisitos que ha de cumplir.

- **Capítulo 2**. Despliegue de librería 3DTI Toolkit en Bela.

En este capítulo se muestra el procedimiento a seguir para reproducir la parte software del prototipo desde cero para cualquier aplicación. Se desarrollan las distintas plataformas y recursos utilizados en el proceso, por qué se han escogido y cuál es su funcionalidad. Se muestra de forma gráfica la configuración necesaria de las distintas aplicaciones para llegar a ejecutar una aplicación que use la librería 3DTI Toolkit en la plataforma Bela mediante Eclipse como IDE.

- **Capítulo 3**. Pruebas para la optimización del sistema.

En este capítulo se exponen los problemas encontrados durante el desarrollo del sistema y que incapacitaban o dificultaban su funcionamiento. También se abordan las distintas pruebas llevadas a cabo para mejorar el rendimiento del sistema y los cambios realizados definitivos que solucionaron los problemas críticos del sistema y lo optimizaron.

- **Capítulo 4**. Aplicaciones.

En este capítulo se detallan las distintas aplicaciones desarrolladas hasta que llegar a la aplicación final. Se centra en aquellas funciones del código que son más importantes para el funcionamiento del sistema y explica de forma gráfica y general qué hace cada bloque de este.

- **Capítulo 5**. Conclusiones y trabajo futuro.

En este capítulo se tratan todas las dificultades encontradas durante el desarrollo del prototipo de este TFG, las limitaciones del sistema y si ha sido satisfactorio. Finalmente, se desarrollan distintas líneas futuras en las que se podría seguir trabajando sobre el prototipo creado.

- **Capítulo 6**. Referencias.

En este capítulo se encuentra la bibliografía desde la que se ha obtenido la información para el desarrollo del proyecto y para la redacción de este TFG.

- **Capítulo 7**. Anexos.

En este capítulo se muestran fragmentos de código utilizados en la aplicación.

Capítulo 2. Despliegue de librería 3DTI Toolkit en Bela

2.1. Bela

Bela [7] es una plataforma computacional empotrada destinada a generar aplicaciones con un entorno de desarrollo relativamente sencillo para atraer a todo tipo de usuarios independientemente de sus conocimientos sobre programación. Está construida en base a BeagleBone [5] [6], una familia de ordenadores empotrados de código abierto que combina su procesamiento con la precisión y conectividad que proporciona el hardware de Bela.

Bela consta de su propio IDE al que se accede mediante cualquier navegador web por la conexión USB, luego, no se necesita de ningún software adicional para trabajar con ella. Esto y sus librerías hacen que el sistema atraiga a toda persona que tenga interés en programar en C++, Pure Data o simplemente, en generar algún sistema interactivo.

Aunque Bela está muy centrada en ser accesible para todos los públicos, realmente destaca por ser una plataforma de ultra baja latencia y alta calidad en el procesamiento de audio [7]. Esto y la existencia de una placa de Bela lo suficientemente pequeña (Figura 2.1) para que no sea incómoda al ponerse sobre la cabeza de una persona hizo que se decidiera realizar el proyecto con ella.

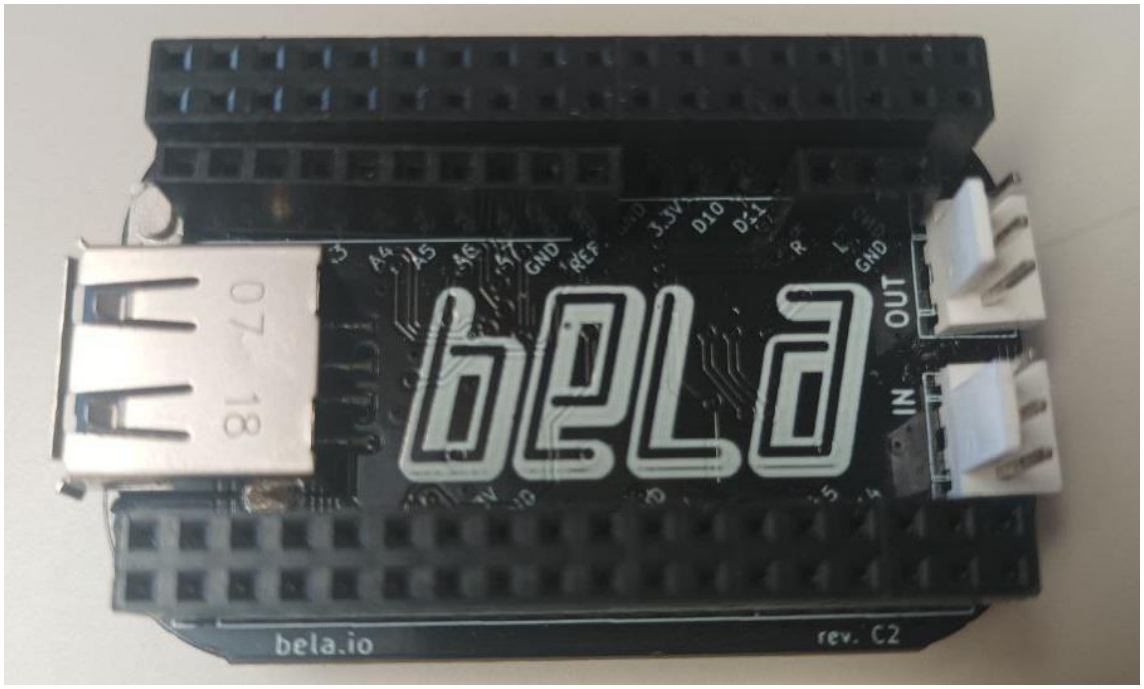

Figura 2.1. Placa Bela Mini utilizada.

A continuación, se exponen los pasos a seguir respecto a Bela para reproducir el sistema prototipo de este TFG.

2.1.1. *Flash* de la tarjeta micro SD

En un primer lugar, es necesario instalar el software de Bela desde el ordenador a una tarjeta micro SD que será introducida en la Bela Mini o en la BeagleBone Black dependiendo de qué dispositivo vayamos a usar aunque el software es el mismo [8].

Para “flashear” la micro SD se ha usado el programa Balena Etcher [20]:

1. Introduce la tarjeta SD en el ordenador (dependiendo del ordenador quizás sea necesario un adaptador a tarjeta SD)

2. Descargar la imagen de Bela que se quiere instalar en la tarjeta (bela_image_vX.X.Xx.img.xz) desde su repositorio en Github (Figura 2.2):

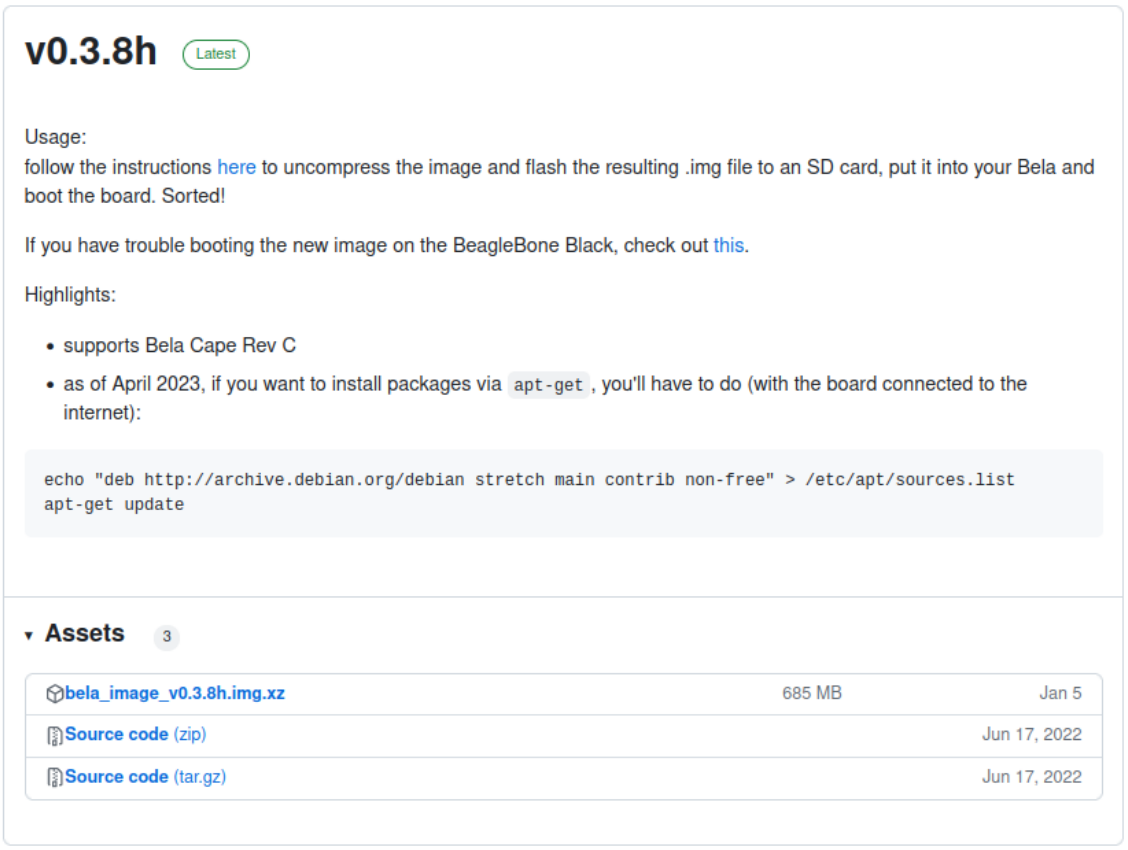

Figura 2.2. Página web de Github donde se puede descargar la imagen de Bela

3. Ejecutar Balena Etcher y seleccionar la imagen de Bela descargada (Figura 2.3).

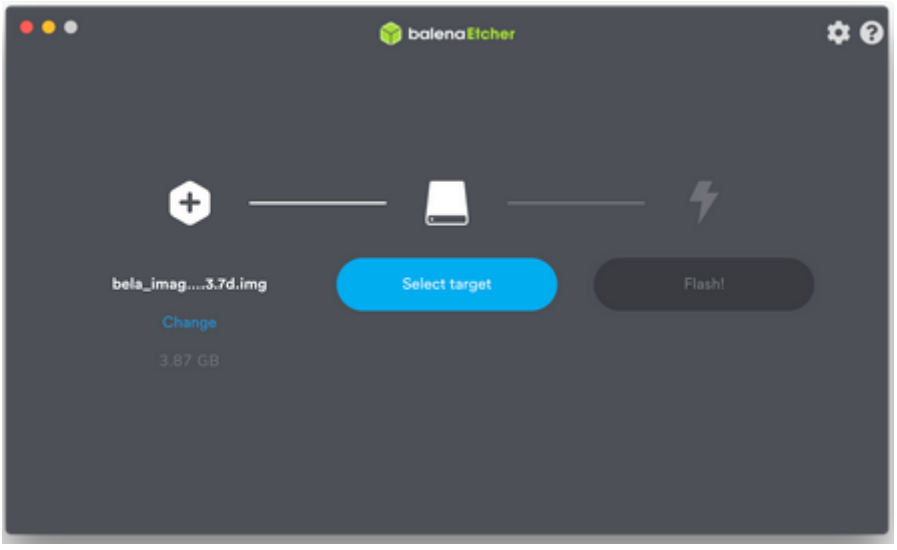

Figura 2.3. Aplicación Balena Etcher con la imagen de Bela seleccionada.

4. Seleccionar la tarjeta SD introducida y flashear (Figura 2.4).

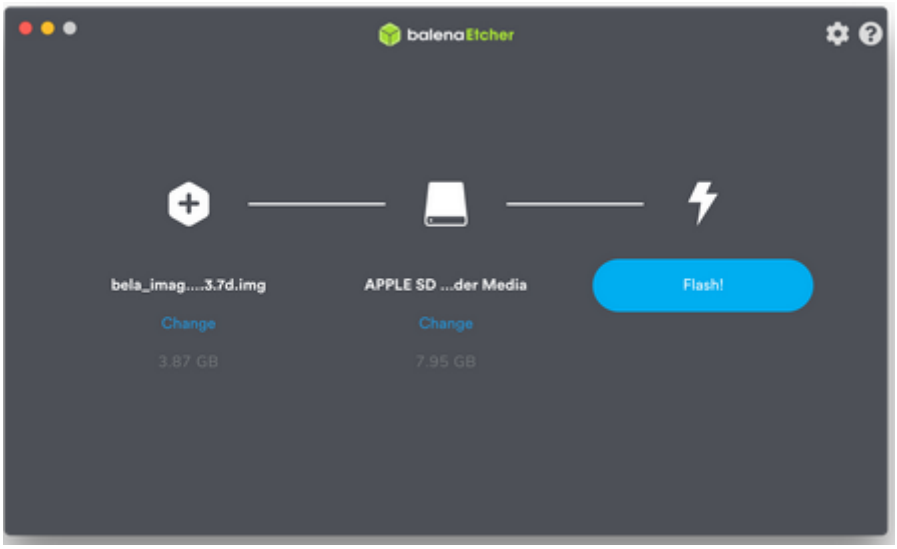

Figura 2.4. Aplicación Balena Etcher con la tarjeta MicroSD seleccionada.

5. Extraer la tarjeta SD del ordenador e insertarla en la ranura de la placa correspondiente.

Bela Mini ya está configurada para que use automáticamente el software de la tarjeta, sin embargo, la BeagleBone Black está naturalmente configurada para que se inicie con el software de su eMMC (*embedded MultiMediaCard*), luego, se debe realizar una serie de acciones para que se inicie con el software de la tarjeta micro SD conectada a la placa.

Para configurar la BeagleBone Black, con la tarjeta SD con el software cargado introducida, presionar el botón S2 de la placa mientras esta se conecta al ordenador y esperar hasta que el LED más cercano a la entrada Ethernet de la placa parpadee al ritmo de un latido. Conectarse al IDE de Bela mediante un navegador web como Google Chrome o Mozilla Firefox y escribiendo en él la dirección IP 192.168.6.2 si está en Windows o la dirección IP 192.168.7.2 si está en Linux. Una vez se ha accedido al IDE de Bela, escribir en la consola el siguiente comando:

`/opt/Bela/bela_bootloader.sh.`

Esto hace que siempre que conectemos la placa, use como sistema operativo aquel que tengamos instalado en la tarjeta SD.

Se debe comprobar que todo ha salido correctamente desconectando Bela y volviéndola a conectar al ordenador. Si con esto automáticamente se puede acceder al IDE mediante el navegador web escribiendo la dirección antes mencionada o “bela.local”, significa que el proceso se ha realizado correctamente. Finalmente, es recomendable compilar un proyecto básico ejemplo de Bela desde el IDE antes de comenzar a programar en Eclipse porque Bela necesita crear algunos archivos internos que genera automáticamente cuando se compila por primera vez desde su IDE. Este proceso se explica detalladamente en la página web oficial de Bela [8] https://learn.bela.io/tutorials/c-plus-plus-for-real-time-audio-programming/setting-up/.

2.1.2. Archivos para compilación cruzada

La compilación cruzada es un proceso de compilación en el que se genera un código ejecutable para una plataforma distinta a aquella en la que el compilador se ejecuta. Esta se llevará a cabo desde un sistema operativo Linux y se generará el código ejecutable para Bela. Es necesario crear algunas carpetas y usar algunos archivos adicionales externos a Bela para poder compilar nuestros programas desde el ordenador [9].

En primer lugar, se crearán las carpetas Debug y Release dentro de Bela para almacenar ahí los ejecutables generados por Eclipse. Se podrían crear las carpetas directamente mediante una carpeta compartida (esta se utilizará y se desarrolla en el siguiente apartado), pero en este caso, se realizarán todas estas acciones mediante la ventana de comandos de Linux en el ordenador. El comando ssh permite la conexión de forma remota a la placa y modificar sus archivos escribiendo las siguientes líneas en la consola:

```
ssh root@192.168.7.2 mkdir -p BelaRemote/Debug
ssh root@192.168.7.2 mkdir -p BelaRemote/Release
```

Instalamos el compilador cruzado en el ordenador para generar el código ejecutable para la arquitectura hardware de la plataforma Bela escribiendo lo siguiente en la consola:

```
mkdir /usr/local/linaro

sudo chmod 777 /usr/local/linaro/

cd /usr/local/linaro

wget https://releases.linaro.org/components/toolchain/binaries/6.3-2017.02/arm-linux-gnueabihf/gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf.tar.xz

tar xf gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf.tar.xz

mv gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf arm-bela-linux-gnueabihf

rm -r gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf.tar.xz
```

Para finalizar, se instalarán una serie de librerías e *includes* de Bela en el ordenador que serán necesarias para la compilación. Para ello, hay que crear la carpeta `/usr/local/linaro/BelaSysroot,` situarse en ella y escribir en la ventana de comandos las siguientes sentencias:

```
wget https://raw.githubusercontent.com/AndrewCapon/OSXBelaCrossCompiler/master/SyncBelaSysroot.sh

chmod +x SyncBelaSysroot.sh

./SyncBelaSysroot.sh
```

2.1.3. Carpeta compartida

Para poder utilizar todos los ficheros de Bela y acceder a la librería 3DTI Toolkit sin tener que cargarla directamente en la placa, se ha creado una carpeta compartida entre el ordenador y Bela. Por este directorio se podrá acceder a los archivos de Bela y se utilizará para los includes y los linkados de la configuración de Eclipse. Esta carpeta desaparece cuando se desconecta Bela del ordenador,

luego, cada vez que se conecte es necesario vincularla de nuevo. Para ello se usa la siguiente línea desde la ventana de comandos:

```
sshfs root@192.168.7.2:/ /home/sergio/BelaWorkspace/BelaShared -o transform_symlinks
```

En este caso, la carpeta compartida es BelaShared en ese determinado directorio. Esta dirección deberá tenerse siempre en cuenta para la compilación del proyecto.

Para el uso de recursos por el programa, como pueden ser ficheros o pistas de audio, será necesario crear una carpeta en la carpeta compartida dentro de la carpeta “root”. En este directorio hay que crear una carpeta llamada “resources” y dentro de esta se creará una carpeta llamada “Samples” (Figura 2.5).

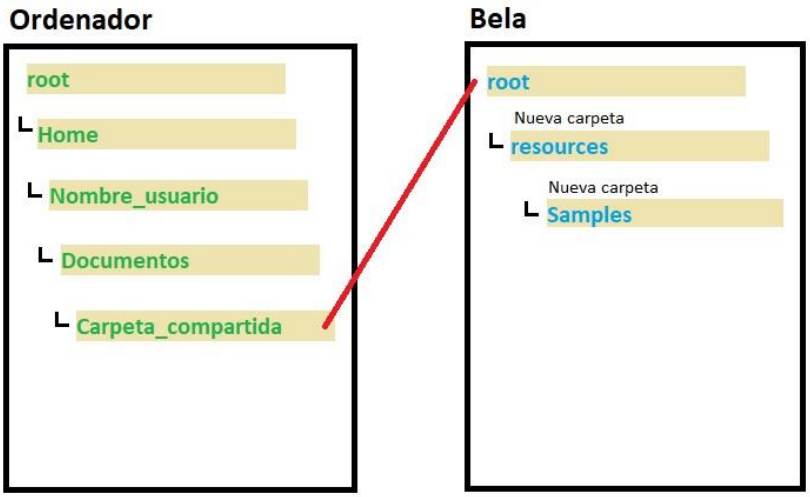

Figura 2.5. Representación del directorio donde hay que crear las carpetas “*resources*” y “*Samples*”.

2.1.4. Comunicación entre Bela y Eclipse

La conexión entre Eclipse y Bela para Debug y Release se realiza mediante gdb, luego, es necesario que Bela tenga instalado gdbserver. Además, para que esto funcione, debido a una diferencia de versiones, es necesario instalar varias librerías antiguas en BeagleBone Black.

- En BeagleBone Black. Conectamos la placa a internet mediante un cable Ethernet y desde la consola del IDE de Bela escribimos los siguientes comandos:

```
sudo apt-get install gdbserver
sudo apt-get install libncurses5:i386
sudo apt-get install libncurses5
sudo apt-get install libpython2.7
```

- En Bela Mini. No tenemos conexión por Ethernet pero no es necesario instalar las librerías antiguas, luego, descargaremos desde el ordenador un gdbserver.deb y lo pondremos dentro de Bela mediante la carpeta compartida. Una vez hecho esto, realizar el siguiente comando en el IDE de Bela:

```
sudo dpkg -i gdbserver.deb
```

2.2. 3DTI Toolkit

Tal y como se describió anteriormente, cada persona sitúa una fuente de audio en el espacio de una forma distinta ya que depende de los rebotes y atenuaciones de las señales de audio en su cuerpo, por lo tanto, para poder repetir esto con una fuente virtual, sería necesario hacer un estudio específico de una persona para generar un filtro que se adecúe a su cuerpo. En cambio, se han creado una serie de filtros genéricos que se asemejan lo suficiente a los específicos de cada individuo para que, de alguna forma, aunque no se adapten perfectamente a sus respectivos HRTF engloben a la mayoría de las personas. La librería 3DTI Toolkit [2], realizada por el grupo de investigación DIANA de la Universidad de Málaga [18], realiza una gran cantidad de cálculos en tiempo real para crear una fuente de audio virtual en el espacio relativo al oyente a partir de estos filtros. Esto requiere de mucho procesamiento y de baja latencia,

características propias de Bela, es por esa razón por la que se adaptaba bien a este proyecto.

Esta librería contiene diversas funciones para modificar las características tanto de las fuentes como del oyente. Algunas de estas funciones no son necesarias en la aplicación desarrollada y otras, aunque puedan ser convenientes, aporten riqueza a la aplicación o sea interesante su estudio como la reverberación de una fuente de audio en el entorno, son demasiado complejas y requieren de un alto procesamiento que, en tiempo real, Bela no es capaz de proporcionar. Por lo tanto, estas funciones no se usan o se deshabilitan en el código. Además, algunas de esas funciones en la librería necesitan de una versión más reciente de C++ que la que usa la imagen de Bela, por lo tanto, se decidió modificar la librería para que solo incluyera aquellos archivos estrictamente necesarios para el funcionamiento de la aplicación desarrollada [2].

A continuación, se exponen los pasos a seguir respecto a la librería 3DTI Toolkit para reproducir el sistema prototipo de este TFG [10].

2.2.1. Descarga de librería

Atendiendo a la decisión de simplificar la librería, se proporciona la versión simplificada junto con este TFG. Sin embargo, también existe la posibilidad de descargar los archivos de la librería oficial desde Github [10] y simplificarla manualmente desde el siguiente enlace https://github.com/3DTune-In/3dti_AudioToolkit_Examples.

2.2.2. Generación de lib3dti.a

Tal y como se ha desarrollado anteriormente, la librería 3DTI Toolkit es relativamente extensa, esto hace que, si se incluye tal y como se descarga en el proyecto de nuestra aplicación, tendrá que compilar todos los archivos de la librería cada vez que se realice algún cambio en el programa. Por lo tanto, se ha creado un archivo Makefile que compila la librería y genera un archivo lib****.a que solo es necesario compilar una vez y se puede incluir en el proyecto como

una librería más, introduciéndola en la orden de linkado del compilador ya que contiene los ficheros .o resultantes de la compilación de todos los ficheros .cpp de la librería 3DTI Toolkit, es decir, de esta forma, nos ahorramos la compilación de la librería cada vez que se desee compilar el código de la aplicación que la usa.

Un Makefile es un fichero donde se pueden agrupar una serie de comandos en lenguaje *make* de tal forma que mediante el comando del mismo nombre *make,* en el directorio del Makefile, se realicen todas las sentencias escritas en el fichero. Supone un ahorro de tiempo en el caso de que sea necesario realizar una serie de líneas de comando de forma sistemática.

Esto es muy conveniente porque en el Makefile se pueden especificar qué archivos de la librería se desea compilar, por lo tanto, no sería necesario eliminar aquellos archivos de la librería que Bela no entienda por su versión de C++, simplemente, se pueden ignorar, es decir, no se compilan ni se incluyen en el archivo .a generado.

```
1
2 3DTI_PATH     =  $(HOME)/Documentos/3dti_toolkit/3dti_AudioToolkit_Examples/3dti_AudioToolkit
3
4 CC            =  /usr/local/linaro/arm-bela-linux-gnueabihf/bin/arm-linux-gnueabihf-g++
5 FLAGS         =  -std=c++11 -fpermissive -lpthread -fPIC
6 INCLUDE1      =  $(3DTI_PATH)/3dti_ResourceManager
7 INCLUDE2      =  $(3DTI_PATH)/3dti_Toolkit
8 INCLUDE3      =  $(INCLUDE1)/third_party_libraries/cereal/include
9 INCLUDE4      =  $(INCLUDE1)/third_party_libraries/boost_circular_buffer
10 LIB          =  $(HOME)/lib
11
12 CFLAGS    =  $(FLAGS) -I. -I$(INCLUDE1) -I$(INCLUDE2) -I$(INCLUDE3) -I$(INCLUDE4)
13 LDLIBS    =
14 LDFLAGS   =
15
16 all : 3dti clean
17
18 3dti:
19
20         $(CC) $(CFLAGS) -c $(3DTI_PATH)/3dti_ResourceManager/HRTF/*.cpp
21
22         $(CC) $(CFLAGS) -c $(3DTI_PATH)/3dti_Toolkit/BinauralSpatializer/*.cpp
23
24         $(CC) $(CFLAGS) -c $(3DTI_PATH)/3dti_Toolkit/Common/*.cpp
25
26         ar rcs lib3dti.a *.o
27
28 clean:
29         rm -f *.o
30
31         rm -f $(3DTI_PATH)/3dti_ResourceManager/HRTF/*.o
32
33         rm -f $(3DTI_PATH)/3dti_Toolkit/BinauralSpatializer/*.o
34
35         rm -f $(3DTI_PATH)/3dti_Toolkit/Common/*.o
36
37
38
39
```

Figura 2.6. Fichero Makefile destinado a la creación de la librería lib3dti.a.

En este Makefile (Figura 2.6) se incluyen todos los archivos .h y librerías adicionales necesarias para la compilación de todos los archivos .cpp que vayamos a usar de la librería 3DTI Toolkit. Se compilan todos los archivos especificados con sus *paths* a sus respectivos directorios mediante el compilador cruzado para Bela y los archivos .o generados se almacenan en un archivo llamado lib3dti.a. Esto es una librería de archivos .o resultantes de la compilación que se puede incluir a un proyecto y, de esta manera, no es necesario compilarlo cada vez que se haga en el proyecto. Finalmente, se eliminan todos los archivos .o generados para mantener limpias las carpetas del ordenador.

2.3. Entorno de desarrollo. Eclipse

Eclipse [11] es un entorno de desarrollo software para C, C++ y Java construido alrededor de un *workspace* (o espacio de trabajo) que puede ser

modificado dependiendo de las funcionalidades que se quiere que tenga la aplicación. Este entorno permite la modificación de todas las características de un proyecto además de la definición de la línea de comando para compilarlo y linkarlo.

Aunque Bela ya consta de un IDE, la distribución y la programación que usa no es conveniente para el desarrollo una aplicación con 3DTI Toolkit por varias razones:

- Ejecuta un proyecto a partir de un archivo llamado render.cpp que no incluye una función main().
- No permite usar librerías externas a Bela porque se obtiene un error al introducir la carpeta de la librería en el directorio del proyecto y como no está en el formato que espera el IDE, no compila el programa.
- No permite depurar el programa. La combinación entre Bela y 3DTI Toolkit (que es una librería compleja) es innovadora, es decir, no se ha hecho antes y se trata de un proceso complejo, luego, necesitamos ser capaces de ver paso a paso si las líneas de código funcionan adecuadamente y en caso de error encontrar donde está el problema.

Para solucionar estos inconvenientes se decidió el uso de Eclipse a través de una carpeta compartida con Bela. Esto permite modificar los archivos internos de Bela para la configuración del proyecto y evita tener que introducir la librería 3DTI Toolkit dentro de Bela ya que mediante compilación cruzada se generará un ejecutable con la aplicación.

Una vez preparadas la librería 3DTI Toolkit, la configuración de los archivos de Bela y creado la carpeta compartida entre el ordenador y la placa, es posible crear un proyecto en Eclipse para generar una aplicación en C++ y que podamos depurarla mediante compilación cruzada. La aplicación se realizará a partir de un ejemplo básico del uso de la librería 3DTI Toolkit, sin embargo, es necesario establecer ciertos parámetros en la creación del proyecto.

A continuación, se exponen los pasos a seguir para crear y configurar un nuevo proyecto en Eclipse que genere una aplicación resultado de la unión entre 3DTI Toolkit y Bela.

2.3.1. Crear proyecto

1- Para crear un nuevo proyecto desde la pantalla inicial de Eclipse, situar el ratón sobre la pestaña File, abrir New Project y seleccionar C++ Project que abrirá una nueva pestaña (Figura 2.7).

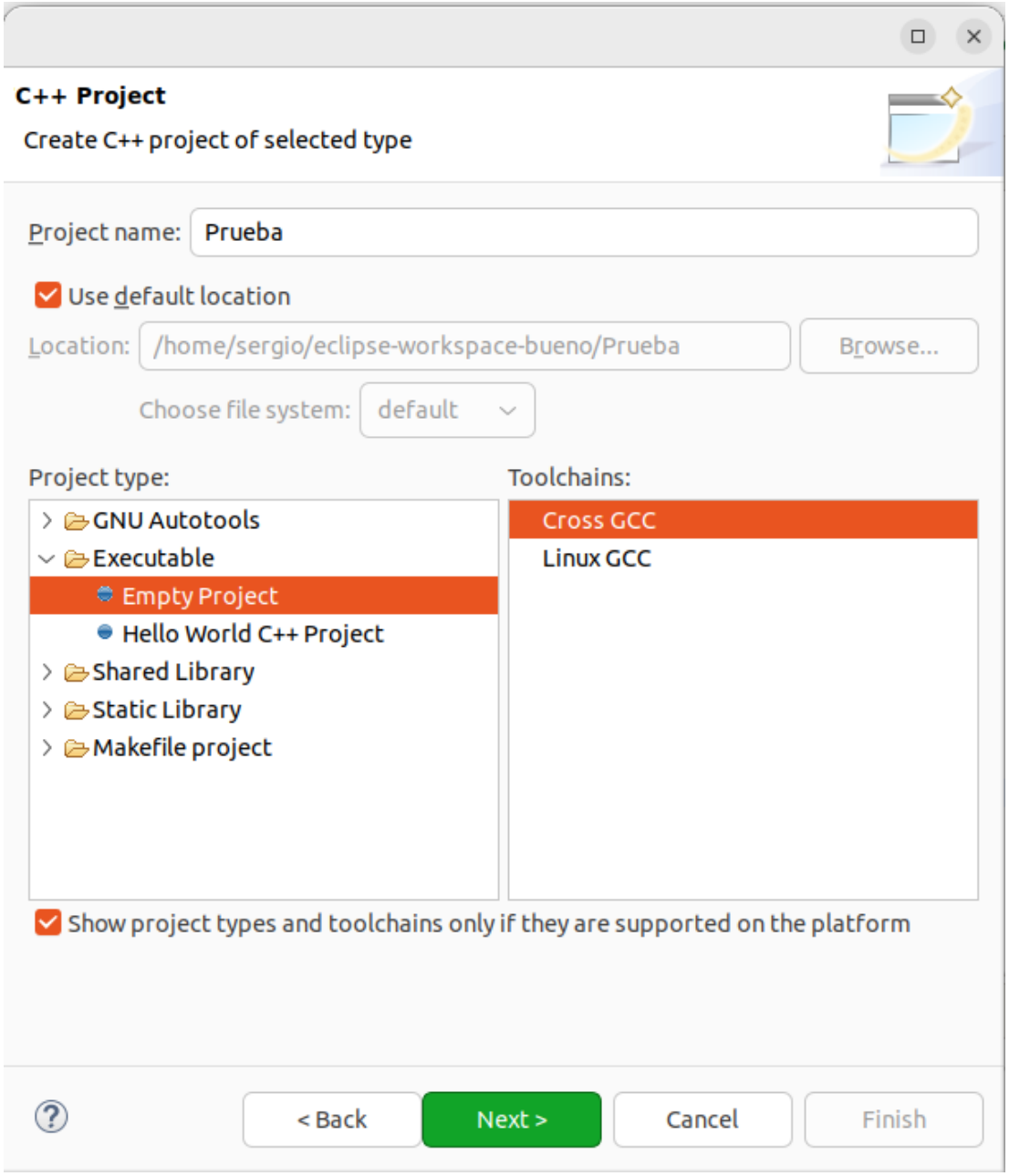

Figura 2.7. Ventana generada en Eclipse al crear nuevo proyecto de C++.

2- Escribir el nombre que se le desea dar al proyecto en *Project name* y seleccionar las opciones *Empty Project* para crear un proyecto vacío y *Cross GCC* para decirle a Eclipse que queremos usar un compilador que no es el que usa Eclipse por defecto. Clic sobre *Next*.

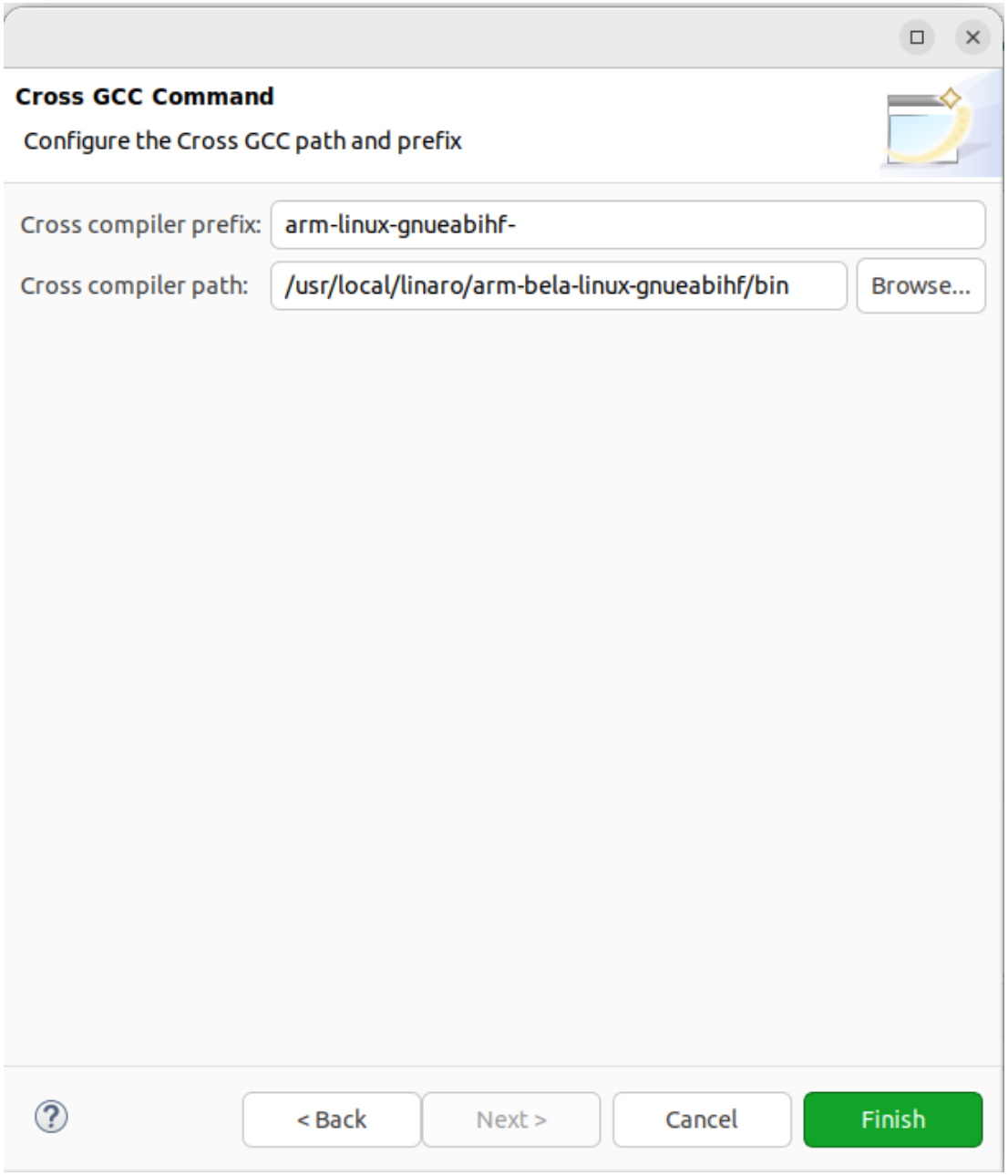

Figura 2.8. Ventana de Eclipse de la creación de un nuevo proyecto donde definir el compilador.

3- Se abrirá una nueva pestaña donde Eclipse pregunta qué compilador queremos usar y dónde está situado. Si se ha instalado el compilador cruzado correctamente como se ha explicado en el anterior apartado de Bela, completar los cajetines tal y como se muestra en la Figura 2.8.

2.3.2. Propiedades

1- Una vez generado el proyecto, hacer clic derecho sobre él y seleccionar la opción Properties. Abrir el desplegable de C/C++ Build y seleccionar Tool Chain Editor. Habrá que asegurarse que en todas las configuraciones, el resto de seleccionables de la pestaña son iguales que en la Figura 2.9 y hacer clic izquierdo sobre Apply.

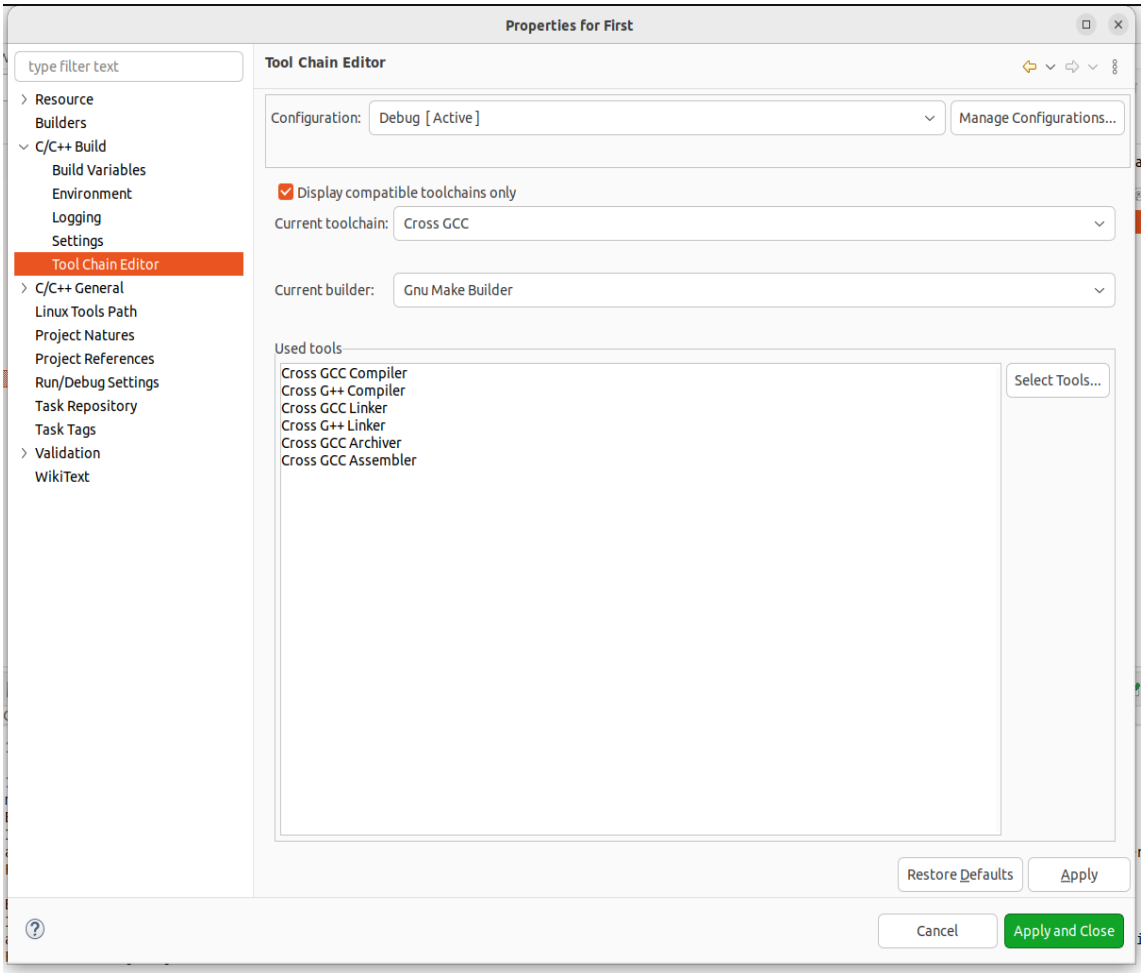

Figura 2.9. Ventana de Eclipse de las propiedades de un proyecto. Tool Chain Editor.

2- A la izquierda en C/C++ Build, seleccionar Settings y en el cajetín de Configuration seleccionar [All Configurations]. Lo primero que debe aparecer es el compilador cruzado que se seleccionó al crear el proyecto (Figura 2.10).

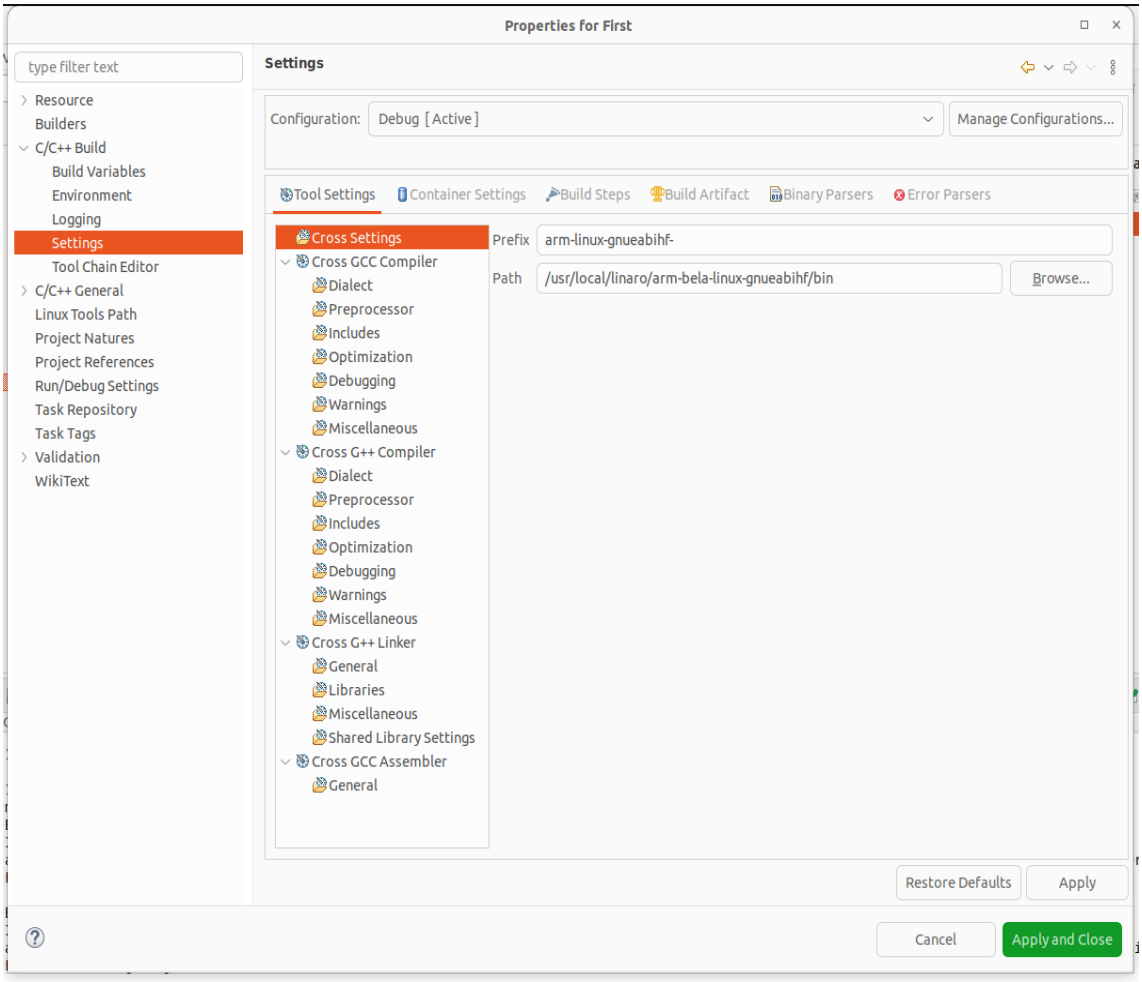

Figura 2.10. Ventana de Eclipse de las propiedades de un proyecto. *Settings* → *Cross Settings*.

3- Hacer clic izquierdo sobre Includes dentro de Cross G++ Compiler y dentro del recuadro Include paths (-I) escribir los directorios donde se encuentran los .h que necesitamos para nuestra aplicación (Figura 2.11).

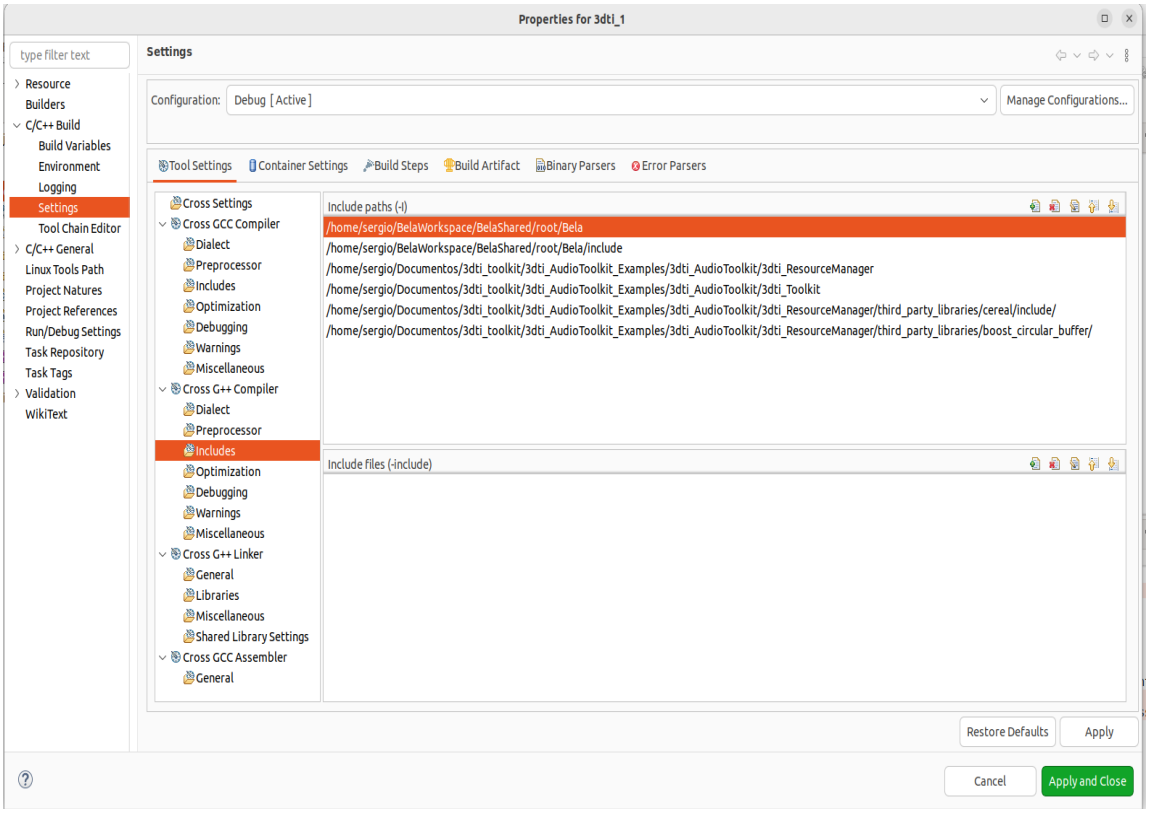

Figura 2.11. Ventana de Eclipse de las propiedades de un proyecto. *Settings* → *Cross G++ Compiler* → *Includes*.

- Mediante la carpeta compartida se accederá a las carpetas que tienen los archivos .h necesarios dentro de Bela. En este caso son:

/home/sergio/BelaWorkspace/BelaShared/root/Bela

/home/sergio/BelaWorkspace/BelaShared/root/Bela/include

- El resto de los directorios llevan a las carpetas de la librería 3DTI Toolkit a partir de la que el programa principal buscará los archivos .h. En este caso son:

/home/sergio/Documentos/3dti_toolkit/3dti_AudioToolkit_Examples/3dti_AudioToolkit/3dti_ResourceManager

/home/sergio/Documentos/3dti_toolkit/3dti_AudioToolkit_Examples/3dti_AudioToolkit/3dti_Toolkit

/home/sergio/Documentos/3dti_toolkit/3dti_AudioToolkit_Examples/3dti_AudioToolkit/3dti_ResourceManager/third_party_libraries/cereal/include/

/home/sergio/Documentos/3dti_toolkit/3dti_AudioToolkit_Examples/3dti_AudioToolkit/3dti_ResourceManager/third_party_libraries/boost_circular_buffer/

4- Hacer clic izquierdo sobre Apply y seleccionar Miscellaneous dentro de Cross G++ Compiler. En Other flags se escribirán flags que caractericen el compilado (Figura 2.12).

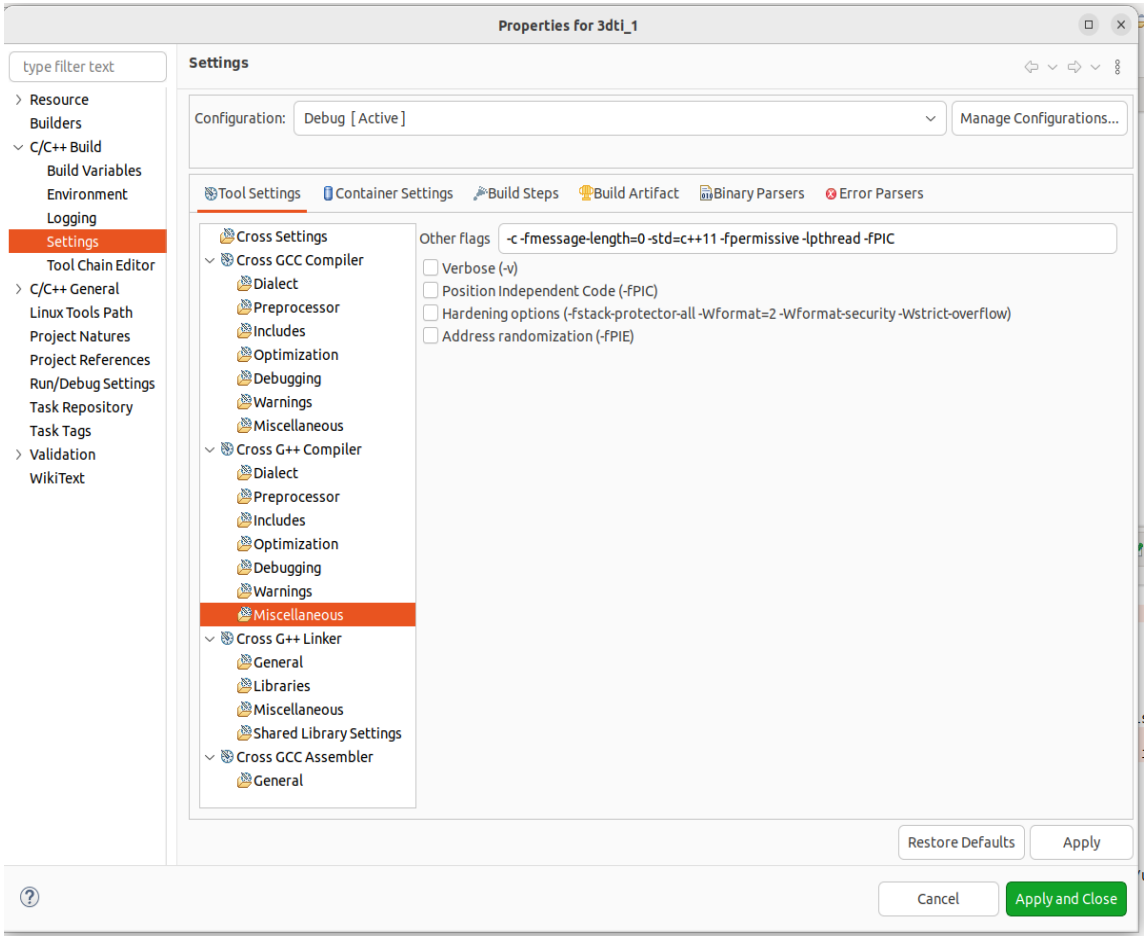

Figura 2.12. Ventana de Eclipse de las propiedades de un proyecto. *Settings → Cross G++ Compiler → Miscellaneous.*

La línea escrita es la siguiente:

-c -fmessage-length=0 -std=c++11 -fpermissive -lpthread -fPIC

5- Tras pulsar sobre Apply, hacer clic izquierdo sobre Libraries dentro de Cross G++ Linker (Figura 2.13).

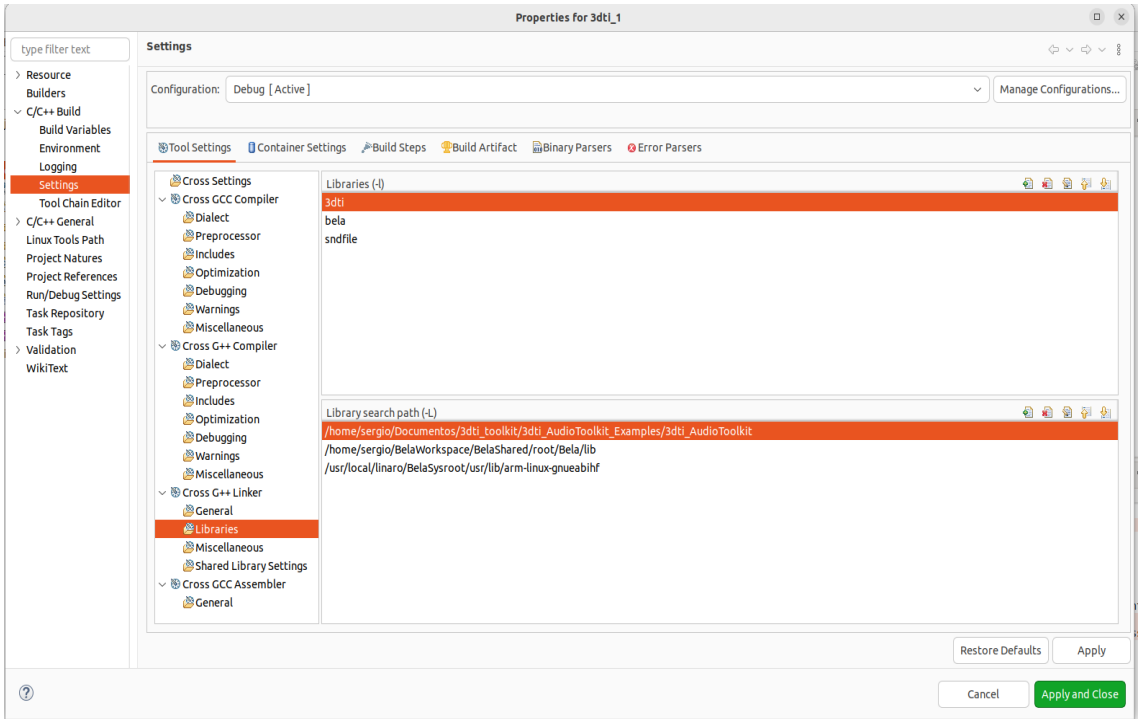

Figura 2.13. Ventana de Eclipse de las propiedades de un proyecto. *Settings → Cross G++ Linker → Libraries.*

- Dentro de Libraries (-l) escribir el nombre de las librerías que se van a usar. “3dti” para la librería 3DTI Toolkit, “bela” para las funciones básicas de Bela y “sndfile” es la librería AudioFile que usa Bela para leer pistas de audio .wav.

  3dti

  bela

  sndfile

- Dentro de Library search path (-L) escribir los directorios donde se encuentran las librerías introducidas en el recuadro superior. En este caso son respectivamente:

/home/sergio/Documentos/3dti_toolkit/3dti_AudioToolkit_Examples/3dti_AudioToolkit

/home/sergio/BelaWorkspace/BelaShared/root/Bela/lib

/usr/local/linaro/BelaSysroot/usr/lib/arm-linux-gnueabihf

6- Pulsar sobre Apply y seleccionar Miscellaneous dentro de Cross G++ Linker (Figura 2.14).

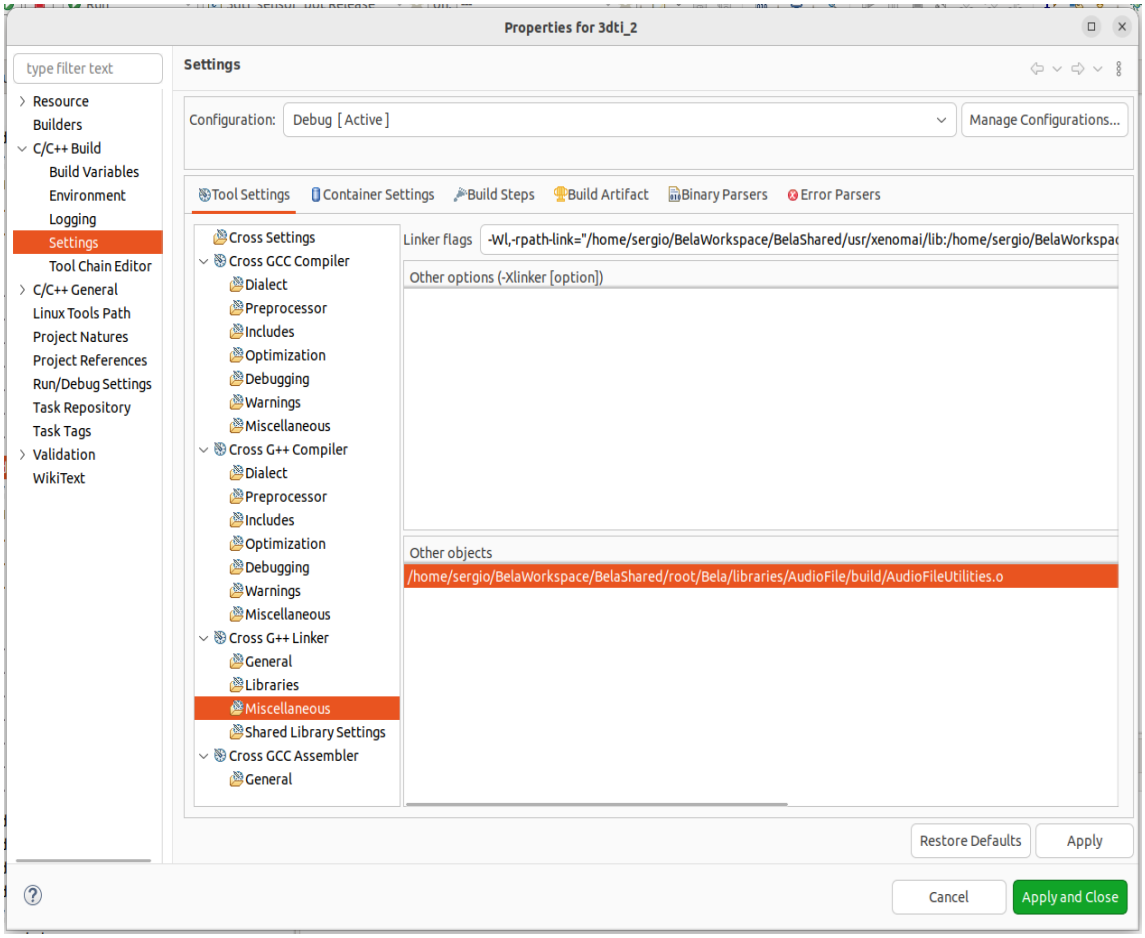

Figura 2.14. Ventana de Eclipse de las propiedades de un proyecto. *Settings → Cross G++ Linker → Miscellaneous.*

- En el cuadro de texto llamado Linker flags se introducirán aquellas librerías que se desea que se usen en tiempo de ejecución mediante el comando -Wl-rpath. La línea es la siguiente:

-Wl,-rpath-link="/home/sergio/BelaWorkspace/BelaShared/usr/xenomai/lib:/home/sergio/BelaWorkspace/BelaShared/usr/local/lib:/home/sergio/BelaWorkspace/BelaShared/usr/lib/arm-linux-gnueabihf"

- En Other objects se pondrá el path al archivo .o de la librería AudioFile de Bela para poder utilizar sus funciones. Para que aparezca este archivo, antes habrá que compilar desde el IDE de Bela un código ejemplo de Bela donde se use esta librería.

/home/sergio/BelaWorkspace/BelaShared/root/Bela/libraries/AudioFile/build/AudioFileUtilities
.o

7- Asegurarse de que las anteriores propiedades se han establecido para todas las configuraciones: Debug, Release y [All Configurations] (Figura 2.15).

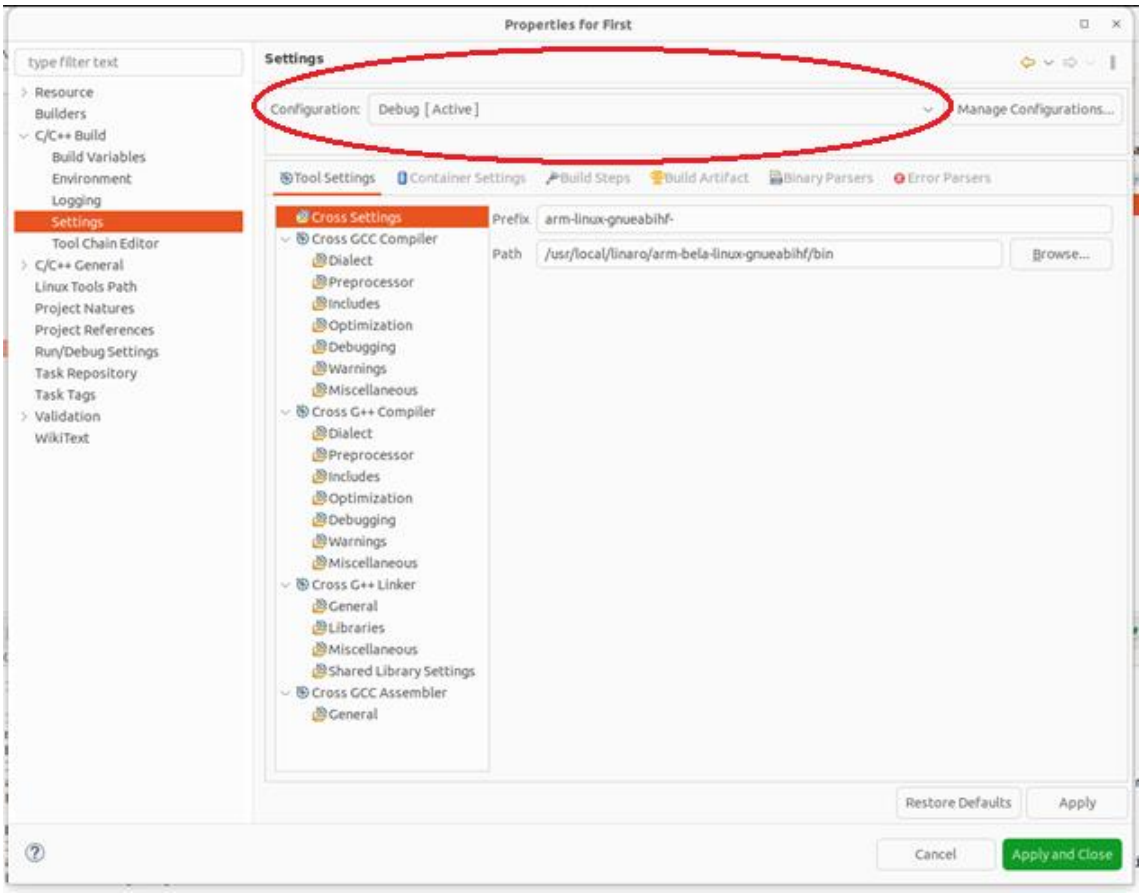

Figura 2.15. Ventana de Eclipse de las propiedades de un proyecto. *Settings → Configuration.*

8- Seleccionar en la pestaña Configurations la opción de Release (Figura 2.16). Hacer clic izquierdo en Optimization dentro de Cross G++ Compiler. Se escogerá la opción -O3. Esto escribirá un flag en el comando de compilación para Release que optimizará el código con el fin de que se ejecute la aplicación lo más rápido posible.

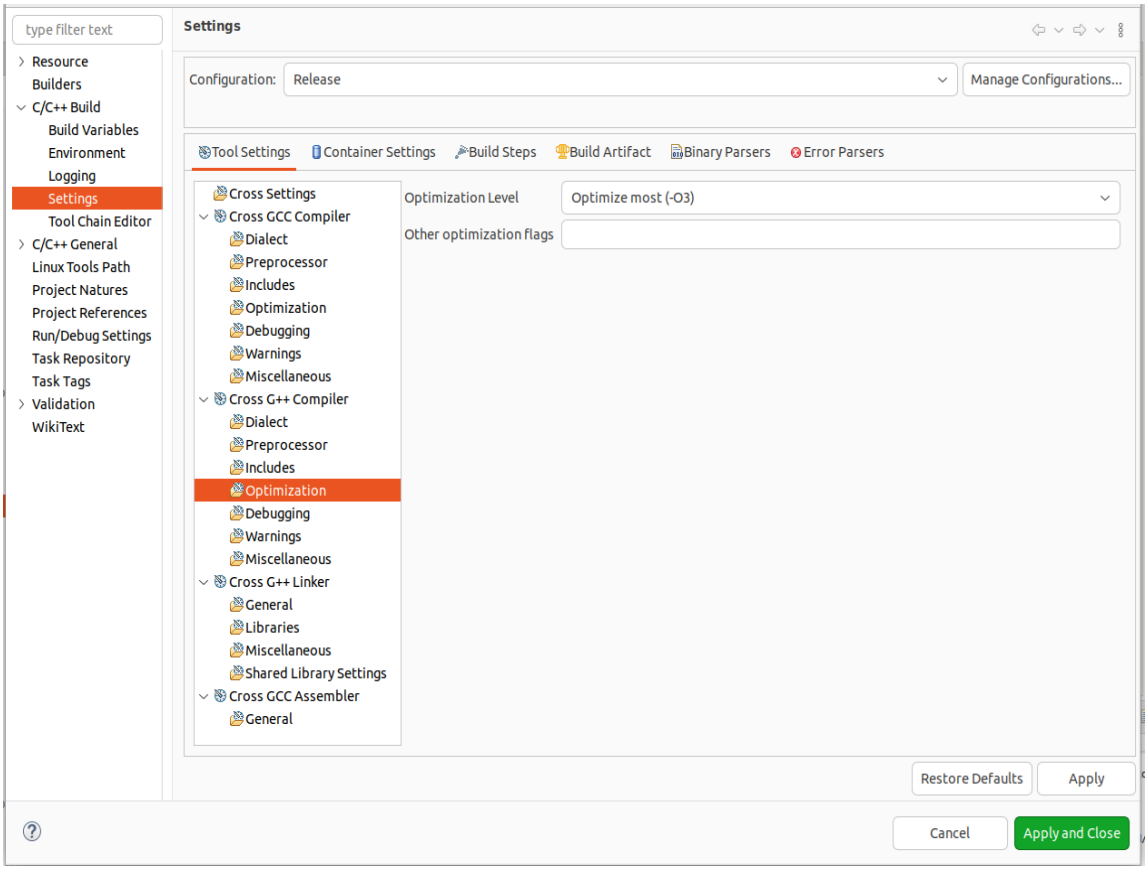

Figura 2.16. Ventana de Eclipse de las propiedades de un proyecto. *Settings → Cross G++ Compiler → Optimization.*

2.3.3. Compilación

Una vez que han sido configuradas todas las propiedades del proyecto, se debería poder compilar la aplicación básica que incorpora la librería 3DTI Toolkit en Bela proporcionada en este TFG y detallada posteriormente en el apartado de Aplicaciones.

1- Incluir el archivo .cpp y el .h en el proyecto creado.

2- Colocar los archivos HRTF .3dti-hrtf dentro de la carpeta compartida en “/root/resources” y las pistas de audio .wav en “/root/resources/Samples”.

3- Clic derecho sobre el proyecto y seleccionar la opción *Build All*.

4- Al cabo de unos segundos se debería haber compilado todo correctamente. En la ventana de problemas de Eclipse aparecerá que la

compilación ha fallado y se han producido una serie de problemas, pero lo importante es que se hayan generado los binarios en el proyecto ya que los errores que aparecen no influyen en el funcionamiento de la aplicación. Si no han aparecido dos binarios en el proyecto (Figura 2.17), entonces existe un problema de algún tipo que ha imposibilitado la compilación y habrá que revisar la configuración del proyecto y el código. Si se han generado los dos binarios significa que se ha compilado correctamente y se puede realizar la configuración de *Debug* y *Release* [9].

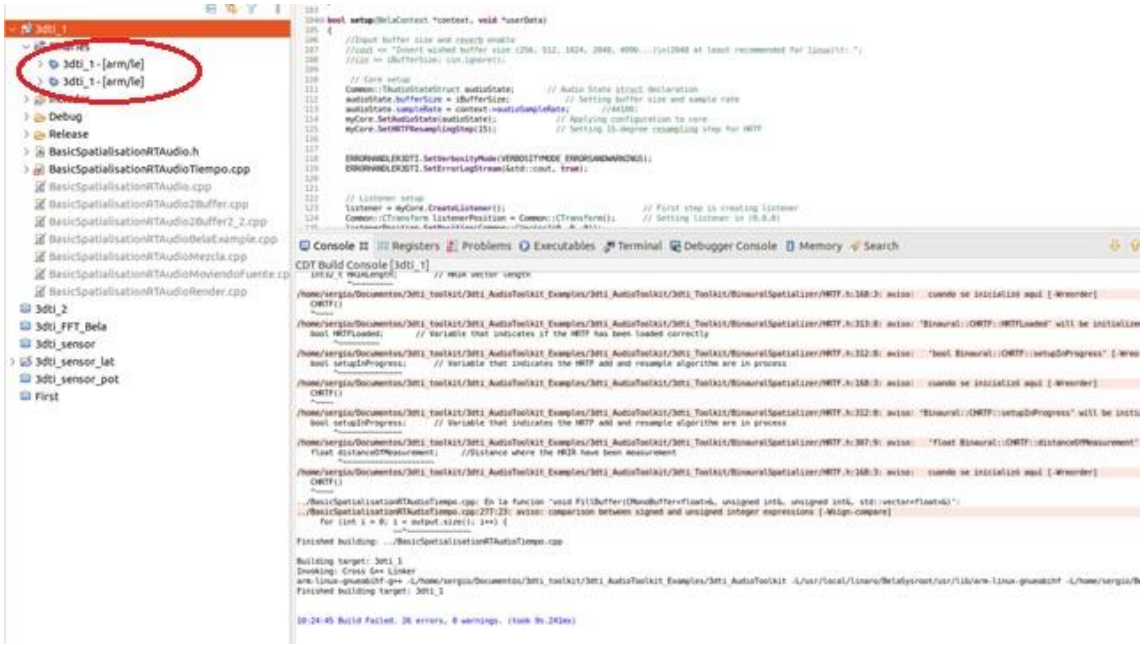

Figura 2.17. Ejemplo de resultado satisfactorio de compilación de este proyecto en Eclipse.

2.3.4. Configuración de *Debug* y *Release*

1- Una vez que se tienen los binarios se puede crear una configuración de *Debug* y de *Release*. Estas configuraciones permiten la depuración del código de forma remota ya que generarán un ejecutable específico para Bela y que se ejecutará en ella. Para *Debug*, hacer clic derecho sobre el proyecto, situar el ratón sobre *Debug* as y seleccionar *Debug Configurations*. Se abrirá una nueva ventana (Figura 2.18) donde hay que hacer doble clic izquierdo sobre la opción *C/C++ Remote Application*.

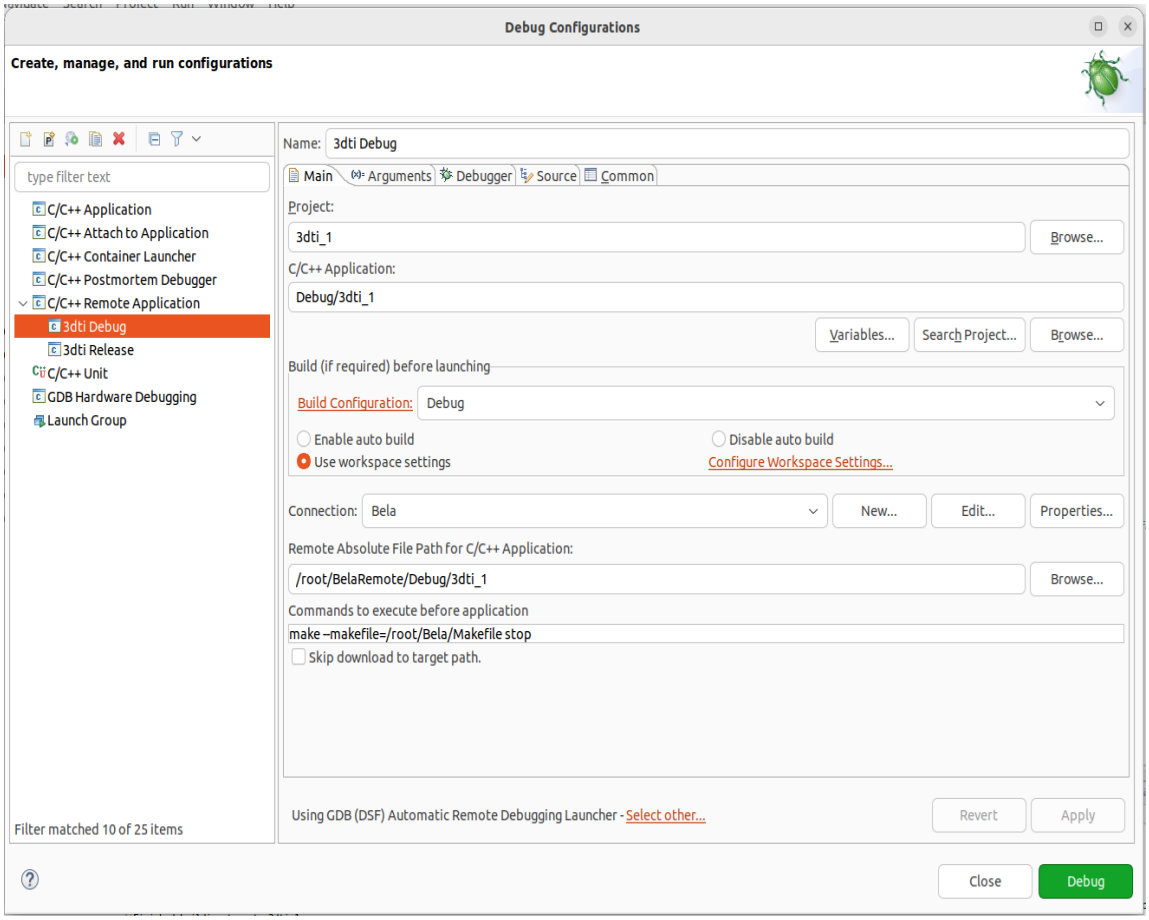

Figura 2.18. Ventana de Eclipse de la configuración de *Debug* para depurar un proyecto. Pestaña *Main*.

- Escribir el nombre que se le desea dar a la configuración en el cajetín *Name*.
- Escoger el proyecto que se va a usar si no está seleccionado pulsando sobre *Browse*… a la derecha del cuadro de texto de Project.
- En el cuadro de texto de *C/C++ Application*, seleccionar mediante la opción de *Search Project…* el binario generado en la compilación para *Debug*.
- En la pestaña desplegable de *Build Configuration*, seleccionar la opción *Debug*.
- En la opción de *Connection*, hacer clic sobre *New…* y se abrirá una nueva ventana (Figura 2.19).

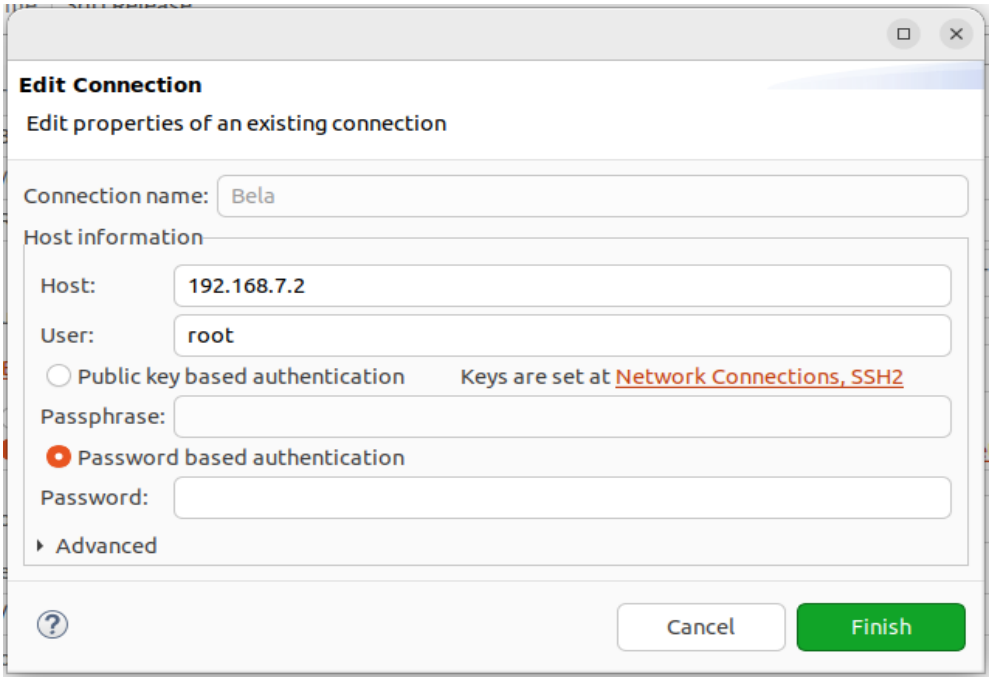

Figura 2.19. Ventana de Eclipse para definir una conexión con otro dispositivo.

- Esta conexión será la que una Eclipse con Bela, luego, en *Connection Name*, escribir su nombre, Bela.
- En el cuadro de texto Host, escribir la dirección que se usa en el navegador web para conectarse al IDE de Bela, 192.168.7.2.
- En *User*, escribir *root*, que es el nombre de usuario que usa Bela.
- Pulsar sobre el botón de *Finish*.

- En *Remote Absolute File Path for C/C++ Application*, pulsar el botón *Browse…* y buscar y seleccionar la carpeta *Debug* dentro de *BelaRemote*.
- En el cuadro de texto *Commands to execute before application*, escribir el siguiente comando:

```
make --makefile=/root/Bela/Makefile stop
```

2- Seleccionar en la parte superior de la ventana la pestaña *Debugger* (Figura 2.20).

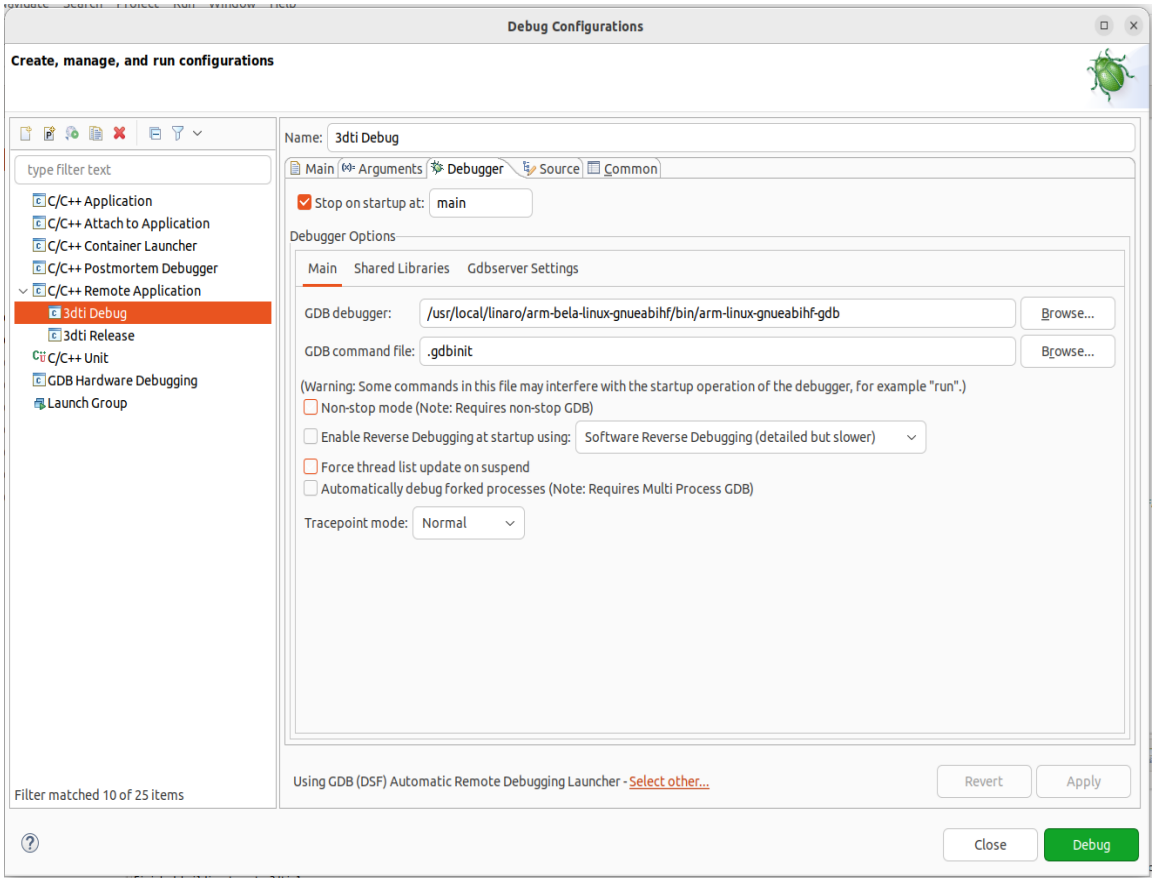

Figura 2.20. Ventana de Eclipse de la configuración de *Debug* para depurar un proyecto. Pestaña *Debugger*.

- En *GDB debugger* seleccionar mediante el botón *Browse*… el archivo *arm-linux-gnueabihf-gdb* que probablemente se sitúe en el mismo sitio que se muestra en la Figura 2.20.

/usr/local/linaro/arm-bela-linux-gnueabihf/bin/arm-linux-gnueabihf-gdb

- Escribir *.gdbinit* dentro del cuadro de texto *GDB command* file
- Activar las opciones de:
  - *Non-stop mode.*
  - *Force thread list update on suspend*.

3- Seleccionar en la parte superior de la ventana la pestaña *Common* (Figura 2.21).

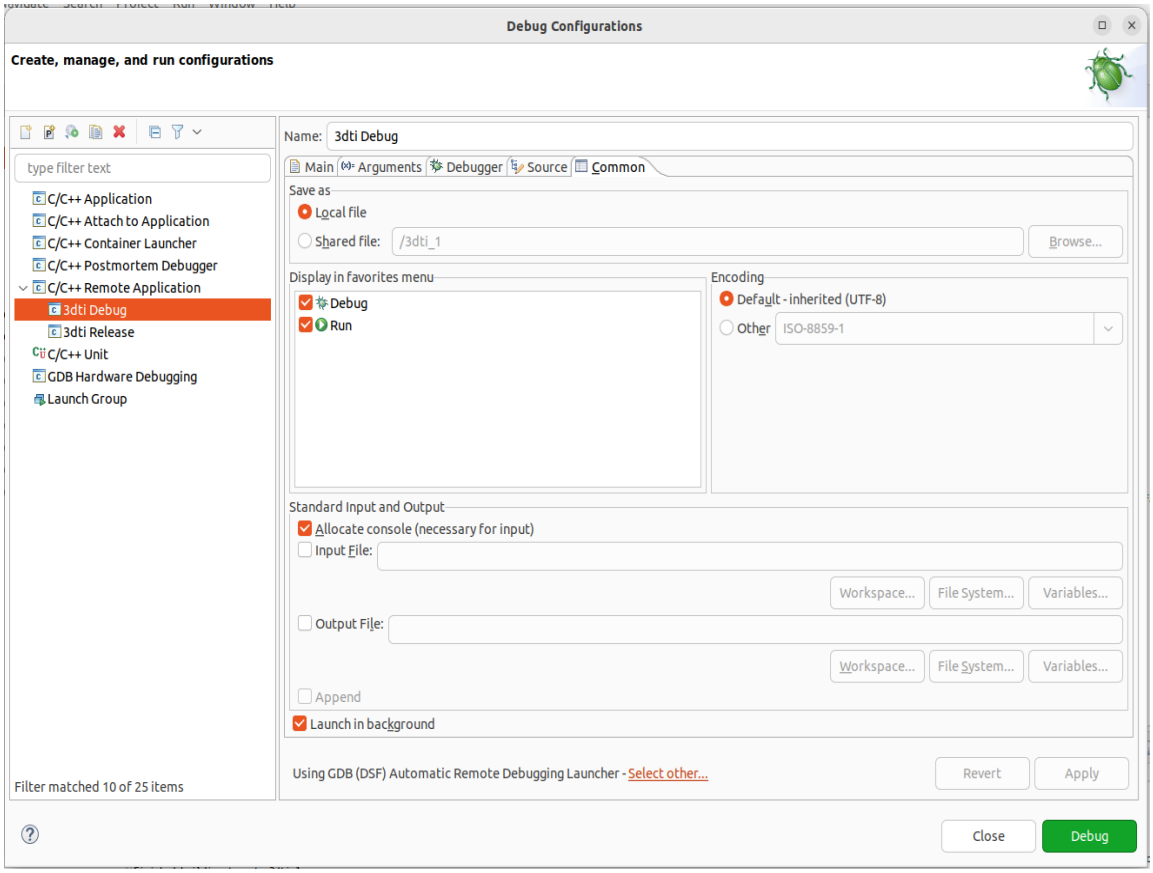

Figura 2.21. Ventana de Eclipse de la configuración de *Debug* para depurar un proyecto. Pestaña *Common*.

- Activar las opciones *Debug* y *Run* tal y como se muestra en la Figura 2.21.

4- Hacer clic sobre *Apply* y cerrar la ventana.

5- Para *Release*, hacer clic derecho sobre el proyecto, situar el ratón sobre *Run As* y seleccionar *Run Configurations* (Figura 2.22). Se abrirá una nueva ventana donde se hará clic derecho sobre la configuración *Debug* que se haya hecho previamente y se seleccionará la opción *Duplicate*.

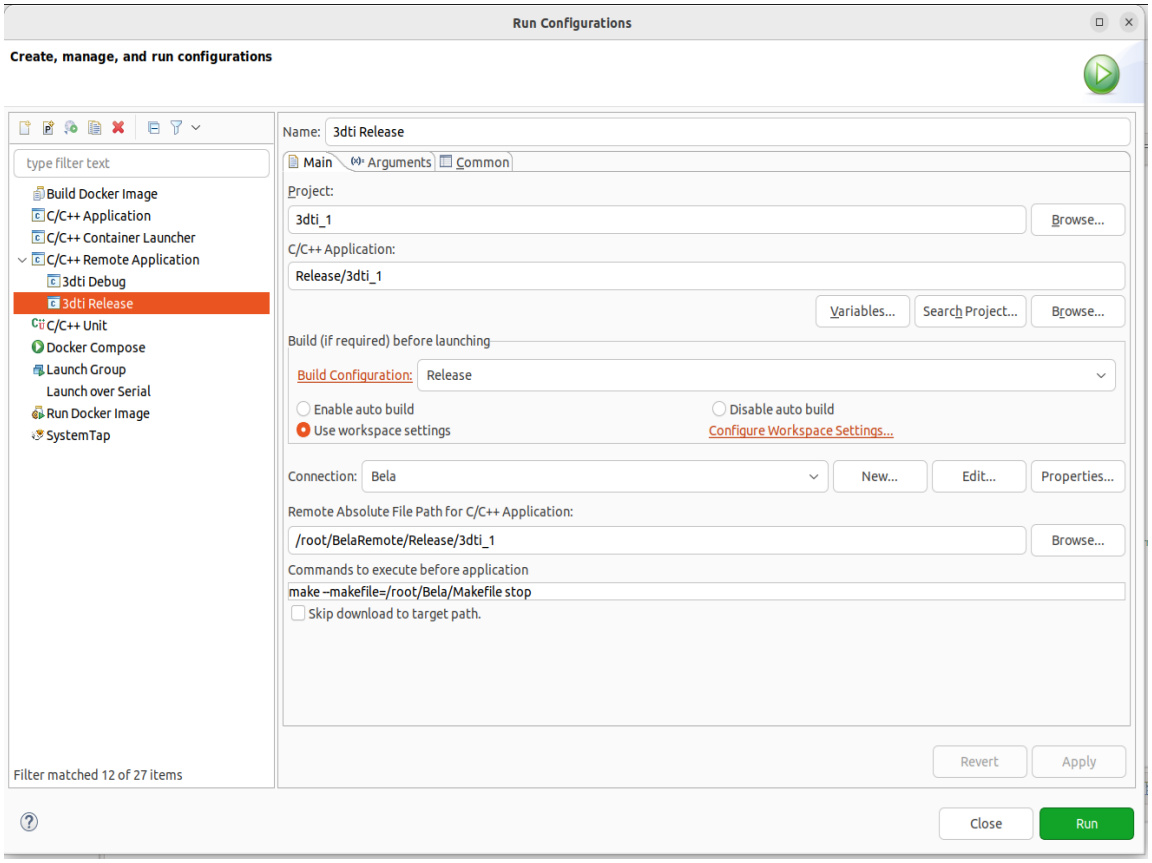

Figura 2.22. Ventana de Eclipse de la configuración de *Release* para obtener una aplicación optimizada. Pestaña *Main*.

- Cambiar el nombre en el cuadro de texto *Name* para identificar esta configuración como *Release*.
- En *C/C++ Application*, pulsar sobre *Search Project* y seleccionar el binario generado para *Release*.
- Seleccionar *Release* en la pestaña desplegable de *Build Configuration*.
- En *Remote Absolute File Path for C/C++ Application*, pulsar el botón *Browse…* y buscar y seleccionar la carpeta *Release* dentro de *BelaRemote*.

6- Hacer clic sobre *Apply* y cerrar la ventana.

7- Abrir la pestaña *Window* de Eclipse y abrir la ventana *Preferences*. Desplegar *Run/Debug* y hacer clic izquierdo sobre *Launching* (Figura 2.23). Asegurarse que está activa la opción *Terminate and Relaunch while launching* y pulsar sobre *Apply and Close*.

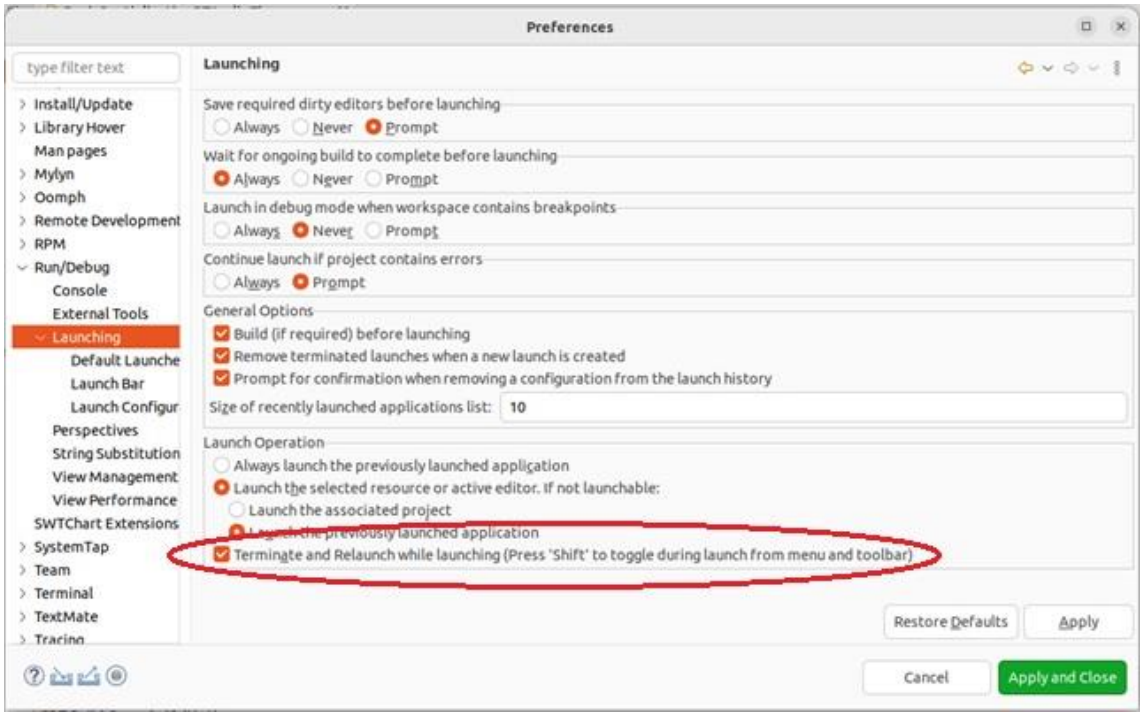

Figura 2.23. Ventana de preferencias de Eclipse. *Window → Preferences → Run/Debug → Launching.*

8- Probar mediante el icono *Debug* que todo funciona correctamente y se puede ejecutar el programa de forma cruzada en Bela.

2.4. Ejecución de aplicaciones

El IDE de Bela tiene una opción en su configuración para ejecutar el proyecto que se seleccione en el momento de encender la placa, es decir, cuando se alimente. El problema está en que tiene que ser un proyecto de los que tenga la placa y debe tener un formato concreto para que funcione. En el caso de los proyectos sencillos de Bela, todos tienen un archivo “render.cpp” que el IDE requiere para poder establecer el proyecto como *run on boot.* Además, usan librerías internas de Bela, el IDE no da opción a la incorporación de librerías externas, luego, el proyecto de este TFG no puede ser ejecutado al encenderse la placa de esta forma en concreto.

A partir de las opciones de *Debug* y *Release* se generan archivos ejecutables dentro de Bela que se pueden asignar para que sean ejecutados al encenderse la placa de una manera alternativa. Existen unos archivos en formato .service con los que se le puede decir a la placa qué ejecutable se quiere llevar a cabo

en segundo plano en cuanto se alimente la placa, por lo tanto, si solo ejecutamos un programa, todos los recursos irán destinados a esta aplicación [12].

1- Dentro de la carpeta compartida, acceder a el directorio "/lib/systemd/system" (Figura 2.24).

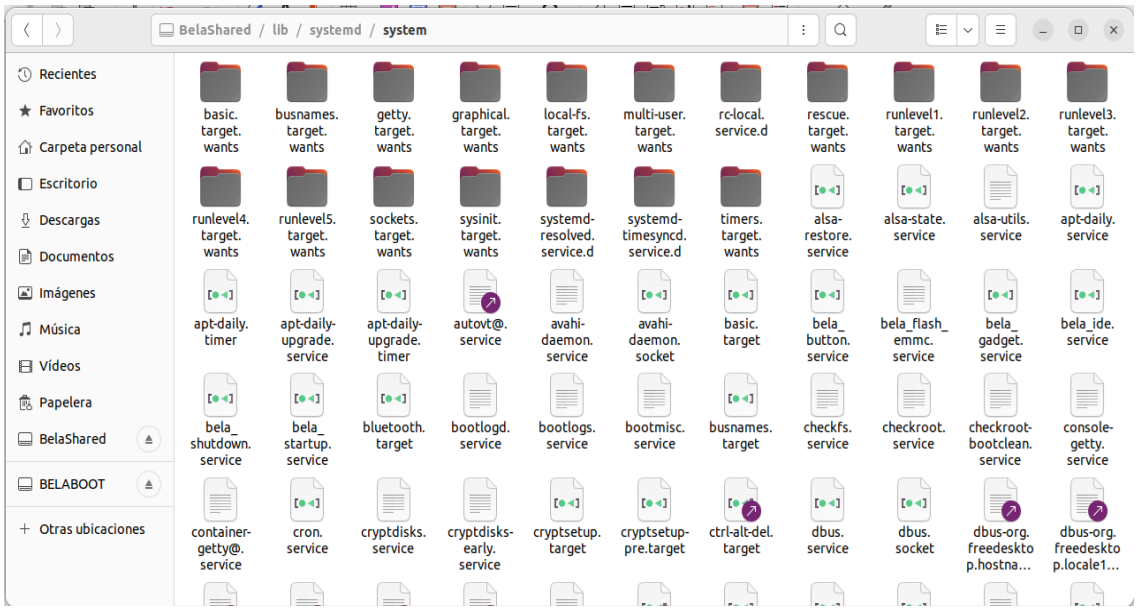

Figura 2.24. Carpeta interna de Bela donde se sitúan los servicios que se ejecutarán. Carpeta compartida → *lib* → *systemd* → *system*

2- Crear un archivo de texto que se llame como el ejecutable que queremos establecer como *run on boot* y acabado en *.service*.

Nombre_ejecutable.service

3- Abrirlo y escribir lo siguiente en su interior sustituyendo Nombre_ejecutable por el ejecutable generado por Eclipse en el modo *Release*:

```
[Unit]
Description=Nombre_ejecutable Launcher
After=network-online.target

[Service]
ExecStart=/root/BelaRemote/Release/Nombre_ejecutable
Type=simple
```

```
Restart=always
RestartSec=1
WorkingDirectory=/root/BelaRemote/Release
Environment=HOME=/root
KillMode=process

[Install]
WantedBy=default.target
```

4- Guardar y cerrar el archivo.

5- En el IDE de Bela, habilitar el ejecutable desde la ventana de comandos con el siguiente comando:

```
systemctl enable Nombre_ejecutable
```

6- Con esto, ya debería ejecutarse la aplicación en el momento de encender la placa. Para deshabilitarlo, acceder al IDE y en la ventana de comandos, escribir los siguientes comandos:

```
systemctl stop Nombre_ejecutable
systemctl disable Nombre_ejecutable
```

2.5. Manual de uso

Una vez se tenga la placa lista para ejecutar con el programa deseado y las pistas de audio en la carpeta “Samples” dentro de Bela, el sistema está listo para su uso.

1- Insertar la placa diseñada para el sensor sobre la Bela Mini (Figura 2.25).

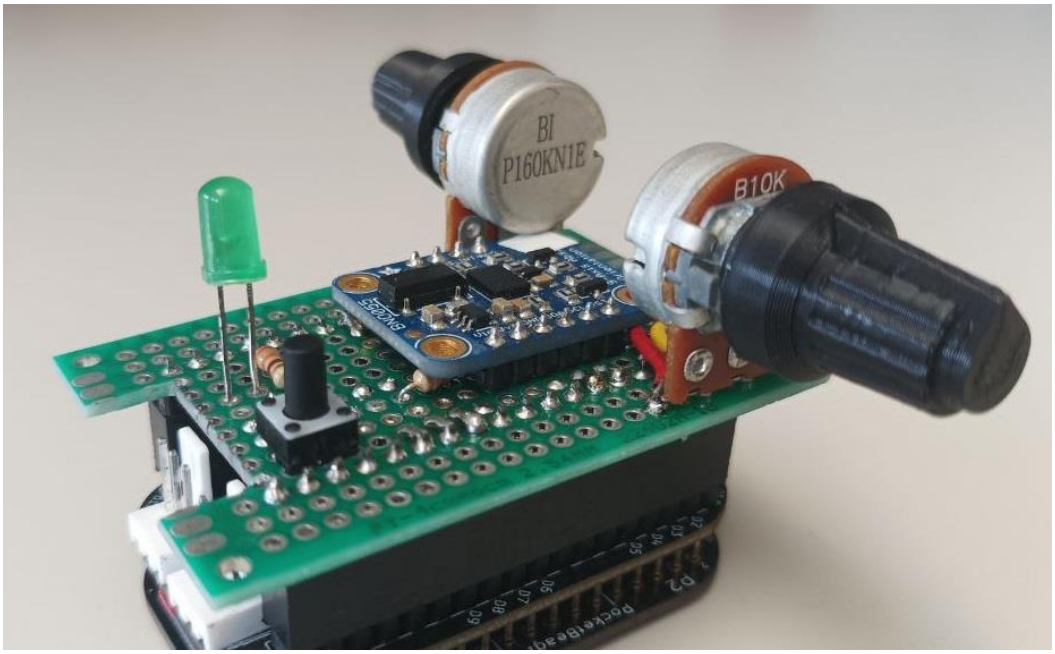

Figura 2.25. Conexión del Hardware del prototipo.

2- Introducir el sistema dentro de la caja, hecha mediante la impresora 3D, hasta que quede fija.

3- Conectar el cable de audio de la tapa en los tres pines de la placa donde está escrito OUT (Figura 2.26). Cerrar la caja con su tapa. Esta tapa debe encajar teniendo en cuenta los huecos para los potenciómetros y un adaptador que debe quedar sobre el botón de la placa.

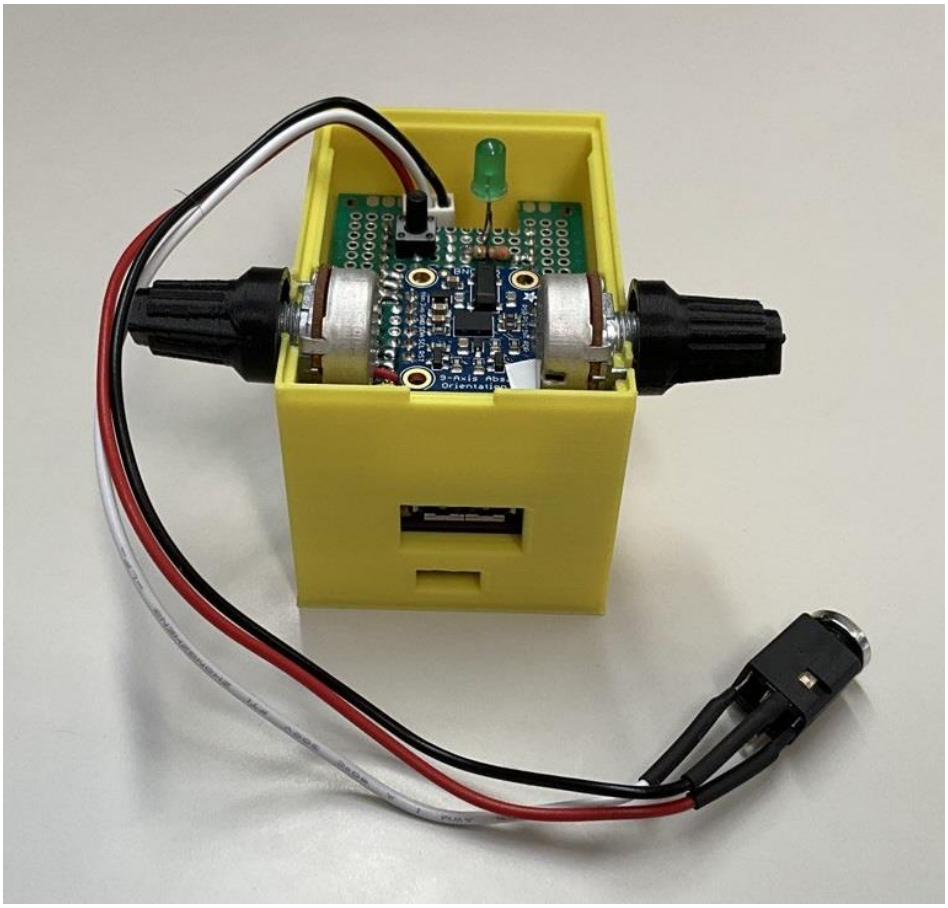

Figura 2.26. Montaje del sistema sobre la caja de impresión 3D.

4- Colocar la caja sobre los auriculares y fijarla mediante bridas a través de un orificio que tiene la caja en la parte inferior (Figura 2.27).

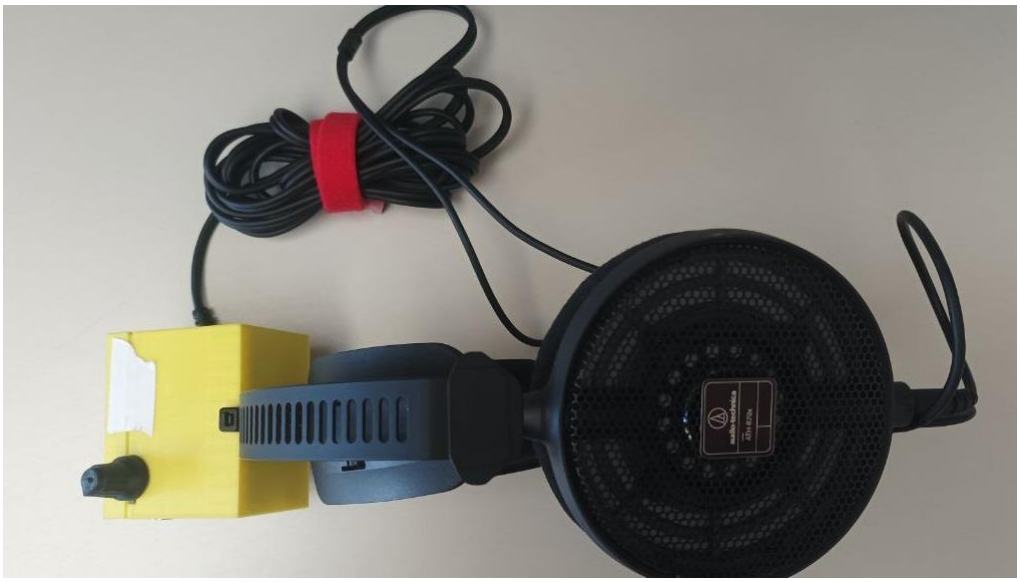

Figura 2.27. Prototipo completo sin las conexiones de audio y alimentación

5- Conectar los auriculares a la caja por la entrada mini-jack.

6- Alimentar la placa con un cable USB-A a una powerbank.

Al cabo de unos segundos, debería comenzar a reproducirse el audio relacionado con el programa cargado.

Capítulo 3. Pruebas para la optimización del sistema

3.1. Primer resultado

En un primer momento, no se sabía si se iba a poder incorporar la librería 3DTI Toolkit dentro de Bela ni qué resultados se obtendrían en caso de que se pudiera. Por lo tanto, se ha llevado a cabo un proceso de adaptación de diversos factores dentro de los archivos y librerías que influyen en la aplicación para ver como estos repercutían en el sistema con una serie de ensayos y pruebas.

Una vez desarrollada una aplicación básica que usara la librería 3DTI Toolkit dentro de Bela, se encontró que tardaba demasiado tiempo en procesar una muestra de audio y reproducirla como para hacerlo en tiempo real. En cada ciclo de la función render, que actúa como una función *Callback* de audio, la mayoría de las veces, no le daba tiempo a procesar una muestra de audio antes de que llegara la siguiente desechando la muestra que estaba siendo procesada y comenzando la siguiente. Esto ocasionaba varios problemas:

- No se reproducía el audio correctamente. Se escuchaba a una velocidad inferior, con mucho ruido y cortes de señal.
- En algunos casos, directamente se bloqueaba el procesador y era necesario apagar la placa.

Para ver cómo de lejos estaba la aplicación de poder reproducir el audio correctamente, se midió el tiempo que empleaba la aplicación en procesar el audio con la librería 3DTI Toolkit tal y como se descargó y se comparó con lo

que dura la función render dependiendo del tamaño del buffer y de la muestra que proporciona el archivo HRTF.

El tamaño del buffer depende del número de datos que se quieran almacenar en él de toda la pista de audio cargada y muestreada. La información almacenada en el buffer se procesará y su resultado se reproducirá mediante Bela. El tiempo que tarde en procesar la muestra depende del tamaño del buffer, mientras menor sea su tamaño, menos tiempo tardará en procesar y, por lo tanto, habrá menos latencia mejorando el proceso en tiempo real. Sin embargo, como el tiempo de procesado de una muestra no es constante, si el buffer es pequeño, tiene menos tolerancia a la variación, luego, hay mayor riesgo de que no haya terminado de procesar la muestra antes de que llegue la siguiente comprometiendo la calidad de la reproducción.

Dentro del procesado de la señal de audio se le aplica un filtro HRTF. Este filtro, como se ha desarrollado anteriormente, simula las modificaciones que realiza el cuerpo de una persona sobre una señal de audio incidente para situar dicha fuente en el espacio. La aplicación de este filtro se realiza mediante la convolución en el dominio de la frecuencia entre los valores del filtro y el buffer donde se recoge la muestra.

La librería 3DTI Toolkit nos obliga a que el tamaño de buffer sea una potencia de dos comprendida entre 128 y 2048. Además, nos recomienda que su valor esté entre 128 y 512, ya que, el tamaño de los filtros HRTF tienen estos valores. Se espera mejor rendimiento cuando el tamaño del buffer sea el mismo que el tamaño de HRTF.

El tiempo que tarda en procesar el audio se obtiene mediante el uso de la función gettimeofday de la librería time.h. Se realiza la diferencia del tiempo tomado antes y después de la función que procesa el audio.

El tiempo que tarde en procesar el audio también depende del tamaño de muestra de HRTF porque la librería 3DTI Toolkit adapta la muestra al tamaño de buffer que se haya especificado. Si el tamaño de muestra es menor que el del buffer, el buffer rellenará con ceros los huecos sobrantes (apenas supone una carga para el microprocesador). Si, en el caso contrario, el tamaño de la muestra

es mayor que el del buffer, se guardarán las muestras para los siguientes buffers que tendrán en cuenta las muestras anteriores (supone un procesamiento extra).

La función render se llevará a cabo cada vez que se rellene un buffer nuevo de entrada, por lo tanto, el tiempo que tarde dependerá del tamaño del buffer y de la frecuencia que tenga el microprocesador para leer audio (Tabla 3.1), en el caso de Bela siempre es 44100 Hz.

$$Tiemporender = \frac{TamañoBuffer}{Frecuencia}$$

		MAX render time
Buffer Size	**128**	2.90ms
	256	5.80ms
	512	11.61ms
	1024	23.22ms
	2048	46.44ms

Tabla 3.1. Tabla con el tiempo máximo que tiene cada muestra para ser procesada dependiendo del tamaño del buffer

Y el porcentaje del tiempo de render que ocupa el procesamiento de audio de la librería 3DTI Toolkit de una fuente se obtiene mediante:

$$\%Procesamiento = \frac{Tiempoquetardaenprocesar}{Tiemporender} * 100$$

Con todo esto, se obtuvieron los siguientes resultados de media:

Librería 3DTI Toolkit				
Con double		HRTF Frame		
		128	256	512
Buffer Size	128	2.97ms (102.33%)	3.90ms (134.37%)	5.80ms (199.83%)
	256	5.90ms (101.64%)	5.91ms (101.80%)	7.97ms (137.30%)
	512	12.19ms (104.99%)	12.20ms (105.08%)	12.19ms (104.99%)
	1024	24.66ms (106.20%)	24.68ms (106.29%)	24.71ms (106.42%)
	2048	51.51ms (110.92%)	51.60ms (111.11%)	51.54ms (110.98%)

Figura 3.1. Tabla con las medidas de tiempo que tarda de media en procesar el audio Bela con la librería 3DTI Toolkit sin modificaciones y el porcentaje de ocupación del tiempo disponible.

Con la Figura 3.1 se puede interpretar que, por norma general, en ningún caso el procesamiento de audio de una muestra acaba antes de que llegue la siguiente muestra (Tabla 3.1), luego, no funciona en ninguno de los casos. Atendiendo a esto, se llega a la conclusión de que hay que optimizar el código de alguna forma.

3.2. Cambio de double a float en FFT

Se tomó la decisión de cambiar los archivos .cpp y .h de la librería 3DTI Toolkit que se encargaban de realizar las transformadas de Fourier de las muestras ya que esto se hacía con una precisión de variable double. Este fichero, con el fin de hacer la transformada, realiza muchos cálculos con variables tipo double aumentando la cantidad de procesamiento requerido cuando para este caso no es necesaria esa precisión. Luego, tras cambiar todas las variables tipo double por float en los archivos fftsg.h y fftsg.cpp y solucionar los conflictos generados sustituyendo las mismas variables en Fprocessor.cpp y Fprocessor.h, se han obtenido la siguiente mejora:

Librería 3DTI Toolkit		Debug		
Con float		HRTF Frame		
		128	256	512
Buffer Size	128	2.58ms (88.9%)	3.49ms (120.3%)	5.43ms (187.1%)
	256	4.82ms (83.0%)	4.83ms (83.2%)	7.09ms (122.1%)
	512	10.08ms (86.8%)	10.10ms (87.0%)	10.09ms (86.9%)
	1024	20.60ms (88.7%)	20.62ms (88.8%)	20.65ms (88.9%)
	2048	41.29ms (88.9%)	41.33ms (89.0%)	41.32ms (89.0%)

Figura 3.2. Tabla con las medidas de tiempo que tarda de media en procesar el audio Bela con la librería 3DTI Toolkit cambiando las variables tipo *double* por *float* y el porcentaje de ocupación del tiempo disponible en *Debug*.

Tal y como representa la Figura 3.2, en la mayoría de los casos se reproduce el audio correctamente, aunque la calidad del audio es la mejor a partir del tamaño de buffer de 512. Sin embargo, este porcentaje de procesamiento es exclusivo para una sola fuente, luego, en ningún caso sería posible incluir otra fuente. Para tener seguridad sobre esto, se ha maximizado la optimización en la configuración de Release y se han dedicado todos los recursos a la aplicación para ver el resultado mediante un atributo de Bela llamado HighPerformanceMode obteniendo los siguientes resultados (Figura 3.3).

Librería 3DTI Toolkit		Release		
Con float		**HRTF Frame**		
		128	**256**	**512**
Buffer Size	**128**	2.15ms (74.1%)	3.06ms (105.4%)	5.05ms (174.0%)
	256	4.60ms (79.2%)	4.58ms (78.9%)	6.84ms (117.8%)
	512	9.53ms (82.1%)	9.56ms (82.3%)	9.52ms (82.0%)
	1024	19.40ms (83.6%)	19.40ms (83.6%)	19.42ms (83.6%)
	2048	39.82ms (85.7%)	39.84ms (85.7%)	39.85ms (85.7%)

Figura 3.3. Tabla con las medidas de tiempo que tarda de media en procesar el audio Bela con la librería 3DTI Toolkit cambiando las variables tipo *double* por *float* y el porcentaje de ocupación del tiempo disponible en *Release*.

Tal y como se preveía, el cambio ha resultado no ser suficiente, luego, como con estas modificaciones y maximizando la optimización no se ha conseguido tener la posibilidad de situar más de una fuente en el espacio, se barajó otra posible modificación.

3.3. Uso de librería de Bela para FFT

Al cambiar las variables de double a float en el fichero que se encargaba de realizar las transformadas de Fourier se esperaba un resultado mucho mejor del obtenido porque se pensaba que Bela usaba un DSP que no era capaz de trabajar con double, luego, dependía del microprocesador para hacer estos cálculos ralentizando el proceso. Sin embargo, Bela no utiliza un DSP, sino que se basa en la librería Ne10 (o NEON) que permite procesar audio con su ARM lo suficientemente rápido como para asemejarse a la velocidad de un DSP [13].

A partir de esta información y sabiendo que la librería que usa Bela para calcular las transformadas de Fourier utiliza la librería NEON [7], se supuso que, si se adaptase la librería 3DTI Toolkit para realizar los cálculos de la transformada de Fourier con las funciones de esa librería, al estar específicamente adaptada para Bela, se agilizaría el proceso.

A la hora de realizar los cambios, hay que tener en cuenta varios factores que diferencian las funciones de Bela y las de 3DTI Toolkit a la hora de calcular las transformadas de Fourier.

3DTI Toolkit:

- Obtiene los resultados de la transformada de Fourier al completo.
- A la salida de la función que calcula la transformada, proporciona un solo array con los valores reales e imaginarios intercalados y, por lo tanto, el buffer es del doble de tamaño que la muestra usada.

FFT de Bela:

- La transformada de Fourier de una señal es simétrica en módulo y antisimétrica en fase, luego, la librería de Bela solo calcula y proporciona a la salida la mitad de los valores ya que considera el resto de los valores redundantes.
- No tiene una función unificada para obtener los valores en su conjunto, tiene una función que devuelve el módulo y otra que devuelve la fase.

Atendiendo a estas diferencias, se modificaron las funciones dentro del fichero FProcessor.cpp para que al cambiar una librería por otra se adaptara el valor que proporciona Bela a su correspondiente salida, independientemente de si se trata de la transformada directa o de la inversa. (Anexo A)

Se creó un proyecto nuevo para comprobar que el código modificado proporcionaba la misma salida con el uso de la nueva librería que con la antigua. Para ello, en una aplicación básica se generó una señal de un diente de sierra, se calculó la transformada de Fourier directa y se guardó en un fichero la salida en el dominio de la frecuencia. Se le aplicó un filtro paso bajo, se calculó la transformada de Fourier inversa y se guardó en un fichero la salida en el dominio

del tiempo. Mediante Matlab, se visualizaron las funciones obtenidas para ambas librerías (Tabla 3.2 y Tabla 3.3).

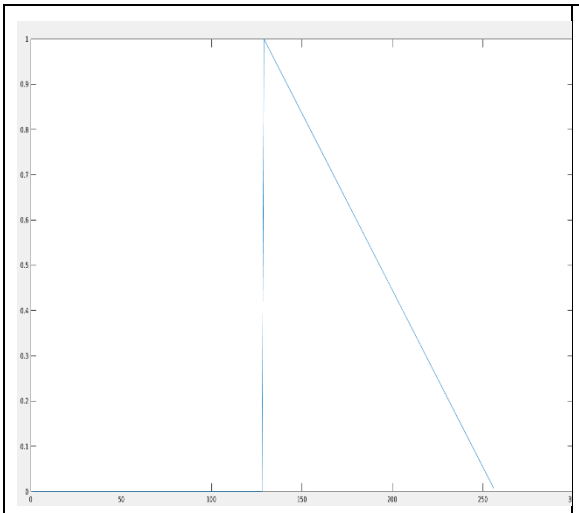	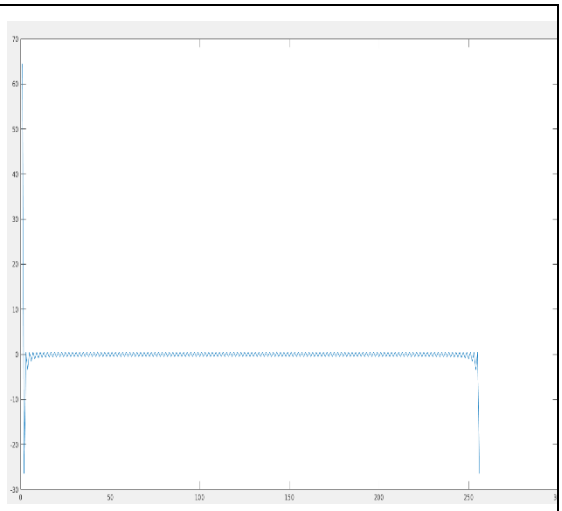
Figura 3.4. Señal de entrada en el tiempo de 256 valores y normalizada. (Diente de sierra)	**Figura 3.5. Transformada de Fourier de la señal de entrada. Parte real en el dominio de la frecuencia.**
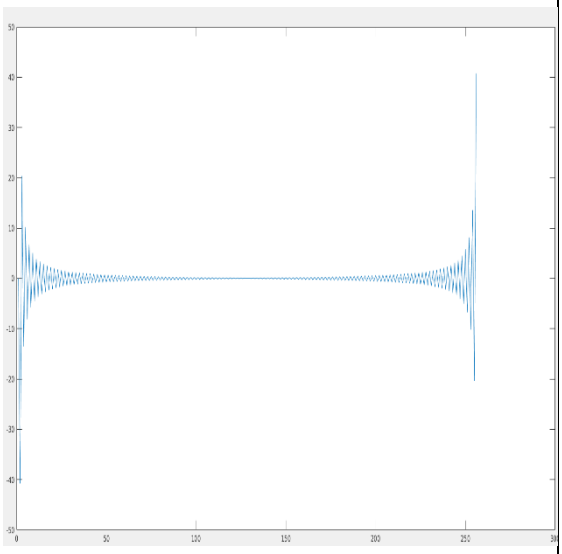	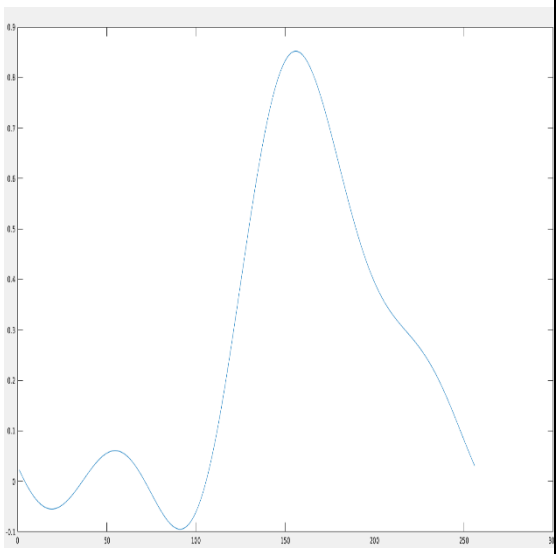
Figura 3.6. Transformada de Fourier de la señal de entrada. Parte imaginaria en el dominio de la frecuencia.	**Figura 3.7. Señal de salida en el tiempo tras aplicar un filtro paso bajo en el dominio de la frecuencia.**

Tabla 3.2. Procesado de audio por pasos con un filtro paso bajo usando funciones de la librería 3DTI Toolkit. Eje X: número de muestras. Eje Y: Amplitud

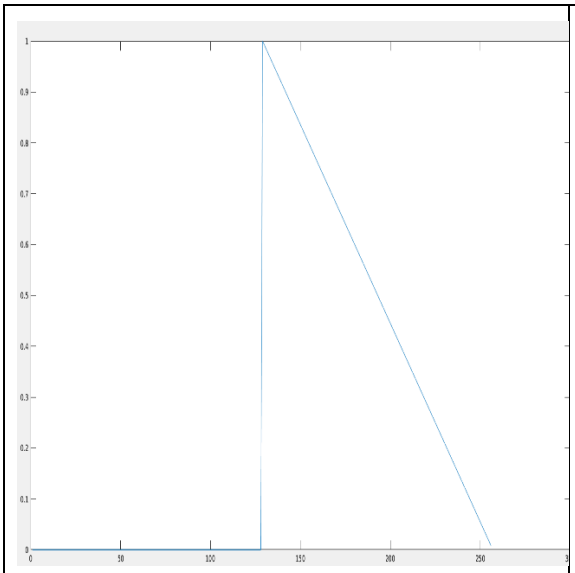

Figura 3.8. Señal de entrada en el tiempo de 256 valores y normalizada. (Diente de sierra)

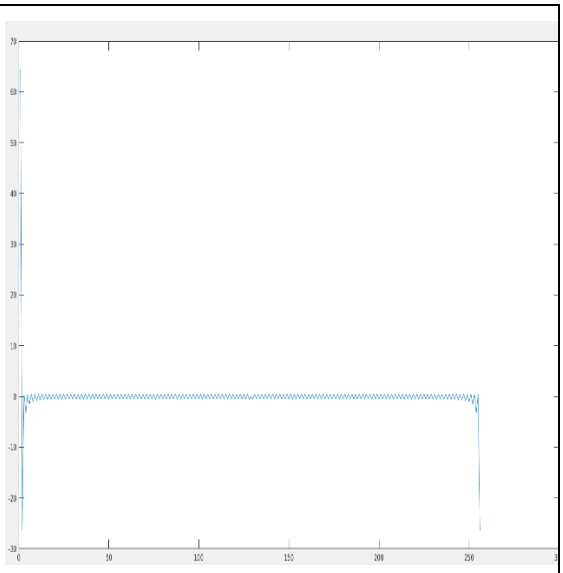

Figura 3.9. Transformada de Fourier de la señal de entrada. Parte real en el dominio de la frecuencia.

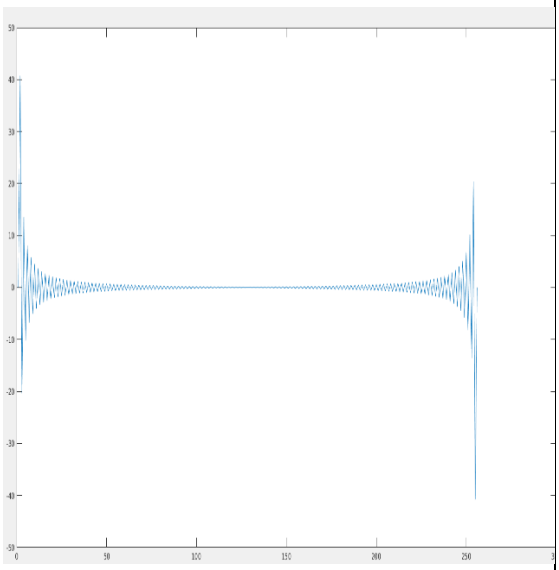

Figura 3.10. Transformada de Fourier de la señal de entrada. Parte imaginaria en el dominio de la frecuencia.

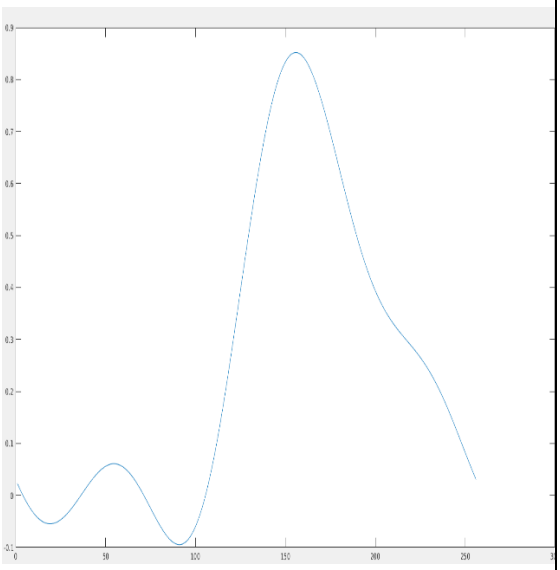

Figura 3.11. Señal de salida en el tiempo tras aplicar un filtro paso bajo en el dominio de la frecuencia.

Tabla 3.3. Procesado de audio por pasos con un filtro paso bajo usando funciones de la librería FFT de Bela. Eje X: número de muestras. Eje Y: Amplitud

Una vez comprobado que el resultado es el mismo (Figura 3.7 y Figura 3.11), se comprobó en el proyecto inicial cómo repercutía la modificación de la

librería 3DTI Toolkit en el tiempo que tardaba en procesar el audio en la función render y se obtuvieron los siguientes resultados mostrados en la Figura 3.12.

Librería Bela				
FFT		**HRTF Frame**		
		128	**256**	**512**
Buffer Size	**128**	2.64ms (90.96%)	3.54ms (121.96%)	5.35ms (184.32%)
	256	4.94ms (85.10%)	4.97ms (85.62%)	7.15ms (123.17%)
	512	10.08ms (86.82%)	10.11ms (87.08%)	10.09ms (86.91%)
	1024	20.44ms (88.03%)	20.42ms (87.94%)	20.42ms (87.94%)
	2048	39.78ms (85.66%)	39.79ms (85.68%)	39.72ms (85.53%)

Figura 3.12. Tabla con las medidas de tiempo que tarda de media en procesar el audio Bela con la librería FFT de Bela y el porcentaje de ocupación del tiempo disponible.

Los resultados de la tabla muestran que no existe mejoría producida por el cambio de la librería, de hecho, en la mayoría de los casos, el rendimiento es incluso peor. Ante estos datos, se decidió volver a la versión anterior de la librería donde no se usaba la librería FFT de Bela y realizar la aplicación final del proyecto sobre una sola fuente.

Capítulo 4. Aplicaciones y pruebas

4.1. Código básico

Este código parte de un ejemplo sencillo del uso de la librería 3DTI Toolkit adaptándolo a Bela cuyos programas tienen una estructura específica [10]. Esta, como se puede observar en su IDE, consta de tres funciones: setup, render y cleanup.

- setup(): Solo se ejecuta una vez al inicio el código.
- render(): Esta función es un bucle infinito a una determinada frecuencia, en este caso, a la que usa Bela para trabajar con audio, 44100 Hz que comienza acto seguido de la función setup().
- cleanup(): Esta función solo se ejecuta al detener el programa, es decir, al final del proceso.

Estas tres funciones son declaradas por una función main() que el IDE de Bela no muestra a los usuarios y que es necesaria para poder programar en un entorno de desarrollo convencional de C++ como es, en este caso, Eclipse. Una vez situada esa función main(), se puede adaptar el código ejemplo de la librería 3DTI Toolkit a esta estructura.

En un primer lugar, hay que eliminar todo lo relacionado con RTAudio ya que se usará Bela como tarjeta de audio y, por lo tanto, usaremos sus funciones para reproducir las señales. Tras esto, la adaptación consiste en situar las declaraciones de objetos situando al oyente y a la fuente en el espacio en la función setup() y realizar el procesado de audio en la función render(). En este caso solo se usa una pista de audio porque interesa ver el funcionamiento básico

del programa y caracterizarlo con el rendimiento y sus retardos característicos, por lo tanto, también se han deshabilitado los efectos adicionales de audio como la reverberación.

Con las librerías de Bela se leerán las pistas de audio en la función setup() y en la función render(), tras procesar una muestra del audio, se reproducirá con bucles que recorren el buffer, lo descomponen en canal izquierdo y derecho y sacan cada valor a su canal correspondiente. Al final de cada vuelta de render(), se mueve un poco la fuente para comprobar que todo funciona correctamente. En la Figura 4.1 se muestra gráficamente cuál es el funcionamiento de esta aplicación.

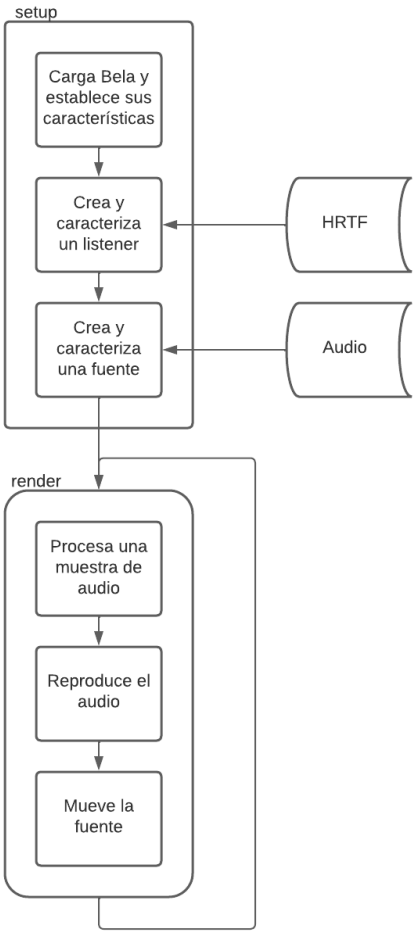

Figura 4.1. Diagrama de flujo de la aplicación básica que usa 3DTI Toolkit en Bela.

4.2. Incorporación del sensor

Se tiene un usuario con este sistema sobre su cabeza y unos auriculares conectados. Al ejecutar esta aplicación, el usuario escucha una fuente de audio proveniente de una posición en el espacio 3D y, si mueve la cabeza, esa fuente de audio se adapta a ese movimiento para que el usuario la sitúe en la misma posición.

Tal y como se ha desarrollado en la introducción de este TFG, para detectar la posición de la cabeza se necesita un sensor inercial. En este caso se barajaron dos opciones, el MPU6050 y el BNO055.

El MPU6050 [20] es un sensor inercial básico que consta de acelerómetro y giroscopio para medir la aceleración y la orientación del sensor. Las funciones de su librería están desarrolladas a un alto nivel, luego, su uso es muy sencillo. Sin embargo, utiliza internamente la librería Wire de Arduino para comunicarse por I2C que supone un problema para usarla en Bela.

El BNO055 [17] de Adafruit es un sensor inercial con mejores características que el MPU6050 y consta de acelerómetro, giroscopio y magnetómetro. Para este sensor ya existe algún proyecto en internet desarrollado sobre Bela, por lo tanto, se decidió partir de él y modificar el controlador que utiliza para el uso del sensor y adaptarlo a este proyecto [14]. La creación del controlador del sensor ha sido desarrollada en el TFG complementario [22] a este.

Figura 4.2. Sensor BNO055.

Para la conexión del sensor BNO055 se ha diseñado un circuito sobre una placa pretaladrada con conexiones de pin de inserción macho. Esta placa se

apila sobre Bela Mini y adapta el sensor a sus pines para la comunicación vía I2C. Además, se ha incorporado un botón para poder establecer la posición actual de la cabeza como posición de referencia. El diseño y el montaje de la placa se detalla en el TFG asociado [22] a este.

Para darle una orientación al objeto oyente creado, hay que pasarle un valor de tipo Quaternion. Este valor se trata de un número complejo con una parte real (w) y tres imaginarias (i, j, k) creado específicamente para tratar rotaciones. Al controlador del sensor BNO055 se le puede pedir que devuelva la rotación mediante un valor de este tipo, luego, solo hay que actualizar la orientación del oyente cada vez que se reciba información del sensor.

Este programa parte de la aplicación de código base eliminando la parte del final de la función render() donde mueve la fuente. Para usar el sensor hay que tener en cuenta que no se desea aumentar más el tiempo de procesamiento dentro de la función render() ya que esto puede comprometer el funcionamiento del sistema, por lo tanto, se ha generado una hebra para la toma de datos del sensor que se ejecuta en paralelo a la función render(). Esta hebra debe estar sincronizada a la lectura de muestras de la fuente de audio porque cuando el oyente mueva la cabeza la posición relativa de la fuente debe adaptarse en tiempo real, es por eso que cada vez que comience la función render(), esta llamará a la hebra del sensor para que se ejecuten simultáneamente.

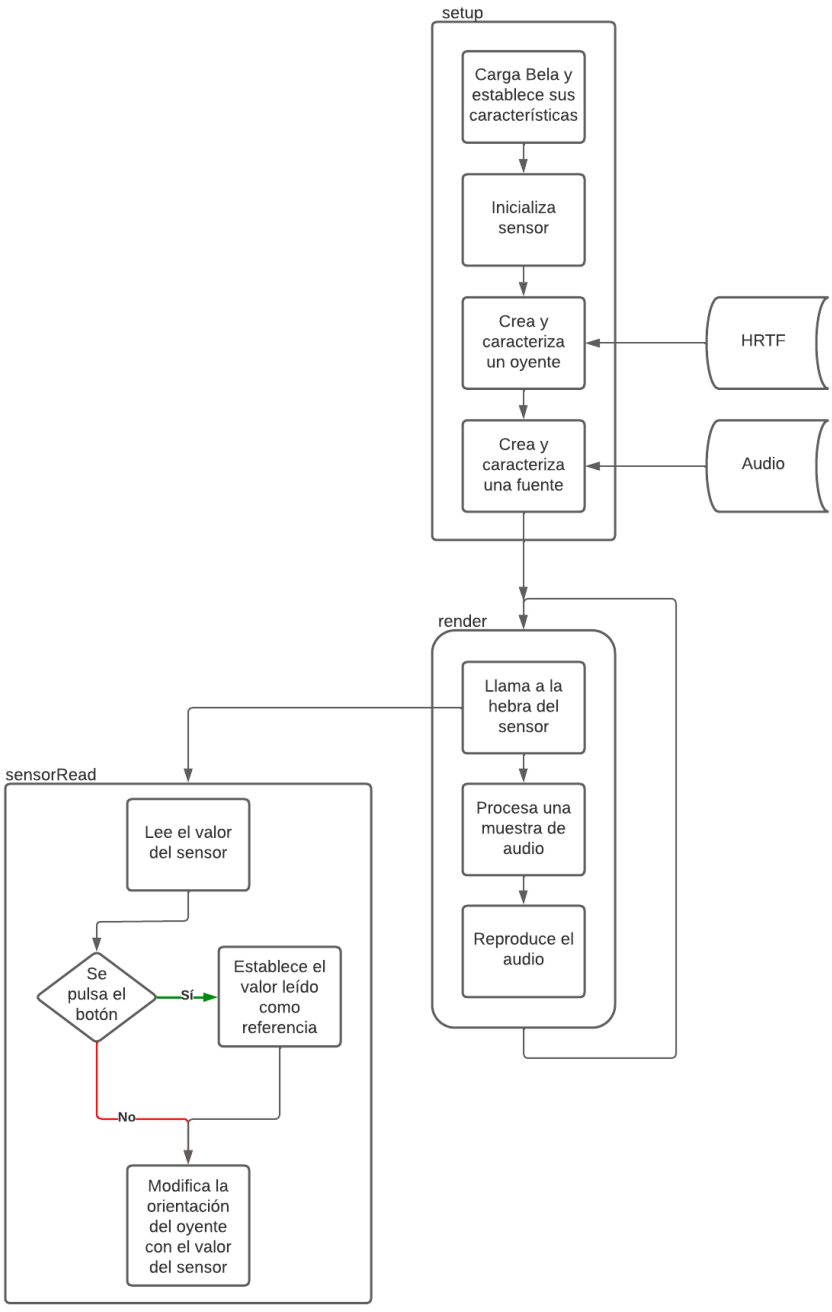

Figura 4.3. Diagrama de flujo de la aplicación con el uso del sensor BNO055.

El funcionamiento del código representado en la Figura 4.3 es el siguiente:

Se declaran dos variables quaternion, una para la orientación inicial y de referencia de la cabeza, que se usará para caracterizar al objeto *listener* (este objeto representa a la persona que usa el prototipo con esta aplicación), y la otra para la variación que devuelve el sensor. También se declara un pin de Bela

como entrada digital para conectar un botón. Cada vez que comienza la función render(), se llama a la hebra del sensor. En esta se devuelve un valor quaternion que se almacena en el del sensor declarado inicialmente. La rotación producida se obtiene, por definición de los quaternion, mediante la multiplicación del valor obtenido y la inversa de la posición de referencia. De esta forma, si el usuario quiere que su orientación actual sea la de referencia solo tiene que pulsar el botón y establecerá el valor de la lectura del sensor como tal.

```
37 Common::CQuaternion initPosHead = Common::CQuaternion(1,0,0,0);
38 Common::CQuaternion sensorData = Common::CQuaternion(1,0,0,0);
39
40 std::string FileSpeech = "/root/resources/Samples/speech.wav";
41 std::string FileSteps = "/root/resources/Samples/steps.wav";
42 std::string hrtf = "/root/resources/3DTI_HRTF_IRC1008_512s_44100Hz.3dti-hrtf";
43
44 void sensorRead(void*)
45 {
46     imu::Quaternion quat = bno.getQuat();
47     sensorData.w = quat.w();
48     sensorData.x = quat.x();
49     sensorData.y = quat.y();
50     sensorData.z = quat.z();
51
52     if(statusButton == 0){
53         initPosHead = sensorData;
54     }
55     sensorData = initPosHead.Inverse() * sensorData;
56     listenerPosition.SetOrientation(sensorData);
57     listener->SetListenerTransform(listenerPosition);
58 }
```

Figura 4.4. Función que ejecuta la hebra del sensor.

Para poder compilar y ejecutar esta aplicación en Eclipse, hay que importar todos los ficheros .cpp y .h del driver del sensor BNO055 [22] al proyecto y hacer un include del fichero Bela_BNO055.h en el fichero principal del proyecto.

4.3. Potenciómetros

El objetivo de esta aplicación es añadir funcionalidades adicionales al sistema que incorpora el sensor para enriquecer la experiencia del usuario. Para ello, se ha desarrollado un control de volumen y un control para cambiar la pista de audio entre varias disponibles. Cada uno de estos controles supone un potenciómetro conectado adecuadamente a una entrada analógica de la Bela Mini. Frente a estos cambios, se ha diseñado una nueva placa insertable sobre Bela Mini y soldada en una placa pretaladrada que lleva el sensor BNO055, un botón, dos

potenciómetros, un LED y la electrónica adicional para adaptar las conexiones (resistencias).

Para el control de volumen, se le ha dado el valor máximo al atributo de Bela que regula el volumen a la salida, 0 dB. Después se ha multiplicado el valor obtenido de la lectura de la entrada analógica, que comprende entre 0 y 1, por 3. El producto entre este valor y el buffer de salida de Bela relaciona directamente el volumen de la señal reproducida con su determinado potenciómetro.

Para el control de la pista, se ha llevado a cabo un proceso parecido. En un primer lugar, se han declarado y guardado varias pistas de audio. Se ha leído el valor de la entrada analógica de su correspondiente potenciómetro y dependiendo del valor obtenido, mediante una serie de condicionales, se procesará una pista u otra.

Esta aplicación es una de las que se podría considerar finales, luego, además del circuito, se ha diseñado una caja de impresión 3D donde situar el sistema sin movimientos innecesarios que puedan afectar al sensor inercial. Se ha fijado a la diadema de unos auriculares y consta de orificios para facilitar la conexión tanto a los auriculares por el conector mini Jack, como a la powerbank mediante el conector USB para la alimentación. La impresión 3D y el hardware se encuentra desarrollado en el TFG complementario a este [22], “Diseño de un tracker cefálico para audio 3D”.

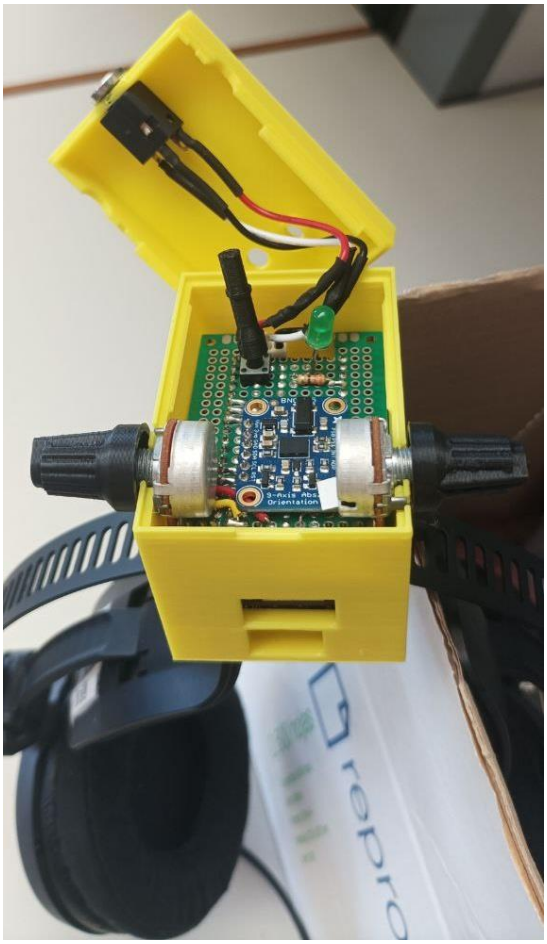

Figura 4.5. Hardware y encapsulado final donde se ejecuta la aplicación.

Este proyecto final se ha presentado en conjunto con un proyecto de la Facultad de Bellas Artes de la Universidad de Málaga al congreso Interacción 2023 formando el artículo “Experimentos de vista y oído, para una pieza de arte sonoro” [23].

A continuación, se muestra el flujograma final del código incorporando todas estas funcionalidades (Figura 4.6).

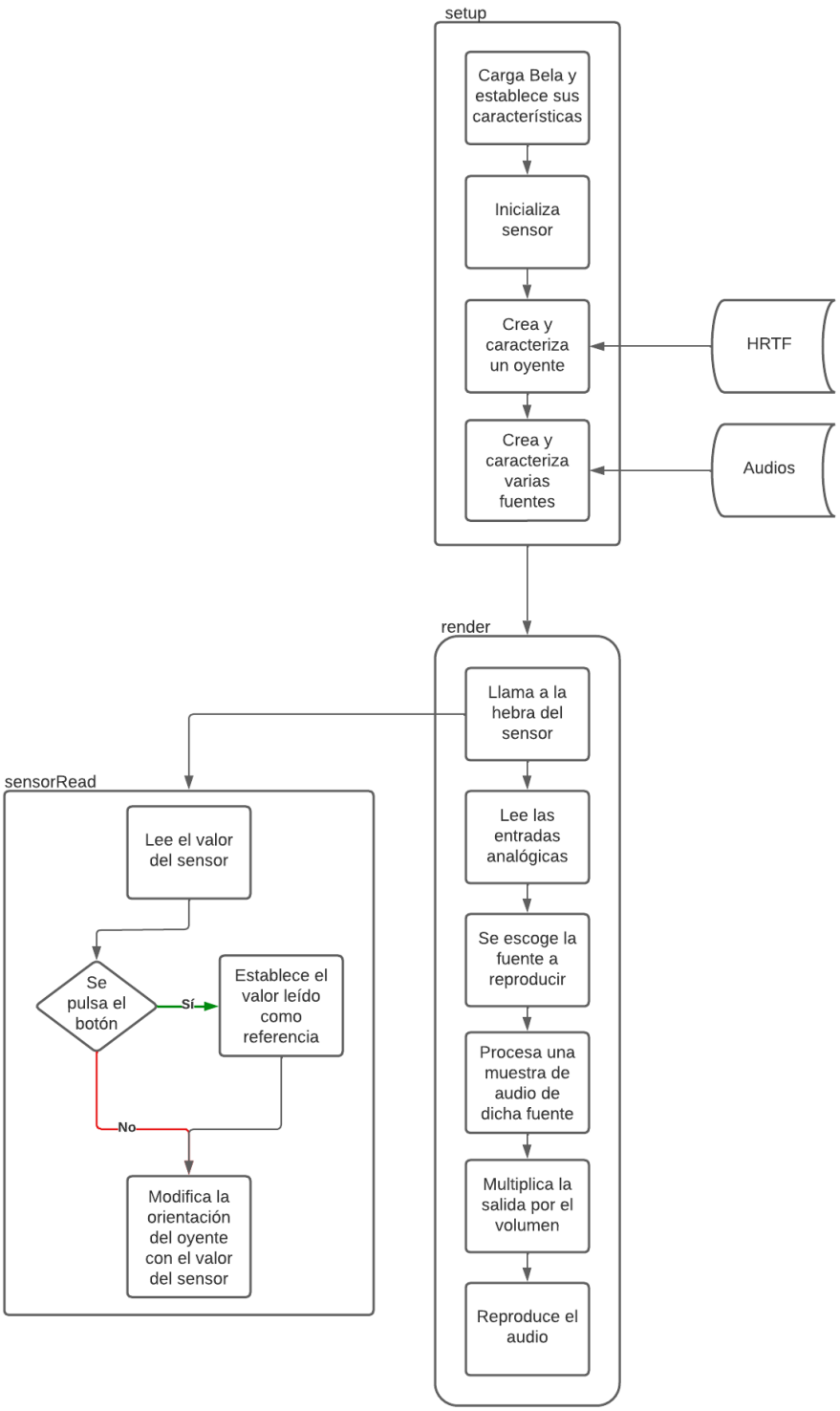

Figura 4.6. Diagrama de flujo del funcionamiento de la aplicación final con controles.

4.4. Pruebas realizadas

Para comprobar que se cumplen los requisitos, se han realizado varias pruebas durante el funcionamiento de la aplicación final desarrollada sobre el prototipo creado en el TFG complementario [22].

El sistema en funcionamiento reproduce una pista de audio que el usuario puede situar en el espacio. Al mover la cabeza, la fuente de audio permanece en el sitio original, luego, detecta el movimiento de la cabeza y actúa sobre la señal de audio en tiempo real para adaptarse al movimiento.

El sistema está compuesto por *PocketBeagle*, *Bela Mini* y una placa diseñada con el sensor inercial BNO055 y varios actuadores, dos potenciómetros y un botón. Un potenciómetro varía el volumen total de la señal de audio y el otro, cambia la pista reproducida por otra distinta, situada en otra posición. Si se mueve la cabeza y se pulsa el botón, la fuente de audio se posiciona en el mismo lugar respecto al oyente.

El prototipo está montado sobre unos auriculares que lleva el usuario. Está alimentado por una *powerbank* y los mismos auriculares se conectan al prototipo permitiendo el movimiento de la persona.

Mediante estas pruebas, se comprueba que se cumplen todos los requisitos menos el requisito R1.5. El sistema no puede reproducir más de una fuente simultáneamente debido a limitaciones de procesamiento del sistema empotrado.

Capítulo 5. Conclusiones y trabajo futuro

5.1. Dificultades

El desarrollo de este proyecto ha supuesto un reto debido a su carácter innovador. Tal y como se ha mencionado en el desarrollo del TFG, la librería 3DTI Toolkit es una librería del grupo de investigación DIANA y está en constante desarrollo. Aunque se ha usado una versión totalmente funcional, no es una librería que esté específicamente pensada para funcionar en ningún sistema empotrado en concreto, de hecho, las demos de la librería se realizan en un ordenador y, por lo tanto, tampoco hay contenido en Internet de personas que lo hayan intentado. Con esto se quiere decir que, aunque teóricamente debía funcionar, no se sabía desde un primer momento qué resultado se iba a obtener ni cómo llegar a ese resultado.

Además, hay que tener en cuenta que Bela también es una placa relativamente nueva y no es tan popular fuera del ámbito del sonido. Esto hace que la información proporcionada por la página oficial esté centrada en el desarrollo desde su propio IDE y con ejemplos relativamente sencillos para captar desarrolladores. Este proyecto necesita de un uso a más bajo nivel de software que el IDE por lo que no hay mucha información y la que ofrece la página oficial para conectarse mediante Eclipse con compilación cruzada está obsoleta.

Teniendo en cuenta estas dos variables, la unión de ellas solo se pudo conseguir mediante ensayo y error e intentando extrapolar problemas y soluciones que hayan tenido otras personas en otros programas a este proyecto.

Tras seguir los pasos que estipula la página Web oficial de Bela para la compilación cruzada con Eclipse, siempre se obtenía un error por una diferencia entre la versión que hace descargar el manual de instrucciones y la versión interna de Bela, por lo tanto, se decidió ignorar esa parte del tutorial e intentar compilar los proyectos directamente con los archivos internos de Bela mediante una carpeta compartida.

Para compilar un proyecto de Bela desde Eclipse, se tuvo que conectar a la placa manualmente mediante el comando SSH e ir compilando poco a poco los proyectos desde la consola para que en los propios errores se pudiesen ver los ficheros que no estaban incluidos. Después hubo que hacerlo con la función de linkado hasta que se encontraron las librerías internas que usa Bela para ejecutar sus proyectos pero que no muestra a sus usuarios desde el IDE. Además, hubo que reestructurar los proyectos de Bela que se quisieran compilar manualmente con la función g++ porque, tal y como se ha comentado anteriormente, el IDE de Bela no muestra la función main() del programa, pero este compilador la necesita.

Tras seguir los pasos, se creó la carpeta compartida y se modificaron los directorios obtenidos en la búsqueda del comando de compilación para que siguieran el camino de la carpeta creada. Finalmente, se consiguió hacer funcionar un proyecto básico de Bela en Eclipse.

Para incorporar la librería 3DTI Toolkit hubo que seguir un proceso parecido ya que había muchos factores que nuestro programa no usaba pero que teníamos que eliminar porque nuestra versión de C++ no entendía. En primer lugar, se probó desde Eclipse pero el proyecto básico no estaba pensado para eso y necesitaba una tarjeta de sonido así que se decidió generar una librería aparte que incluir en el proyecto y usar Bela como tarjeta de sonido directamente. Mediante los errores mostrados a la hora de intentar compilar con la consola, se recogieron las direcciones de los ficheros que necesitaba el proyecto básico de 3DTI Toolkit para compilar y linkar. Estos se pudieron introducir en Eclipse pero tardaba mucho tiempo en compilarlos, luego, se decidió crear la librería con un Makefile.

Una vez que se consiguió introducir la librería 3DTI Toolkit en Bela, debido a las especificaciones de Bela y conociendo el rendimiento de la librería, se tenían altas expectativas respecto al resultado. Sin embargo, solo se pudo reproducir una fuente y por muchos cambios que se realizaron para optimizar el programa, no fue lo suficiente para mejorarlo.

5.2. Resultados

Aunque no se ha conseguido optimizar el sistema lo suficiente como para que sea posible reproducir más de una fuente, con el resultado obtenido se pueden hacer la mayoría de las pruebas y experimentos que necesita el grupo de investigación DIANA para continuar con el desarrollo de su librería.

Anteriormente, se usaban gafas de realidad virtual para distinguir la orientación de la cabeza en las pruebas de la librería 3DTI Toolkit. El sistema desarrollado en este TFG permite hacerlo con un hardware más sencillo y directo.

Además, con este proyecto se ha creado un sistema medidor de latencia que no necesita más de una fuente. Dándole uso, se ha podido obtener la latencia en distintos sistemas que usan la librería 3DTI Toolkit y compararlos llegando a la conclusión de que, aunque Bela no procese las muestras de audio tan rápido como se esperaba, sí que es la más veloz de entre las opciones. Estas medidas están descritas en profundidad en el TFG asociado [22] y cómo interviene este TFG en las medidas se explica detalladamente como una línea futura ya que es una utilidad que se le puede dar a este proyecto.

El sistema desarrollado se puede considerar una base desde la que partir ya que existen varias líneas futuras que continuarían avanzando el proyecto pudiendo llegar a obtener un resultado de gran valía en el sector. Es por eso por lo que este trabajo de fin de grado se ha centrado especialmente en la capacidad de reproducir los pasos para recrear el sistema.

Por todas estas razones y teniendo en cuenta el balance de requisitos cumplidos, se considera que el resultado es satisfactorio.

5.3. Líneas futuras

5.3.1. Medida de latencia

A partir de la aplicación de la incorporación del sensor, previamente desarrollada, se han realizado varias modificaciones para medir la latencia entre el movimiento de la cabeza y la reproducción del audio.

La idea para medir la latencia es la siguiente:

1- Se ha creado un HRTF llamado Hemineglect y se ha cargado al programa (Figura 5.1). Este HRTF hace que cuando el oyente tenga una orientación en el plano x-y de entre 0 y 180º se silenciarán las fuentes. En el caso contrario, cuando el oyente tenga una orientación entre 180 y 360º, las fuentes tendrán su máxima amplitud, por lo tanto, el cambio de una orientación a otra generaría un cambio brusco en la reproducción de audio generando un flanco.

2- Se ha creado una pista de audio en formato WAV y se ha cargado al programa (Figura 5.1). Esta pista es un tono cuadrado centrado en 0V y de 440Hz para tener una señal constante de referencia.

```
39
40 std::string FileSpeech = "/root/resources/Samples/TONO_440_Squ.wav";
41 std::string hrtf = "/root/resources/Hemineglect_44100_256.3dti-hrtf";
42
```

Figura 5.1. Declaración del filtro HRTF y de la señal de audio que se usa para medir la latencia.

3- Se ha deshabilitado la interpolación de la fuente de audio porque, en caso contrario, se suavizaría el cambio brusco entre la reproducción de audio y el silencio y viceversa dificultando la medida.

La latencia es una variable esencial a la hora de caracterizar un sistema, especialmente si este realiza sus funciones en tiempo real. Saber el tiempo de respuesta de un sistema frente a variaciones define su calidad y eficacia. El sistema de medida (Figura 5.2), el procedimiento llevado a cabo y los valores obtenidos en las pruebas están desarrollados de forma más detallada en el TFG asociado [22] “Diseño de un *tracker* cefálico para audio 3D”.

Figura 5.2. Sistema de medición de latencia.

5.3.2. Imagen experimental

Existe una imagen experimental de Bela en la que se ha actualizado el sistema operativo a Debian 11 (Bullseye) que permitiría la compilación de C++ en su versión 13. Con esto, se podría usar la versión más reciente de la librería 3DTI Toolkit que ha sido remodelada y depurada para que su funcionamiento

sea más eficiente. Entre las posibles optimizaciones del código previamente desarrolladas se consideró esta opción. Se habló con los desarrolladores de Bela y se preguntó si la imagen experimental era funcional. Tras su respuesta afirmativa se intentaron realizar estos cambios, sin embargo, debido al desconocimiento y a la serie de errores que surgieron, se vio que era necesaria una comunicación constante con el equipo desarrollador de Bela para darle uso a la imagen. Todo esto suponía una cantidad de tiempo adicional de la que no se disponía, luego, se llegó a la conclusión de dejar este desarrollo para un futuro.

5.3.3. Sensor y protocolo OSC

Una de las opciones consideradas a la hora de decidir el proyecto trata sobre el diseño e implementación de una placa con su propio microcontrolador que solo se encargue de tomar los valores de la orientación de la cabeza y los envíe por protocolo OSC a un ordenador. El ordenador se encargaría de reproducir las pistas de audio y las procesaría dependiendo de los datos recibidos por el mensaje OSC. Esta opción no necesitaría de Bela ya que estaría conectada directamente al ordenador y este se encargaría del procesado. Se podría considerar que la conexión entre la placa y el ordenador fuese vía Wifi e incluso que los auriculares se conectasen por Bluetooth para una conexión totalmente inalámbrica. Todos estos casos estarían pendientes de medir la latencia para caracterizar el sistema, pero, ya solo al incorporar un ordenador con una tarjeta de sonido al conjunto, garantizaría la posibilidad de reproducir una mayor cantidad de fuentes de audio y aplicar efectos de sonido adicionales.

Referencias

[1] M. Cuevas Rodríguez, «Spatial Hearing,» de *3D Binaural Spatialisation for Virtual Reality and Psychoacoustics*, Málaga, Universidad de Málaga, 2022, pp. 1-9.

[2] M. Cuevas-Rodríguez, L. Picinali, D. González-Toledo, C. Garre, E. de la Rubia-Cuestas, L. Molina-Tranco, A. Reyes-Lecuona, “3D Tune-In Toolkit: An open-source library for real-time binaural spatialisation”, PLoS One, vol. 14, no. 3, p. e0211899, marzo, 2019, DOI: 10.1371/journal.pone.0211899.

[3] Reyes-Lecuona, “Audio inmersivo, personalizado e interactivo para RV/RA e investigación en psicoacústica, proyectos SONICOM y SAVLab”, actas de *INTERACCIÓN 2022*, Teruel (España), septiembre, 2022.

[4] E. Aguilera, J. Lopez, P. Gutierrez, M. Cobos, "An Immersive Multi-Party Conferencing System for Mobile Devices Using Binaural Audio", proc. of *55th International Conference: Spatial Audio*, Paper 4-3, Valencia (España), agosto, 2014.

[5] McPherson, V. Zappi, “An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black”, proc. of *Audio Engineering Society Convention 138*, Paper 9331, Varsovia (Polonia), mayo, 2015.

[6] C. Long, J. Kridner, “Meet Beagle™: Open Source Computing”, beagleboard.org, noviembre, 2021. Disponible en: https://beagleboard.org/ [Última visita: 13 de junio, 2023].

[7] Bela, «What is Bela?,» [En línea]. Available: https://bela.io/about. [Último acceso: 13 de junio, 2023].

[8] Bela, «Managing your SD Card,» [En línea]. Available: https://learn.bela.io/using-bela/bela-techniques/managing-your-sd-card/. [Último acceso: 13 junio 2023].

[9] Bela, «Compiling Bela projects in Eclipse,» [En línea]. Available: https://learn.bela.io/using-bela/advanced-topics/compiling-bela-projects-in-eclipse/. [Último acceso: 13 junio 2023].

[10] Reyes-Lecuona, L. Picinali, M. Cuevas-Rodríguez, D. González-Toledo, L. Molina-Tranco, C. Garre, E. de la Rubia-Cuestas, A. Rodríguez-Rivero, “3dti AudioToolkit”, github.com, febrero, 2023. Disponible en: https://github.com/3DTune-In/3dti_AudioToolkit [Última visita: 13 de junio, 2023].

[11] Eclipse Foundation, «Eclipse platform overview,» Marzo 2023. [En línea]. Available: https://help.eclipse.org/2023-03/index.jsp?nav=%2F0. [Último acceso: 13 Junio 2023].

[12] Bela, «Running a program as a service,» [En línea]. Available: https://learn.bela.io/using-bela/bela-techniques/running-a-program-as-a-service/. [Último acceso: 13 Junio 2023].

[13] ARM Developer, «Neon - Technical Information,» [En línea]. Available: https://developer.arm.com/Architectures/Neon#Technical-Information. [Último acceso: 13 Junio 2023].

[14] M. Romanov, P. Berghold, D. Rudrich, M. Zaunschirm, M. Frank, and F. Zotter, “Implementation and Evaluation of a Low-cost Head- tracker for Binaural Synthesis,” in 142nd Convention Audio Engi- neering Society, Berlin, Germany, 2017, pp. 1–6.

[15] Bela, «Bela Mini,» [En línea]. Available: https://learn.bela.io/products/bela-boards/bela-mini/. [Último acceso: 22 junio 2023].

[16] C. Long, J. Kridner, “PocketBeagle”, beagleboard.org, Marzo, 2021. Disponible en: https://beagleboard.org/pocket [Última visita: 22 de junio, 2023].

[17] Bosch, «BNO055. Intelligent 9-axis absolute orientation sensor,» Noviembre 2014. [En línea]. Available: https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf. [Último acceso: 22 06 2023].

[18] diana Research Group, «3DI-DIANA - Grupo de Investigación,» [En línea]. Available: https://www.diana.uma.es/?page_id=53. [Último acceso: 22 06 2023].

[19] Stanford University, «OpenSoundControl.org,» 13 Agosto 2021. [En línea]. Available: https://ccrma.stanford.edu/groups/osc/index.html. [Último acceso: 22 Junio 2023].

[20] Balena, «balenaEtcher,» [En línea]. Available: https://etcher.balena.io/. [Último acceso: 22 Junio 2023].

[21] InvenSense, «MPU-6000 and MPU-6050 Product Specification,» 19 Agosto 2013. [En línea]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. [Último acceso: 23 Junio 2023].

[22] J. J. Navarrete Gálvez, Diseño de un tracker cefálico para audio 3D, Málaga: Universidad de Málaga, 2023.

[23] B. Montalvo, A. Reyes-Lecuona, J. Artero Flores, A. Cajigal, S. Florido y J. J. Navarrete, *Experimentos de vista y oído, para una pieza de arte sonoro,* Málaga, 2023.

Anexos

A. Funciones del código FProcessor.cpp modificadas para que usen la librería de Bela.

i. Transformada directa de Fourier

```
//Calculate the FFT of the input signal in order to convolved it with other signal
void CFprocessor::CalculateFFT(const std::vector<float>& inputAudioBuffer_time, std::vector<float>& outputAudioBuffer_frequency, int irDataLength)
{
int inputBufferSize = inputAudioBuffer_time.size();

ASSERT(inputBufferSize > 0, RESULT_ERROR_BADSIZE, "Bad input size when setting up frequency convolver", "");
ASSERT(irDataLength > 0, RESULT_ERROR_BADSIZE, "Bad ABIR size when setting up frequency convolver", "");
```

```
if ((inputBufferSize > 0) && (irDataLength > 0)) //Just in
case error handler is off
{
////////////////////////////////
// Calculate FFT/output size //
////////////////////////////////
int FFTBufferSize = inputBufferSize + irDataLength;
//Check if if power of two, if not round up to the next
highest power of 2
if (!CalculateIsPowerOfTwo(FFTBufferSize)) {
FFTBufferSize = CalculateNextPowerOfTwo(FFTBufferSize);
}
//FFTBufferSize *= 2;                                    //We
multiplicate by 2 because we need to store real and imaginary
part
//In Bela, we don't multiplicate by 2 because there's an
internal complex array

////////////////////////////////////////////////////////////
////////////////////
// Bela Fft library
////////////////////////////////////////////////////////////
////////////////////
Fft fourier(FFTBufferSize);
fourier.setup(FFTBufferSize);

//////////////
// Make FFT //
```

```
//////////////

fourier.fft(inputAudioBuffer_time);     //Make the FFT


//////////////////

// Prepare Output //

//////////////////

//Copy to the output float vector

if          (outputAudioBuffer_frequency.size()          !=
(FFTBufferSize*2))                                        {
outputAudioBuffer_frequency.resize(FFTBufferSize*2); }

for (int i = 0; i < FFTBufferSize/2; i++) {

outputAudioBuffer_frequency[2         *         i]         =
static_cast<float>(fourier.fdr(i));

outputAudioBuffer_frequency[2     *     i     +     1]     =
static_cast<float>(fourier.fdi(i));


if(i != 0){

outputAudioBuffer_frequency[(FFTBufferSize*2) - (2 * i)] =
static_cast<float>(fourier.fdr(i));

outputAudioBuffer_frequency[(FFTBufferSize*2) - (2 * i + 1)]
= - (static_cast<float>(fourier.fdi(i)));

}

}

}

}
```

ii. Transformada inversa de Fourier

```
//Calculate the IFFT of the output signal
void CFprocessor::CalculateIFFT(const std::vector<float>& inputAudioBuffer_frequency, std::vector<float>& outputAudioBuffer_time)
{
int inputBufferSize = inputAudioBuffer_frequency.size();
ASSERT(inputBufferSize > 0, RESULT_ERROR_BADSIZE, "Bad input size", "");

if (inputBufferSize > 0) //Just in case error handler is off
{
//////////////////////////////
// Calculate output size //
//////////////////////////////
int FFTBufferSize = inputBufferSize / 2;          //We have an array with real and imaginary values but our fft function
//asks for two buffers, one of real values and another with imaginary values
//that's why we divide by 2

////////////////////////////////////////////////////////////
////////////////////
// Bela Fft library
////////////////////////////////////////////////////////////
////////////////////
Fft ifourier(FFTBufferSize);
```

```
ifourier.setup(FFTBufferSize);
     //Bela fft functions for declaring and initializing a
fourier transform


//////////////

// Make IFFT //

//////////////

std::vector<float>
inputAudioBuffer_frequency_Re(FFTBufferSize, 0.0f);    //
Auxiliary array

std::vector<float>
inputAudioBuffer_frequency_Im(FFTBufferSize, 0.0f);


for(int i = 0; i < FFTBufferSize; i++){

inputAudioBuffer_frequency_Re[i]                              =
inputAudioBuffer_frequency[i * 2];      // Divide into real
part

inputAudioBuffer_frequency_Im[i]                              =
inputAudioBuffer_frequency[i * 2 + 1]; //   and   imaginary
part

}

ifourier.ifft(inputAudioBuffer_frequency_Re,
inputAudioBuffer_frequency_Im);    //Make the IFFT


//////////////////

// Prepare Output //

//////////////////

int outBufferSize = inputAudioBuffer_frequency.size() / 2;
     //Locar var to move throught the outbuffer
```

```
if (outputAudioBuffer_time.size() != outBufferSize) {
outputAudioBuffer_time.resize(outBufferSize); }

//float normalizeCoef = 2.0f / FFTBufferSize;
     //Store the normalize coef for the FFT-1

//Fill out the output signal buffer

for (int i = 0; i < outBufferSize; i++) {

outputAudioBuffer_time[i]                                    =
static_cast<float>(CalculateRoundToZero(ifourier.td(i)));
//* normalizeCoef)); Bela Fft lib already normalize the data

}

}

}
```