UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

“Lector y reproductor de partituras musicales mediante técnicas de reconocimiento de imagen”

TRABAJO FINAL DE GRADO

Autor/a:
Sergio Pérez Bernabéu

Tutor/a:
José Ignacio Herranz Herruzo

GANDIA, 2020

Resumen

En este documento se detalla el diseño y funcionamiento de una aplicación programada en MatLab, cuyo principal propósito es facilitar el aprendizaje a aquellos músicos inexpertos que se hayan aventurado en el mundo del piano y la música clásica y necesiten una pequeña ayuda extra.

El algoritmo de la aplicación es capaz de reproducir la melodía, añadir el nombre de las notas y mostrar un teclado animado a partir de una simple imagen de una partitura creada en el editor de partituras MuseScore. Para la detección de las figuras musicales se realizan técnicas de reconocimiento de imagen.

Palabras clave: Partitura musical, reconocimiento de imagen, piano, aplicación, MatLab

Abstract

This document details the design and operation of an application programmed in MatLab, whose main purpose is to facilitate learning for inexperienced musicians who have ventured into the world of piano and classical music and need a little extra help.

The application's algorithm is able to play the melody, add the name of the notes and display an animated keyboard via a simple image of a score created in MuseScore score editor. Image recognition techniques are used to detect musical figures.

Keywords: Music score, image recognition, piano, application, MatLab

Tabla de contenido

Introducción 5
Objetivos 5
Estudio de la simbología musical 6
Estrategia 10
Pasos para la detección de objetos y la codificación de la partitura 11
Funcionamiento de las funciones ofrecidas 38
Diseño y uso de la interfaz gráfica 43
Comparación con otros mecanismos similares 45
Mejoras a largo plazo 45
Conclusiones 46
Referencias 47

Ilustración 1 Logo de MuseScore 6
Ilustración 2 Exportar partitura en MuseScore 6
Ilustración 3 Opción "Fondo transparente" en MuseScore 6
Ilustración 4 Ejemplo de resolución en imagen exportada 7
Ilustración 5 Árbol de decisión de figuras musicales 8
Ilustración 6 Partitura usada durante el estudio de la simbología musical 9
Ilustración 8 Partes de una figura musical 9
Ilustración 7 Ejemplos de corcheas grupales 9
Ilustración 9 Funcionamiento de líneas del pentagrama y llaves 9
Ilustración 10 Fragmento de *imagebin* 12
Ilustración 11 Líneas del pentagrama detectadas 12
Ilustración 12 Valor de *y* de la línea central 13
Ilustración 13 Superposición de figura y líneas de pentagrama 13
Ilustración 14 Posición La 4 16
Ilustración 15 Fragmento del resultado de borrar el pentagrama 16
Ilustración 16 Objeto de compás 18
Ilustración 17 Ejemplos de corcheas grupales 18
Ilustración 18 Corchea con núcleos en horizontal y línea aditiva 19
Ilustración 19 Grosor de línea aditiva 19
Ilustración 20 Anchura de semicorchea doble 19
Ilustración 21 Corchea con línea aditiva recortada 20
Ilustración 22 Corcheas grupales aisladas 20
Ilustración 23 Ubicación de núcleo en figuras de distinta anchura 21
Ilustración 24 *BoundingBox* de un objeto 22
Ilustración 25 Ejemplo de erosión morfológica 22
Ilustración 26 Efecto del borrado de plicas en corcheas grupales 22
Ilustración 27 Fragmento del resultado de borrar plicas a la imagen de las blancas 23
Ilustración 28 Línea aditiva sobrante en una corchea 23
Ilustración 29 Núcleos y corchetes aislados en corcheas y semicorcheas 24
Ilustración 30 Núcleos horizontales separados 24
Ilustración 31 Núcleos conectados 25
Ilustración 32 Las líneas en rojo señalan la posición central de núcleos conectados 25
Ilustración 33 Datos en PC (Partitura codificada) 26
Ilustración 34 Posición de un núcleo para detectar los corchetes que actúan sobre él 27
Ilustración 35 Ejemplos de objetos no interpretables 28
Ilustración 36 Puntillo 29
Ilustración 37 Notas consecutivamente a un semisalto de distancia 29
Ilustración 38 Resultados en *PartCodi* de obtener *N* y *P* 31
Ilustración 39 Notas simultáneas, alineadas verticalmente 31
Ilustración 40 Problema del silencio de redonda 31
Ilustración 41 En orden Becuadro, bemol y sostenido 32
Ilustración 42 Sostenido tras *'skel'* y *'endpoints'* 32
Ilustración 43 Resultado de obtener *N* y *P* en *Alteraciones* 33
Ilustración 44 Relación entre líneas divisorias y *rangopentagramas* 34
Ilustración 45 Alteración de armadura, en la posición de un *Do 5* 35
Ilustración 46 Lectura errónea de las alteraciones (izquierda) respecto la lectura correcta (derecha) 36
Ilustración 47 Fragmento de *ContrAlt* 37
Ilustración 48 Ejemplo del funcionamiento de *Texto* 38
Ilustración 49 Proceso de generación de *cancion* mediante *tmenor* y *tmenorPA* 39
Ilustración 50 Múltiples voces en un pentagrama 40
Ilustración 51 Primera vista de la aplicación 43
Ilustración 52 Aspecto de la aplicación una vez procesada la imagen 44

Introducción

Leer partituras musicales puede ser difícil al principio, y engorroso cuando todavía es necesaria un poco más de práctica. La simbología empleada puede no ser muy sencilla al principio, y en la mayoría de casos la gente termina perdiendo su paciencia, perdiéndose el resultado final de ese esfuerzo. Sin embargo, sacando provecho de las técnicas de reconocimiento de imagen actuales, es posible facilitar el proceso y comenzar directamente con la práctica del instrumento.

Aunque podría ser usado de muchas otras maneras, una de las utilidades de este proyecto es que sirva como complemento a clases musicales, en especial para aquellos alumnos que todavía no logran leer con fluidez una partitura y necesitan de un profesor que les ayude y corrija. En estos casos, los alumnos podrán seguir estudiando sus partituras, ya sean piezas simples o ejercicios, sin la presencia del profesor, pues la aplicación de la que trata este documento les puede servir de guía. En el caso de estudiantes más expertos, es probable que les interese una manera de escuchar la melodía de la partitura antes de ponerse a estudiarla, o por ejemplo, en el caso de los estudiantes de Eso y Bachiller que estudian música, seguramente prefieran una manera rápida de anotar el nombre de las notas a tener que hacerlo de manera manual, ahorrando así tiempo que pueden invertir a la práctica de la flauta. Por supuesto, este tipo de aplicaciones también podría servir de utilidad a aquellas personas que simplemente se sientan interesadas por la música clásica, o que sientan algún tipo de curiosidad por la lectura de partituras.

Dado que cada vez es más común el uso de partituras digitales gracias al continuo progreso de la tecnología, la efectividad de este *Lector de partituras* se centra en aquellas de naturaleza digital. Además, y para que sirva como punto de partida, el algoritmo se centrará en la lectura de partituras para piano, concretamente.

Objetivos

Principales

El objetivo principal de este proyecto es diseñar una aplicación en MatLab que detecte los objetos de una partitura musical en formato digital, de manera que los interprete correctamente, en la medida de lo posible, y construya con ellos una versión codificada de la partitura (a lo que llamaremos de aquí en adelante *Partitura codificada*), de manera que esta se pueda reproducir, mostrar el nombre de las notas en la imagen y un teclado que señale las teclas a pulsar para ejecutar la melodía.

Además, la estructura del algoritmo deberá permitir que, a largo plazo, se soporte la lectura de partituras para otros instrumentos (como por ejemplo la flauta dulce o el violín). También ha de ser sencillo y rápido, en relación a lo que se esperaría de una aplicación así.

Por último, la interfaz gráfica ha de ser lo más simple e intuitiva posible, dado que la mayoría de usuarios potenciales se espera que sean de corta edad.

Secundarios

Para lograr el objetivo principal, se seguirán las siguientes tareas:

- Elegir un editor de partituras y estudiar su simbología
- Dar con una estrategia que permita la reproducción, adición de nombres de nota y visualización de un teclado animado a partir de la imagen.
- Diseñar e implementar el algoritmo correspondiente en MatLab.
- Diseñar una interfaz gráfica que permita al usuario hacer uso de las funciones ofrecidas.

Estudio de la simbología musical

Existen muchos programas de edición de partituras actualmente, sin embargo, cada uno utiliza una simbología musical ligeramente distinta. Aunque esto no impide la elaboración de un algoritmo que interprete correctamente las notas al menos en la mayoría de editores de partituras, el desarrollo de este TFG se basa en la simbología usada por el software MuseScore.

Ilustración 1 Logo de MuseScore

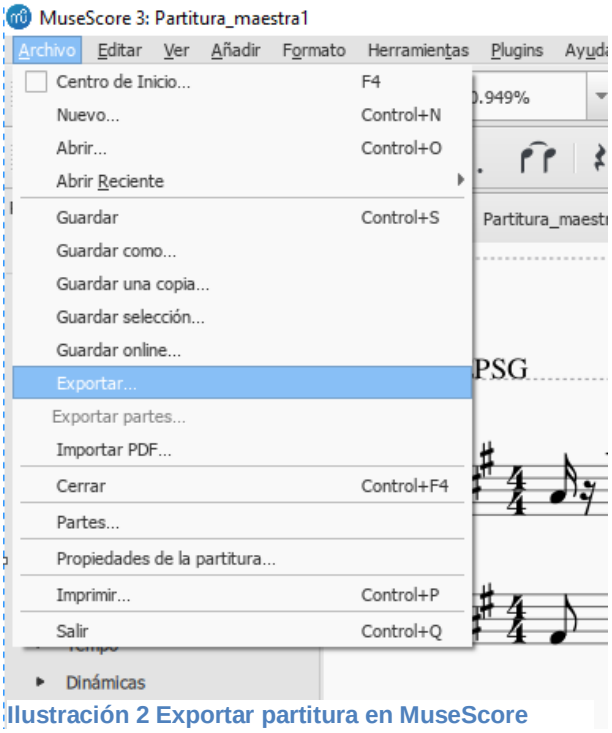


Ilustración 2 Exportar partitura en MuseScore

Se ha elegido este editor de partituras digitales fundamentalmente por dos razones; resulta ser gratis a la vez que muy popular. Prueba de esto son las apariciones en portalprogramas.com [1] como el software de edición de partituras gratis mejor votado, y en neoteo.com [2] como el primero en la lista de "programas para crear partituras".

Además, una de las características más importantes en relevancia a este proyecto, es que MuseScore permite la exportación de las partituras a imagen de una manera rápida y sin complicaciones, siendo estas de gran resolución y alta calidad.

Para exportar a imagen una partitura en MuseScore, basta con acceder a la pestaña "Archivo" y seguidamente clicar en "Exportar…".

Esta es una tarea sencilla para, por ejemplo, un profesor de música que pretende repartir muestras digitales de una partitura a sus alumnos para que ensayen en casa, luego de haber diseñado unos ejercicios específicos.

Sin embargo, hay que prestar atención a las posibles opciones de configuración que estén activadas en el momento de exportar. Por ejemplo, al menos para esta versión del *Lector de partituras,* es muy importante que la opción "Fondo transparente" no esté activada [*ver ilustración 3*].

Ilustración 3 Opción "Fondo transparente" en MuseScore

Observemos también cómo la resolución a la que MuseScore exporta de manera predeterminada sus partituras resulta muy apropiada. La resolución de 360DPI ofrece imágenes de 2977x4208px. A continuación, se observa un fragmento de una partitura exportada a esta resolución en escala 1:1.

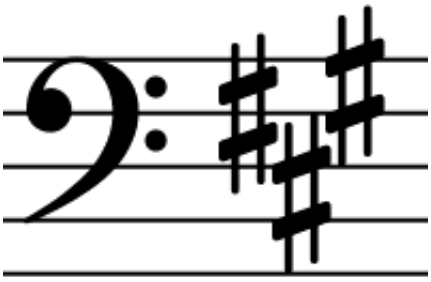
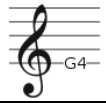
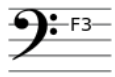
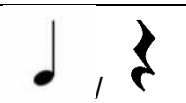
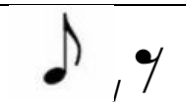
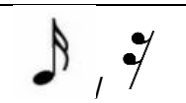
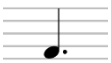


Ilustración 4 Ejemplo de resolución en imagen exportada

En un futuro es posible adaptar el algoritmo de esta aplicación de manera que se interpreten correctamente partituras de menor resolución. Sin embargo, además de que existe cierto límite (pues se necesita un nivel mínimo de detalle para el reconocimiento de las figuras), complicaría enormemente el procedimiento a seguir. Por ejemplo, a baja resolución, dos figuras musicales muy próximas entre sí podrían superponerse, perdiendo una manera simple de distinguir ambas figuras.

Continuando con el estudio de la simbología, se muestra una tabla descriptiva de los símbolos que estudiaremos:

Símbolo	Nombre	Descripción
	Llave de Sol	Asigna a la quinta línea del pentagrama, contando desde arriba, la nota Sol 4
	Llave de Fa	Asigna a la segunda línea del pentagrama, contando desde arriba, la nota Fa 3.
	Redonda / Silencio de redonda	Si no es silencio, indica la altura de la nota a tocar. Su duración es de 4 pulsos.
	Blanca / Silencio de blanca	Si no es silencio, indica la altura de la nota a tocar. Su duración es de 2 pulsos.
	Negra / Silencio de negra	Si no es silencio, indica la altura de la nota a tocar. Su duración es de 1 pulsos.
	Corchea / Silencio de corchea	Si no es silencio, indica la altura de la nota a tocar. Su duración es de ½ pulso.
	Semicorchea / Silencio de semicorchea	Si no es silencio, indica la altura de la nota a tocar. Su duración es de ¼ de pulso.
	Sostenido	Aumenta en un semitono las notas a su misma altura.
	Bemol	Disminuye en un semitono las notas a su misma altura.
	Becuadro	Cancela el efecto de sostenidos y bemoles.
	Puntillo	Alarga la duración de la nota ½ de su duración original
	Línea Divisoria	Indica la finalización de un compás.

Gracias al software de MatLab y alguna de sus aplicaciones internas como ImageViewer, podemos estudiar cada uno de estos símbolos a fin de crear un filtro que distinga cada uno de los símbolos. Para ello, y para facilitar una futura compatibilidad con más editores de partituras, trataremos de utilizar características generales. Aprovecharemos el hecho de que los símbolos son proporcionales al pentagrama, y agruparemos los símbolos según una serie de umbrales.

Para esto, definimos *espacio* como la distancia en píxeles entre dos líneas consecutivas del mismo pentagrama. Esta distancia permanece constante para todos los pentagramas (al menos en el caso de MuseScore).

Así pues, se incluye abajo el árbol de decisión usado para la distinción de objetos, paso que detallaremos más adelante [*ver Detección de objetos pág.16*]

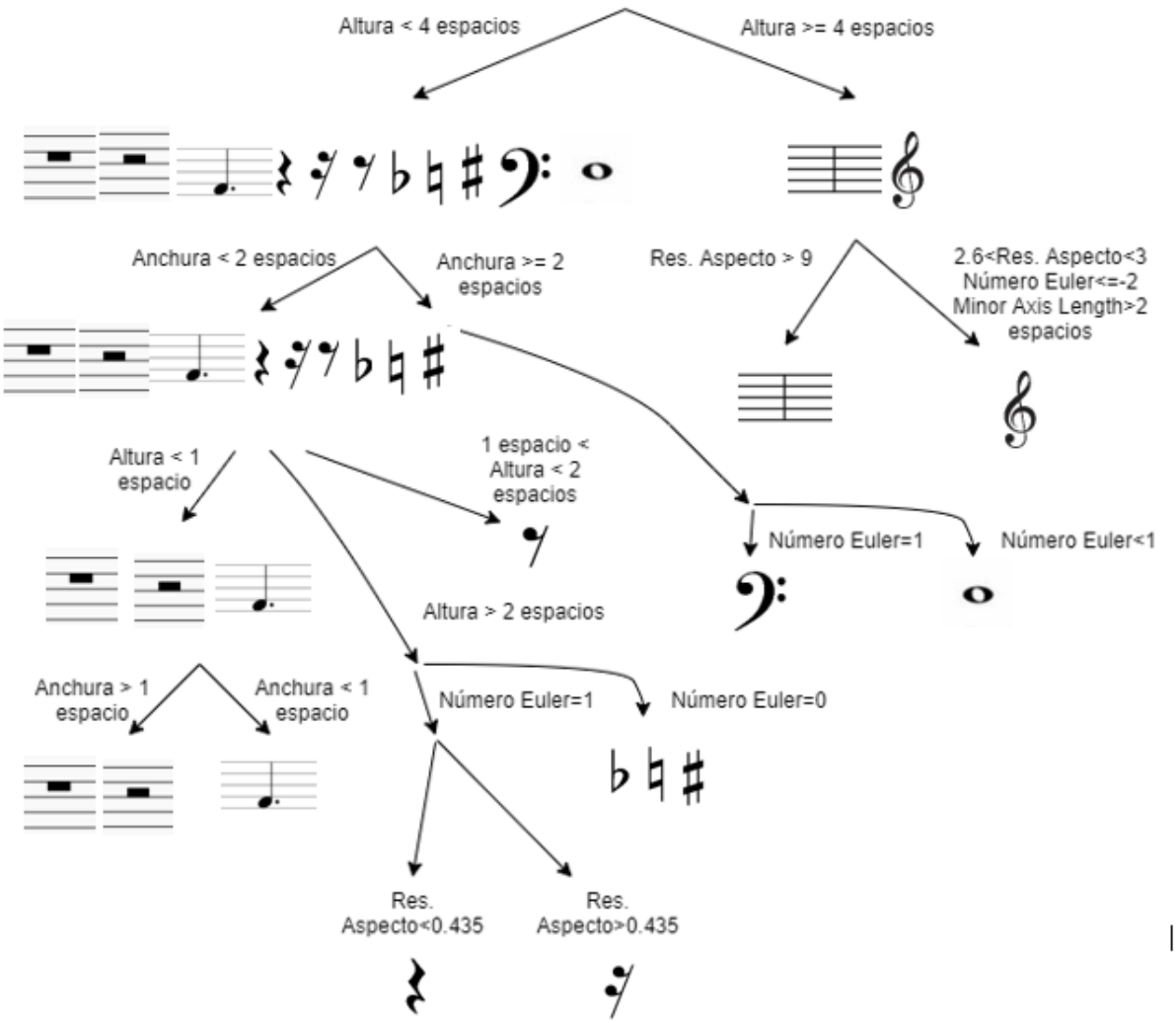


Ilustración 5 Árbol de decisión de figuras musicales

Figuras restantes tras la aplicación del árbol de decisión: 

Como se puede observar de la ilustración, el árbol de decisión no distingue directamente entre silencios de redonda y de blanca, aunque para ello basta con calcular la distancia respecto la segunda línea del pentagrama en el que se encuentra. Tampoco distingue entre los tres tipos de alteraciones, ni en las notas 'tocables', es decir, no silencios (las figuras restantes). Estas distinciones se detallarán más adelante, pues requieren de procesado extra.

Para el estudio de los símbolos musicales, se ha diseñado una partitura que contiene las variantes más interesantes de los símbolos vistos anteriormente. En ella se incluyen los casos más 'conflictivos' o complicados que se ha decidido soportar en esta versión de la aplicación. Se adjunta a continuación la imagen:

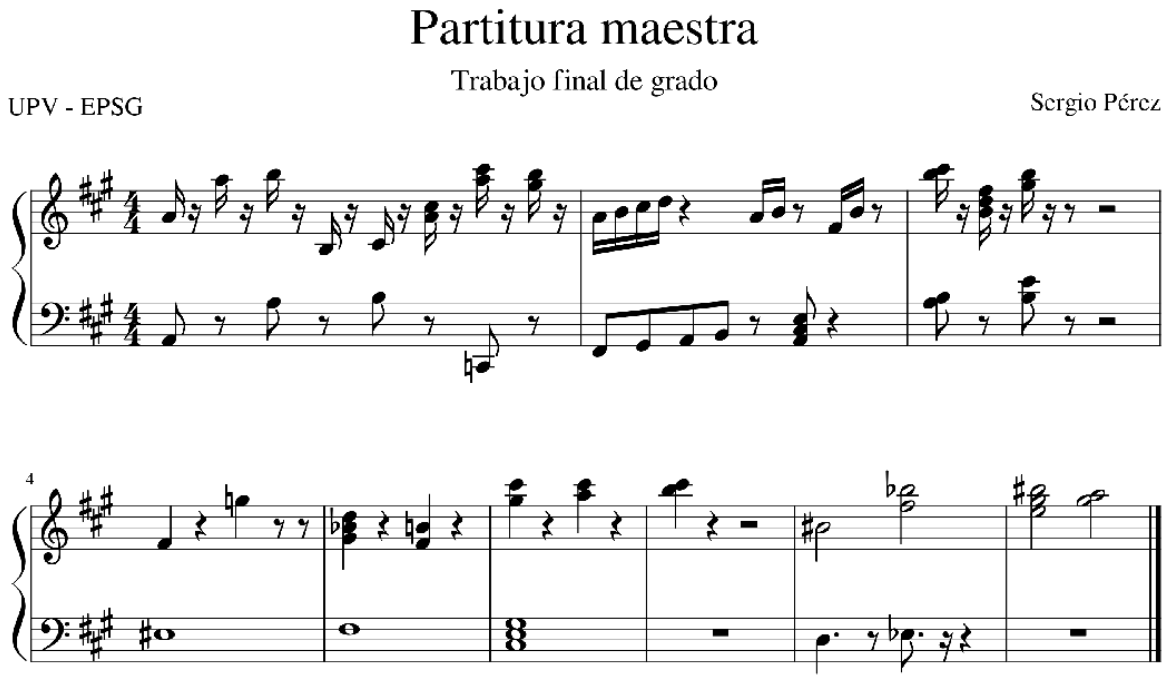


Ilustración 6 Partitura usada durante el estudio de la simbología musical

Para el entendimiento de lo que queda de documento, es interesante conocer las diversas partes de una figura musical, y lo que conoceremos a partir de ahora como 'corcheas grupales' (grupos de corcheas o semicorcheas que se agrupan bajo el mismo corchete).

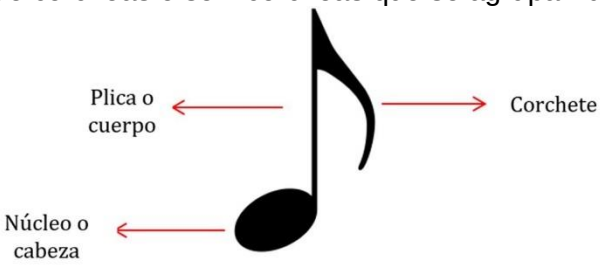


Ilustración 8 Partes de una figura musical

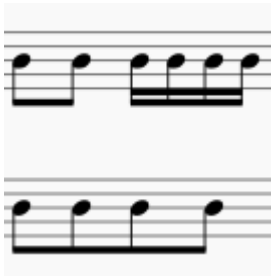


Ilustración 7 Ejemplos de corcheas grupales

Además, en la ilustración de abajo se puede observar las distintas posiciones en las que puede encontrarse una figura musical en el pentagrama, y de qué manera se relacionan la llave de Sol y la de Fa.

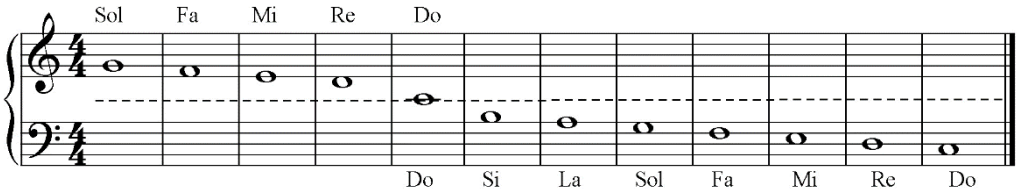


Ilustración 9 Funcionamiento de líneas del pentagrama y llaves

Estrategia

Pentagrama

Para poder clasificar y tomar en consideración cada figura musical, hemos de aislar cada una de ellas del resto de objetos. Como se puede observar, lo único que nos impide esto en primera estancia son las líneas del pentagrama. Por lo tanto, uno de los primeros pasos del algoritmo será la eliminación de las líneas del pentagrama.

Partitura codificada

Para poder realizar las funciones deseadas, las cuales llamaremos a partir de ahora *Melodía*, *Texto* y *Modo Práctica,* hemos de transformar la información visual de la imagen en otro tipo de valores que contengan la información esencial de la partitura, de manera que estos se puedan interpretar de una manera u otra a fin de lograr la función deseada. Para esto, tengamos en cuenta qué tipo de información se necesita para cada una:

- *Melodía*: Necesitaremos toda la información tonal y rítmica. Esto quiere decir que no basta con codificar las figuras que representen notas a tocar, si no también aquellas figuras que alteren el ritmo o la altura. Esto incluye las alteraciones, las llaves y los puntillos. Además, habrán de ser procesadas con anterioridad las líneas divisorias, pues de estas depende el efecto de las alteraciones y los puntillos.

  La duración de las notas viene dada por la forma de la figura en cuestión, pero en el caso de la altura, por las coordenadas *y* respecto la línea central del pentagrama al que pertenezca. Además, las coordenadas *x* serán importantes para determinar el orden en el que sonarán las notas.

- *Texto*: El mejor lugar para añadir el nombre de las notas es entre pentagrama y pentagrama, debajo de la nota a la que se nombra. Por eso, necesitaremos el número de pentagrama al que pertenece la nota, así como su coordenada en *x*.

- *Modo Práctica*: Además de la altura y la duración de las notas, necesitaremos valores de *x* de manera que se pueda seguir la partitura mediante un indicador que recorra la melodía conforme el teclado la reproduce.

Además de todo ello, añadiremos un parámetro necesario en las tres funciones; es imprescindible conocer cuándo las notas van a ser tocadas simultáneamente. De esta manera, en un acorde, las notas sonarán a la vez, aunque se hayan identificado de forma separada, y el nombre de cada una de sus notas se mostrará uno arriba del otro, sin superponerse. Llamaremos a este parámetro *Simul* (de simultáneo).

Así pues, hemos de obtener una cadena de valores, a la que llamaremos a partir de ahora *Partitura codificada,* con el siguiente formato:

$$\begin{bmatrix} N_1 & \cdots & Nn \\ T_1 & \cdots & Tn \\ X_1 & \cdots & Xn \\ P_1 & \cdots & Pn \\ S_1 & \cdots & Sn \end{bmatrix}$$

Donde:

- *n* equivale al número de notas totales, incluidos los silencios.
- *N* hace referencia a la información tonal de la nota. Llamaremos a este parámetro *Número de nota,* y servirá tanto para asociar cada nota a una tecla del piano como para obtener su frecuencia en *Hz*. El valor de 0 representa un *La 4,* y el 99 representa un silencio.
- *T* hace referencia a la información rítmica de la nota. Llamaremos a este parámetro *Tipo de nota,* pues viene dado principalmente por el tipo de figura musical. Representa el número de pulsos de duración de la nota.

- *X* equivale a las coordenadas en *x* del objeto en las que se encuentra la nota.
- *P* equivale al número de pentagrama al que pertenece la nota.
- *S* (*Simul*) hace referencia a la simultaneidad de las notas. Este valor es 1 cuando se trata de la última nota del acorde, es decir, se considerarán simultáneas esa nota y las consecutivas anteriores que tengan un valor de *Simul* igual a 0.

Pasos para la detección de objetos y la codificación de la partitura

A continuación, se detallará el proceso del que hace uso la aplicación para detectar figuras musicales en una imagen y obtener la codificación de la partitura.

Parámetros iniciales

Ahora que conocemos la estrategia a seguir, desarrollaremos el código con el que va a funcionar el *Lector de partituras*.

En primer lugar, estableceremos unos parámetros iniciales. La posición en el pentagrama de una nota determina el *Número de nota* de la figura previo al efecto de las alteraciones. *EscalaNatural* representa estos valores para una tesitura de 5 octavas, comenzando desde el *Do 2*. Además, se establece un valor para *BPM* ("Beats per minute*",* en español pulsos por minuto) y para *Fs* ("Sampling frequency", en español frecuencia de muestreo) de manera predeterminada. *Ts* representa el periodo de muestreo, y equivale a la inversa de *Fs*.

En esta versión de la aplicación se considera un umbral de color fijo de 25. Esto quiere decir que, durante el proceso de umbralización, se convertirán a negro los valores inferiores a este valor y a blanco los superiores. Esto considerará las sombras (grises) como parte del objeto.

```
EscalaNatural=[-33 -31 -29 -28 -26 -24 -22 -21 -19 -17 -16 -14 -12 -10 -9 -7
-5 -4 -2 0 2 3 5 7 8 10 12 14 15 17 19 20 22 24 26 27 29 31 32 34 36 38 39];
BPM=100;
Fs=8192;
Ts=1/Fs;
threshold=25;
```

Primer procesado

Para garantizar un buen funcionamiento independientemente de la naturaleza de la imagen, nos aseguraremos de que esta esté en escala de grises, es decir, no contenga información de color. Además, mediremos el alto y el ancho de la imagen, que servirá en múltiples ocasiones.

Por último, debido a que las herramientas de reconocimiento de imagen consideran que hay un objeto cuando los pixeles están a 1 (color blanco en binario, equivalente al 255 en escala de grises) guardaremos una copia de la imagen con el color invertido. En esta imagen, las figuras musicales se verán de color blanco y podrán ser detectadas como objetos.

```
if  size(imageoriginal,3)==3
    imageoriginal=rgb2gray(imageoriginal);
end

[alto, ancho]=size(imageoriginal); %medimos
imagen=255.-imageoriginal; %invertimos
```

Binarización previa a Hough

Para poder realizar la transformada de Hough y detectar las líneas del pentagrama, hemos de obtener la binarización de la imagen. Usando un valor de umbral de 25, [*ver Parámetros iniciales, esta misma pág.*] convertimos la imagen en escala de grises a una imagen binaria, comprobando píxel a píxel su valor.

```
imagebin=zeros(alto, ancho); %imagen vacía
for i=1:alto
    for b=1:ancho
        if imagen(i, b)<threshold
            imagebin(i, b)=0;
        else
            imagebin(i,b)=255;
        end
    end
end
```

Ilustración 10 Fragmento de *imagebin*

Nótese que en este momento conservamos tres imágenes, la original *imageoriginal*, la invertida *image* y la invertida y binarizada *imagebin*, de la cual podemos ver un fragmento en la ilustración de la derecha.

Transformada de Hough (líneas de pentagrama)

Wikipedia describe la transformada de Hough como "una técnica para la detección de figuras en imágenes digitales (…) tales como rectas, circunferencias o elipses." [3]. En este caso, nos servirá para detectar las líneas del pentagrama como paso previo a la eliminación de este.

Se pretende detectar cada una de las líneas de 1 píxel de grosor correspondientes al pentagrama. Para ello, tendremos en cuenta las características más distintivas:

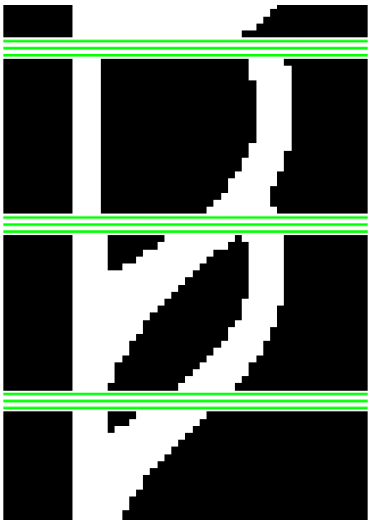


Ilustración 11 Líneas del pentagrama detectadas

- Son perpendiculares al eje *x*. Esto se representa con un valor de theta de -90.
- Su longitud es mayor a la mitad del ancho de la hoja.
- No suele ser común ver más de 10 pentagramas en una misma página. Dado que, a la resolución por defecto, las líneas del pentagrama miden 3 píxeles de grosor, y cada pentagrama contiene 5 de estas líneas, necesitamos un total de 150 picos de Hough. Este número equivale al número de líneas de 1 píxel de grosor que consideraremos. En el código aparece como 160 por motivos de seguridad (alguna línea podría medir inesperadamente más de 3 píxeles, o la imagen podría ser de otra resolución).

En la ilustración de la izquierda podemos ver cómo la transformada de Hough detecta satisfactoriamente cada línea de 1 píxel de grosor del pentagrama (marcadas en verde).

La información más interesante que obtenemos de cada una de las líneas son las coordenadas de su punto inicial y final.

```
%Detecto cada linea de pixeles que pertenecen al pentagrama
[H,T,R] = hough(imagebin, 'theta', -90);
P  = houghpeaks(H,160,'NHoodSize', [1 1]);
lines = houghlines(imagebin,T,R,P,'FillGap',15,'MinLength',ancho*1/2);
```

Ordenamiento de las líneas obtenidas

Dado que las líneas que hemos procesado son totalmente paralelas al eje *x*, podemos definir su posición con una única coordenada en el eje *y*. Además, la información del vector que genera la transformada de Hough por defecto está desordenada. Para simplificar pasos posteriores, generaremos el vector *LineasOrdenadas*, el cual contiene los valores de *y* ordenados de cada línea procesada anteriormente.

```
numlineasHough=size(lines, 2);
LineasOrdenadas=zeros(numlineasHough, 1);
for i=1:numlineasHough
    LineasOrdenadas(i)=lines(i).point1(2);
end
LineasOrdenadas=sort(LineasOrdenadas);
```

Obtención de LineasInd y color de líneas

Como hemos visto antes, necesitaremos la altura de las figuras respecto la línea central del pentagrama para obtener la nota a la que representa *[ver Partitura codificada (Melodía), pág. 10*]. Para ello, obtendremos otro vector que contenga el valor de *y* de la línea central de cada grupo de líneas de 1 píxel de grosor que forman una línea de pentagrama. Por ejemplo, en la ilustración 12 se observa que 592 es el valor de *y* mencionado de la línea de pentagrama en cuestión.

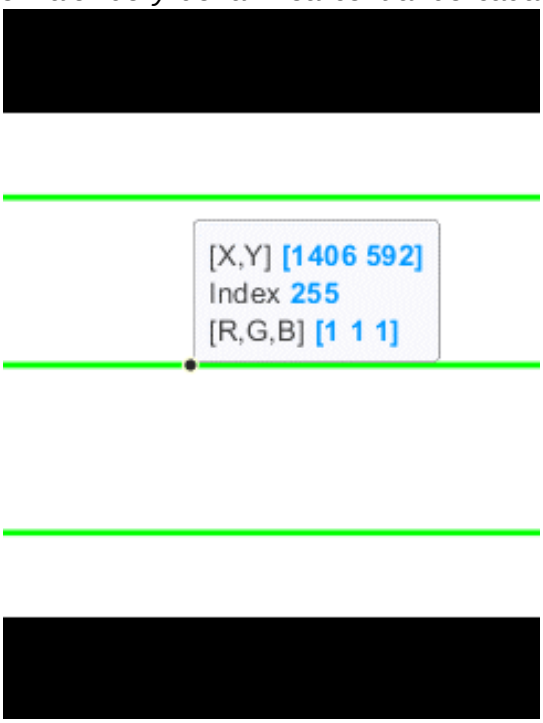


Ilustración 12 Valor de *y* de la línea central

Además, para el borrado del pentagrama necesitaremos conocer el color de cada una de las líneas de 1 píxel. La razón de esto es la siguiente: si pusiéramos a 0 todos los píxeles de la línea, no solo borraríamos la línea, si no que trocearíamos todas las figuras musicales que se posan sobre ella. Esto complicaría mucho el proceso de reconocimiento de objetos, por lo que no se hará así en este proyecto.

La manera usada de distinguir figuras musicales de las líneas del pentagrama es basándose en el color. Cuando una figura se superpone a una línea del pentagrama, sus valores aumentan, tendiendo al blanco. Únicamente cuando la línea ya es lo más blanca posible (valor 255), se necesita comprobar el color de los píxeles vecinos para conocer si hay una figura superpuesta o no.

Aunque no resulta común en MuseScore, la obtención del color de las líneas se hará realizando el promedio del valor de todos los pixeles de la línea que no tenga una figura superpuesta, dado que sería posible que la línea varíe ligeramente de color durante su trazo. Además, esto resulta más estable y seguro frente a errores o casos excepcionales.

En el código puede verse como se agrupan las líneas consecutivas para obtener la información comentada.

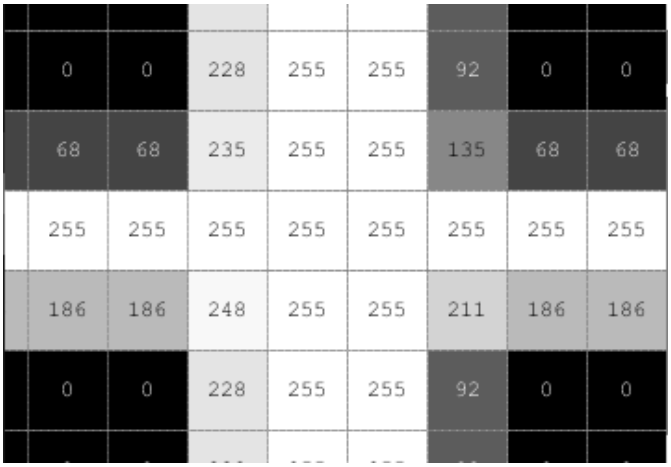


Ilustración 13 Superposición de figura y líneas de pentagrama

```
LineasInd=[]; %Vector de líneas centrales
LAgr=1; %Líneas agrupadas
coloresquevoyviendo=[];
color=zeros(numlineasHough, 1);
for i=1:numlineasHough
    %Si la línea siguiente es consecutiva, la agrupo
    if i~=numlineasHough && LineasOrdenadas(i)+1==LineasOrdenadas(i+1)
        LAgr=[LAgr; i+1];
    else
        %Termino de agrupar. Calculo color y linea central
        for b=1:length(LAgr) %Para cada linea agrupada
            for c=lines(LAgr(b)).point1(1):lines(LAgr(b)).point2(1) %Desde
su punto de inicio a su punto final
                %Si no hay objetos tiñiendo la línea, guardo su color
                if imagen(LineasOrdenadas(LAgr(1))-1, c)<threshold &&
imagen(LineasOrdenadas(LAgr(end))+1, c)<threshold

coloresquevoyviendo(end+1)=imagen(LineasOrdenadas(LAgr(b)), c);
                end
            end
            color(LAgr(b))=mode(coloresquevoyviendo); %Promedio
            coloresquevoyviendo=[];
        end

        LineasInd(end+1)=median(LineasOrdenadas(LAgr)); %Línea central

        %Preparo LAgr para la siguiente iteración
        if i~=numlineasHough
            LAgr=i+1;
        end
    end
end
```

Comprobación de número de líneas

Dado que el pentagrama siempre está formado por 5 líneas, es importante comprobar que el número de líneas individuales (centrales) que hemos obtenido sea múltiplo de 5. El código siguiente hace saltar un error si lo anterior no se cumple.

```
numlineas=length(LineasInd);
numpent=numlineas/5;
if mod(numlineas,5)~=0
    f = errordlg('Numero incoherente de lineas de pentagrama' ,'Algo ha ido
mal...');
end
```

Rango de pentagramas

Tanto para la adición de texto, como para mostrar la imagen del pentagrama correspondiente durante la visualización del teclado animado, necesitamos conocer qué rango comprende cada pentagrama. Además, será útil para asignar a cada figura el pentagrama que le corresponda. Calcularemos este rango como los valores de *y* que equidisten de dos pentagramas consecutivos.

Como se entenderá, el primer y último valor no se pueden calcular de la misma manera, por lo que se considerará que el primer pentagrama abarca desde una distancia igual al ancho del pentagrama multiplicado por un factor de $3/2$ más arriba de su primera línea hasta el punto que equidista con el segundo pentagrama, y de similar manera para el último pentagrama, sumándole lo mismo hacia abajo.

Por último, redondearemos los valores, pues no podremos referirnos a píxeles de valor decimal.

```
rangopentagramas=zeros(numpent+1, 1);
margen=(LineasInd(5)-LineasInd(1))*1.5;

for i=1:numpent-1
     %valor medio entre linea 5 y 6, 10 y 11...
    rangopentagramas(i+1)=LineasInd(5*i)+(LineasInd(5*i+1)-
LineasInd(5*i))*0.5;
end

rangopentagramas(1)=LineasInd(1)-margen; %primer valor
rangopentagramas(end)=LineasInd(end)+margen; %último valor

for i=1:numpent+1 %Redondeamos
   rangopentagramas(i)=round(rangopentagramas(i));
end
```

Borrado de imagen no necesaria

Como se puede ver en la ilustración 6 [*ver pág. 9*], usualmente las partituras contienen el título y demás información en la parte superior de la imagen. Esta información resulta molesta durante el proceso de reconocimiento de objetos y ralentiza procesos posteriores. Por ello, nos deshacemos de tal fragmento de la imagen. Al hacer esto, la información que hemos obtenido en el proceso anterior se ve afectada [ver Rango de pentagramas, pág. anterior]. Sin embargo, basta con conocer el tamaño del fragmento eliminado para recuperar dicha información.

Dado que la adición del texto y los pentagramas que se muestran durante el *Modo Práctica* [*ver Partitura codificada, pág. 10*] funcionan a partir de la imagen original (es decir, no recortada), salvaremos los valores obtenidos en el apartado anterior en un nuevo vector llamado *ytexto*.

```
%Me quedo una copia para Texto y los fragmentos de imagen
for i=1:numpent+1
    ytexto(i)=rangopentagramas(i);
end

if rangopentagramas(1)<0
    rangopentagramas(1)=1;
end

%Borrado de imagen no necesaria
imagen=imagen(rangopentagramas(1):rangopentagramas(end), :);

%Actualizamos la información
[alto, ancho]=size(imagen);

for i=1:numlineasHough
    LineasOrdenadas(i)=LineasOrdenadas(i)-rangopentagramas(1)+1;
end

for i=1:numlineas
    LineasInd(i)=LineasInd(i)-rangopentagramas(1)+1;
end

for i=length(rangopentagramas):-1:1
    rangopentagramas(i)=rangopentagramas(i)-rangopentagramas(1)+1;
end
```

Almacenamiento del La 4 por pentagrama

Para facilitar la asignación de un *Número de nota* a las figuras, almacenaremos el valor de *y* que correspondería para un *La 4*. Este valor se calcula como el valor que equidista de la 3ª y 4ª línea de cada pentagrama. Por ejemplo, la ilustración del lado izquierdo muestra una figura ubicada en ese lugar.

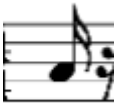


Ilustración 14
Posición La 4

Además, aprovechamos el momento para introducir la variable *espacio*, la cual hemos definido en la página 8.

```
La4=zeros(1, numpent);
salto=LineasInd(2)-LineasInd(1);
semisalto=salto/2;

for i=1:numpent
    La4(i)=LineasInd(i*5-2)+semisalto;
end
```

Borrado de pentagrama

Llegados a este punto estamos en disposición de borrar las líneas del pentagrama para avanzar hacia el reconocimiento de objetos. Para ello seguiremos el siguiente criterio [*ver Obtención de LineasInd y color de líneas, pág. 13*]:

- Si el color de la línea no es lo más blanco posible (podremos distinguir un objeto superpuesto) borramos todos aquellos píxeles cuyo color sea igual o menor al de la línea.
- Si el color de la línea es prácticamente puro blanco, borraremos aquellos píxeles cuyos vecinos superiores sean negros (valor menor al umbral [*ver Parámetros iniciales, pág. 11*]).

Una vez borrado el pentagrama, la imagen está lista para ser binarizada y filtrada por el árbol de decisión.

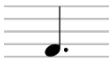
Ilustración 15 Fragmento
del resultado de borrar el pentagrama

```
for i=1:numlineasHough
    if color(i)<250
        for pixel=1:ancho
            if imagen(LineasOrdenadas(i),
pixel)<=color(i)+5
                %si el color es <= al del pentagrama
                imagen(LineasOrdenadas(i), pixel)=0;
            end
        end
    else
        for pixel=1:ancho  %avance hacia la derecha
            if imagen(LineasOrdenadas(i)-1,
pixel)<threshold
                %si su vecino superior no tiene objetos
                imagen(LineasOrdenadas(i), pixel)=0;
            end
        end
    end
end
```

Nótese que el pentagrama se borra en la imagen no binarizada. La imagen binarizada únicamente se usa para detectar las líneas del pentagrama.

Detección de objetos

Una vez aislados los objetos, llega el momento de reconocer los objetos según su forma, de la misma forma que hemos estudiado en el árbol de decisión [*ver Estudio de la simbología musical, pág. 8].* No se añadirá el código íntegro correspondiente a esta parte por motivos de espacio, pero se detallará a continuación cómo se almacenará la información esencial de cada figura una vez reconocida.

Figura(s)	Nombre	Método de almacenamiento
	Silencios (Todos)	Matriz *PC* (*Partitura codificada*) con el formato: $\begin{bmatrix} N_1 & \cdots & Nn \\ T_1 & \cdots & Tn \\ X_1 & \cdots & Xn \\ Y_1 & \cdots & Yn \end{bmatrix}$ Donde: • *N* corresponde al *Número de nota.* En este momento se introduce un 99 si se trata de un silencio o un 0 si se trata de una redonda • *T* corresponde a la duración en pulsos • *X* corresponde a las coordenadas en el eje *x* • *Y* corresponde a las coordenadas en el eje *y*
	Redondas	
	Puntillos	Matriz *Puntillos* y *LineasDivisorias* respectivamente con el formato: $\begin{bmatrix} X_1 & \cdots & Xn \\ Y_1 & \cdots & Yn \end{bmatrix}$ Donde: • *X* corresponde a las coordenadas en el eje *x* • *Y* corresponde a las coordenadas en el eje *y*
	Líneas divisorias	
	Alteraciones (Todas)	Se guarda la información visual en una imagen aparte (*Alt*). Requerirá de procesado extra (insertar referencia)
	Llaves (Todas)	Matriz *Llaves* con el formato: $\begin{bmatrix} T_1 & \cdots & Tn \\ X_1 & \cdots & Xn \\ Y_1 & \cdots & Yn \end{bmatrix}$ Donde: • *T* corresponde al tipo de llave, donde 0 corresponde a la llave de Sol y 1 a la de Fa • *X* corresponde a las coordenadas en el eje *x* • *Y* corresponde a las coordenadas en el eje *y*

Cada vez que se llega a reconocer un objeto y se almacena su información, este se elimina de la imagen. Además, para el caso de las líneas divisorias, se comprobará su altura de la manera en la que sigue:

```
if stats(i).BoundingBox(4)>0.75*LineasInd(10)-LineasInd(1)
        PentagramaDoble=1;
end
```

En el caso de que al menos una de ellas mida más del 75% la distancia entre la primera línea de un pentagrama y la última del pentagrama siguiente, se considerará que el pentagrama es doble, es decir, típico de una partitura para piano. Esto se hace así para poder soportar partituras de otros instrumentos en el futuro. Una vez todos los elementos han sido filtrados y eliminados, nos queda una imagen donde solo quedan las figuras:  .

Borrar el objeto de compás

Al igual que hicimos con el fragmento de la partitura donde aparece el título de la partitura [*ver Borrado de imagen no necesaria, pág. 15*], nos desharemos del objeto que indica el tipo de compás que se ha de seguir. Nos referimos al objeto que aparece en la ilustración 16. Esto se hace porque no es relevante, pues el ritmo es un concepto a tener en cuenta solo en la interpretación de la pieza, y su simulación en una aplicación de este tipo tendría un efecto apenas notable.

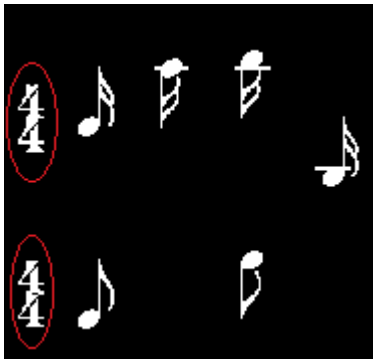


Ilustración 16 Objeto de compás

El criterio empleado para borrar este objeto es el siguiente:

- Buscamos objetos cuya anchura sea menor a la mínima típica presente en una corchea doble y que presenten una alta compacticidad, es decir, un gran número de píxeles en un área reducida. Esto se hace así porque las corcheas grupales [*ver ilustración 17, en esta pág.*] pueden cumplir el criterio por el lado del corchete [*ver ilustración 5, pág. 9*] cuando este es paralelo al eje *x.*

- De ellos, borramos aquellos objetos cuya relación entre su altura y los píxeles que se encuentren en el límite superior o inferior sea mayor a 7. Esto se hace así porque, como se puede intuir de la ilustración 16, el resto de objetos son puntiagudos en sus extremos.

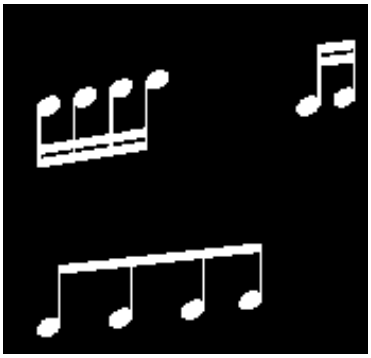


Ilustración 17 Ejemplos de corcheas grupales

```
for i=1:CCobj.NumObjects
    if stats(i).BoundingBox(3)<LineasInd(4)-LineasInd(1) &&
stats(i).Area/stats(i).BoundingBox(3)/stats(i).BoundingBox(4)>0.4
%No corchea y compacto
        puntosqueveo=0;
        x=stats(i).BoundingBox(1)+0.5; %Coordenadas esq. superior izquierda
        y=stats(i).BoundingBox(2)+0.5;

        for pixel=x:x+stats(i).BoundingBox(3)
            if imageobj(y, pixel)==1
                puntosqueveo=puntosqueveo+1;
            end
        end

        if stats(i).BoundingBox(4)/puntosqueveo<7 %Cumple el criterio
            imageobj(CCobj.PixelIdxList{1,i}) = 0; %Elimino
        else

            puntosqueveo=0; %Compruebo límite inferior

            for pixel=x:x+stats(i).BoundingBox(3)
                if imageobj(y+stats(i).BoundingBox(4)-1, pixel)==1
                    puntosqueveo=puntosqueveo+1;
                end
            end

            if stats(i).BoundingBox(4)/puntosqueveo<7 %Cumple el criterio
                imageobj(CCobj.PixelIdxList{1,i}) = 0; %Elimino
            end
        end
    end
end
```

Recorte de líneas aditivas

Las líneas aditivas son líneas que se comportan de la misma manera que las líneas de pentagrama, y se usan cuando la nota a tocar es más grave o aguda de lo que un pentagrama puede representar con 5 líneas.

El criterio más eficaz para distinguir las corcheas grupales de los objetos restantes es su anchura. A diferencia de ellos, las corcheas grupales no presentan una anchura inferior a 3 *espacios* [*ver Estudio de la simbología musical, pág. 8*]. Sin embargo, esto no es siempre así.

Cuando cualquier otro objeto de los no reconocidos por el árbol de decisión contiene dos núcleos en horizontal atravesados por una línea aditiva [*ver ilustración nº18, en esta pág.]* su anchura se puede confundir con la de una corchea grupal [*ver ilustración nº20, en esta misma pág.].*

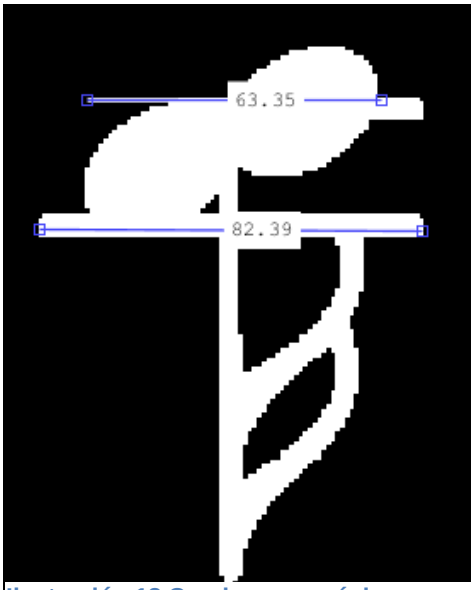


Ilustración 18 Corchea con núcleos en horizontal y línea aditiva

Nótese que, si la figura de la ilustración 18 no tuviera la línea aditiva, su anchura sería menor a 3 *espacios* (75 píxeles en este caso) y podría ser fácilmente descartada como corchea grupal.

Conociendo que las líneas aditivas presentan un grosor equivalente a *espacio* dividido entre 5 (5 píxeles en este caso) *[ver ilustración nº19, en esta pág.],* podemos recortar las líneas aditivas siguiendo el siguiente criterio:

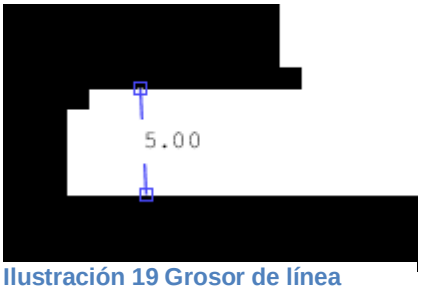


Ilustración 19 Grosor de línea aditiva

- Obtenemos el fragmento de la figura que corresponda con la primera columna de píxeles comenzando por la izquierda
- Consideramos los objetos (píxeles a 1) comprendidos en él y obtenemos el área de cada uno.
- Eliminamos todas las columnas que no contengan objetos de área (en este caso igual a la altura) mayor a $espacio/5$, es decir, columnas que solo contengan línea aditiva.
- Si la columna contiene objetos de área mayor a la mencionada, se termina la operación.
- Repetimos los pasos anteriores por el lado de la derecha.

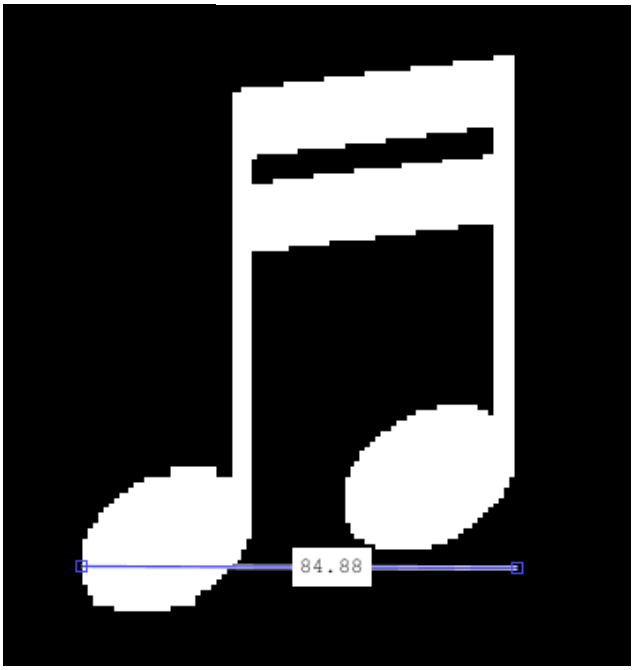


Ilustración 20 Anchura de semicorchea doble

La eliminación de las líneas aditivas por el lado de la izquierda se realiza con el código de a continuación.

```
CCobj = bwconncomp(imageobj);
stats = regionprops(CCobj,'BoundingBox');

 for i=1:CCobj.NumObjects
     for
x=stats(i).BoundingBox(1)+0.5:stats(i).BoundingBox(1)+st
ats(i).BoundingBox(3)-0.5
y=stats(i).BoundingBox(2)+0.5:stats(i).BoundingBox(2)+st
ats(i).BoundingBox(4)-0.5;
         CCLadi = bwconncomp(imageobj(y, x));
         statsLadi=regionprops(CCLadi, 'Area');

         for a=1:size(statsLadi, 1)
             if statsLadi(a).Area>espacio/5
                 nohayLadi=1;
                 break
             else
                 imageobj(y, x)=0;
                 nohayLadi=0;
             end
         end

         if nohayLadi==1
             break
         end
     end
 end
```

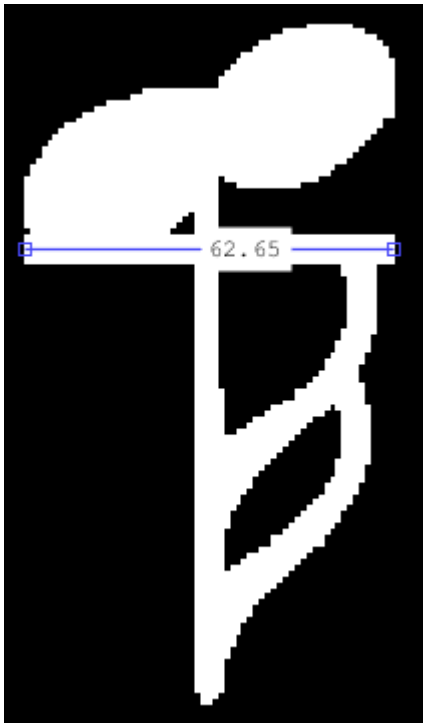


Ilustración 21 Corchea con línea aditiva recortada

Como se puede observar en la ilustración 21, la anchura de este objeto ya se puede distinguir de la de las corcheas grupales.

Distinción y aislamiento de corcheas grupales

Gracias al paso anterior, podemos aislar las corcheas grupales del resto de figuras de forma sencilla con el siguiente código:

```
CCobj = bwconncomp(imageobj);
stats = regionprops(CCobj,'BoundingBox');
CG=zeros(alto, ancho);

for i=1:CCobj.NumObjects
    if stats(i).BoundingBox(3)>3*espacio
        CG(CCobj.PixelIdxList{1,i}) = 1;
        imageobj(CCobj.PixelIdxList{1,i}) = 0;
    end
end
```

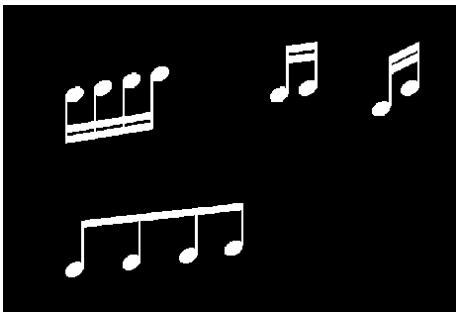


Ilustración 22 Corcheas grupales aisladas

Esto se hace porque la manera elegida de procesar corcheas grupales es bastante particular por sí misma, y complicaría mucho el procedimiento intentar procesar estos objetos a la vez que los demás.

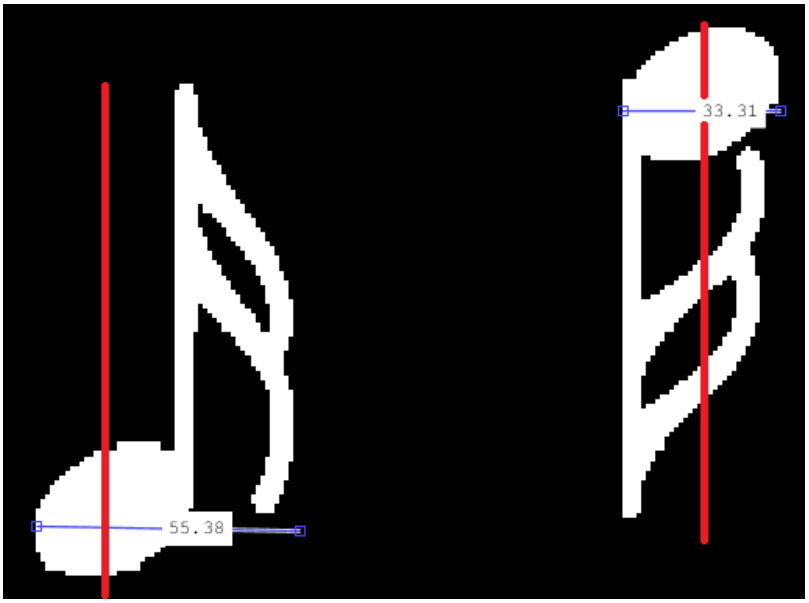
Como se puede comprobar de la partitura maestra [*ver ilustración nº6, pág. 9*], el resultado, observable en la ilustración nº22 es correcto; hemos aislado todas las corcheas grupales.

Seguiremos distinguiendo las figuras entre sí para simplificar el proceso de interpretación.

Distinción y aislamiento de blancas

La diferencia más característica entre las blancas y el resto de figuras (recordemos que, ahora mismo, contenemos en una misma imagen las figuras correspondientes a; blancas, negras, corcheas y semicorcheas individuales) es que su núcleo está vacío. Sin embargo, las blancas son muy susceptibles a fragmentarse al realizar sobre ellas operaciones morfológicas como la erosión [*ver Borrado de plicas, pág. 23*].

La manera en la que distinguiremos los núcleos vacíos de las blancas del resto de ellas, es comprobar si, en el lugar donde se encuentra alguno de sus núcleos (solo basta comprobar uno de ellos), existe una línea ininterrumpida de un tamaño igual o mayor al típico de un núcleo, el cuál es de 1 *espacio* (25 píxeles en este caso).



Dónde se va a encontrar el núcleo de la figura depende de su anchura. Como se puede ver abajo en la ilustración nº22, el núcleo se encuentra en el eje central en los casos donde la anchura de la figura no supera la del núcleo. Cuando la figura resulta ser más ancha (ya sea porque tiene corchete o porque existen dos núcleos en horizontal, como hemos visto en las ilustraciones nº18 y nº21 [*ver págs. 19 y 20*]), el núcleo se encuentra a ¼ de la anchura total, contando desde la izquierda, como se observa en la ilustración de la izquierda.

Ilustración 23 Ubicación de núcleo en figuras de distinta anchura

```
CCobj = bwconncomp(imageobj);
stats = regionprops(CCobj,'BoundingBox');
blancas=zeros(alto, ancho);

for i=1:CCobj.NumObjects
    if stats(i).BoundingBox(3)>=2*espacio
        x=round(stats(i).BoundingBox(1)+0.5+stats(i).BoundingBox(3)/4);
    else
        x=round(stats(i).BoundingBox(1)+0.5+stats(i).BoundingBox(3)/2);
    end

    y=stats(i).BoundingBox(2)+0.5:stats(i).BoundingBox(2)+stats(i).BoundingBox(4)-1;
    CCLadi = bwconncomp(imageobj(y, x));
    statsLadi=regionprops(CCLadi, 'Area');
    nucleonegra=0;

    for a=1:CCLadi.NumObjects
        if statsLadi(a).Area>espacio*3/4 %Aproximamos por seguridad
            nucleonegra=1;
        end
    end
    if nucleonegra==0
        imageobj(CCobj.PixelIdxList{1,i}) = 0;
        blancas(CCobj.PixelIdxList{1,i}) = 1;
    end
end
```

Como se puede ver en el código, se considera que hay un núcleo de negra (relleno) cuando se encuentra una línea mayor a ¾ partes de la altura típica de un núcleo (1 *espacio*). Esto se hace por seguridad, pues es posible que las coordenadas de *x* calculadas no resulten ser exactamente las del centro del núcleo, sino las de un punto próximo. De todas maneras, el trazo de un núcleo de blanca es muy fino y jamás llega a estar cerca del umbral mencionado.

Guardado de los "BoundingBox" de los objetos aislados

Para extraer la información que se necesita de cada figura musical, hemos de aislar nuevamente, para cada una de ellas, el corchete o corchetes (si tiene) y los núcleos, y en todo caso habremos de eliminar las plicas. Para poder recuperar la asociación entre núcleos y corchetes y poder asignar cada uno de estos fragmentos a su objeto original, y aprovechando que hemos terminado de aislar figuras entre sí en función de su tipo (corcheas grupales, redondas y figuras de núcleo relleno), guardaremos la información relativa al "BoundingBox" de cada figura antes de seguir operando con ellas.

El "BoundingBox" de una figura es el rectángulo de menor tamaño en el que esta figura cabe. Un ejemplo de ello se puede observar en la ilustración de la derecha.

Ilustración 24 *BoundingBox* de un objeto

```
CCnegras = bwconncomp(imageobj); %Nucleos negros
negrasStats=regionprops(CCnegras, 'BoundingBox');

CCCG = bwconncomp(CG); %Corcheas grupales
CGStats=regionprops(CCCG, 'BoundingBox');

CCblancas = bwconncomp(blancas); %blancas
blancaStats=regionprops(CCblancas, 'BoundingBox');
```

Borrado de plicas

Para conocer la altura de los núcleos y así obtener su altura, necesitaremos aislarlos del resto de píxeles. Borraremos las plicas de todos los elementos, excepto las blancas, de la misma manera: mediante operaciones morfológicas de erosión.

La erosión es una operación morfológica que superpone a una imagen (generalmente binaria) un elemento estructurante (una matriz de unos de una forma concreta), de manera que el centro del elemento estructurante se sitúe en cada píxel de la imagen original. Se eliminan todos aquellos píxeles de la imagen donde el elemento estructurante no se haya visto totalmente contenido por los píxeles activos (en 1) de la imagen original. La ilustración de la derecha es un ejemplo donde el elemento estructurante es un disco [4].

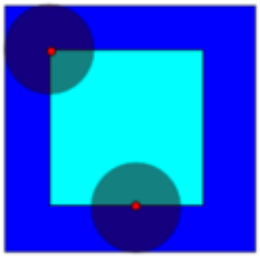


Ilustración 25 Ejemplo de erosión morfológica

```
nucleos=imerode(imageobj, [0 0 0; 1 1 1; 0 0 0]);
nucleos=imerode(nucleos, [0 0 0 ; 1 1 1; 0 0 0]);
nucleos=imerode(nucleos, [0 0 0 ; 1 1 1; 0 0 0]);
```

Los comandos utilizados, en el caso de las figuras de núcleo relleno (recordamos que son negras, corcheas y semicorcheas individuales) son los siguientes:

Ilustración 26 Efecto del borrado de plicas en corcheas grupales

Se realizarán los mismos pasos para las corcheas grupales:

```
nucleosCG=imerode(CG, [0 0 0; 1 1 1; 0 0 0]);
nucleosCG=imerode(nucleosCG, [0 0 0 ; 1 1 1; 0 0 0]);
nucleosCG=imerode(nucleosCG, [0 0 0 ; 1 1 1; 0 0 0]);
```

Como hemos dicho anteriormente, no se puede realizar los mismos pasos para el caso de las blancas si pretendemos que los núcleos no se fragmenten [ver Distinción y aislamiento de blancas, pág. 21].

Ilustración 27 Fragmento del resultado de borrar plicas a la imagen de las blancas

Para ello, volveremos a utilizar la transformada de Hough para detectar la plica. Tendremos en cuenta que:

- El grosor de una plica es de 4 píxeles. Aproximando y añadiendo un pequeño margen, podemos decir que no supera $1/5$ del espacio entre dos líneas consecutivas del pentagrama. Entonces, en el grosor de una plica hay $espacio/5$ líneas de 1 píxel de grosor. Teniendo en cuenta que hay n objetos, tendríamos $n * espacio/5$ líneas que conformarían todas las plicas de la imagen.

- Una plica mide, idealmente, 3 *espacios*. Añadiremos un margen y nos aseguraremos suponiendo que la plica jamás mide menos de 2 *espacios*.

- La plica (o las líneas de 1 píxel que la conforma) siempre es perpendicular al eje *x*. Esto equivale a un ángulo *Theta* igual a 0.

```
CCredondasnucleos = bwconncomp(blancas);
[H,T,R] = hough(blancas, 'theta', 0);
P  = houghpeaks(H,CCredondasnucleos.NumObjects*espacio/5,'NHoodSize', [1 1]);
linesR = houghlines(blancas,T,R,P, 'MinLength', 2*espacio);

for i=1:size(linesR, 2)
    x=linesR(i).point1(1);
    y=linesR(i).point1(2):linesR(i).point2(2);
    blancas(y,x)=0;
end
```

El resultado se observa en la ilustración nº27 en esta misma página. Como se puede ver, la posición de los núcleos y su altura (útil para determinar el número de núcleos sin operar sobre la imagen) es fácilmente conocible ahora.

Además, es interesante observar como este método separa los núcleos unidos de manera horizontal, lo cual trataremos de conseguir con el resto de figuras.

Borrado de líneas aditivas

Tras el borrado de plicas, es posible que nos encontramos con el caso de la imagen de la derecha. Al borrar el segmento que unía la línea aditiva, se nos queda un rectángulo paralelo al eje *x*. Dado que pretendemos mantener núcleos y corchetes en una misma imagen, y aprovechando que estos elementos son fácilmente distinguibles, los eliminaremos de la imagen. A continuación, el código usado en el caso de las figuras de núcleo relleno

.

```
CCnucleos = bwconncomp(nucleos);
Nucleostats = regionprops(CCnucleos,'BoundingBox',
'Orientation');

for i=1:CCnucleos.NumObjects
    if Nucleostats(i).Orientation==0
        nucleos(CCnucleos.PixelIdxList{1,i}) = 0;
    end
end
```

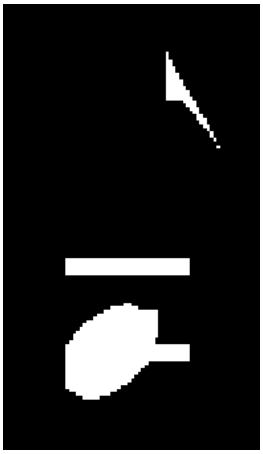


Ilustración 28 Línea aditiva sobrante en una corchea

Para el caso de las blancas y las corcheas grupales, el procedimiento será muy similar, salvando ligeras particularidades de cada caso.

Aislamiento de corchetes

En este punto, tan solo deberíamos contener núcleos y corchetes en todas las imágenes. Para simplificar todavía más los pasos posteriores, filtraremos todos los objetos obtenidos y aislaremos los corchetes. Tengamos en cuenta que las figuras de blanca ya están aisladas y distinguidas, por lo que este paso nos permitirá distinguir negras (no tiene corchete), corcheas y semicorcheas individuales.

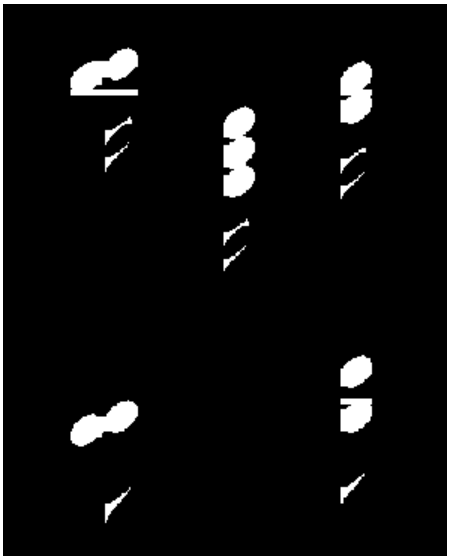


Ilustración 29 Núcleos y corchetes aislados en corcheas y semicorcheas

Comenzaremos por el caso de las corcheas. El criterio a usar para aislar los corchetes es muy sencillo. Nos basaremos en el hecho de que un núcleo no puede medir menos de *espacio* en ninguno de sus ejes, lo cual no se cumple en el caso de los corchetes. Para ello, haremos uso del parámetro "Minor Axis Length"'.

"Minor Axis Length" es una propiedad de un objeto que MathWorks define como la "longitud (en píxeles) del eje menor de la elipse que tiene los mismos segundos momentos centrales normalizados que la región, devuelto como un escalar." [5]

```
CCnucleos=bwconncomp(nucleos);
Nucleostats = regionprops(CCnucleos,'MinorAxisLength',
'Area');
corchetes=zeros(alto, ancho);

for i=1:CCnucleos.NumObjects
    if Nucleostats(i).Area<espacio
        nucleos(CCnucleos.PixelIdxList{1,i}) = 0;
    elseif Nucleostats(i).MinorAxisLength<espacio*3/4
        corchetes(CCnucleos.PixelIdxList{1,i}) = 1;
        nucleos(CCnucleos.PixelIdxList{1,i}) = 0;
    end
```

Como se puede ver en el código, se ha decidido incluir un filtro de objetos pequeños. Aunque no es común, es posible que la erosión realizada durante el borrado de plicas [ver Borrado de plicas, pág. 22] deje pequeños grupos de píxeles sueltos que pueden malinterpretarse como un corchete extra. Además, se ha incluido un factor de ¾ multiplicando a *espacio* que sirve de margen y evita efectos no deseados.

Separación de núcleos

Ahora que tenemos los núcleos de las negras, corcheas y semicorcheas individuales aislados, podemos observar un fenómeno del que hemos hablado anteriormente [*ver Borrado de plicas, pág. 23]*. El hecho de que dos núcleos se sitúen en horizontal puede complicar los cálculos de altura. En primer lugar, habría que comprobar si el núcleo que estemos comprobando es uno de estos casos, y si lo es, comprobar su altura tomando dos valores de *x* distintos, uno en cada columna de núcleos. Y, aunque no es común encontrar una figura musical con dos columnas de núcleos, no es deseable que la lectura de una partitura fallase solo por encontrarse con un caso excepcional.

Así pues, se ha dado con una solución más simple. Se comprobará cada núcleo, y si su anchura encaja con la típica de dos núcleos en horizontal (mayor a 2 *espacios*) se añadirá una columna de ceros en la mitad del objeto, separando ambas columnas. De esta manera, podremos tratarlas por separado, permitiendo crear un algoritmo más general, pues habrá desaparecido el problema mencionado.

Ilustración 30 Núcleos horizontales separados

En la ilustración nº30 de la página anterior, podemos ver un fragmento del resultado de este paso. Sin embargo, se puede observar cómo, fruto de la separación de los núcleos, un fragmento de línea aditiva queda expuesto. Por esta razón, es recomendable volver a repetir los pasos enunciados en *Borrado de las líneas aditivas [ver pág. 23]*.

```
Nucleostats = regionprops(nucleos,'BoundingBox');
for i=1:size(Nucleostats, 1)
    if Nucleostats(i).BoundingBox(3)>2*espacio

x=round(Nucleostats(i).BoundingBox(1)+Nucleostats(i).BoundingBox(3)/2);
        for y=Nucleostats(i).BoundingBox(2)-
0.5:Nucleostats(i).BoundingBox(2)+Nucleostats(i).BoundingBox(4)+0.5
            nucleos(y, x)=0;
        end
    end
end
```

Lectura de notas

Estamos en disposición de interpretar los núcleos y los corchetes y completar la *Partitura codificada*. El procedimiento general será el siguiente:

- Conociendo las coordenadas del "BoundingBox" del objeto original, buscamos en la imagen de los núcleos aquellos que se encuentren en esa área, y lo mismo para los corchetes.
- Procesando cada uno de los núcleos detectados, se comprueba el número de corchetes que afecta a la duración de la nota.
- Se calculan las coordenadas *x* e *y* de cada núcleo y se agrega la información al vector *PC* (*Partitura Codificada*).

Sin embargo, cada caso tiene sus particularidades que detallaremos a continuación. Recordemos los parámetros que hay que agregar a la matriz de la *Partitura Codificada*:

- *N (Número de nota):* por ahora introduciremos un 0.
- *T* (tipo): duración en pulsos. Si es blanca, será 2; si es negra, 1. En los demás casos dependerá del número de corchetes que afecten al núcleo.
- *X*: coordenadas en el eje *x* del objeto al que pertenece.
- *Y*: coordenadas en el eje *y* del núcleo.

Lectura de notas (blancas)

Este es el caso más fácil. La duración la conocemos pues sabemos que se trata de blancas. Solo nos queda calcular las coordenadas de cada núcleo. Sin embargo, hay que tener en cuenta el fenómeno presente en la ilustración de la derecha: cuando dos o más núcleos se sitúan próximos (más concretamente a un intervalo de tercera) los núcleos se conectan, de manera que MatLab los interpreta como un solo objeto. Aun así, podemos conocer cuántos núcleos están presentes en realidad teniendo en cuenta que la altura típica de un núcleo, como se ha repetido en este documento, es *espacio.*

Ilustración 31 Núcleos conectados

Ilustración 32 Las líneas en rojo señalan la posición central de núcleos conectados

Las coordenadas en el eje *y* de cada núcleo se puede calcular con la fórmula de abajo, donde:

- *BB(1)* es la coordenada en *x* de la esquina superior izquierda del "BoundingBox" del objeto
- *Altura* es la altura del objeto
- *Nº núcleos* es el número de núcleos totales del objeto, conocido por la altura del objeto dividido espacio
- *D* es el núcleo que se está procesando: será 1 cuando se procese el primero, 2 cuando se procese el segundo…

$$BB(1) + Altura * \frac{d * 2 - 1}{nº\ núcleos * 2}$$

105	106	107
0	0	0
2	2	2
2549	2549	2549
850.50	876.50	902.50

De la imagen de la izquierda se puede observar cómo se ha procesado correctamente el caso mencionado. Las tres columnas de la matriz contienen la misma coordenada *x* pues forman parte del mismo objeto, pero distinta *y*.

Ilustración 33 Datos en PC (Partitura codificada)

```
%Lectura de blancas
CCredondasnucleos = bwconncomp(blancas);
redondasnucleostats = regionprops(CCredondasnucleos,'BoundingBox', 'Centroid');

for i=1:CCblancas.NumObjects
    nucleosdelobjeto=[];
    %Que nucleos forman parte del objeto
    for a=1:size(redondasnucleostats, 1)
        if redondasnucleostats(a).Centroid(1)>=blancaStats(i).BoundingBox(1)-
0.5+blancaStats(i).BoundingBox(3)
            %Me he pasado, no van a haber más nucleos pertenecientes
            break
        elseif redondasnucleostats(a).Centroid(1)>blancaStats(i).BoundingBox(1) &&
redondasnucleostats(a).Centroid(1)<blancaStats(i).BoundingBox(1)+blancaStats(i).Bo
undingBox(3) %la x esta dentro del BB del objeto original
            if redondasnucleostats(a).Centroid(2)>blancaStats(i).BoundingBox(2) &&
redondasnucleostats(a).Centroid(2)<blancaStats(i).BoundingBox(2)+blancaStats(i).Bo
undingBox(4) %la y esta dentro del BB del objeto original
                nucleosdelobjeto(end+1)=a;
            end

        end
    end

    %Introducimos los datos en PC
    for c=1:length(nucleosdelobjeto)


numbolitas=round(redondasnucleostats(nucleosdelobjeto(c)).BoundingBox(4)/espacio);

        for d=1:2:numbolitas*2-1
            PC(2, end+1)=2; %T
            PC(3,
end)=blancaStats(i).BoundingBox(1)+blancaStats(i).BoundingBox(3)/2; %X
            PC(4,
end)=redondasnucleostats(nucleosdelobjeto(c)).BoundingBox(2)+redondasnucleostats(n
ucleosdelobjeto(c)).BoundingBox(4)*d/(numbolitas*2); %Y
        end
    end
end
```

Lectura de notas (negras y corcheas y semicorcheas individuales)

Este caso es muy similar al anterior. De hecho, la única diferencia son los corchetes. Como decíamos en el método general de lectura [*ver Lectura de notas, pág. 25]*, para conocer el tipo (es decir, la duración en pulsos) hay que conocer el número de corchetes que afectan a los núcleos.

La duración en pulsos (*T*) de este tipo de figuras se calcula como:

$$T = 2^{-nº\ corchetes}$$

Como se aprecia en la fórmula, la duración de la nota es menor conforme más corchetes actúen sobre ella.

Lectura de notas (corcheas grupales)

Aunque en esta versión de la aplicación se ha priorizado la lectura correcta de semicorcheas, y no figuras de menor duración, se ha buscado un algoritmo independiente de la subdivisión empleada, de manera que se puedan reconocer fusas o semifusas, aunque estas figuras no entran en el enfoque de este proyecto.

La diferencia respecto el caso anterior es que aquí no todos los núcleos se ven afectados por todos los corchetes. Solo los corchetes que estén conectados a un núcleo mediante su plica afectarán a dicho núcleo. Sin embargo, hemos borrado las plicas, así que nos las ingeniaremos de otra manera.

La idea es la siguiente: generalmente el corchete que se encuentra por arriba de un núcleo es el que afecta al mismo. Si esto fuera así, lo lógico sería escanear la imagen para la *x* en la que se encuentra el núcleo, y contar el número de corchetes con los que me encuentro. Sin embargo, en muchas ocasiones, los corchetes que afectan al núcleo se encuentran a la *x* a la que se encontraría si al núcleo se le hubiera dado la vuelta, como colocando un espejo en uno de sus lados.

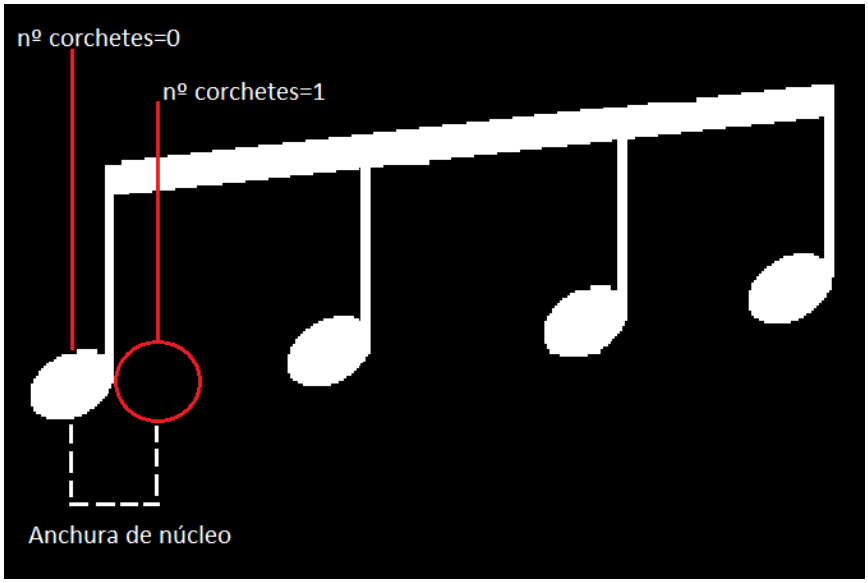


Ilustración 34 Posición de un núcleo para detectar los corchetes que actúan sobre él

Además, para conocer a qué lado colocar el 'espejo' en estos casos, hay que comprobar a qué lado queda el corchete más corto, pues siempre irá colocado hacia el lado donde quede este. Esto es así sin importar el número de corchetes. Compruebe la ilustración nº33 para entender mejor el ejemplo del espejo.

Se hará lo siguiente: dado un núcleo se calcularán una serie de variables. A partir de esas variables, se calculará la coordenada *x* mediante la cual se deducirá el número de corchetes que afectan al núcleo dado, mediante un escaneo en la imagen de los corchetes. Las variables son las siguientes:

- *corchetemaslargo* y *corchetemascorto*: como su nombre indica, se almacenará el corchete (más bien su número de corchete, según su aparición) más largo y más corto.
- *cubierto:* esta variable será 1 cuando el núcleo quede "cubierto" por el corchete más corto. Si no, *cubierto* valdrá 0. No depende de la orientación de la figura, sino que calcula si el núcleo queda en el rango que delimitan los extremos del corchete más corto.
- *lado:* lado indica si el núcleo en cuestión está a la derecha o a la izquierda del corchete más corto. Si está a la izquierda, lado valdrá 0. De otra manera, valdrá 1.
- *normal:* hace referencia a la orientación del objeto. Si vale 1, indica que los corchetes quedan por arriba. Si vale 0, serán los núcleos los que queden por arriba. Esto es importante considerarlo pues cambia de lado los núcleos (a lo espejo).

La variable *lado* tan solo será relevante cuando *cubierto* valga 0. La razón de esto es evidente, el núcleo ni está a la derecha ni a la izquierda de un corchete cuando está debajo o arriba de él. Así pues, la coordenada *x* resulta ser, si *cubierto* vale 0:

$$x = x\ central\ del\ núcleo + (lado - normal) * (-anchura\ del\ núcleo)$$

Si cubierto vale 1 quiere decir que el núcleo se encuentra en la misma vertical que el corchete más corto. Si contamos los corchetes con los que nos encontramos en la coordenada en la que se encuentra el núcleo, habremos obtenido el número correcto de corchetes que actúan sobre tal núcleo.

El resto de cálculos son idénticos a los realizados para las demás figuras.

Ordenación de PC (Partitura codificada) I

Realizados los pasos anteriores, hemos debido de codificar todas las notas de la partitura. Llega el momento de ordenar estos valores y preparar la matriz para los últimos pasos.

Al igual que hace un músico, que lee las notas de izquierda a derecha, desde el primer pentagrama hasta el último, terminaremos ordenando las notas de esa misma manera para que se interpreten en ese orden. Sin embargo, por ahora tan solo ordenaremos de izquierda a derecha.

```
%Ordeno valores de PC en función de X
numnotas=size(PC, 2);
X=zeros(1, numnotas);

for i=1:numnotas %Almaceno valores de x
    X(i)=PC(3, i);
end

[~,I]=sort(X); %Ordeno valores de x
ParCod=zeros(4, numnotas); %Creo una nueva matriz para PC

%Almaceno en la nueva matriz los valores ordenados
for i=1:numnotas
    ParCod(:, i)=PC(:, I(i));
end
```

Eliminación de objetos no interpretables

En una partitura pueden aparecer símbolos irrelevantes a la melodía, aunque contengan cierta información musical. Este es el caso de, por ejemplo, el número de compás y la llave que asocia los dos pentagramas como uno solo, señalados en la ilustración de la derecha. Como se puede imaginar, estos objetos también resultan inútiles para nuestro propósito de reproducir la melodía.

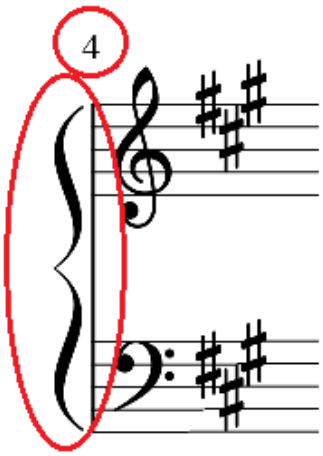


Ilustración 35 Ejemplos de objetos no interpretables

```
i=1;
while 1
    if ParCod(3,
i)<=Llaves(2, 1)
        i=i+1;
    else
        break
    end
end

ParCod=ParCod(:, i:end);
numnotas=size(ParCod, 2);
```

Sin embargo, es muy fácil detectar cuándo un objeto, del cual ya hemos introducido su información en *PC*, se trata de uno de estos casos: para ello solo hay que comprobar si se encuentran anteriores a la primera llave, ya sea de Sol o de Fa.

Puntillos

Aprovechando el nuevo estado de la *Partitura codificada*, le aplicaremos a la misma el efecto de los puntillos ya detectados.

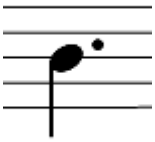


Ilustración 36 Puntillo

Para ello, tengamos claro cómo afecta a las notas. El puntillo afecta a las notas que se sitúen inmediatamente a su izquierda, y prácticamente en su misma vertical. Esto último no se cumple estrictamente en los casos como el de la ilustración nº36, en el que, dado que el centro del núcleo se posa sobre la línea del pentagrama, sería difícil reconocer a simple vista el puntillo, que también caería en esa posición. A las notas a las que afecta, les multiplica su duración actual por un factor de $\frac{3}{2}$, es decir, aumenta la mitad de su duración.

```
%Puntillos
for i=1:numnotas
    for b=1:size(Puntillos, 2)
        if ParCod(4, i)+semisalto>Puntillos(2, b) && ParCod(4, i)-
espacio<Puntillos(2, b)%Coincide en Y
            if Puntillos(1, b)-ParCod(3, i)>0 && Puntillos(1, b)-ParCod(3,
i)<espacio*2   %Coincide en x
                ParCod(2, i)=ParCod(2, i)*(3/2);
                break
            end
        end
    end
end
```

Llaves

Para agregar un *Número de nota* a las notas que no son silencios, necesitamos conocer la última llave que ha aparecido en el pentagrama donde se encuentran, pues de ello depende que las líneas del pentagrama signifiquen una nota u otra.

Para facilitar la asignación del *Número de nota*, y recordando que los parámetros almacenados de las llaves detectadas son el tipo de llave y las coordenadas en *x* y en *y*, traduciremos la coordenada vertical al número de pentagrama en el que se encuentra, conociendo el rango de los pentagramas [*ver Rango de pentagramas, pág. 14*].

```
%Llaves; paso de Y a P
for i=1:size(Llaves, 2)
    for p=1:size(rangopentagramas, 1)-1 %P
        if Llaves(3, i)>=rangopentagramas(p) && Llaves(3,
i)<=rangopentagramas(p+1)
            Llaves(3, i)=p;
        end
    end
end
```

Partitura codificada, obtención de P y N

Teniendo las llaves, podemos saber el pentagrama y el *Número de nota* de las notas almacenadas en la *Partitura codificada*, en ese orden.

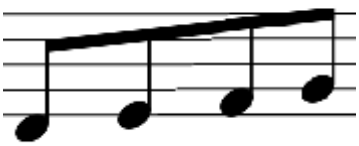


Ilustración 37 Notas consecutivamente a un semisalto de distancia

Como se ha dicho, el *Número de nota* depende de la posición vertical del núcleo en el pentagrama al que pertenece. Cuando se trate de una llave de Sol, la cual recordamos que asigna a la cuarta línea del pentagrama (contando desde arriba) la nota Sol, calcularemos el *Número de nota N* a partir del número de *semiespacios* [*ver ilustración 37, en esta pág.]* que existen entre el núcleo y el punto que equidista de la tercera y cuarta línea, es

decir, la posición correspondiente al *La 4*. Recordemos también que hemos asignado al *La 4* el *Número de nota* 0, por lo que un número de *semiespacios* respecto la posición del *La 4* de 0 equivaldría al mismo *Número de nota*.

Sin embargo, el *Número de nota* no equivale al número de *semiespacios*. Esto es porque en la escala natural (es decir, sin tener en cuenta las alteraciones), no todas las notas están a la misma cantidad de tonos de distancia. De hecho, entre el *Mi* y el *Fa* y entre el *Si* y el *Do* solo existe un semitono de distancia. De ahí viene la creación del vector *Escalanatural* [*ver Parámetros iniciales, pág. 11*], y es que, gracias a este vector, podemos asociar el número de *semiespacios* respecto la posición del *La* a un *Número de nota,* pues este número de *semiespacios* equivale a la posición en *Escalanatural* en la que se encuentra el *Número de nota* correspondiente a la nota en cuestión.

Para cada nota en la matriz de la *Partitura codificada*, calcularemos su pentagrama, y con esta información, su *Número de nota.* Sabemos que la nota es de cierto pentagrama cuando su coordenada en *y* pertenece al rango vertical que abarca ese pentagrama.

Respecto al *Número de nota*, hay que saber que, para el caso de la llave de *Fa*, el procedimiento será el mismo que para la de *Sol*, pero tomando la posición que queda 12 casillas a la izquierda.

```
%Partitura codificada; paso de Y a P y obtención de N
for i=1:size(ParCod, 2)

    y=ParCod(4, i);

    for p=1:size(rangopentagramas, 1)-1 %P
        if y>=rangopentagramas(p) && y<rangopentagramas(p+1)
            break
        end
    end

    if ParCod(1, i)~=99
        nsemi=round((La4(p)-y)/semiespacio);
        for b=size(Llaves, 2):-1:1
            if Llaves(3, b)==p
                if Llaves(1, b)==1 %Llave de fa
                    ParCod(1, i)=EscalaNatural(nsemi+8);
                else %Llave de sol
                    ParCod(1, i)=EscalaNatural(nsemi+20);
                end
            end
        end
    end

    ParCod(4, i)=p;
end
```

Ordenación de ParCod (Partitura codificada) II

Ordenaremos ahora la *Partitura codificada* por pentagrama, consiguiendo un orden de lectura pleno. Para ello, introduciremos en otra matriz, los elementos de *ParCod* (*Partitura codificada*) que vayan coincidiendo con el número de pentagrama que se plantee. Para mantener el orden de izquierda a derecha logrado en pasos anteriores [ver Ordenación de PC (Partitura codificada) I, pág. 28], comprobaremos cada elemento en ese mismo orden.

```
PartCodi=zeros(5, numnotas);
a=1;

for p=1:numpent
    for i=1:numnotas
        if ParCod(4, i)==p
            PartCodi(1:4, a)=ParCod(:, i);
            a=a+1;
        end
    end
end
```

Se observa lo conseguido en la ilustración de abajo. Observamos *Números de notas* (1ª fila) y números de pentagrama (4ª fila).

	1	2	3	4	5	6
1	0	99	12	99	14	99
2	0.25000	0.25000	0.25000	0.25000	0.25000	0.25000
3	530.50	579.10	643	701.10	764	822.10
4	1	1	1	1	1	1
5	0	0	0	0	0	0

Ilustración 38 Resultados en *PartCodi* de obtener *N* y *P*

Agregación de Simul

Como hemos dicho anteriormente, es interesante la inclusión de un parámetro que indique la simultaneidad de las notas [*ver Estrategia (Partitura Codificada), pág. 10*].

En una partitura se sabe las notas que van a ser tocadas a la vez a simple vista, pues estas están alineadas verticalmente.

El criterio es simple; si la nota que estamos comprobando está a un valor de *x* igual o prácticamente igual al de la siguiente, se le coloca un 0 en la última fila de *Partitura codificada*. En caso contrario, se le da valor 1. Sin embargo, durante la inclusión de este parámetro sólo tiene en cuenta las figuras de un mismo pentagrama. Esto se hace para resolver el "Problema del silencio de redonda" [*ver ilustración nº 40, en esta pág.]*

```
%Agregación de simul
for i=1:numnotas
    if i==numnotas ||  PartCodi(4, i)~=PartCodi(4, i+1)
        PartCodi(5, i)=1;
    elseif PartCodi(3, i+1)-PartCodi(3, i)>espacio*3/2
        PartCodi(5, i)=1;
    end
end
```

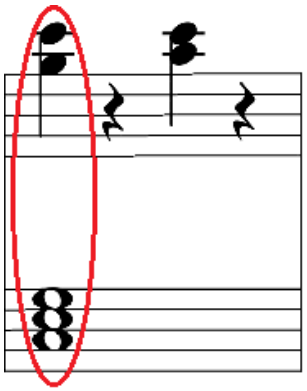
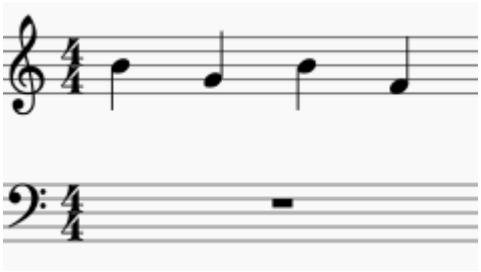


Ilustración 39 Notas simultáneas, alineadas verticalmente



En la ilustración de la izquierda se puede comprobar que el silencio de redonda ha de comenzar al mismo tiempo que la primera negra del pentagrama superior. Sin embargo, estas dos figuras no están alineadas en el eje *x.*

Ilustración 40 Problema del silencio de redonda

Interpretación y almacenamiento de alteraciones

Recordemos que durante el proceso de reconocimiento de objetos [*ver ilustración 5 "Árbol de decisión", pág. 8*] aislamos las alteraciones en una misma imagen. Sin embargo, es necesario distinguir cada uno de los tres tipos de alteraciones y conocer la posición de cada figura.

El proceso utilizado para ello viene motivado por la siguiente idea: si observamos los tres tipos de alteraciones, podemos comprobar que la principal característica que las permite distinguirse entre sí es el número de extremidades. El becuadro dispone de 2, el bemol de 1 y el sostenido de 8.

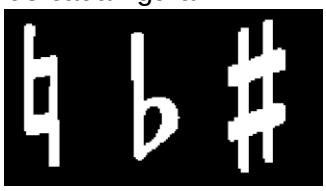


Ilustración 41 En orden Becuadro, bemol y sostenido

La manera en la que se ha llevado a cabo la idea anterior es realizando dos operaciones morfológicas a la imagen de las alteraciones *Alt:* primero *'skel'*, de "esqueleto" en español, y seguidamente '*endpoints*', "puntos finales". La primera comprime las figuras hasta convertirlas en líneas de 1 píxel de grosor unidas por un solo píxel y la segunda conserva únicamente los puntos de los extremos. Entonces, se puede identificar cada tipo de alteración por el número de puntos que se encuentren en el área donde se encontró la figura por primera vez.

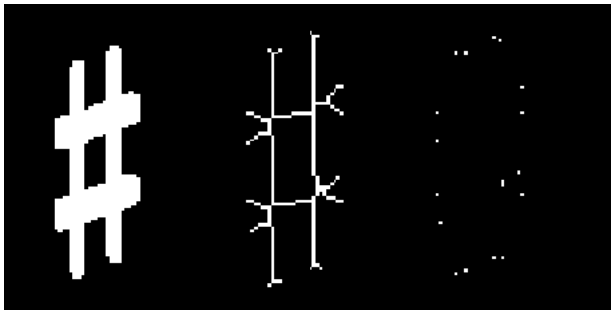


Ilustración 42 Sostenido tras *'skel'* y *'endpoints'*

```
Alt = bwmorph(Alt,'skel', Inf);
AltCC = bwconncomp(Alt);
Altstats = regionprops(AltCC,'BoundingBox', 'Centroid');
altend=bwmorph(Alt, 'endpoints');

for i=1:AltCC.NumObjects
    puntos=0;
    Alteraciones(3, end+1)=Altstats(i).Centroid(1); %X
    Alteraciones(4, end)=Altstats(i).Centroid(2);
    for
x=Altstats(i).BoundingBox(1)+0.5:Altstats(i).BoundingBox(1)+Altstats(i).Boun
dingBox(3)-0.5
        for
y=Altstats(i).BoundingBox(2)+0.5:Altstats(i).BoundingBox(2)+Altstats(i).Boun
dingBox(4)-0.5
            if altend(y, x)==1
                puntos=puntos+1;
            end
        end
    end

    if puntos>=2 && puntos<=4 %Bemol
        tipo=-1;
        Alteraciones(4,
end)=Altstats(i).BoundingBox(2)+Altstats(i).BoundingBox(4)*3/4; %Reajustamos
Y
    elseif puntos>=5 && puntos<=12 %Becuadro
        tipo=0;
    elseif puntos>=13 %Sostenido
        tipo=1;
    end

    Alteraciones(2, end)=tipo;
end
```

Las alteraciones se identifican en la matriz *Alteraciones* por el número de semitonos con el que afectarán a las notas dentro de su rango: 1 para el sostenido y -1 para el bemol. El becuadro se almacena con un 0, pues cancelará el efecto de las dos anteriores como se verá. (insertar referencia)

De las figuras originales se almacenan las coordenadas en *x* e *y*. Sin embargo, hay que considerar que, a diferencia del resto de figuras, el punto que nos interesa del bemol no es su centro exacto, sino aproximadamente el punto que queda a ¾ de su altura, contando desde arriba. Esto se debe a que es la única figura asimétrica.

Por último destacar que el número de puntos que se detectan tras las dos operaciones morfológicas no suele variar. Se obtienen 6, 3 y 18 para los casos del becuadro, bemol y sostenido respectivamente.

Alteraciones, obtención de P y N

Alteraciones

4x21 double

	1	2	3	4
1	12	-12	12	-12
2	1	1	1	1
3	324.27	324.27	324.27	324.27
4	1	2	3	4

Ilustración 43 Resultado de obtener *N* y *P* en *Alteraciones*

Al igual que se ha hecho con la *Partitura codificada* y para facilitar pasos posteriores, se obtendrá el *Número de nota* y el número de pentagrama de cada alteración.

El código usado en este proceso es prácticamente idéntico al usado durante la obtención de P y N de la partitura codificada [ver pág. 31], por lo que se omitirá.

Ordenación de Alteraciones

Por la misma razón que el caso anterior, ordenaremos los elementos de *Alteraciones* de manera que queden, de menor a mayor, primero por coordenada en *x* y luego por pentagrama.

Es interesante observar que, dado que la detección de objetos durante el árbol de decisión [*ver Estudio de la simbología musical, pág. 8*] se realiza de izquierda a derecha, los elementos ya se tienen ordenados por coordenada en *x*.

La ordenación se realiza de manera prácticamente idéntica a la usada con la *partitura codificada* [*ver Ordenación de ParCod (Partitura codificada) II, pág. 30*], por lo que se omitirá el código.

Distinción de armadura y alteraciones accidentales

En música, “armadura” hace referencia a un conjunto de alteraciones que afectan de manera general a todas las notas que continúen. Estas se colocan generalmente al inicio de una pieza o de sección, y suele ir acompañada de una llave (como la de Sol o la de Fa). En concreto, se sitúan sobre su derecha.

Sin embargo, también es común el uso de alteraciones accidentales, las cuáles afectan a las notas del mismo compás, únicamente. Sabiendo esto, el criterio usado para distinguir qué alteraciones forman parte de la armadura y cuáles no, se basa en los siguientes principios:

- Una alteración que esté **próxima** a una llave, forma parte de la armadura.
- Si una alteración está **muy próxima** a una alteración que forme parte de la armadura, esta también forma parte de la armadura.

Mediante observación, se puede comprobar que un umbral apropiado para definir qué es **próximo** y **muy próximo**, es respectivamente, la distancia equivalente a 3 *espacios* y la equivalente a 2 *espacios.*

```
AltAcc=[];
AltArm=[];

for i=1:size(AlteracionesP, 2)
    HaSidoArm=0;
    for a=size(Llaves, 2):-1:1 %busco ultima llave del mismo P
        if Llaves(3, a)==AlteracionesP(4, i) && Llaves(2,
a)<AlteracionesP(3, i) %La llave coincide en p
            if AlteracionesP(3, i)-Llaves(2, a)<espacio*3+semiespacio %La
alteracion esta proxima a la llave encontrada
                AltArm(:, end+1)=AlteracionesP(:, i); %Es armadura
                HaSidoArm=1;
            end
        end
    end
    if HaSidoArm==0
        if AlteracionesP(3, i)-AltArm(3, end)<espacio*2
        AltArm(:, end+1)=AlteracionesP(:, i); %Es armadura
        else
        AltAcc(:, end+1)=AlteracionesP(:, i); %Es accidental
        end
    end
end
```

Líneas divisorias, obtención de P

Para facilitar el proceso donde se tiene en cuenta el efecto de las alteraciones, y aprovechando que hemos almacenado la coordenada en *y* central de cada línea divisoria detectada en la partitura, convertiremos esta información en el número de pentagrama en el que se encuentra la línea.

Es importante tener en cuenta lo siguiente si se trata, como es el caso, de mantener cierta compatibilidad con partituras de otros instrumentos. Cuando el pentagrama es doble, por ejemplo, en el caso de partituras para piano, cada línea divisoria tiene su efecto en dos pentagramas a la vez. Sin embargo, esto no sucede en el resto de casos, por ejemplo, en el caso de partituras para flauta dulce.

Así pues, el algoritmo usado para este paso está basado en lo siguiente: Si el pentagrama es doble, cada línea divisoria representa dos de ellas, donde:

- Ambas coexisten en la misma coordenada en x.
- La primera pertenece al pentagrama impar i y la segunda al pentagrama $i + 1$.

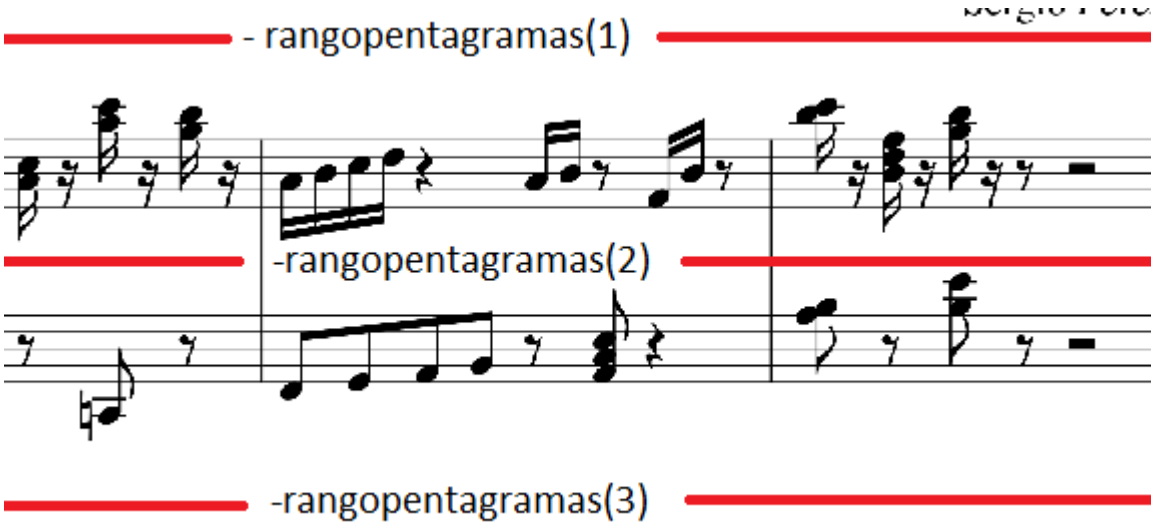


Ilustración 44 Relación entre líneas divisorias y *rangopentagramas*

```
%Lineas divisorias; paso de Y a P
if PentagramaDoble==1 %Partitura para piano
    LineasDivisorias2=zeros(2, size(LineasDivisorias, 2)*2);
    for i=1:size(LineasDivisorias, 2)
        for p=1:numpent/2
            if LineasDivisorias(2, i)>rangopentagramas(2*p-1) &&
LineasDivisorias(2, i)<rangopentagramas(2*p+1)
                LineasDivisorias2(1, 2*i-1)=LineasDivisorias(1, i); %X
                LineasDivisorias2(2, 2*i-1)=2*p-1; %Pentagrama impar
                LineasDivisorias2(1, 2*i)=LineasDivisorias(1, i); %X
                LineasDivisorias2(2, 2*i)=2*p; %Pentagrama par
                break
            end
        end
    end
else
    LineasDivisorias2=zeros(2, size(LineasDivisorias, 2));
    for i=1:size(LineasDivisorias, 2)
        for p=1:numpent
            if LineasDivisorias(2, i)>rangopentagramas(p) &&
LineasDivisorias(2, i)<rangopentagramas(p+1)
                LineasDivisorias2(2, i)=p;
                break
            end
        end
    end
end
```

Contribución de la armadura

Las primeras alteraciones de las cuáles tendremos que considerar primero su aplicación son aquellas que pertenecen a la armadura, dado que su efecto es global.

Consideremos también en qué consiste el efecto de las alteraciones de la armadura:

- Una alteración de armadura afecta no solo a aquellas notas situadas sobre su misma línea (o posición) en el pentagrama, sino también a aquellas situadas a una línea que represente cualquier octava de la misma nota. Un ejemplo es el mostrado en la ilustración nº45, a la derecha: en la armadura encontramos un sostenido en la posición de un *Do 5*, sin embargo, afectará a cualquier Do que se encuentre en el mismo pentagrama (por ejemplo un *Do 4*).

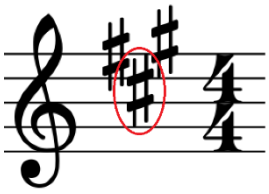


Ilustración 45 Alteración de armadura, en la posición de un *Do 5*

- Las alteraciones afectan sólo a las notas que se lean después de ellas, es decir, aquellas que se encuentren a su derecha en un mismo pentagrama.
- El efecto de las alteraciones no es acumulativo, sino sustituyente. Esto quiere decir que solo la última alteración que haya sido leída conservará sus efectos, hasta que una nueva alteración que afecte a las mismas notas la reemplace. Esto quiere decir que, para una nota cualquiera, sólo la última armadura que haya sido leída hasta entonces tiene efecto.

  Llegados a este punto, estamos en condiciones de considerar el efecto de la(s) armadura(s) detectadas. Sin embargo, ese efecto no se aplicará inmediatamente sobre la matriz de la *Partitura codificada*, sino que se preparará en una matriz aparte. La explicación de esto viene más adelante.

```
%Preveo el efecto de la armadura
ContrAlt=zeros(1, numnotas);

for i=1:size(AltArm, 2)

    Nalt=AltArm(1, i);
    a=AltArm(1, i)-12;

    while a>=EscalaNatural(1)
        Nalt(end+1)=a;
        a=a-12;
    end

    a=AltArm(1, i)+12;

    while a<=EscalaNatural(end)
        Nalt(end+1)=a;
        a=a+12;
    end

    for b=1:numnotas
        if AltArm(4, i)==PartCodi(4, b)  &&  PartCodi(3, b)>AltArm(3, i)
%Coincide en P y es posterior a la alteración
            for c=1:size(Nalt, 2)
                if Nalt(c)==PartCodi(1, b) %Coincide en número de nota
                    ContrAlt(b)=AltArm(2, i);
                end
            end
        end
    end
end
```

Contribución de alteraciones accidentales

El efecto de las alteraciones accidentales es puntual y tan sólo dura un compás. Se resume bajo dos principios.

- Una alteración accidental afecta a aquellas figuras que representen la misma nota (sin importar la octava) y que se encuentren entre ella y la próxima línea divisoria.
- El efecto de las alteraciones es sustituyente, por lo que se aplican de izquierda a derecha.

En la siguiente imagen vemos una demostración del segundo principio. Si, por ejemplo, tuviéramos en cuenta el efecto de las alteraciones por orden de tipo, no lograríamos la lectura correcta.

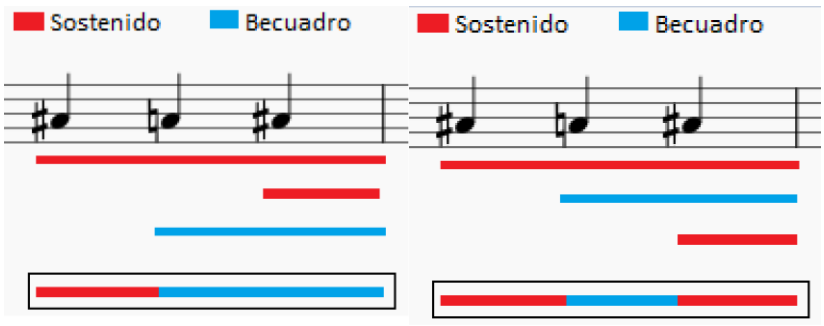
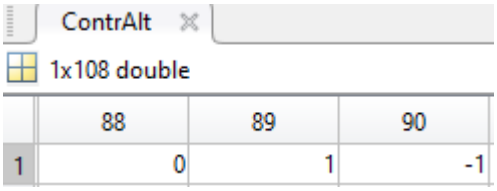


Ilustración 46 Lectura errónea de las alteraciones (izquierda) respecto la lectura correcta (derecha)



A la izquierda, un fragmento del vector *ContrAlt* (contribución de alteraciones), donde se observa que la nota número 89 se ve afectada por un sostenido, y la siguiente por un bemol.

Ilustración 47 Fragmento de *ContrAlt*

```
%Preveo el efecto de las alteraciones accidentales
for i=1:size(AlteracionesP, 2)
    for a=1:size(LineasDivisorias2, 2)
        if LineasDivisorias2(2, a)==AlteracionesP(4, i) &&
LineasDivisorias2(1, a)>AlteracionesP(3, i)
            for b=1:numnotas
                if PartCodi(4, b)==AlteracionesP(4, i) && PartCodi(1,
b)==AlteracionesP(1, i) && PartCodi(3, b)>AlteracionesP(3, i) && PartCodi(3,
b)<LineasDivisorias2(1, a)
                    ContrAlt(b)=AlteracionesP(2, i);
                end
            end
            break
        end
    end
end
```

Aplicación del efecto de las alteraciones

Una vez tenemos en una matriz aparte la suma de los efectos de las alteraciones, podemos aplicarlo a la *Partitura codificada*. El método es muy sencillo: el nuevo *Número de nota* de cada elemento será igual a él mismo más el número que se encuentre en la misma posición dentro de la matriz *ContrAlt.*

```
%Tengo el cuenta la contribución de las alteraciones
for i=1:numnotas
    PartCodi(1 ,i)=PartCodi(1 ,i)+ContrAlt(i);
end
```

Una vez terminado este paso, *Partitura codificada* está completa.

Funcionamiento de las funciones ofrecidas

Disponemos de toda la información esencial de la partitura en una sola matriz ordenada. Es el momento de reinterpretar esta cadena de números en función de nuestro propósito.

Texto

Una característica que ofrece nuestra aplicación y que puede servir de utilidad a aquellos músicos que tan solo necesiten conocer el nombre de las notas para interpretar la melodía es *Texto*.

El funcionamiento es el siguiente: por cada nota que no sea silencio, se obtiene el nombre de la nota. Esto se consigue reduciendo el *Número de nota* al rango de una sola octava. El número restante se asocia a una cadena de caracteres con el nombre de la nota correspondiente. Además, el nombre de la nota se sitúa en la misma vertical que la figura musical, de manera que quede centrado. En el eje *y* se situará entre pentagrama y pentagrama (haciendo uso de la matriz *rangopentagramas* *[ver Rango de pentagramas, pág. 14]*), teniendo en cuenta qué, si ya se ha puesto un nombre en ese lugar, (debido a que se trata de notas simultaneas) se colocará algo más abajo.

En la ilustración de abajo podemos comprobar el resultado de este proceso:

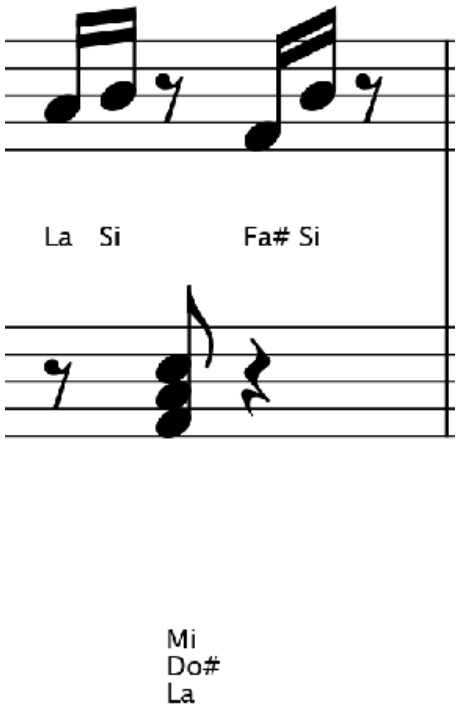


Ilustración 48 Ejemplo del funcionamiento de *Texto*

```
%Le pongo texto a la imagen
n=0;
for i=1:numnotas
    if PartCodi(1, i)~=99
        a=PartCodi(1, i);
        while a<0 || a>11 %reduzcoN
            if a<0
                a=a+12;
            elseif a>11
                a=a-12;
            end
        end

        switch(a)
            case 0
                a='La';
            case 1
                a='La#';
            case 2
                a='Si';
            case 3
                a='Do';
            case 4
                a='Do#';
            case 5
                a='Re';
            case 6
                a='Re#';
            case 7
                a='Mi';
            case 8
                a='Fa';
            case 9
                a='Fa#';
            case 10
                a='Sol';
            case 11
                a='Sol#';
        end
        x=round(PartCodi(3, i))-
espacio;
        y=round(ytexto(PartCodi(4,
i)+1)-espacio+n*espacio);
        imageoriginal =
insertText(imageoriginal,[x
y],a,'FontSize', espacio,
'BoxColor', 'white', 'BoxOpacity',
0);
    end

    if PartCodi(5, i)==0
        n=n+1;
    else
        n=0;
    end
end
```

Melodía

Para que el usuario pueda escuchar la partitura, la aplicación construirá una matriz con información de audio digital reproducible a partir de la *Partitura codificada*.

Para entender el proceso, hemos de entender mínimamente el concepto de audio digital: como se sabe, el sonido son vibraciones en el aire. Un altavoz o unos auriculares emiten sonido porque hacen oscilar una membrana que empuja el aire. El audio digital son una cadena de números que representan el 'empujón' que debería estar pegando la membrana al aire en determinado instante. Generaremos esta cadena de números a partir de cada columna de la *Partitura Codificada*: *el Número de nota* define la frecuencia con la que la membrana del altavoz oscilará y el parámetro *Tipo* define el tiempo durante el cual lo hará.

Para convertir los parámetros de *Partitura Codificada* en sonido, utilizaremos la función "sin()" de MatLab, ya que, a partir de los parámetros mencionados antes, genera la cadena de valores que representa la oscilación típica de una sinusoide. La frecuencia de la sinusoide la obtendremos de la siguiente expresión [6], donde 440 equivale a la frecuencia de afinación estándar del *La 4*:

$$f\ (Hz) = 440 \cdot 2^{\frac{Número\ de\ nota}{12}}$$

Además, habrá que tener en cuenta el concepto de la simultaneidad. Durante la generación de la matriz, se solaparán tanto las notas indicadas por el parámetro Simul [*ver Agregación de Simul, pág. 31*] como cada pentagrama par sobre el pentagrama anterior siempre y cuando se haya considerado que el pentagrama es doble.

La ilustración de abajo muestra el uso de dos contadores, *tmenor* y *tmenorPA* (en la imagen simplificados como *t* y *tPA*), para lograr el solape de pentagramas en el caso de un pentagrama doble. *tmenor* corresponde al primer número de muestra vacío (es decir, donde se va a empezar a escribir) de la matriz *cancion,* la cual contendrá toda la información auditiva de la partitura.

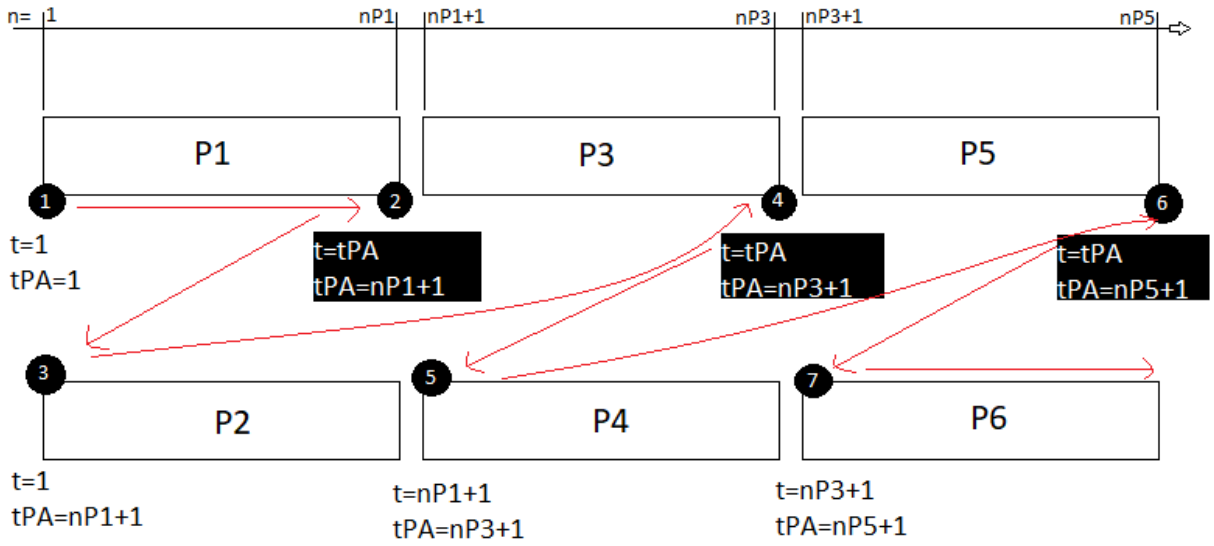
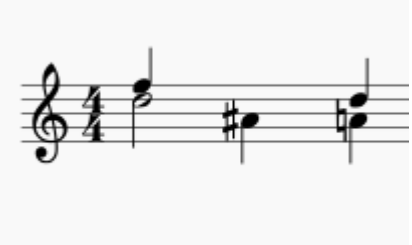


Ilustración 49 Proceso de generación de *cancion* mediante *tmenor* y *tmenorPA*

En la imagen, *nP1* equivale al número de muestras que dura el pentagrama uno, y respectivamente con *nP3* y *nP4*. Hay que aclarar que este número se calcula calculando el tamaño de la matriz *cancion* al momento de actualizar los parámetros *tmenor* y *tmenorPa*, cuyos momentos vienen representados por un cuadrado negro.

tmenor se inicializa con un valor de 1, y se incrementa por el número de muestras de la nota más corta de todas aquellas que se supongan simultáneas. Esto se realiza para poder soportar múltiples voces [*ver ilustración nº50*] y para evitar el problema del silencio de redonda [ver ilustración nº40, pág. 31]. *tmenorPA (*PA de Pentagrama Anterior*)* sirve para guardar el valor de *tmenor* antes de sobrescribirlo, pues volveremos a ese número de muestra cada vez que terminemos un pentagrama impar.



En la ilustración de la izquierda vemos como, si la segunda nota (la que viene acompañada del sostenido) se reprodujera al terminar la blanca del primer grupo de dos notas, la negra de ese mismo grupo habría dejado de sonar hace un pulso. Sin embargo, esto no es correcto, pues dicha pausa no se ha representado con un silencio en la partitura. Lo correcto es que la segunda nota empiece a sonar en cuanto termine la primera negra; eso sí, respetando los dos pulsos de duración de la blanca.

Ilustración 50 Múltiples voces en un pentagrama

```
%Método para obtener audio a partir de PartCodi
tmenor=1;
tmenorPA=1;
tipos=[];
cancion=[];
ntplaysp=zeros(1, numpent/(1+PentagramaDoble)+1);

for i=1:numnotas

    T=0:Ts:PartCodi(2, i)*(60/BPM)-Ts; %Creo un vector con su tamaño
    tipos(end+1)=length(T);
    if PartCodi(1, i)~=99 %es una nota
        f=440*2^(PartCodi(1, i)/12); %Convierto su N a frecuencia
    else %Es un silencio
        f=0;
    end
    cancion=[cancion zeros(1, tmenor-1+length(T)-length(cancion))]; %añado
tantos ceros como falten
    cancion(tmenor:tmenor+length(T)-1)=cancion(tmenor:tmenor+length(T)-
1)+sin(2*pi*f*T);

    if PartCodi(5, i)==1
        tmenor=tmenor+min(tipos);
        tipos=[];
    end

    if i==numnotas || PartCodi(4, i)~=PartCodi(4, i+1) %Pentagrama está
terminado
        if PentagramaDoble==1 && mod(PartCodi(4, i), 2)==0
            ntplaysp((PartCodi(4,
i)/2)+1)=round(tmenor/Fs/(60/BPM)/Tplay)+ntplaysp(PartCodi(4, i)/2);
        elseif PentagramaDoble==1 && mod(PartCodi(4, i), 2)==1
            tmenor=tmenorPA;
            tmenorPA=length(cancion)+1;
        elseif PentagramaDoble==0
            ntplaysp(PartCodi(4, i))=tmenor/Tplay+ntplaysp(PartCodi(4, i));
        end
    end

end
```

Luego de la creación del audio, los valores se normalizan para evitar el efecto “clipping”.

Modo Práctica

El *Modo Práctica* consiste en una ventana aparte que contiene un teclado y una pantalla, aparte de unos botones de reproducción.

Cuando se le da a "play" el teclado empieza a mostrar las teclas que se están pulsando virtualmente, de manera que el usuario pueda imitar el tecleo a tiempo real. Además, la pantalla muestra el pentagrama que se está reproduciendo, indicando con una línea vertical, la figura musical que se está interpretando en ese momento.

Para el funcionamiento de esta característica, se ha dado con la creación de una matriz aparte llamada *Teclado*, la cual define, en el dominio temporal, el inicio y final del pulsado de cada tecla.

Para ello, es imprescindible el concepto *Tplay*. *Tplay* es básicamente el máximo común divisor de las duraciones de las notas que aparezcan en *Partitura Codificada.* Este es el valor máximo de tiempo que debe pasar entre comprobación y comprobación de qué teclas deben estar pulsadas (y sonando) en ese momento. Por ejemplo, si en la partitura solo existen negras (duración 1) y corcheas (duración 0.5), *Tplay* adquiriría el valor 0.5 por tratarse del máximo común divisor de los dos.

```
%Calculo de Tplay
distipos=PartCodi(2, 1);
for i=1:numnotas
    for j=1:length(distipos)
        if PartCodi(2, i)==distipos(j)
            break
        elseif PartCodi(2, i)~=distipos(j) &&
j==length(distipos)
            distipos(end+1)=PartCodi(2, i);
        end
    end
end

Tplay=double(gcd(sym(distipos)));
```

El inicio del pulsado de una tecla se indica en *Teclado* mediante el número que corresponde con la duración de la nota en pulsos multiplicado por 60 y por la frecuencia de muestreo *Fs* [*ver Parámetros iniciales, pág.11*], y el final mediante un 1. El resto de casillas quedan vacías. Esto ayuda a realizar el cálculo que permite reproducir las notas con su duración asignada.

Además, la última fila de *Teclado* contiene la información relevante a la línea que sigue el ritmo de la lectura en la pantalla, en concreto la coordenada *x* de las notas que se están reproduciendo en ese momento. Para su cálculo también hay que tener en cuenta el "problema del silencio de redonda" [*ver pág. 31*]. La solución es elegir, de las *x* obtenidas de cada nota que suene a cada *Tplay,* la menor de entre los dos pentagramas (cuando el pentagrama sea doble). Esta solución se basa en el mismo concepto que el contador *tmenor* durante la creación de la matriz de audio.

```
%Obtención de x para la pantalla de modo práctica
for i=1:ntplaysp(end)+1
    if xlinea(1, i)~=0 && xlinea(2, i)~=0
        Teclado(end, i)=min(xlinea(:, i));
    elseif xlinea(1, i)~=0
        Teclado(end, i)=xlinea(1, i);
    elseif xlinea(2, i)~=0
        Teclado(end, i)=xlinea(2, i);
    end
end
```

La matriz *Teclado* se genera con el código de a continuación:

```
%Creacion del vector 'Teclado'
teclamasgrave=-33;
teclamasaguda=26;
Teclado=zeros(teclamasaguda-teclamasgrave+1+1, ntplaysp(end)); %+1 para
Xlinea
p=1;
tmenor=1;
tipos=[];
xlinea=zeros(2, ntplaysp(end)+1);
while p<=numpent
    for i=1:numnotas
        if PartCodi(4, i)==p
            tipos(end+1)=PartCodi(2, i);
            if PartCodi(1, i)~=99
                Teclado(PartCodi(1, i)+1-teclamasgrave, tmenor)=PartCodi(2,
i)*60*Fs;
                Teclado(PartCodi(1, i)+1-teclamasgrave, tmenor+PartCodi(2,
i)/Tplay)=1;
            end
            if PartCodi(5, i)==1
                xlinea(1+mod(p, 2), tmenor)=PartCodi(3, i)*804/ancho;
                tmenor=tmenor+min(tipos)/Tplay;
                tipos=[];
            end

            if i==numnotas || PartCodi(4, i)~=PartCodi(4, i+1)
                break
            end
        end
    end
    if i==numnotas
        break
    elseif PartCodi(4, i+1)==numpent
        p=2;
        tmenor=1;
    else
        p=p+2;
    end
end
```

Por último, para que la pantalla muestre el pentagrama correcto a cada instante, de la imagen original de la partitura se recortan diversos fragmentos que se guardan en la matriz *ImagenEnPentagramas*. El algoritmo comprueba la matriz *ntplaysp* (vector generado durante la generación de la matriz de audio con el número de *Tplays* que han de pasar hasta que cierto pentagrama termine) para conocer cuándo es el momento correcto para pasar a la imagen del pentagrama siguiente.

```
%Almaceno fragmentos de imagen para visualización en Modo Práctica
a=1;
for i=1:1+PentagramaDoble:numpent
    fragmento=imageoriginal(ytexto(i):ytexto(i+1+PentagramaDoble),1:ancho);
    fragmento=imresize(fragmento,[189 804]);
    ImagenEnPentagramas{a}=fragmento;
    a=a+1;
end
```

El método de lectura de *Teclado* usado durante el módo práctica se puede resumir de la siguiente manera:

- Durante un *Tplay* se “pulsa” cada tecla indicada en la columna que se esté procesando mostrando un rectángulo sobre ellas. De la misma manera, se “suelta” una teclaocultando el mismo rectángulo.
- Cada vez que se “pulsa” una tecla, se reproduce el sonido adecuado durante el tiempo indicado. En realidad, se reproducen *n* muestras, donde *n* equivale a dividir el número indicado en *Teclado* por el número de *BPM* que se haya elegido, y restarle el número de muestras equivalentes a multiplicar el tiempo que llevamos desde el comienzo del último *Tplay* por la frecuencia de muestreo *Fs.*

Cabe destacar que los sonidos son generados por MatLab durante la apertura de la aplicación, y que estos se crean para la mayor duración encontrada en *Partitura Codificada* para esa misma nota, lo que quiere decir que ni los audios son más largos de lo necesario ni se crean audios que no se vayan a utilizar.

Diseño y uso de la interfaz gráfica

Durante el diseño de la interfaz gráfica, se pensó en los más jóvenes para garantizarles un uso sin complicaciones de la aplicación desde la primera vez. Por ello, lo más prioritario durante su diseño ha sido en lograr una interfaz lo más simple e intuitiva posible. A continuación, un ejemplo de ello.

Nada más abrir la aplicación [ver ilustración nº51], nos encontramos con tan solo dos pequeñas ventanas. En una, leemos en grande “Haz click aquí para cargar la imagen”, lo cual se explica solo, y la otra tiene forma de botón, pero no es interactuable hasta el momento. La razón de ello es evidente.

Al clicar para cargar la imagen, nos aparece una ventana que nos pide el archivo a cargar. Una vez localizado, y luego de darle doble click, el archivo se carga y automáticamente se muestra en la misma ventana. En tal momento, el botón grande con la palabra “Procesar” escrita en ella y con un pequeño led verde que nos anima a hacerlo, se vuelve presionable.

Si le damos al botón, el led cambiará a botón rojo, indicando que ha comenzado el procesado. Además, el botón dejará de ser interactuable durante este mismo tiempo.

Una vez termina el procesado, se muestran todas las opciones de las que va dotada esta versión del *Lector de partituras*.

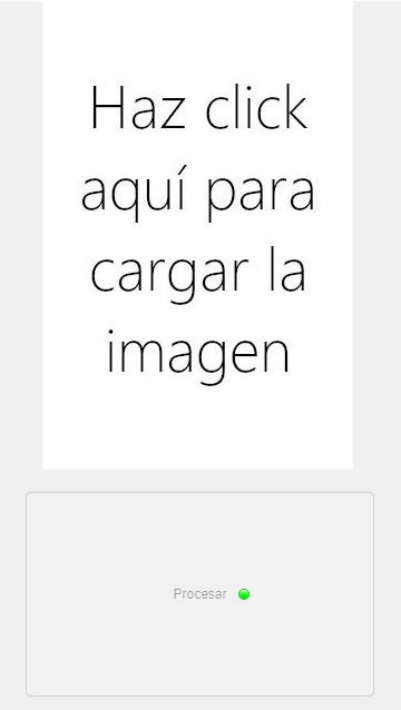
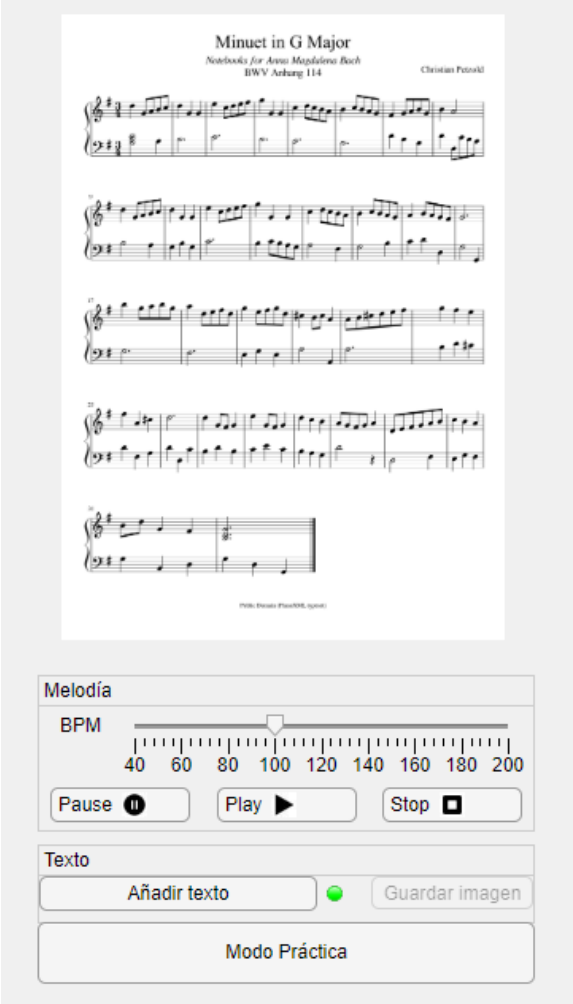


Ilustración 51 Primera vista de la aplicación

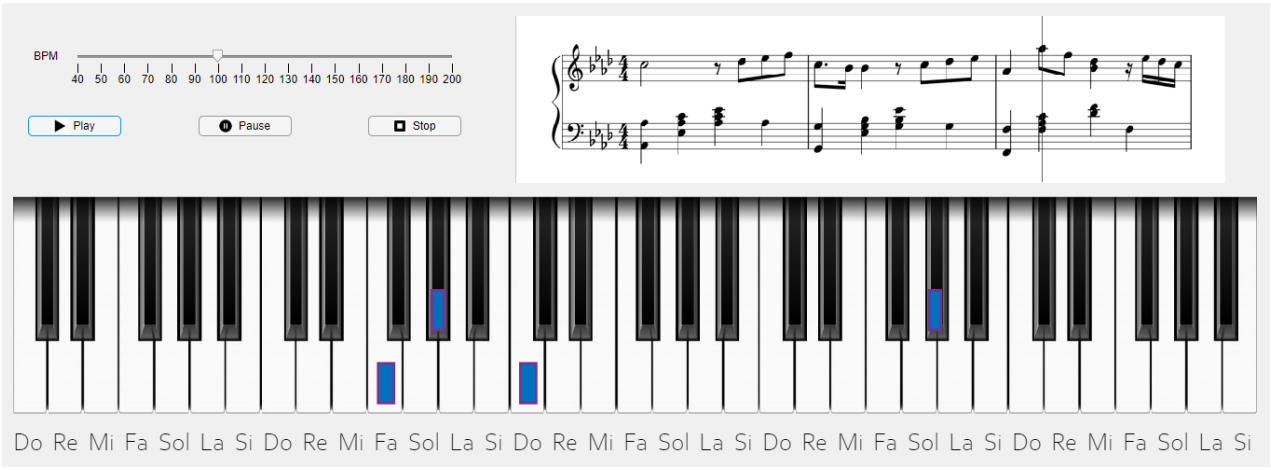


Los botones que se muestran tras la carga y procesado de la partitura tienen las siguientes funciones:

- **BPM**: barra que permite seleccionar el número de pulsos por minuto con los que se reproducirá la pieza al darle al "Play". El rango de valores posibles está comprendido entre 40 y 200. Es muy poco común encontrar obras para piano a un tempo fuera de ese rango.
- **Play**: la melodía reanuda la reproducción. Si es la primera vez que se pulsa, comenzará desde el principio.
- **Pause**: pausa la reproducción.
- **Stop**: deja de reproducir. Útil para volver al principio de la reproducción.
- **Añadir texto**: añade el texto debajo de las notas. Muestra un led verde que cambia a color rojo cuando ha sido accionado por el usuario, aunque vuelve a verde al terminar de procesar la orden.
- **Guardar imagen**: guarda la imagen en el directorio elegido. Solamente es interactuable cuando "Añadir texto" ha sido pulsado.
- **Modo Práctica**: abre una ventana donde se muestran las características del Modo Práctica.

Ilustración 52 Aspecto de la aplicación una vez procesada la imagen

Si elegimos abrir el *Modo Práctica*, la ventana que se nos abre tiene el aspecto que se puede ver en la ilustración de abajo. De la misma manera que la ventana principal del *Lector de Partituras*, contiene los botones **Play**, **Pause** y **Stop** y una barra de **BPM.**



Además, en la imagen se puede observar la manera utilizada para señalizar las teclas, y el estilo de línea vertical que sigue la lectura del pentagrama en pantalla.

Comparación con otros mecanismos similares

Existen diversas aplicaciones con una finalidad similar a la que presenta el *Lector de partituras*. Sin embargo, el uso de esta herramienta en lugar de otras puede resultar en un mayor beneficio. A continuación, se detallan los casos más fundamentales:

Objeto o aplicación	Descripción	Diferencias respecto la aplicación propuesta
Partitura de papel	Hoja o folio de papel, generalmente escaneada, que representa una pieza musical mediante la simbología correspondiente	Generalmente, no aparece el nombre de las notas. Además, no indica la digitación a ser usada en un instrumento
Archivos *midi*	Archivos que contienen información musical, incluyendo altura y duración de notas y el tipo de instrumento a sonar	No aparece el nombre de las notas. Además, no indica la digitación a ser usada en un instrumento, ni el usuario inexperto puede seguir la partitura tan solo usando un *midi*.
Crescendo, MusicScore, Guitar Pro…	Programas de edición de partituras	Resultan programas complejos, poco enfocados a la práctica del instrumento, y algunos de ellos son de pago
CamScanner, Music Pal, Note Reader…	Aplicaciones para dispositivos móviles de reconocimiento y lectura de partituras actuales	No indica la digitación a ser usada en un instrumento

Mejoras a largo plazo

El *Lector de partituras* está lejos de ser una aplicación terminada o completamente desarrollada. De hecho, existe una multitud de formas en las que la aplicación podría mejorar. A continuación, una lista detallada de algunas de ellas:

- Código de programación: aunque el procesado de imagen del *Lector de partituras* se realiza muy efectivamente en MatLab, no es tan efectiva la manera de lidiar con aspectos más gráficos, como el teclado del *Modo Práctica*. La aplicación mejoraría enormemente su entorno gráfico si estuviera programada en un lenguaje de programación más afín, como podría ser Java.
- Simbología musical: aunque el catálogo de figuras musicales que el *Lector de partituras* es capaz de detectar es bastante amplio, está lejos de ser completo. Los símbolos más prioritarios de ser añadidos en una versión posterior de la aplicación son: ligaduras, símbolos de repetición, trinos y mordentes, entre otros.
- Compatibilidad con editores de partituras: aunque se ha visto que MuseScore es un buen comienzo, no basta con soportar la simbología de un solo editor de partituras para convertir el *Lector de partituras* en una aplicación general de lectura de partituras.
- Edición de partituras: el proceso de reconocimiento de objetos y obtención de la *Partitura codificada* permite la recreación de otra partitura. Esto puede ser útil para convertir una imagen en una partitura editable, sobre todo cuando la aplicación sea compatible con una gran cantidad de tipos de partituras.
- Compatibilidad de nuevos instrumentos. El *Lector de partituras* no busca ser un aliado de los pianistas, sino de los músicos en general. La inclusión de nuevos instrumentos en la aplicación podría ayudar a darle una finalidad más general al programa y abrir nuevas posibilidades.

Conclusiones

A partir de técnicas de reconocimiento de imagen, se ha logrado diseñar y poner en funcionamiento una aplicación en MatLab que, a partir de la imagen de una partitura del programa MuseScore; reproduce la melodía, añade el nombre de las notas y propone una nueva manera de practicar música.

A pesar de lograr los objetivos de este proyecto, el *Lector de partituras* está todavía en proceso de creación, y las oportunidades de mejorar son incontables. Sin embargo, resulta ser una aplicación decente y bastante novedosa. Su método de reconocimiento de figuras musicales, con el cuál obtiene la codificación de la partitura introducida, resulta ser un método eficaz, simple y rápido, además de fácilmente adaptable a otros editores de partituras. Todo esto anima a pensar que, si la aplicación se desarrolla correctamente, podría llegar a ser un buen complemento para las clases musicales, pues sustituiría en cierta forma al profesor en aquellos casos donde no se puede contar con él, ya sea por falta de tiempo o por cualquier otra circunstancia.

Cabe añadir que la función *Modo Práctica* no funciona con la ligereza que se esperaba en un principio, al menos en ordenadores portátiles, aunque parece que su uso es viable para ordenadores de torre o de mayor capacidad de procesamiento. Se sospecha que parte de la culpa de esto se debe a la incapacidad de MatLab de manejar aspectos gráficos de la misma manera que otros lenguajes de programación, pues se puede comprobar que el algoritmo sobre el cual está basado esta característica de la aplicación es extremadamente simple y poco redundante.

Referencias

[1] - *Programas de Partituras gratis:* https://www.portalprogramas.com/descargar/partituras
[2] - *Programas para crear partituras:* https://www.neoteo.com/programas-para-crear-partituras/
[3] - *Transformada de Hough*: https://es.wikipedia.org/wiki/Transformada_de_Hough
[4] - Erosión (morfología): https://es.wikipedia.org/wiki/Erosi%C3%B3n_(morfolog%C3%ADa)
[5] – regionprops: https://es.mathworks.com/help/images/ref/regionprops.html?lang=en
[6] – La física de la música: http://ondasquenosrodean.blogspot.com/p/la-fisica-de-la-musica.html