



# MezzoReader

SMII 2020/2021

Colom Colom, Joan  
([joacoco@inf.upv.es](mailto:joacoco@inf.upv.es))

# Índice

1. Resumen de las ideas clave	5
2. Introducción	5
3. Objetivos	5
4. Planificación	5
5. Desarrollo	6
5.1. Identificación de pentagramas	7
5.2. Identificación de símbolos	7
5.3. Análisis de tonos	9
5.4. Ampliando la identificación de símbolos	10
5.4.1. Identificación de corcheas	12
5.5. Reproducción de notas musicales	13
5.6. Interfaz	14
5.7. Últimos ajustes del proceso de reconocimiento	14
5.8. Pictoforms	15
6. Funcionalidad	16
7. Conclusiones	17
8. Trabajos futuros	17
9. Bibliografía	18
10. Anexo I	19

## Índice de figuras

Figura 1: resultados de la ejecución de tutorial de operaciones morfológicas de OpenCV [3].	7
Figura 2: ejemplo de la aplicación de detección de círculos sobre una partitura.	9
Figura 3: plantillas utilizadas para la detección de notas negras y blancas respectivamente.	9
Figura 4: esquema de medición de tonos.	10
Figura 5: plantillas utilizadas para la detección de notas redondas y silencios de negra, redonda, blanca y corchea respectivamente.	11
Figura 6: ejemplo del análisis de pentagramas mediante ventana.	11
Figura 7: ejemplos de la detección de barras en corcheas. A la izquierda la imagen antes de la erosión y dilatación, a la derecha el resultado de dichas operaciones.	13
Figura 8: plantillas utilizadas para la identificación de la “cola” de una corchea en posición inferior y de una corchea en posición superior respectivamente.	13
Figura 9: ejemplo de pictoform.	15

## Índice de tablas

Tabla 1: planificación inicial del proyecto. \_\_\_\_\_6

Tabla 2: duración de los silencios y notas cubiertos por el proyecto. \_\_\_\_\_12

# 1. Resumen de las ideas clave

A lo largo de este trabajo vamos a presentar y describir el desarrollo de un programa capaz de procesar imágenes que contengan pentagramas monofónicos en clave de sol y extraer las melodías que en ellos se describen, al cual hemos bautizado con el nombre de MezzoReader.

## 2. Introducción

Los trabajos previos de Marc Escrig Villalonga [1] y Jorge Alcañiz Villanueva [2] en el procesamiento de partituras musicales nos han servido de referencia y apoyo en el desarrollo de este proyecto, obteniendo de ellos ideas de cómo abordar el problema y las dificultades que encontraríamos. No obstante, también nos planteamos soluciones que en estos trabajos no se abordan para observar y comparar la utilidad y eficacia de estas alternativas.

El desarrollo del proyecto ha sido posible gracias al uso de las librerías OpenCV y OpenAL en el lenguaje de programación C++. Además, hemos llevado a cabo el desarrollo sobre el sistema operativo macOS en su versión Big Sur (11.2.3).

## 3. Objetivos

Los objetivos principales del proyecto se definieron en su inicio como:

- Implementar la detección de los pentagramas contenidos en la imagen.
- Implementar la detección y diferenciación de las notas redondas, blancas, negras y corcheas.
- Implementar la detección y diferenciación de los silencios de redonda, blanca, negra y corchea.
- Calcular la altura de cada nota en función a su posición relativa dentro de el pentagrama.
- Reproducir la melodía extraída a partir de la reproducción individual y secuencial de cada nota identificada.

Además, también se definieron una serie de objetivos secundarios cuyo desarrollo se llevaría a cabo solo si el tiempo lo permitía, atendiendo a las dificultades surgidas durante el desarrollo de los objetivos principales, el tiempo disponible y la carga de trabajo. Así, estos objetivos secundarios eran:

- Implementar la detección y diferenciación de las notas semicorcheas y fusas.
- Implementar la detección y diferenciación de los silencios de semicorchea y fusa.
- Implementar la corrección automática de imágenes inclinadas, tomando como referencia el trabajo de Jorge Alcañiz Villanueva [2].

Como podremos ver a continuación, los objetivos principales han sido cumplidos. No obstante, los objetivos secundarios no han sido abordados finalmente debido a que nos pareció poco apropiado tratar de expandir las capacidades de reconocimiento sin refinar de forma adecuada primero el reconocimiento de los símbolos que constituían nuestros objetivos principales, y por tanto nuestra atención se ha centrado en estas tareas.

## 4. Planificación

Tareas a completar y calendario		
Tarea	Inicio	Fin
Detección de pentagramas.	15/2/2021	27/2/2021
Detección de notas negras y corcheas sin diferenciación.	18/2/2021	2/3/2021

Tareas a completar y calendario		
Tarea	Inicio	Fin
Detección de notas blancas y redondas sin diferenciación.	1/3/2021	12/3/2021
Medición de la altura de las notas.	18/2/2021	27/2/2021
Diferenciación de notas negras y corcheas.	19/3/2021	31/3/2021
Diferenciación de notas blancas y redondas.	13/3/2021	19/3/2021
Detección de silencios de negra.	9/4/2021	12/4/2021
Detección de silencios de corchea.	13/3/2021	16/4/2021
Detección y diferenciación de silencios de blanca y redonda.	19/4/2021	24/4/2021
Reproducción de la melodía extraída.	27/4/2021	5/5/2021

Tabla 1: planificación inicial del proyecto.

La planificación que se estableció en la fase de planteamiento del proyecto puede observarse en la Tabla 1 y en el diagrama de Gantt del Anexo I. Esta planificación ha sufrido cambios a la hora de su puesta en práctica debido a que ciertas tareas resultaron suponer una duración distinta a la inicialmente prevista. Por ejemplo, la detección de redondas, y silencios de negra, blanca y redonda resultaron ser muy sencillos de incorporar una vez diseñado el sistema de detección para notas negras y blancas. Por otro lado, la detección de corcheas ha sido una tarea que nos ha tomado más tiempo del previsto inicialmente, pasando por diferentes versiones tanto del algoritmo particular para la identificación de corcheas como del sistema general de reconocimiento de símbolos, tareas que no estaban previstas pero fueron necesarias. No obstante, en general, el orden de los pasos ha sido respetado, a excepción de la detección de corcheas por necesitar mayor reflexión y la reproducción de la melodía que se desarrolló mientras planteábamos la estrategia a seguir para las corcheas. Así, la planificación nos ha dado margen para el refinamiento del sistema e incluso en desarrollo de los pictoforms, que no se encontraban en el planeamiento original del proyecto.

## 5. Desarrollo

El desarrollo de nuestra aplicación parte del código proporcionado en el tutorial de operaciones morfológicas de OpenCV *Extract horizontal and vertical lines by using morphological operations* [3]. En este ejemplo, se aplican las operaciones `erode(...)` y `dilate(...)` [8] sobre la imagen de un pentagrama para obtener a partir de ella, por un lado, la imagen conteniendo únicamente las líneas horizontales de la misma y, por otro lado, eliminar de la imagen las líneas del pentagrama conservando el resto de la imagen. No obstante, este ejemplo realmente no lleva acabo ningún tipo de procesamiento de la información contenida en la imagen y se limita sencillamente a aplicar estas operaciones a modo de filtros para obtener las imágenes mencionadas. Por tanto, el primer paso para nuestro proyecto fue modificar este código [3] para que fuera capaz de detectar y almacenar los pentagramas presentes en la imagen.

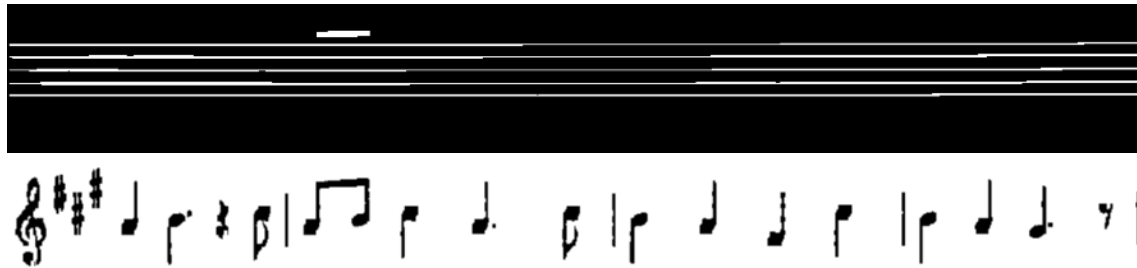


Figura 1: resultados de la ejecución de tutorial de operaciones morfológicas de OpenCV [3].

## 5.1. Identificación de pentagramas

La detección de pentagramas en una imagen se reduce a la identificación de líneas rectas horizontales en la imagen. Por esta razón, se puede partir del proceso de erosión y dilatación aplicado en el tutorial [3] y, una vez obtenida la imagen que contiene únicamente las líneas horizontales, esta se puede procesar para almacenar todas las filas de la imagen con un alto porcentaje de color. Además, podemos asumir que todas aquellas filas de la imagen que sean candidatas a contener una línea horizontal y que se encuentran muy cercanas entre sí representan realmente la misma línea, con lo que haciendo la media de la posición de las filas candidatas consecutivas muy cercanas obtendremos la posición de la línea real en la imagen. Si tras aplicar este proceso de detección y filtrado de líneas horizontales el número total de líneas encontradas es múltiplo de cinco, significará que hemos encontrado tantos pentagramas en ella como grupos de cinco líneas haya. En caso de que el número de líneas encontradas no fuera múltiplo de cinco, sería necesario un análisis más riguroso para poder identificar la presencia o no de pentagramas.

Esta forma de abordar la identificación de pentagramas probó inicialmente ser muy efectiva en la aplicación sobre imágenes que contenían pentagramas horizontales. No obstante, más adelante contemplaremos ciertas dificultades que pueden surgir con este procedimiento y cómo las hemos abordado, aunque cabe destacar que la idea principal no ha cambiado. Las funciones que implementan estas operaciones las podemos encontrar en la clase `MezzoUtilities` y son:

- `find_horizontal_lines(...)`: encargada de extraer todas las filas de la imagen que contengan un alto porcentaje de píxeles en blanco tras aplicar las operaciones morfológicas y que son por tanto candidatas a ser líneas de un pentagrama.
- `filter_horizontal_lines(...)`: encargada de, dada una lista de filas candidatas, fusionar aquellas que por su proximidad se puede considerar que contribuyen a la misma línea.
- `extract_all_staffs(...)`: encargada de obtener todos los pentagramas de una imagen. Esta función hace uso de `find_horizontal_lines(...)` y `filter_horizontal_lines(...)` para extraer los pentagramas. Los pentagramas son modelados mediante la clase `Staff` que contendrá las cinco líneas con lo conforman, el espaciado medio entre línea y los límites superiores e inferiores de alcance en la imagen.

Con esto, ya teníamos una de las partes más importantes de nuestro proyecto ya que era la base para su desarrollo y el siguiente paso fue llevar a cabo el análisis de los pentagramas que acabábamos de identificar.

## 5.2. Identificación de símbolos

La primera aproximación para la identificación de notas musicales que consideramos fue el uso de las mismas operaciones de erosión y dilatación que utilizamos para la detección de las líneas horizontales pero mediante la aplicación de una plantilla elíptica con `MORPH_ELLIPSE` del mismo

tamaño que el espacio medio entre líneas del pentagrama. Una vez aplicadas las operaciones morfológicas, bastaba con analizar la imagen resultante para encontrar todos los cúmulos de píxeles blancos, calculando su altura, anchura y centro.

Esta técnica probó ser eficaz para la detección de negras, corcheas y semicorcheas, teniendo un alto ratio de eficacia en la identificación de estas notas, aunque sin ser capaz de diferenciarlas entre ellas. Esto, unido con la alta eficacia de la detección de líneas, parecía indicar que las operaciones morfológicas podían ser un método efectivo para la lectura de pentagramas. Por esta razón, decidimos tratar de llevar a cabo la identificación de notas blancas siguiendo el mismo procedimiento. Para ello, empleamos dos plantillas elípticas, una de mayor tamaño que la otra, y aplicamos su diferencia centrada para obtener la plantilla final con la característica forma de elipse hueca de las notas blancas. Sin embargo, esta solución probó dar muy malos resultados.

Con el fin de tratar de mejorar los resultados, optamos por utilizar como plantilla un recorte de la propia nota blanca que obtuvimos a partir del pentagrama que estábamos tratando de analizar. No obstante, esta solución tampoco parecía permitir la identificación de las notas blancas.

Tras llegar a un callejón sin salida para el análisis de notas blancas mediante operaciones morfológicas, decidimos optar por otras técnicas. La primera de estas alternativas fue la detección de círculos [4][5], y para ello decidimos llevar a cabo pruebas sobre el ejemplo del tutorial sobre detección de círculos de OpenCV [6].

Con el código proporcionado [6], vimos que la identificación de círculos presentaba problemas para discernir entre notas y letras. Esto probaba ser un problema importante, ya que muchas partituras incluyen la letra de la canción bajo los pentagramas. Por esta razón se optó por descartar esta alternativa.

En este punto decidimos recuperar la idea empleada por Marc Escrig en su trabajo [1] para el cual utilizó la función `matchTemplate(...)` de OpenCV [7]. Dicha función permite encontrar el punto de una imagen con más similitud con la plantilla proporcionada, obteniendo de forma cuantificada esta similitud. No obstante, en lugar de utilizar plantillas de la nota entera, optamos por la identificación de únicamente su cabeza, por lo que las plantillas usadas en las primeras pruebas eran la cabeza de una nota negra y la cabeza de una nota blanca. Finalmente, esta alternativa probó dar buenos resultados y ser eficaz para la identificación y diferenciación de notas negras y blancas.

Antes de proseguir con la identificación de símbolos, cabe hacer ciertas consideraciones sobre el método *match template* para la identificación de símbolos:

- Debido a que el tamaño de la plantilla buscada es relevante para su correcta identificación, será necesario escalar las plantillas al tamaño apropiado en relación al espacio medio entre las líneas del pentagrama. Para ello utilizaremos la función `resize(...)`.
- Optamos por el uso de plantillas de gran resolución para tratar de asegurar que el escalado sea siempre de reducción, ya que un escalado de magnificación causa una mayor distorsión que puede afectar al proceso de identificación.
- Para la función `matchTemplate(...)` utilizaremos el método `TM_CCORR_NORMED` ya que nos ha proporcionado mejores resultados.

Llegados a este punto, ya podemos pasar a considerar la extracción de la información relativa a los tonos de las notas identificadas.



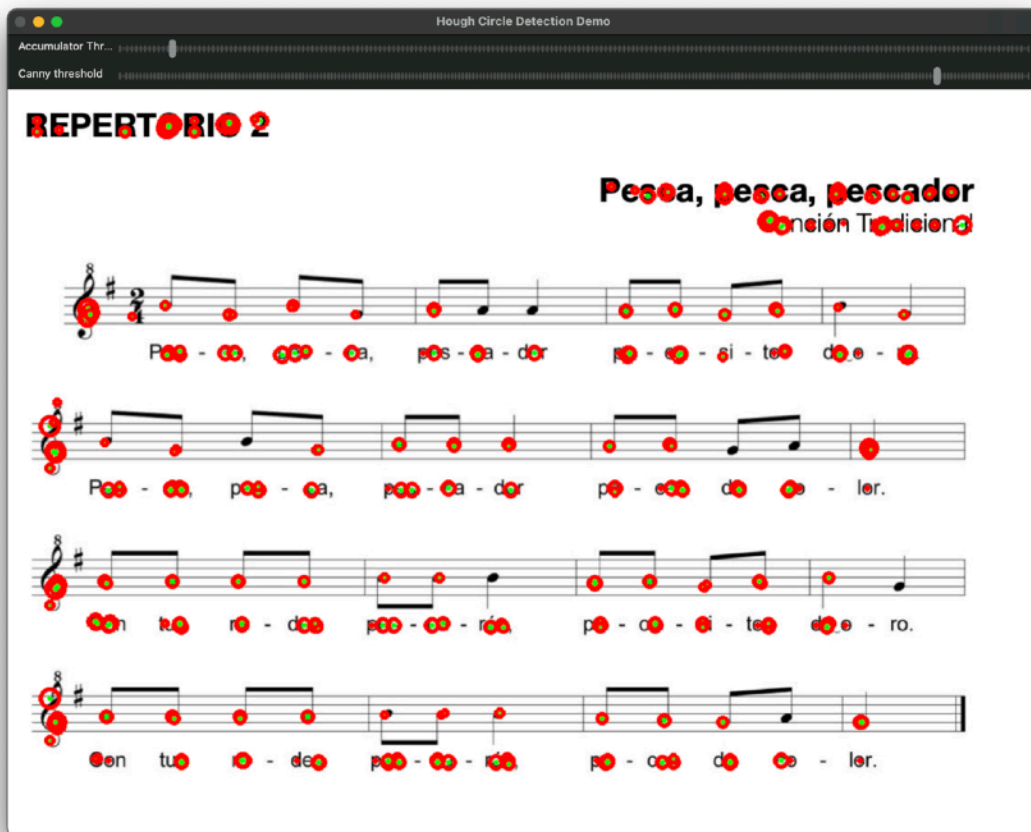


Figura 2: ejemplo de la aplicación de detección de círculos sobre una partitura.

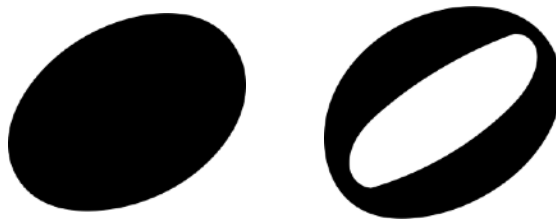


Figura 3: plantillas utilizadas para la detección de notas negras y blancas respectivamente.

### 5.3. Análisis de tonos

A partir de la posición de una nota en la imagen y de la posición de la línea inferior del pentagrama al que pertenece, podemos obtener el tono de la nota midiendo su altura relativa con respecto al pentagrama. De esta manera, si la nota se encuentra en la línea inferior del pentagrama, que denominaremos línea base, su altura relativa será cero. Si se encuentra en el primer espacio en blanco, su altura relativa será uno, y por tanto si se encuentra en la segunda línea será dos. Siguiendo esta progresión, podemos medir la altura relativa de la nota en unidades que representen la mitad del espacio medio entre líneas, de tal forma que si la altura de la nota es par estaremos en una línea y si es impar estaremos en un espacio. Con esta medida, teniendo en

cuenta que en las partituras en clave de sol la línea base representa la nota mi, podemos determinar tanto el tono como la escala de la nota.

Para facilitar la tarea, representamos las notas mediante la clase `Note` que contendrá la posición de la nota en la imagen, el tono y la duración, de la cual hablaremos más adelante.

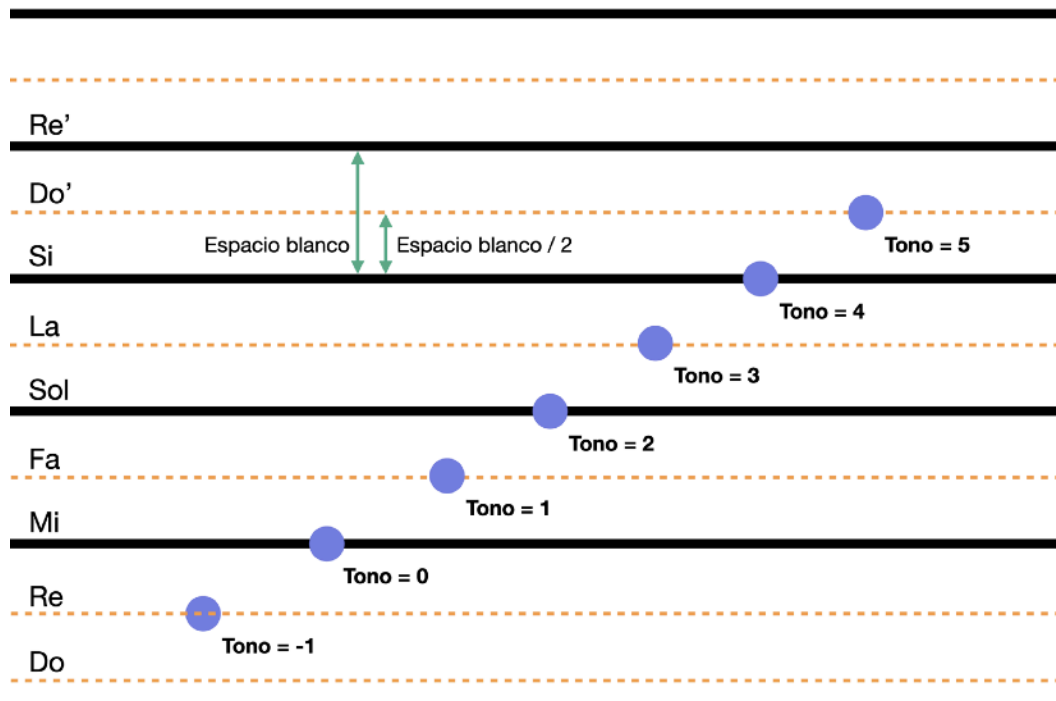


Figura 4: esquema de medición de tonos.

## 5.4. Ampliando la identificación de símbolos

Hasta el momento, el reconocimiento de los símbolos contenidos en un pentagrama consistía en tratar de identificar todas las notas negras presentes en el área de alcance del pentagrama dentro de la imagen, y a continuación se procedía a hacer el proceso con las notas blancas. Este proceso se podría haber extendido para añadir más símbolos, encontrando todos los símbolos de las sucesivas clases. No obstante, este método de funcionamiento presentaba varios inconvenientes que llevaron a la remodelación del proceso de identificación:

- Había una mayor fragilidad a los falsos positivos ya que era posible que un mismo objeto se diera como bueno para distintas clases al compararse sucesivamente, lo que para poder solucionarse hubiera requerido repasar la lista de anteriores hallazgos para ver si habían conflictos en la misma posición de la imagen para solucionarlos.
- Dificultaba la extensión de las capacidades de reconocimiento del programa, siendo farragosa y complicada la incorporación de nuevos símbolos.
- Los símbolos se identificaban de forma no ordenada, por lo que la secuencia que se obtenía como resultado representaba el conjunto de símbolos encontrados, pero no la secuencia de símbolos presentes en el pentagrama, lo que era fundamental para permitir la posterior reproducción de la melodía y facilitar la visualización gráfica del proceso de identificación en tiempo real.

Por estas razones, en primer lugar, procedimos al diseño de dos nuevos componentes:

- La clase `Symbol`, encargada de representar los símbolos que se pretendía reconocer así como sus características y parámetros, como la ruta en el sistema de archivos a la plantilla que lo representa, la altura medida en espacios de pentagrama, un identificador y varios parámetros para ajustar el reconocimiento del mismo.

- La cabecera `Alphabet` donde se declaran los diferentes símbolos que queremos identificar y la lista de los mismos. De esta manera para extender el número de símbolos a reconocer únicamente es necesario añadir la descripción del mismo a `Alphabet`. Se incorporan por tanto la descripción de notas redondas, así como también los silencios de negra, blanca, redonda y corchea.



Figura 5: plantillas utilizadas para la detección de notas redondas y silencios de negra, redonda, blanca y corchea respectivamente.

A continuación, remodelamos el algoritmo de identificación de símbolos para llevar a cabo un análisis secuencial del pentagrama de izquierda a derecha haciendo uso de una ventana, tratando de simular el proceso de lectura humano. De esta forma, la ventana se define entre la columna de inicio y la de fin dentro de la imagen, incrementando la posición de la columna de fin hasta llegar al final del pentagrama mientras no se detecte ningún símbolo dentro de la ventana, y actualizando la columna de inicio a la posición de fin cuando se encuentra un símbolo.



Figura 6: ejemplo del análisis de pentagramas mediante ventana.

Para la identificación de símbolos dentro de la ventana, se compara su contenido con cada uno de los símbolos definidos en la lista de `Alphabet`, devolviendo como resultado el índice del símbolo que presentó un mayor grado de similitud con el contenido de la ventana, así como también su posición. En caso de no encontrar ningún símbolo en la ventana, el índice devuelto será -1. Las funciones encargadas de implementar estos procedimientos son `extract_notes_v2(...)` y `find_most_suitable_template(...)` presentes en la clase `MezzoUtilities`.

Gracias a este procedimiento, podemos construir la lista ordenada de notas y silencios contenidos en un pentagrama. La duración de los mismos vendrá determinada por la clase de símbolo que los represente, aplicando los valores vistos en la Tabla 1. La posición en la imagen se obtiene a partir de la posición en la ventana, y para el cálculo del tono se utiliza la función `get_note_tone(...)` que implementa la funcionalidad descrita en el apartado 4.3.

Nombre	Nota	Silencio	Duración
Redonda			x4 tiempos







Nombre	Nota	Silencio	Duración
Blanca			x2 tiempos
Negra			x1 tiempo
Corchea			x0,5 tiempos

Tabla 2: duración de los silencios y notas cubiertos por el proyecto.

### 5.4.1. Identificación de corcheas

La identificación de corcheas supone una dificultad adicional al reconocimiento de los símbolos mencionados hasta el momento. Esto se debe a que los símbolos que habíamos tratado hasta el momento presentan características visuales muy diferentes entre sí que hacen que las plantillas se puedan comparar de forma individual con la ventana, garantizando que dichas plantillas son lo suficientemente diferentes como para asegurar una correcta clasificación del contenido de la ventana, motivo por el cual hasta el momento solo nos habíamos centrado en identificar la “cabeza” de las notas ya que era su rasgo distintivo. No obstante, las corcheas presentan una alta similitud con las notas negras que dificulta su diferenciación. Además, las corcheras pueden presentar diferentes formas dependiendo de a qué altura se encuentren y de si se encuentran agrupadas con otras corcheas o no. Todo esto nos llevó a sacrificar la generalidad del algoritmo que habíamos planteado para la identificación de los símbolos para poder añadir los casos específicos de las corcheas y tener un reconocimiento más informado.

La identificación de corcheas parte de la identificación de la “cabeza” de una nota negra, ya que este rasgo tanto en negras como en corcheas es idéntico. Una vez identificada la misma, habrá que proceder a determinar si realmente se trata de una nota negra o, de lo contrario, estamos ante la presencia de una de las formas que puede tomar la corchea.

Para llevar a cabo esta identificación, tenemos que tratar de determinar si podemos encontrar la “cola” característica de las corcheas individuales o la barra que une las agrupaciones de corcheas cerca de la nota encontrada. Esto supone que será necesario analizar los alrededores de la nota identificada, para lo que creamos una nueva ventana que representa el área de interés para estudiar la nota encontrada. En esta nueva ventana aplicamos los siguientes pasos:

1. Comparamos el contenido del área de interés con la plantilla de una corchea en posición superior. Si esta plantilla no es encontrada pasamos al siguiente paso. En caso contrario, ya podremos afirmar que hemos encontrado una corchea.
2. Tratamos de identificar la presencia de la barra de unión entre corcheas, para lo que aplicamos un método de erosión y dilatación basado en el utilizado de la identificación de líneas horizontales. De esta manera, conseguiremos obtener una imagen que contendrá únicamente la barras encontradas. Si la cantidad de color presente en esta imagen no supera un umbral mínimo, continuaremos en el siguiente paso. En caso contrario, ya podemos afirmar que hemos encontrado una corchea.
3. Se compara el contenido de la ventana con la plantilla de la “cola” de una corchea en posición inferior. Si esta se encuentra, marcaremos la nota como corchea, pero en caso contrario la dejaremos clasificada como una nota negra.

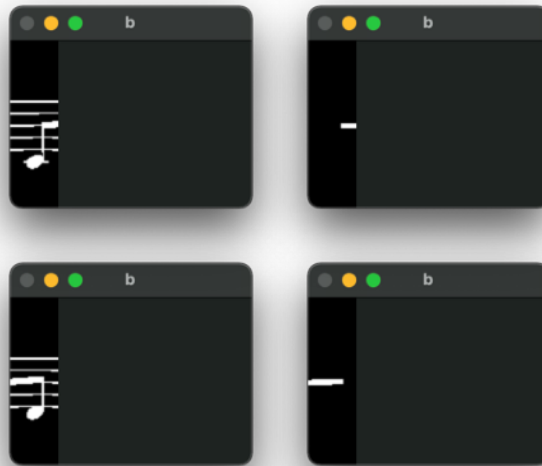


Figura 7: ejemplos de la detección de barras en corcheas. A la izquierda la imagen antes de la erosión y dilatación, a la derecha el resultado de dichas operaciones.

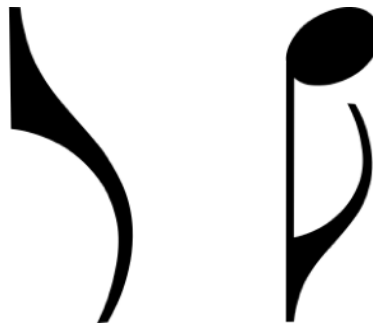


Figura 8: plantillas utilizadas para la identificación de la “cola” de una corchea en posición inferior y de una corchea en posición superior respectivamente.

El orden de estos pasos no es arbitrario. La identificación de la plantilla completa de una corchera en posición superior ha probado dar un menor número de falsos positivos que la identificación de corcheras individuales en posición inferior. Por tanto, aunque la identificación de barra posee una eficacia mayor en la mayoría de casos, necesita un mayor estudio del área de interés y por tanto se realiza en segundo lugar para tratar de reducir en número de veces que se llevará a cabo el proceso.

Así, la elección de las plantillas utilizadas tampoco es casual. La plantilla de la nota entera probó ser más fácilmente identificable por el sistema para la corchea en posición superior y permitir una mayor efectividad. Por otro lado, para la corchea de posición inferior decidimos utilizar tan solo la “cola” para tratar de focalizar el reconocimiento al rasgo más distintivo, ya que al usar la plantilla completa las notas negras eran confundidas como corcheas en posición inferior muy fácilmente.

## 5.5. Reproducción de notas musicales

Con el objetivo de facilitar al usuario de nuestra aplicación la comprensión de la notación musical y las melodías contenidas en la partituras, incorporamos la posibilidad de reproducir las notas

extraídas durante el proceso de análisis de la partitura. La clase `MezzoPlayer` es la encargada de implementar dicha funcionalidad mediante el uso de la librería OpenAL [9].

El primer paso para permitir la reproducción de las notas era disponer de los archivos de audio de las siete notas de la escala central. Estos nos fueron proporcionados inicialmente por nuestro profesor y nos permitieron empezar el desarrollo del reproductor. No obstante, con el desarrollo del proyecto se nos hizo necesario que los sonidos fueran de una duración exacta. Por ello, mediante el uso de la aplicación GarageBand [10], generamos un archivo WAV para cada una de las siete notas siendo la duración de los mismos la de un compás de cuatro por cuatro, equivalente a una duración de dos segundos. Esta duración nos ha permitido ajustar más fácilmente la duración de las notas reproducidas de acuerdo a la Tabla 1 sabiendo que el audio completo equivale a una redonda.

El constructor de `MezzoPlayer` se encarga de crear el oyente y la fuente necesaria para reproducir el sonido en el origen de coordenadas, así como también de la carga en siete bufares distintos de los sonidos de cada una de las notas. Una vez creado, mediante la función `MezzoPlayer::play(...)` podremos reproducir una nota dada, seleccionándose el buffer de audio y el pitch de la fuente adecuado para la reproducción de dicha nota. Esta función llevará a cabo una espera activa mientras no se complete la duración de la nota, en cuyo caso se detendrá la reproducción de la fuente y se finalizará la espera activa.

## 5.6. Interfaz

Prosiguiendo con nuestro objetivo de facilitar la comprensión y aprendizaje de la notación musical por parte del usuario, introducimos la posibilidad de mostrar de forma gráfica el proceso de análisis en tiempo real, mostrando la ventana de análisis y resaltando los símbolos reconocidos, indicando el tono de los mismos en caso de tratarse de notas. Además se proporciona también un deslizador en la ventana que muestra en proceso de análisis para poder ajustar la velocidad del mismo.

De la misma manera, también añadimos una nueva ventana para la reproducción de la partitura, la cual marca en tiempo real la nota que se esta reproduciendo. Esto facilita en seguimiento de la melodía y la comprensión de la partitura.

## 5.7. Últimos ajustes del proceso de reconocimiento

Con la intención de probar la eficacia de `MezzoReader` sobre más casos de uso, llevamos a cabo el escaneo de antiguas partituras que teníamos en casa para probar a llevar a cabo el reconocimiento de las mismas. No obstante, para nuestra sorpresa, el reconocimiento de las mismas falló, ya que `MezzoReader` era incapaz de detectar ninguno los pentagramas que estas imágenes contenían.

Siendo la primera vez que la identificación de pentagramas fallaba, procedimos a tratar de averiguar cuál era el motivo. Finalmente resultó ser que una elevada resolución de las imágenes, combinada con el fino grosor de las líneas de los pentagramas provocaba un alto escalonamiento en la representación de las mismas dentro de la imagen. Dicho escalonamiento imposibilitaba el reconocimiento de las líneas ya que el programa únicamente consideraba como candidatas a líneas aquellas filas de la imagen que tuvieran un 70% de píxeles coloreados y el escalonamiento provocaba que no se alcanzara dicho porcentaje.

Para solucionar este problema, y en vistas de que encontrar un porcentaje adecuado genérico que tuviera la mayor efectividad sería imposible, decidimos llevar a cabo la adición sobre la función `extract_notes_v2(...)` de un modo adaptativo que, dado el número real de pentagramas que la imagen contiene, va considerando de forma progresiva diferentes porcentajes tratando de encontrar el porcentaje apropiado para identificar los pentagramas. La exploración de dichos porcentajes se hace de forma descendente a diferentes etapas, siendo cada etapa más precisa

pero más lenta que la anterior, con la idea de tratar de acotar lo máximo posible el rango de porcentajes factibles para la última etapa mediante las primeras.

Esta técnica nos ha permitido una mayor flexibilidad en el reconocimiento de los pentagramas. No obstante, su uso no implica la identificación infalible de estos pentagramas, ya que al tener que rebajar el porcentaje necesario para considerar una línea, aumenta el riesgo también de que se produzcan falsos positivos que causen un reconocimiento erróneo.

Otro posible problema que hemos observado con respecto a las líneas del pentagrama es que su grosor puede afectar al reconocimiento de las corcheas con barra. Esto se debe a que no todas las partituras tienen el mismo grosor de línea para sus pentagramas y en aquellas que su línea es más gruesa, las operaciones morfológicas que se encargan de aislar la barra de las corcheas no son capaces de eliminar las líneas del pentagrama por su grosor. Para poder tratar de solucionar esto, vimos que en estos casos basta con dibujar una fina línea en blanco sobre las líneas del pentagrama para que éstas no pasen el filtro de las operaciones morfológicas. No obstante, aplicar este procedimiento en partituras con líneas más finas ha probado inducir problemas e interferir con la identificación de las barras, con lo que hemos optado empíricamente por aplicar este procedimiento únicamente cuando el grosor medio de línea sea superior al 20% del tamaño del espacio entre líneas medio.

## 5.8. Pictoforms

Una alternativa que empezamos a considerar tras el desarrollo del algoritmo de pentagramas y notas así como la reproducción de las melodías contenidas en ellos fue la posibilidad de no solo encontrarlas y reproducirlas, sino también ser capaces de almacenarlas. De esta idea surgió lo que hemos denominado *pictoforms*.

Los pictoforms se inspiran en la idea de códigos QR [11] o ArUco markers [12], de forma que consisten en imágenes que codifican una secuencia de notas y silencios, representando su tono y duración. De esta manera, estas imágenes constituyen una forma de codificar las melodías extraídas durante la fase de reconocimiento de nuestra aplicación para poder almacenarlas y poder, de igual manera, cargar posteriormente dichas imágenes para escuchar la melodía que estas codifican.

Las notas se representan en estos pictoforms mediante el uso de cuadrados de tres por tres píxeles. La fila superior de este cuadrado codifica el tono de la nota utilizando cada canal de color de dichos píxeles como un bit para codificar el valor numérico del tono. De igual forma, la fila inferior codifica el identificador numérico del tipo de nota de que se trata, a partir del cual podremos obtener la duración de la misma. El píxel central se utiliza para indicar la presencia de una nota en el cuadro actual y por último el primer píxel de la segunda fila representa si estamos ante una nota o un silencio. Las funciones `decode_pictoform(...)` y `encode_pictoform(...)` de la clase `MezzoUtilities` implementan la lectura y generación de este tipo de imágenes.

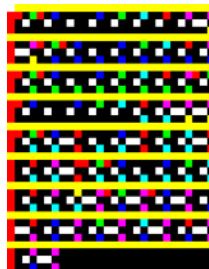


Figura 9: ejemplo de pictoform.



Los pictoforms han sido pensados para que puedan tener cualquier escala y puedan contener melodías polifónicas. No obstante, estas cuestiones no han sido implementadas ya que el desarrollo de los pictoforms y sus posibilidades no es el tema central de nuestro proyecto.

## 6. Funcionalidad

La interfaz final de nuestra aplicación es de la forma:

Usage: MezzoReader [params]

```
-?, -h, --help, --usage (value:true)
Usage examples:
./MezzoReader -i=images/notes.png -o=analysed_image.png -p -v
./MezzoReader -i=images/when_im_gone.jpg -a=7 -n=5
-a, --staffs (value:<none>)
Adaptive mode given the number of staffs
contained in the image.
-d, --decode (value:<none>)
Load a melody from a pictoform
-e, --encode (value:<none>)
Save the extracted melody as pictoform.
-i, --input (value:images/notation.png)
Input image.
-n, --precision (value:3)
Precision for the adaptive mode as 10e-n.
-o, --output (value:pentagrama_analizado.png)
Output image.
-p, --play (value:<none>)
Play music after the analysis.
-v, --visual (value:<none>)
Visual mode.
```

Mediante la opción `-i` podemos indicar la ruta a la imagen que queremos analizar y la opción `-o` nos permitirá indicar el lugar y nombre de la imagen que se generará como resultado del análisis. Las opciones `-v` y `-p` servirán para activar el modo gráfico de análisis y la reproducción final de la melodía respectivamente. En caso de no proporcionar el parámetro `-i`, la aplicación tratará por defecto de acceder a la imagen *notation.png* en la carpeta *images*, la cual debe encontrarse en el mismo directorio que la aplicación. Si no se proporciona el parámetro `-o`, se generará por defecto una imagen *pentagrama\_analizado.png* en el mismo directorio en el que se encuentra la aplicación. Para permitir la reproducción de la melodía, la carpeta *sounds* debe encontrarse en el mismo directorio que la aplicación.

Las opciones `-a` y `-n` se utilizan para el modo adaptativo. El parámetro `-a` indica cuántos pentagramas contiene la partitura de la imagen proporcionada, y el sistema intentará adaptar sus parámetros internos para encontrar el número de pentagramas dado en la imagen. El parámetro `-n` se puede utilizar conjuntamente con el parámetro `-a` para indicar la precisión del decremento más pequeño utilizado por el algoritmo adaptativo. De esta manera para el valor por defecto  $n=3$  se utilizará como unidad de decremento más pequeña del algoritmo adaptativo el valor  $10^{-3}$ .

Finalmente, las opciones `-e` y `-d` se utilizan para tratar con pictoforms. MezzoReader puede escribir la melodía reconocida en una pictoform usando la opción `-e` y puede leer y reproducir los sonidos contenidos en una pictoform usando el parámetro `-d`. El parámetro `-d` tiene preferencia sobre el resto de los parámetros, lo que hace que el sistema ignore todos los demás cuando está presente `-d`.



Así, algunos ejemplos de ejecución de la aplicación son:

```
$ ./MezzoReader -i=images/notes.png -o=analysed_image.png -p -v
$ ./MezzoReader -i=images/when_im_gone.jpg -a=7 -n=5
$ ./MezzoReader -i=images/a_dormir.jpeg
$ ./MezzoReader -v -p -e=pictoform_notacion.png
$ ./MezzoReader -d=pictoform_notacion.png
```

## 7. Conclusiones

El análisis de partituras musicales es una tarea compleja, comparable a las tareas de reconocimiento de objetos y de análisis de textos en imágenes. Las técnicas de reconocimiento de plantillas de OpenCV han probado ser útiles para situar e identificar los símbolos contenidos en un pentagrama, pero también han mostrado tener sus limitaciones. La identificación de símbolos pequeños y de rasgos finos ha mostrado ser difícilmente abordable con estas técnicas, pudiéndose ver en la complejidad y la falta de precisión en la identificación de los rasgos de una corchea, así como también en la imposibilidad de detección de los silencios de corchea en la mayoría de los casos, a excepción de en algunas imágenes con una muy alta resolución y nitidez. Además, estas técnicas muestran ser muy rígidas en el reconocimiento, siendo necesaria una mayor flexibilidad para poder tolerar las diferencias tipográficas entre unas partituras y otras. La calidad del análisis obtenido dependerá de la resolución y nitidez de la imagen proporcionada, así como de la orientación de la partitura en la imagen proporcionada, el grosor de las líneas del pentagrama y la tipografía de las notas.

Con todo esto, llegamos a la conclusión de que la aplicación de técnicas de inteligencia artificial y aprendizaje profundo, como la red convolucional utilizada por Jorge Alcañiz [2], se intuye como una alternativa más adecuada para abordar el problema del análisis e interpretación de partituras por computador.

## 8. Trabajos futuros

El desarrollo del proyecto nos abre muchas alternativas para trabajos futuros, ya que queda mucho por hacer en el reconocimiento y análisis pentagramas. Estas tareas se podrían abordar con la incorporación de técnicas de aprendizaje profundo e inteligencia artificial para poder llevar a cabo el reconocimiento de símbolos más pequeños y sutiles como los puntos, las ligaduras o un mejor reconocimiento de los silencios de corchea. De igual forma, se podría extender el reconocimiento a la identificación de partituras en diferentes claves, y tanto monofónicas como polifónicas.

Una mejor y más interactiva interfaz para animar y facilitar el aprendizaje de la notación musical tanto por parte de niños como adultos es otra área con muchas posibilidades para el crecimiento de proyecto. Alternativas para dar soporte a un mayor abanico de sonidos de diferentes instrumentos a poder reproducir o incluso la posibilidad de poder comparar la partitura con la entrada proveniente de un instrumento periférico.

Otra área que se nos presenta es la explotación de las posibilidades de los pictoforms para compartir y guardar melodías. De igual que con la cámara de nuestros dispositivos móviles podemos leer códigos QR el cualquier momento que nos den información adicional de un determinado contexto, el reconocimiento de los pictoforms por parte de dispositivos con cámara podría ser una manera permitir el enriquecimiento de contextos mediante la reproducción de melodías descritas en estos códigos. Una forma de integrar la música con la realidad aumentada.

## 9. Bibliografía

- [1] Marc Escrig Villalonga. (2018). *Sistema de análisis de imágenes de partituras musicales para su digitalización y su reproducción* <<https://riunet.upv.es/handle/10251/112316>>.
- [2] Jorge Alcañiz Villanueva. (2020). *Aplicación de ayuda a la iniciación al uso de partituras musicales* <<https://riunet.upv.es/handle/10251/151758>>.
- [3] Theodore Tsesmelis. *Extract horizontal and vertical lines by using morphological operations* <[https://docs.opencv.org/master/dd/dd7/tutorial\\_morph\\_lines\\_detection.html](https://docs.opencv.org/master/dd/dd7/tutorial_morph_lines_detection.html)>.
- [4] H.K. Yuen, J. Pricen, J. Illingworth and J. Kittler. (1989). *A Comparative Study of Hough Transform Methods for Circle Finding* <<http://www.bmva.org/bmvc/1989/avc-89-029.pdf>>.
- [5] Adrian Rosebrock. (2014). *Detecting Circles in Images using OpenCV and Hough Circles* <<https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/>>.
- [6] OpenCV. *Hough Circle Transform* <[https://docs.opencv.org/3.4/d4/d70/tutorial\\_hough\\_circle.html](https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html)>.
- [7] OpenCV. *Object Detection* <[https://docs.opencv.org/3.4/df/dfb/group\\_imgproc\\_object.html#ga586ebfb0a7fb604b35a23d85391329be](https://docs.opencv.org/3.4/df/dfb/group_imgproc_object.html#ga586ebfb0a7fb604b35a23d85391329be)>.
- [8] OpenCV. *Image Filtering* <[https://docs.opencv.org/master/d4/d86/group\\_imgproc\\_filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c](https://docs.opencv.org/master/d4/d86/group_imgproc_filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c)>.
- [9] OpenAL. *OpenAL: Cross Platform 3D Audio* <<https://openal.org>>.
- [10] Wikipedia, la enciclopedia libre. *GarageBand* <<https://es.wikipedia.org/wiki/GarageBand>>.
- [11] Wikipedia, la enciclopedia libre. *Código QR* <[https://es.wikipedia.org/wiki/Código\\_QR](https://es.wikipedia.org/wiki/Código_QR)>.
- [12] OpenCV. *Detection of ArUco Markers* <[https://docs.opencv.org/master/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html)>

## 10. Anexo I

Tarea	Inicio	Fin	15/2/21	22/2/21	1/3/21	8/3/21	15/3/21	22/3/21	29/3/21	5/4/21	12/4/21	19/4/21	26/4/21	3/5/21
Detección de pentagramas.	15/2/21	27/2/21												
Detección de notas negras y corcheas sin diferenciación.	18/2/21	2/3/21												
Detección de notas blancas y redondas sin diferenciación.	1/3/21	12/3/21												
Medición de la altura de las notas.	18/2/21	27/2/21												
Diferenciación de notas negras y corcheas.	19/3/21	31/3/21												
Diferenciación de notas blancas y redondas.	13/3/21	19/3/21												
Detección de silencios de negra.	9/4/21	12/4/21												
Detección de silencios de corchea.	13/3/21	16/4/21												
Detección y diferenciación de silencios de blanca y redonda.	19/4/21	24/4/21												
Reproducción de la melodía extraída.	27/4/21	5/5/21												