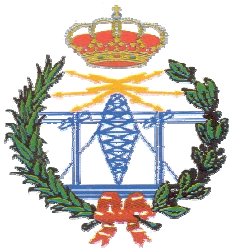
UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
ESPECIALIDAD: SONIDO E IMAGEN



PROYECTO FIN DE CARRERA

SISTEMA DE RECOCIMIENTO DE PARTITURAS MUSICALES

Autor:
DAVID CARRETERO DE LA ROCHA

Tutora:
Dra. RAQUEL M. CRESPO GARCÍA
(Departamento Ingeniería Telemática)

Leganés, junio de 2009

Lo que sabemos es una gota de agua;
lo que ignoramos es el océano.

- *Isaac Newton*

La batalla de la vida no siempre la gana el hombre más fuerte, o el más ligero, porque tarde o temprano, el hombre que gana es aquel que cree poder hacerlo.

- *Napoleón Hill*

Agradecimientos

En primer lugar, debo dar gracias a mis padres por haberme facilitado los medios para poder cursar una carrera, una Ingeniería Técnica de Telecomunicación con especialidad en Sonido e Imagen, en una de las mejoras universidades del panorama español, la Universidad Carlos III. Gracias por vuestra comprensión y ayuda, por haberme animado en los momentos de bajón y enseñarme que todo se puede conseguir con esfuerzo.

En cuanto al proyecto fin de carrera, debo dar gracias a mi tutora Raquel, ya que me dio la oportunidad de trabajar en uno de los proyectos más interesantes y amenos que había visto hasta entonces. Gracias por haber confiado en mí y haberme dedicado tu tiempo a pesar de tu apretada agenda.

Gracias a mi familia, a mis padres: Alfonso y Ana, y a mi hermano Alberto, por haber estado ahí apoyándome y recordándome que ya mismo tengo el título en mis manos. Gracias a Rocío, por ser la mejor persona y haberme dado todo el cariño y apoyo que necesitaba en estos momentos y durante la carrera. Por haberme escuchado atentamente, darme consejo, ayudarme a olvidar los problemas… Y por supuesto, por tantos momentos bonitos y felices y ¡por los que nos quedan!

También dar las gracias a mis abuelos: Pablo y Petra, por confiar en su nieto y ayudarle en todo lo necesario con las dudas de teoría musical. Igualmente gracias a Paqui y a Manolo, por acogerme y tratarme tan bien, porque gracias a vosotros y a mis padres estamos cumpliendo un sueño.

También quiero dar gracias a mis compañeros de piso, Carlos y Daniel, por esos momentos buenos y malos que hemos pasado juntos con las clases y los exámenes. Por las mil “movidas” que hemos tenido en el piso, por las tonterías que habremos hecho, por las noches hasta las tantas estudiando o haciendo prácticas… O por las noches que simplemente nos daba por tocar la guitarra y hacer el tonto.

Por supuesto, no me voy a olvidar de mi mejor amigo Alberto, que me ha ayudado y me ha acompañado durante todo el transcurso de la carrera, compartiendo anécdotas, historias o consejos y recordándome que debo tomarme la vida con más filosofía y salir más los fines de semana. Como tú mismo me dijiste: ¡Qué grande eres!

Tampoco quiero olvidar a algunos de los compañeros de clase más importantes que he tenido: César, Tony (Pakstor), José Luís (Aldovera), las dos Beatriz, Estela, Jessica, Pablo, Efrén y Juanlu. Gracias por esos momentos en clase.

Y finalmente, tampoco me voy a olvidar de mis colegas del barrio, los mejores amigos que se pueden tener: Diego, Guille, Dani (Roa), Jesús (Guayer), Jose Carlos, Gonzalo, Jorge (Menelcar)… Gracias por estar ahí aunque últimamente nos veamos de poco en poco.

A todos vosotros, ¡Gracias! ¡Este proyecto va por ustedes!

~ ÍNDICE ~

~ ÍNDICE DE FIGURAS ~

Resumen

El objetivo principal del sistema reconocedor de partituras musicales desarrollado en este proyecto, es la elaboración de un programa capaz de analizar partituras, ya sean manuscritas o no manuscritas, y digitalizar sus elementos, pudiendo tratar dichos datos y representarlos en distintos formatos.

Para ello, se hará uso de un clasificador simple basado en el algoritmo k-NN (k-*Nearest Neighbor*) que, junto a una base de datos de símbolos, será capaz de asignar a cada elemento una clase concreta dentro de las mencionadas en dicha base de datos.

Por lo tanto, la implementación de este sistema diseña y elabora un sistema tradicional de reconocimiento óptico de música que destaca por la rapidez de ejecución y los resultados obtenidos para ambas clases de partituras (manuscritas o no manuscritas), en contraposición a la gran mayoría de los sistemas OMR desarrollados que centran su atención en los símbolos no manuscritos.

Abstract

The main goal of the musical scores recognition system developed in this project is to make a program capable of analizing scores, wheter or not handwritten musical scores, and digitalize their symbols, for be able to treat these data and represent them in various formats.

To do this, it will use a simple classifier based on k-NN algorithm (*k-Nearest Neighbor*) which, with a symbols database, it will be able to assign to each element a class of this database.

Therefore, the implementation of this system designed and developed a tradicional system of optical music recognition that highlights the speed of execution and the results for both kinds of scores (handwritten musical scores or not handwritten musical scores), in contrast to most developed OMR systems that focus their work in the recognition of not handwritten musical scores.

1. Introducción

1.1. Motivación

Actualmente, la gran mayoría de la música se encuentra en formato digital. Sin embargo, existen composiciones que requieren de un procesado especial para su conversión como ocurre con las partituras musicales. La utilización del formato digital proporciona unas ventajas entre las que se pueden destacar:

- Almacenamiento perpetuo en el tiempo de la información musical en formato digital
- Tratamiento de los datos digitales (modificación, transmisión…)
- Conversión de los datos a otros formatos (gráfico, acústico…)
- Gestión más eficaz de las obras (mediante bases de datos)
- Estudio de las obras
- Globalización de la música, mediante la expansión y el intercambio de ficheros

Para la conversión de las partituras a formato digital, en primer lugar, se ha tenido en cuenta la utilización de un sistema de reconocimiento óptico de música (OMR, "*Optical Music Recognition*"). Otra opción para convertir la música impresa a música digital sería la utilización de programas de ordenador que, nota a nota, permitieran al usuario incluir símbolos digitales en un pentagrama hasta codificar la partitura. Esta tarea puede ser tediosa, sobretodo si tenemos en cuenta que existen obras con más de 300 páginas de partituras, como "*Aida*" de Verdi.

Mientras el primer procedimiento mencionado se realiza de forma automática (o semiautomática), el segundo requiere de un usuario que disponga de conocimientos musicales para transcribir la partitura. Este factor plantea una ventaja evidente de los sistemas OMR sobre el resto de opciones.

Además, la creación de un sistema OMR se puede aprovechar para realizar otros sistemas que utilicen sus características. Como ejemplo, se puede mencionar un sistema inteligente de dictado musical, como el propuesto en la Figura 1.

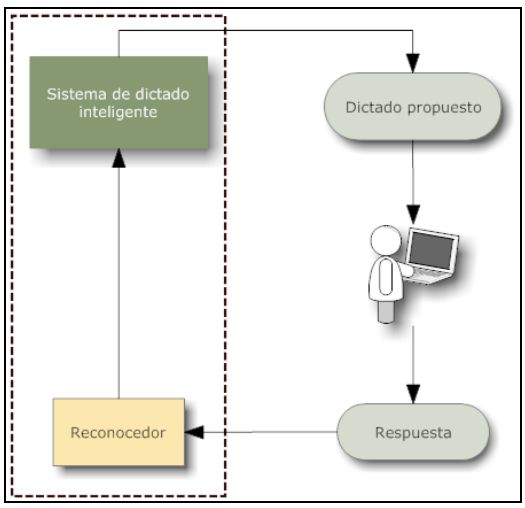


Figura 1 - Sistema de dictado musical inteligente

Dicho sistema constituiría un programa capaz de generar dictados musicales para que un usuario, que se supone estudiante de la música, fuera capaz de transcribir los datos sonoros a una partitura en papel o a un dispositivo capturador, con el fin de digitalizarlos. Posteriormente, el sistema de dictado analizaría la partitura de respuesta mediante un sistema de reconocimiento óptico de música (OMR) y proporcionaría un mensaje o unos resultados mostrando los fallos y los aciertos obtenidos.

Independientemente de estos elementos, se pueden buscar y encontrar más aplicaciones para los sistemas OMR. Para ilustrarlo, se verá el ejemplo de una editorial que quiera hacer nuevas ediciones de los libros antiguos de partituras musicales y que solo disponga del original en papel. La empresa podría elaborar un sistema OMR propio o utilizar alguno existente para digitalizar todas las partituras, y así poder lanzar nuevas ediciones. De forma similar, el autor de una obra musical manuscrita no tiene porqué digitalizar una a una las notas con un programa de ordenador o contratar a alguien para que lo haga, sino que puede utilizar el sistema OMR para digitalizar los símbolos manuscritos y obtener una versión digitalizada de su partitura. Así puede crear copias y modificar los elementos que considere oportunos.

Sin embargo, para este último ejemplo propuesto se deben tener en cuenta los pobres resultados que obtienen la gran mayoría de los múltiples sistemas OMR con partituras manuscritas, los cuales se centran en el reconocimiento de las notaciones modernas hechas a máquina. Esto plantea una serie de elementos a tener en cuenta para la superación de este desafío:

- **Carácter manuscrito de los símbolos:** cada autor puede proporcionar un estilo de escritura propio que diferencie claramente las notas de una clase con las notas de esa misma clase realizada por otro autor.

- **Varianza entre símbolos del mismo género:** un autor no genera todas las notas de una misma clase iguales. Aunque se parezcan, existen pequeñas diferencias entre cada símbolo que dificultará la tarea del reconocimiento.

Todas las ventajas comentadas de la digitalización de las partituras musicales junto a la necesidad de mejorar los sistemas OMR actuales para que trabajen adecuadamente con partituras manuscritas serán motivos que impulsen la creación de este proyecto. En él, se intentarán abordar los retos mencionados para crear un sistema OMR completo, que funcione para ambos casos de partituras manuscritas y no manuscritas.

1.2. Objetivos

El objetivo fundamental de este proyecto es la elaboración de un sistema de reconocimiento óptico musical (OMR) que permita detectar, a partir de la imagen en mapa de bits de la partitura (ya sea escaneada o extraída directamente del ordenador) los distintos símbolos musicales que lo componen, así como la duración de la nota y otros elementos del pentagrama. Dicho sistema también debe poder trabajar con notas aisladas para realizar la clasificación.

La creación del sistema se desarrollará con la capacidad de soportar notación musical moderna, ya sea manuscrita o no manuscrita. Para ello, debe contar con una base de datos de símbolos musicales que proporcionen los datos necesarios para realizar la correcta clasificación. La creación de dicha base de datos será uno de los objetivos colaterales del proyecto.

Para lograr el objetivo de la creación del sistema OMR se debe desglosar el problema en objetivos de más bajo nivel. La superación de las metas básicas permitirá la superación de las de mayor nivel:

1. Análisis del estado del arte referente al reconocimiento de partituras y símbolos musicales.

2. Creación de un bloque que adapte las características locales de la imagen de la partitura convenientemente

3. Diseño e implementación de un bloque que separe los pentagramas de la partitura y los símbolos de cada pentagrama

4. Creación de un clasificador que reconozca los símbolos proporcionados

5. Utilización de reglas de teoría musical para comprobación de errores y búsqueda de nuevos símbolos compuestos

6. Definición de un formato de representación de los datos digitalizados de la partitura

Además, el sistema debe cumplir una serie de requisitos de funcionalidad que se detallan a continuación:

- Debe trabajar con partituras manuscritas o no manuscritas
- Rápida ejecución del proceso completo
- Facilidad de uso
- Debe permitir trabajar con partituras en blanco y negro
- Debe permitir trabajar con partituras cuyas líneas de pentagrama sean de otro color distinto al de los símbolos
- Debe utilizar una serie de datos de configuración para ejecutar el programa

En cuanto a los requisitos de implementación, es necesario que el programa se construya de forma modular, mediante funciones que ejecuten determinadas tareas y conformen determinados bloques de funcionalidad.

1.3. Estructura de la memoria

El proyecto cuenta con nueve capítulos, los cuales disponen de diversos subapartados. Para la realización de este proyecto se ha intentado que los capítulos sean independientes y que sigan un orden concreto que ayude al lector a la correcta compresión del trabajo realizado.

La presente memoria se organiza como se explica a continuación

1. **Introducción**. En este capítulo se tratan los motivos fundamentales que han suscitado la creación del proyecto, los objetivos propuestos para su desarrollo y la estructura que se utilizará en la memoria.

2. **Estado del arte.** Presenta un estudio detallado de las investigaciones realizadas por otros autores en el campo del reconocimiento óptico de música. También explica algunas de las técnicas utilizadas para elaborar los sistemas OMR y determinados ejemplos de sistemas ya implementados.

3. **Gestión del proyecto.** Este capítulo resume las tareas en las cuales se ha organizado el desarrollo del proyecto y los tiempos de ejecución, e incluye un apartado con el presupuesto del sistema en el caso de que se tratara de un sistema comercial

4. **Descripción del sistema.** Consituye el resumen de todo lo que se ha hecho. Se podría considerar el apartado más importante de la memoria ya que detalla la resolución del sistema y las decisiones de desarrollo e implementación.

5. **Resultados.** En este capítulo de muestran los resultados obtenidos y se evalúa el sistema para comprobar el correcto funcionamiento de cada etapa y del conjunto global.

6. **Conclusiones.** Este capítulo presenta una serie de conclusiones que determinan si se han conseguido cumplir los objetivos marcados inicialmente y en qué medida, a partir de la observación de los resultados.

7. **Líneas de trabajos futuros.** El desarrollo del programa no constituye un trabajo cerrado, sino que se puede mejorar con la implementación de determinados métodos e ideas que se comentan en este apartado.

8. **Anexos.** Este capítulo incluye documentos de interés que completan el trabajo realizado con la creación del sistema y la redacción de esta memoria. Concretamente, se han definido cuatro anexos. El primero conforma una breve introducción al formato del lenguaje *Lilypond*. El segundo es el manual de usuario del programa y el tercero el manual de referencia. En el cuarto se muestran algunos de los símbolos que se han utilizado para crear determinadas bases de datos y ejemplos manuscritos.

9. **Bibliografía.** Contiene las referencias bibliográficas que se han consultado para realizar el proyecto.

2. Estado del arte

En la actualidad, el desarrollo de sistemas de reconocimiento de símbolos, sean caracteres, símbolos aislados, notas musicales u otros elementos, es llevado a cabo por empresas comerciales con el fin de obtener una mayor capacidad productiva. Estos sistemas facilitan el trabajo del usuario permitiéndole la posibilidad de digitalizar gran cantidad de datos de forma automática, rápida y sencilla. Mediante estos sistemas se pueden llevar a cabo diversas tareas, tales como el almacenamiento de documentos escaneados en formato textual, evitando formatos de mapas de bits (reduciendo así en gran medida la tasa de bits del fichero), la lectura de formularios de forma automática, la reproducción de una canción a partir de su partitura escaneada…

Aunque el campo que abarca el reconocimiento de símbolos es relativamente amplio (ya que se pueden enfocar los sistemas hacia el reconocimiento de cualquier cosa), los sistemas que se han desarrollado se han encaminado a detectar y reconocer símbolos bajos unas determinadas condiciones. En el caso del reconocimiento de caracteres, por lo general, se propone el uso de un estilo de escritura particular para el buen funcionamiento del sistema, de manera que la forma de escribir de cada usuario siga unas pautas determinadas, disminuyendo con esto la personalidad de la escritura. Algo similar ocurre con los formularios de lectura automática; el proceso requiere que las casillas que el usuario marque se rellenen de una forma y color determinado, evitando cruces o tachones. En el caso de la lectura de símbolos musicales, tendremos grandes problemas para reconocer partituras manuscritas ya que la mayor parte de los sistemas desarrollados han sido enfocados en el reconocimiento de notaciones hechas a máquina, evitando el aprendizaje del reconocedor, necesario para la escritura manual en la mayor parte de los casos.

El estudio del reconocimiento de notación musical nos plantea un problema de reconocimiento bidimensional. Por una parte, la dirección horizontal de dicho espacio puede estar asociada con la duración temporal de la nota mientras que la dirección vertical puede estar asociada con el tono.

Dentro de dicho estudio, la mayor parte del trabajo realizado abarca la lectura de partituras hechas a máquina, aunque en los últimos años la tendencia es que se investiguen sistemas capaces de soportar distintas notaciones y pentagramas manuscritos. La razón es obvia. Dada la explosión de contenidos digitales de hoy en día, se intenta digitalizar toda la información existente para poder analizarla y transmitirla de una forma eficaz. Esto nos permitirá, entre otros, crear grandes bases de datos de información disponibles para todo el público a partir de manuscritos antiguos que nos permitan estudiar de forma automática distintos estilos musicales.

No obstante, la tarea del reconocimiento OMR no será algo trivial. Como se verá más adelante el reconocimiento de símbolos musicales es un trabajo difícil, no sólo por el carácter personal que cada autor puede proporcionar a su partitura (en el caso manuscrito), sino también por la amplia variedad de combinaciones de los elementos musicales que debe ser capaz de soportar el sistema reconocedor de notas.

Para realizar esta tarea se dispone de dos procedimientos que engloban distintas técnicas y sistemas. Por un lado están los sistemas tradicionales de reconocimiento óptico de música – "*Optical Music Recognition - OMR*" – y por otro con los sistemas adaptativos de reconocimiento

óptico de música – "*Adaptive Optical Music Recognition - AOMR*" de Fujinaga[1]–. La diferencia fundamental entre ambos procedimientos reside en que los sistemas OMR son capaces de reconocer partituras y símbolos musicales, dentro de un conjunto limitado de símbolos y bajo unas determinadas directrices, mientras que los sistemas AOMR, más recientes, son capaces de adaptar su motor de reconocimiento para poder aprender nuevos símbolos y notaciones a partir del reconocimiento tradicional OMR.

El proyecto desarrollado se encuadrará dentro del grupo de los sistemas OMR tradicionales, utilizando técnicas que permitirán la utilización de partituras manuscritas y no manuscritas.

Aunque Fujinaga fuera uno de los autores pioneros en proponer un motor de reconocimiento adaptable, no será el único. Como se verá posteriormente, estos sistemas evolucionarán y llegarán a utilizar redes neuronales simulando el funcionamiento del cerebro humano para el reconocimiento de las notas.

Los sistemas OMR que se estudiarán en apartados posteriores engloban distintas etapas comunes: adquisición de la partitura, segmentación y clasificación de las notas. Como etapas adicionales se pueden citar representación en un sistema musical y su reproducción.

2.1. Revisión de artículos de otros autores

Para acercarnos a los orígenes del reconocimiento de música impresa, debemos acudir a los trabajos realizados por Pruslin[2] en 1966 y Prerau[3] a finales de 1960 y a principios de 1970, dos trabajadores del Instituto Tecnológico de Massachussets (MIT). Los dos trabajos doctorales realizados por estos investigadores han sido los puntos de partida para las investigaciones de otros autores. Sin embargo, la mayor parte de ellos se han realizado a comienzos de 1980, dada la disponibilidad de los escáneres ópticos. Algunos de los proyectos de investigación más recientes han sido publicados en números de la revista "*Computing in Musicology*"[4]

La mayor parte del trabajo de investigación realizado en esta área ha sido desarrollado por trabajadores Japoneses. Matsushima creó un sistema de visión de partituras para el robot *WABOT-2* [5][6] en 1985 y posteriormente, en 1988, integraría un sistema que facilitaba la producción de una partitura en braile[7]. Nakamura[8], en 1978, desarrolló un sistema de lectura de partituras con el objetivo de utilizarlo para crear una base de datos para analizar canciones del folclore japonés. Aoyama y Tojo[9], en 1982, también desarrollaron un sistema de lectura de partituras para almacenar la información de las partituras impresas en una base de datos.

Algunos de los otros trabajos de investigación que se realizaron en el área del reconocimiento de partituras impresas incluyen los trabajos realizados por Wittlich[10] en 1974, Andronico y Ciampa[11] en 1982, Tonnesland[12] en 1986, Martin[13] en 1987 y Roach y Tatum[14] en 1988.

Cabe destacar también el trabajo realizado por Michale Kassler[15][16], en 1970, quien publicó dos trabajos en los que se realiza un estudio sobre distintos puntos acerca del campo del reconocimiento automático de la música impresa. En la primera de las publicaciones, "*An Essay Towars the Specification of a Music Reading Machine*" define lo que una máquina debería ser capaz de reconocer mientras que en la segunda "*Optical Character-Recogntion of*

Printed music: a Review of Two Dissertations" discute los trabajos realizados por los pioneros del campo del reconocimiento de música impresa: Pruslin y Preau.

Cada uno de estos trabajos se enfocó de una forma distinta, para conseguir un objetivo determinado, aunque en todos los casos se tratasen de estudios relacionados con el reconocimiento óptico de música. Dichos objetivos abarcaban ámbitos que iban desde la formación de una base de datos (Aoyama y Tojo, Nakamura) hasta la provisión de un sistema de visión robótica (Matsushima). Para Nakamura, éstos se centraron en producir una base de datos de las canciones folclóricas japonesas con el fin de estudiarlas y manipularlas, así como reproducirlas mediante una partitura impresa, usando un ordenador. Aoyama y Tojo también centraron sus objetivos en producir una base de datos, pero enfocándolo de una forma más general, como parte de un sistemas interactivo de edición. En contraposición, los trabajos de Roach y Tatum fueron propulsados por la economía de la publicación musical, donde una gran cantidad de dinero se gasta en convertir los manuscritos de los compositores en notación a máquina.

De igual forma, los trabajos realizados intentaron solventar el problema del reconocimiento de distintas maneras. A modo de ejemplo, el trabajo de Pruslin y Prerau se dirigió para reconocer un número limitado de símbolos para aplicaciones de ámbito general mientras que el de Roach y Tatum se centró en el problema a bajo nivel, reconociendo las componentes de los símbolos.

Los dos autores del MIT descritos anteriormente, Pruslin y Prerau, destacaron la distinción entre aquellos símbolos que no varían su apariencia y, por tanto, pueden ser tratados como caracteres que faciliten su reconocimiento y los símbolos musicales que pueden tener variaciones en sus parámetros gráficos, como ocurre con las corcheas, ya que pueden aparecer aisladas o unidas a otras corcheas. Este último fenómeno lógicamente dificulta en gran medida el reconocimiento del símbolo.

A continuación, se detallarán en profundidad algunos de los trabajos realizados dentro del campo del reconocimiento musical.

2.1.1. Aoyama y Tojo (1982) [9]

La investigación realizada por estos autores en 1982, publicada solamente en japonés, contiene muchas de las técnicas que son usadas actualmente en distintas investigaciones sobre los sistemas de reconocimiento óptico musical. El sistema propuesto en este trabajo esta dividido en tres etapas: entrada, segmentación y reconocimiento. En la primera etapa la imagen de entrada es convertida a binario (imagen en blanco y negro), y se localizan las líneas del pentagrama. Posteriormente se obtiene la anchura entre dichas líneas, así como la altura entre ellas. En la siguiente etapa, segmentación, las líneas del pentagrama se eliminan y los símbolos son segmentados usando análisis por componentes conectados. Finalmente los símbolos segmentados son clasificados y verificados.

La etapa del reconocimiento puede ser ejecutada mediante algoritmos de reconocimiento de patrones o mediante análisis de la estructura de las notas. La entrada del sistema es una partitura impresa libre de símbolos rotos. Se permite que tenga cualquier tamaño y que las uniones entre las notas (corcheas, semicorcheas…) estén ligeramente borrosas debido al proceso de impresión o digitalización de la imagen. El sistema implantado por el autor utilizó un escáner de 254 dpi (puntos por pulgada – *dots per inch* –) con 8 niveles de grises.

Antes de ejecutar el proceso de segmentación y de reconocimiento, se debe llevar a cabo la detección del pentagrama para su eliminación. Esto permitirá eliminar información innecesaria que puede afectar a la eficacia de la etapa de reconocimiento. Para ello, una vez se obtiene la partitura escaneada, se realiza la proyección sobre el eje Y de la imagen digitalizada, obteniendo un histograma cuyos máximos indican los posibles candidatos a líneas del pentagrama. Sabiendo dónde se encuentran situadas, se procede a su eliminación.

La eliminación de las líneas del pentagrama puede causar que se produzcan cortes en los elementos musicales provocando que el sistema realice una segmentación excesiva, considerando las notas partidas u otros elementos como notas aisladas, lo cual resulta nocivo para el proceso de reconocimiento. Para evitar esto, las regiones de las líneas del pentagrama más próximas a las notas no se eliminan (ver Figura 2).

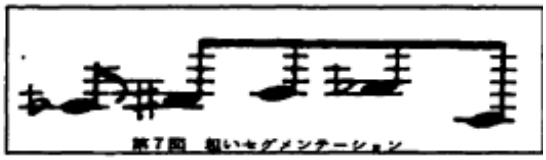


Figura 2 - Eliminación de las líneas exceptuando las regiones cercanas a las notas

Una vez eliminadas dichas líneas, se procede a la segmentación de notas. Para ello se buscan las notas que se encuentren sobre una línea o dos, mediante un ejemplo prototipo, para eliminarlas temporalmente. Esto se realiza para encontrar huecos dentro de las notas; huecos resultantes de la eliminación de las líneas del pentagrama. Una vez se detectan los símbolos se pueden marcar de forma que cuando se eliminen el resto de las líneas los símbolos no se fragmenten. Los huecos son detectados por el sistema mediante la búsqueda de pequeñas agrupaciones de píxeles blancos entre líneas del pentagrama. Una vez los huecos se marcan, las notas se pueden restaurar y las líneas nocivas se pueden eliminar sin problema.

La imagen resultante se segmenta a través del método de análisis por componentes conectados. La altura y la anchura de la caja imaginaria que rodea a cada segmento se utiliza para etiquetar de forma tosca los componentes conectados de píxeles en diez grupos (ver Figura 3). La altura y la anchura se normaliza usando la altura del espacio entre las líneas del pentagrama.

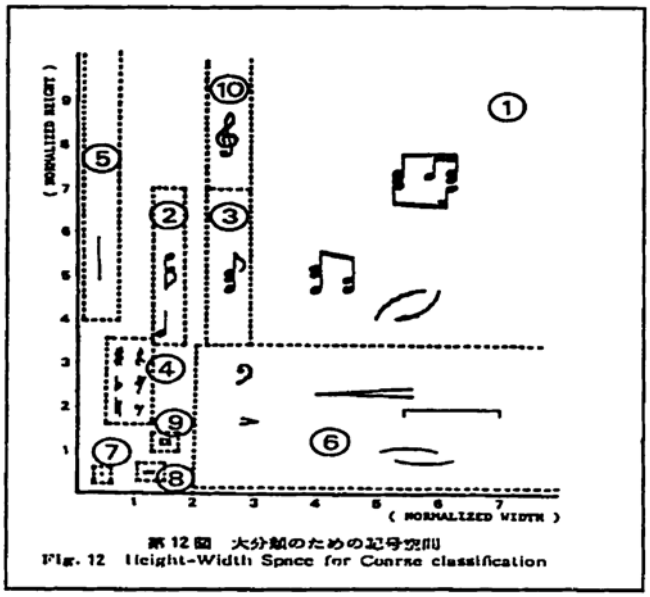


Figura 3 - Clasificación de los elementos musicales en 10 grupos según el tamaño del símbolo respecto a su anchura y altura

El grupo uno compuesto por notas y uniones de notas, las plicas y los corchetes de las notas, así como sus uniones, se separan de la cabeza de la nota mediante la eliminación de las regiones delgadas. Posteriormente se realiza un análisis de la nota para obtener parámetros tales como anchura, altura, centro de gravedad, cociente entre área y área de la caja imaginaria que recubre dicha nota, etc. Estos parámetros servirán para que el clasificador pueda llevar a cabo su trabajo de forma eficiente.

En el grupo cuatro, compuesto por silencios y alteraciones (sostenidos, bemoles, becuadros…), se utilizará un clasificador en árbol basados en códigos RLE horizontales y verticales para separar los miembros de esta clase.

Finalmente, las reglas del sintaxis que conciernen la posición de los símbolos y el número constante de compases en una medida son usadas para chequear el resultado del reconocimiento. Aunque no se desarrolló, el autor propuso la posibilidad de reconocimiento de marcas de cinética musical (*andante, vivace, presto*…).

2.1.2. Maenaka y Tadokoro (1983) [17]

El objetivo principal de estos investigadores fue construir un sistema que pudiera ser portátil, compacto, fácil de usar y barato. Para conseguirlo, utilizaron un microprocesador de 8 bit (MC6809) y una cámara de TV como dispositivo de entrada. También trataron la posibilidad de utilizar una maquina de telefax como alternativa. En la Figura 4 se muestra un ejemplo de la arquitectura del sistema.

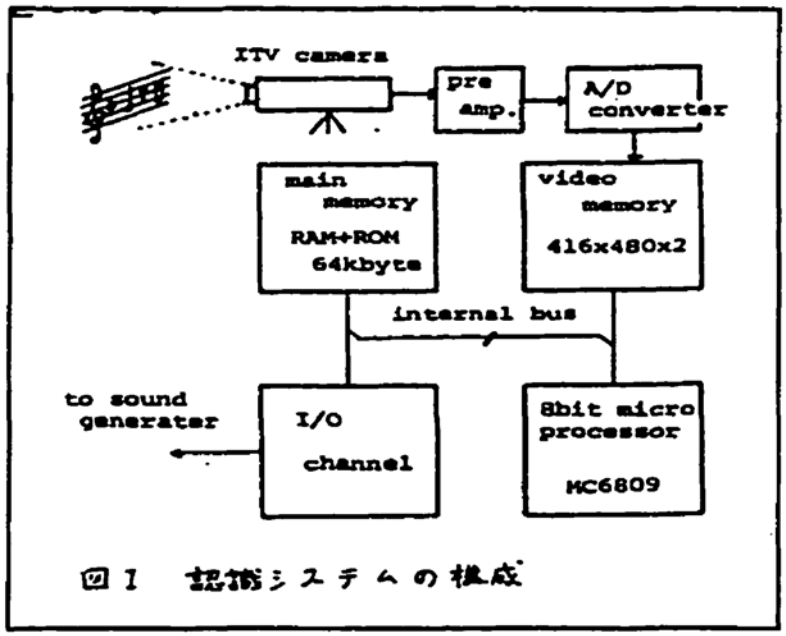


Figura 4 - Arquitectura del sistema de Maenaka y Tadokoro (1983)

La cámara de TV esta equipada con una lente de aumento y lentes de primer plano. Se utilizan lámparas de 100 W para iluminar. Debido a las limitaciones de la cámara, la hoja de la partitura de tamaño A4 tiene que ser dividida en cuatro secciones. Ajustando la ganancia del conversor analógico-digital y la iluminación, se elimina la necesidad de usar métodos para preprocesar la imagen. Un método de umbral simple es suficiente para el reconocimiento de patrones. Sin embargo, a causa de las características ópticas de las lentes de primer plano, las cuatro esquinas de la imagen se distorsionan. La investigación también discute el problema de cambiar el ratio de aspecto durante la adquisición.

En este proyecto se definen un conjunto mínimo de fuentes musicales que deben ser reconocidas por el sistema: la clave de Sol, las barras divisorias del pentagrama, las barras dobles, la barra final del pentagrama, símbolo de repetición de pentagrama, redondas, blancas, negras, corcheas, semicorcheas, uniones de corcheas y semicorcheas, silencio de redonda, de blanca y de negra, bemoles, sostenidos, becuadros, puntos y ligaduras.

La existencia de las plicas y el corchete de las corcheas puede ser determinado mediante el muestreo de las regiones fijas vecinas. Para distinguir entre una nota negra y una blanca, se utilizan dos algoritmos distintos dependiendo de si la nota esta situada encima de una línea de pentagrama o entre ellas.

Para la nota entre dos líneas, las líneas equidistantes del pentagrama son escaneadas de izquierda a derecha. Si el píxel negro cambia a blanco antes de que la note termine, la nota se considera blanca, en otro caso negra. Para la nota que esta sobre la línea del pentagrama, el área alrededor de la nota se escanea de forma vertical para buscar transiciones de blanco a negro. Ese escaneo es desarrollado varias veces en diferentes posiciones a lo largo del eje horizontal. Si sólo un número muy pequeño de líneas verticales tienen la transición, entonces se considera negro, en otro caso, blanco (ver Figura 5).

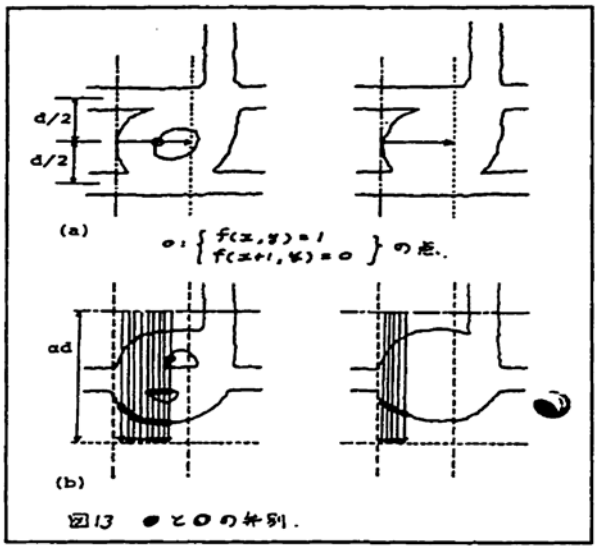


Figura 5 - Método desarrollado para identificar una blanca y una negra: a) caso de nota entre dos líneas de pentagrama, b) caso de nota encima de la línea

Los resultados de las pruebas de estos investigadores fueron diversos. Por una parte, debido a la pobre calidad de la imagen y el ruido, algunos de los algoritmos no fueron tan robustos como en un principio se esperaba. Además, debido al gran número de parámetros, tales como pesos para el muestreo fijo y la anchura del enventanado de la plica, las elecciones correctas fueron difíciles de elegir. Además, estos valores tuvieron que ser cambiados dependiendo del contraste de la imagen de entrada. La tasa de error ha sido menor de un error por imagen (1/4 de página). La precisión puede ser incrementada al aumentar los puntos de muestreo, pero esto también puede aumentar el tiempo de procesado. En general, dependiendo de la partitura, el proceso necesita de 4 a 10 minutos para procesar una línea de música monofónica.

2.1.3. Kim, Chung y Bien (1987) [18]

Este trabajo muestra un sistema completo OMR que utiliza una cámara de TV como dispositivo de entrada y un robot mecánico como reproductor. A diferencia del sistema *WABOT-2* de Matsushima (1985), este fue diseñado para reconocer partituras musicales con distinto tamaño de letra bajo unas condiciones de escasa iluminación y sin hardware especial. Las cinco etapas del sistema son: preprocesado, clasificación tosca, clasificación fina, chequeo de la sintaxis musical e interfaz para interpretar la música mediante el dispositivo.

El sistema realiza las siguientes presunciones:

- Los símbolos musicales son más oscuros que el fondo.
- Los símbolos están distribuidos de forma aleatoria en el pentagrama.
- La distancia entre dos símbolos es más grande que un cuarto de la altura entre las líneas del pentagrama.

En el preprocesado, la imagen en escala de grises se mejora mediante un operador convolucional Laplaciano de 3x3, para eliminar el desenfoque entre los símbolos adyacentes:

$$H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Después dicha imagen se convierte en imagen binaria mediante el establecimiento de un umbral adaptativo. Al mismo tiempo, cada elemento del pentagrama, como las líneas y símbolos se separan unos de los otros.

Para eliminar las líneas del pentagrama, cada punto de una línea, se mantiene si la vecindad vertical satisface una de tres condiciones: si solo un píxel de encima es negro, o si ambos de los dos píxeles de abajo son negros, o si los cuatro píxeles de encima y los cuatro píxeles de abajo contienen al menos cinco píxeles negros. De otra forma, el punto es eliminado.

La proyección sobre el eje X se utiliza para la segmentación de cada símbolo. Se utiliza una burda clasificación para cada símbolo segmentado utilizando la altura y la anchura de la caja imaginaria que rodea al símbolo después del normalizar la altura entre las líneas del pentagrama.

2.1.4. Carter (1988) [19][20]

Este autor propuso la idea de segmentar la partitura escaneada mediante la utilización de Grafos de Líneas Adyacentes (LAG). Este método consiste en el escaneo de la imagen de forma horizontal y vertical en busca de cadenas de píxeles negros. Estos forman nodos correspondientes a las uniones de los segmentos adyacentes que se solapan. El grafo obtenido se analiza después para detectar las líneas del pentagrama y los símbolos que se solapan con dichas líneas. Mediante la utilización de esta técnica, se obtienes los siguientes elementos:

- Identificación de las áreas vacías sin el pentagrama
- Identificación del pentagrama, incluso si la imagen esta rotada hasta 10 grados
- Identificación de las líneas del pentagrama que se encuentran ligeramente desviadas, rotas o cuyo grosor varia

Las fases de este sistema OMR son:

- Producción de LAG mediante el aislamiento de las partes vacías de las líneas del pentagrama. Esta etapa también aísla todos los símbolos y los grupos de símbolos conectados o solapados.
- Clasificación de los símbolos según el tamaño de la caja imaginaria que rodea al símbolo y organización de sus secciones constituyentes.

2.1.5. McGee y Merkley (1991) [21]

Estos investigadores trabajaron en el reconocimiento de notas musicales cuadradas procedentes de las notaciones musicales antiguas (ver Figura 6). La eliminación de las cinco líneas del pentagrama se realiza mediante la búsqueda de líneas horizontales delgadas de suficiente longitud. Al mismo tiempo son enderezadas. La clasificación se realiza usando un

conjunto de rectángulos que abarcan el área de una nota aunque los autores también experimentaron con un método originalmente desarrollado para la identificación de huellas dactilares. La resolución de entrada es de 300 dpi.

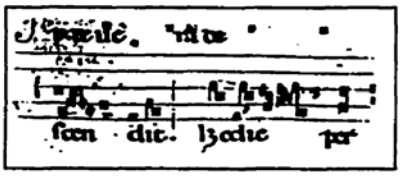


Figura 6 - Notación musical antigua

2.1.6. Miyao (1992) [22]

Las dos características principales de este sistema son que incorpora una gramática de notación musical para ayudar al reconocimiento, y que, de forma distinta a la mayor parte de los sistemas, las líneas del pentagrama son eliminadas después de que las notas sean extraídas.

Se realizan tres observaciones sobre las características de la notación musical:

- La posición de la clave, del compás y del tiempo pueden ser predichas a partir de la posición del pentagrama y las barras.
- Otros símbolos, incluyendo puntos, ligaduras, ligaduras de expresión, marcas de acento, etc., son posicionados de forma relativa a las notas dentro del pentagrama
- El tamaño de los símbolos va en concordancia a la altura del espacio entre las líneas del pentagrama.

Las etapas que forman el sistema son: detección de la posición de las líneas del pentagrama, búsqueda de notas y eliminación del pentagrama. Después de que las líneas del pentagrama se eliminen, los símbolos remanentes son agrupados toscamente en concordancia a su tamaño y posición y los símbolos son clasificados utilizando características estructurales o mediante la comparación con ejemplos.

Este sistema utiliza la transformada de Hough lineal para encontrar las líneas del pentagrama. Las barras separadoras del pentagrama se localizan utilizando proyecciones en el eje X. Las notas negras se extraen usando máscaras rectangulares. Las notas blancas se distinguen de las negras por el número de píxeles blancos en el área de la máscara. Una redonda y una blanca se distinguen mediante comparación con ejemplos.

Las notas candidatas encontradas fuera del pentagrama se verifican mediante la búsqueda de líneas adicionales. Si no se encuentran dichas líneas, la candidatura es revocada. Para ello, dada una nota, las líneas se buscan mirando a los bordes de la izquierda y de la derecha de la nota. Si no se encuentra, la candidatura de la nota es rescindida también. Existen reglas de notación que se utilizan para que al detectar estos elementos, los símbolos sean gramaticalmente correctos (por ejemplo, no son válidas más de tres líneas adicionales en una nota). Después de eliminar las líneas del pentagrama, las componentes conectadas son agrupadas, según la altura, anchura y la posición relativa sobre la mitad de las líneas del pentagrama.

Los símbolos de tamaño fijo agrupados de forma tosca son además clasificados usando patrones mallados de 6x6. El símbolo se divide en una malla de 6x6 y cada malla es representada por el cociente del número de píxeles negros a píxeles blancos. Los 36 números son representados como vectores y comparados con los vectores de los prototipos utilizando medidas de distancias Euclídeas. Los símbolos sin clasificar son reconectados insertando las líneas del pentagrama previamente eliminadas, y el cálculo de la distancia se repite. Para los símbolos que varían con el tamaño, como ocurre con las ligaduras de expresión, los píxeles con el mismo valor horizontal y vertical son usados para la clasificación. Finalmente, las reglas espaciales son usadas para finalizar la decisión de clasificación.

Mediante este sistema se consigue una eficacia del 93% al 98% con un tiempo de procesamiento de 3 a 20 minutos por página usando una estación Sony (NWS-821). El escáner de entrada tiene una resolución de 240 dpi.

2.1.7. Modayur (1992) [23]

Este sistema compuesto por dos niveles utiliza algoritmos morfológicos para la detección de símbolos a bajo nivel y un módulo a alto nivel que verifica la salida del módulo de bajo nivel. También incorpora notación de sintaxis para ayudar en el posicionamiento espacial de los símbolos. Los autores afirman que la tarea de reconocimiento puede ser realizada casi en tiempo real y logran eficacias del 95% en los ejemplos procesados, con una eficacia de pico del 99.7% para las negras y las corcheas.

Los autores realizaron algunas suposiciones:

- Las líneas del pentagrama, que son cinco, se encuentran igualmente espaciadas
- El tamaño de símbolos es relativo
- La imagen no tiene mucha distorsión por inclinación
- Las notas se posicionan de forma relativa a la duración de la misma
- Las alteraciones son situadas directamente delante de la nota que alteran
- Las plicas de las notas, en general, se sitúan debajo de la nota cuando son adjuntos a la izquierda de la nota y arriba cuando van adjuntos a la derecha
- Los silencios de negra se centran en el pentagrama
- El silencio de blanca toca la tercera línea por encima, mientras que el silencio de redonda toca la cuarta línea del pentagrama por debajo

El sistema es capaz de reconocer veinte símbolos: claves de Sol y de Fa, sostenidos, bemoles, redondas, blancas, negras, corcheas, silencio de negra, plicas de la nota, uniones entre notas, y medias uniones. El sistema funciona en una estación de procesamiento de imagen MVI-Genesis 2000 y tarda 2 minutos en procesar una imagen de 512 x 480 píxeles.

2.1.8. Roth (1994) [24]

El autor desarrolló un sistema OMR cuya salida se utilizaba de forma conjunta con el editor de notación musical Lipsia con el fin de reproducir de forma digital la partitura original. El sistema se compone de siete etapas:

- **Rotación**. Para corregir inclinaciones, la imagen es rotada por deformación horizontal y vertical. La cantidad de deformación se determina manualmente.

- **Estadísticas de cadenas verticales de píxeles con el mismo valor**. La media de las longitudes de las cadenas verticales de píxeles blancos y negros se usan para estimar la anchura de las líneas del pentagrama (a partir de los píxeles negros) y los espacios entre las líneas (a partir de los píxeles blancos). Se asume que el tamaño de todos los pentagramas en una página el mismo.

- **Localización y eliminación de líneas del pentagrama**. Las líneas del pentagrama son localizadas mediante la búsqueda de grupos de cinco picos en la proyección en el eje Y. Una vez localizadas, las líneas son eliminadas de la imagen. Para no afectar mucho a los símbolos, las líneas solo son eliminadas cuando su altura esta cerca del ancho global de las líneas.

- **Localización y eliminación de las líneas verticales**. Examinando la proyección X de cada pentagrama, se detectan las líneas verticales. Notar que cualquier línea vertical, siempre que sea suficientemente delgada, es eliminada.

- **Etiquetado de componentes conectados**. Las componentes restantes son identificadas. Se crea una lista de componentes y referencias de cada píxel al componente al que pertenece.

- **Reconocimiento de símbolos**. Antes de que los símbolos sean clasificados, las notas blancas separadas (debido a la eliminación de las líneas del pentagrama) se fusionan y las notas negras conectadas (debido a los acordes) se separan usando heurística. Roth empleó un complejo árbol de decisión para clasificar varios símbolos musicales usando las siguientes características: altura, anchura, área y centro de gravedad. La localización con respecto a otros componentes, líneas verticales y líneas del pentagrama también se toman en consideración.

- **Generación de documentación en notación Lipsia**. Finalmente el elemento reconocido es reproducido utilizando el editor de notación musical Lipsia.

2.1.9. Fujinaga (1996) [1][25]

Este investigador desarrolló como tesis doctoral un sistema OMR capaz de adaptarse a los símbolos musicales que se le presentan, o lo que es lo mismo, es capaz de aprender.

Dicha habilidad es el punto fuerte del sistema, denominado “Reconocimiento Óptico de Música Adaptativo” (AOMR), al permitir el aprendizaje de nuevos símbolos musicales y notaciones escritas a mano. Este también mejora la eficacia en el reconocimiento de estos objetos mediante el ajuste interno de parámetros.

La implementación del sistema adaptativo esta basado en el aprendizaje constante mediante ejemplos base, análogo a la idea de aprendizaje por ejemplos, que identifica objetos desconocidos por su similitud a uno o más ejemplos conocidos almacenados. El proceso entero de basa en dos algoritmos simples: el clasificador k-NN (*Nearest Neightbour*) y el algoritmo genético. Usando estos algoritmos, el sistema es capaz de incrementar su eficacia en el tiempo según se procesan más datos.

Este sistema esta compuesto fundamentalmente por una base de datos y tres procesos independientes:

- **Reconocedor**: localiza, separa y clasifica los símbolos musicales en categorías musicalmente significativas. Como ya se dijo, dicha clasificación esta basada en la regla k-NN y el algoritmo genético, la base de datos de símbolos y las características coleccionadas de sesiones anteriores.

- **Editor**: la salida del reconocedor es corregida mediante un operador humano musicalmente entrenado que utiliza un editor de notación musical. El editor puede proveer a la vez una previsualización visual y acústica de la salida. En este caso, el autor utilizó un editor musical de dominio público, el "*Nutation*" de Glen Diener, el cual muestra y reproduce el resultado del proceso de reconocimiento. El resultado obtenido se almacena en la base de datos de símbolos mediante el clasificador y la etapa de aprendizaje.

- **Aprendizaje**: esta etapa mejora la rapidez y eficacia de futuras sesiones de reconocimiento mediante el reordenamiento continuo de la base de datos y la optimización de las estrategias de clasificación.

Hay que destacar que el sistema puede disminuir el tiempo de reconocimiento propio. En el sistema de clasificación k-NN, el tiempo de reconocimiento es proporcional al tamaño de la base de datos. Según esto, la reducción de la base de datos implica la reducción de tiempo de clasificación. Esto también se traduce en un mayor grado de error y una menor precisión.

2.1.10. Bainbridge (1991 - 1997) [26][27][28][29]

En algunos de los documentos de Bainbrigde citados en la bibliografía (ver "*An extensible optical music recognition system*" [26] y "*Optical Music Recognition: a generalised approach*" [28]) el autor describe un sistema capaz de reconocer diferentes notaciones musicales como la notación común, la notación de percusión o las tablaturas (comúnmente utilizadas para guitarras). Para ello, en primer lugar describe una manera de detectar las líneas del pentagrama mediante la proyección vertical de los píxeles negros de cada fila. Antes de proceder a la eliminación de dichas líneas, se realiza un procesado OCR para eliminar todo el texto presente en la partitura.

Este autor también introdujo un lenguaje denominado "Primera" que permitía especificar cómo ensamblar las primitivas juntas. Es común encontrar sistemas cuyas primitivas han sido codificadas de forma que es difícil extender el sistema. El lenguaje soporta cuatro tipos de técnicas de reconocimiento de patrones: proyecciones, transformada de Hough, *template matching* (reconocimiento mediante prototipos) y técnicas *slicing* (ver "*Preliminary Experiments in Musical Score Recognition*" [29] del mismo autor).

El sistema fue probado con diferentes ejemplos de música con notación tradicional y partituras de cantos Gregorianos, obteniendo resultados satisfactorios.

2.1.11. O^{3}MR (2001) [30]

Bellini, Bruno y Nesi en su trabajo "*Optical music sheet segmentation*" describen un sistema denominado O^{3}MR (*Object Oriented Optical Music Recognition*) cuyo módulo de segmentación no elimina las líneas del pentagrama en contraposición al resto de los sistemas. Para realizar la segmentación, el sistema se basa en perfiles de proyecciones para extraer símbolos básicos que contribuyen a la formación de la partitura musical. Esta arquitectura ha sido diseñada para trabajar únicamente con la notación tradicional con música monofónica en la que los pentagramas están compuestos por cinco líneas. La arquitectura de este sistema comprende cuatro etapas:

- Segmentación
- Reconocimiento básico de símbolos
- Reconocimiento de símbolos de notación musical
- Refinado del modelo de notación musical

El proceso de segmentación de la arquitectura O^{3}MR se puede dividir en tres niveles:

- **Nivel cero**: se encarga de obtener el grosor de las líneas del pentagrama y la distancia entre ellas. Estos dos parámetros servirán para aislar cada pentagrama de la partitura original, almacenándolos en imágenes en las que sólo aparezca un pentagrama.

- **Nivel uno**: este nivel trabaja con los segmentos producidos por el nivel anterior para extraer segmentos de la imagen que contengan símbolos musicales

- **Nivel dos**: se encarga de descomponer la salida del nivel uno en símbolos básicos (cabeza de la nota, plicas, uniones…)

El módulo de reconocimiento de símbolos básicos se encarga de reconocer los símbolos que han sido obtenidos por el nivel dos del proceso de segmentación. Esta salida se normaliza y se inserta en la entrada de una red neuronal, que es la que se encarga del proceso de reconocimiento de símbolos de notación musical.

2.2. Software disponible

El primer producto comercial que permitió la digitalización de una partitura musical mediante técnicas OMR fue lanzado en 1993 a cargo de la empresa *Musitek*[31]. Esta empresa fundada en Mayo de 1991 por Christopher Newell llevó a cabo una serie de investigaciones en el campo del reconocimiento musical para desarrollar y lanzar en Agosto de 1993 el programa "*MIDISCAN*" para Microsoft Windows. Este programa, pionero en software OMR, esencialmente escaneaba una partitura y la reproducía. Las primeras versiones tenían una tasa de acierto del 25% de la notación total de la partitura, lo que se traducía en que el usuario tenía que arreglar posteriormente el 75% que se había reconocido mal. Después, con la versión 2.0 del programa, la tasa de reconocimiento aumentó a un 80 %, destacando la capacidad de leer sobre partituras escritas a mano, con una tasa de reconocimiento del 50 %.

Años más tarde, la empresa *Musitek* cambió su enfoque para desarrollar un producto nuevo que integrara la tecnología desarrollada de escaneo de música junto a un editor de partituras

en ordenador, utilizando MIDI para grabar y sintetizar la música, creando "*SmartScore*" para *Microsoft Windows* y *Macintosh*, en 1996.

Posteriormente, la empresa londinense *Neuratron* [32] lanzó en invierno de 1997 el software "*Optical Manuscript*" para plataformas *Acorn* [33][34]. El lanzamiento de este programa estaría respaldado por "*Sibelius*", un importante programa de edición de partituras que integró la capacidad de trabajar con los archivos del programa de *Neuratron* en su versión 7. Con la decadencia de la plataforma *Acorn*, la empresa decidió crear nuevas versiones del programa para *Microsoft Windows* y *Macintosh*. Estas versiones fueron renombradas a "*Neuratron PhotoScore*" y vieron la luz por primera vez incluidos en una versión de "*Sibelius*" a finales de 1998 (para la versión de *Windows*) y 1999 (para la versión de *Macintosh*).

Sin embargo, la contribución más importante de dicha empresa en el campo del reconocimiento musical fue realizada en Junio de 2007 cuando, después de lanzar varias versiones del programa, se puso a la venta "*PhotoScore Ultimate 5*", el primer producto del mundo ideado para reconocimiento de música escrita a mano, que también incluía un sistema de reconocimiento de música impresa, convirtiéndose también en el primer programa con dos motores de reconocimiento musical.

A partir de entonces, se han desarrollado multitud de programas de reconocimiento basados en técnicas OMR. Actualmente la eficacia de estos programas puede llegar al 99% cuando la partitura escaneada esta limpia (no aparecen deformaciones, ni manchas, ni cortes y discontinuidades en la imagen) y la notación que se utiliza no contiene símbolos inusuales (símbolos que, aunque existen, son relativamente poco utilizados).

2.2.1. OMR comerciales

A continuación se mencionará parte del software OMR comercial que podemos encontrar, aparte de los ya tratados:

- **Capella – Scan** [35]: este programa creado por la empresa *Capella*, permite capturar la partitura impresa, detectar los símbolos y reproducirlos. Incluye la posibilidad de exportar la partitura a distintos formatos, entre los que destaca el formato MusicXML, utilizado por programas como "*Sibelius*" o "*Finale*". No permite la corrección de las notas mal reconocidas dentro del programa, por lo que hay que utilizar otro software para llevar a cabo esta tarea. También incluye un motor OCR de reconocimiento de texto para extraer las letras de las canciones en el caso que el pentagrama lo tuviera. No permite trabajar con partituras hechas a mano.

- **SharpEye** [36]: este programa desarrollado por *Visiv*, sólo disponible para *Microsoft Windows*, permite convertir la imagen escaneada de un pentagrama a un fichero MIDI, NIFF o con formato MusicXML. También incluye un editor para poder corregir las notas mal detectadas. Destacar que este editor no funciona con partituras hechas a mano.

- **PDFtoMusic** [37]: este programa creado por la empresa *Myriad* permite extraer la partitura de una imagen escaneada a partir del archivo PDF que contiene dicho documento. El programa, aparte de extraer la partitura, permite reproducirla o exportarla a otros formatos (MIDI, MusicXML, Myr, WAV, AIFF…). Funciona tanto en *Microsoft Windows* como en *Macintosh*.

- **VivaldiScan** [38]: otro programa más con capacidades de reconocimiento musical desarrollado por *Mediasoftel* que permite obtener la partitura a partir de una imagen para posteriormente corregirla y exportarla a distintos formatos. Cuenta con un grado de eficacia del 99 % sobre partituras limpias o casi limpias. Sólo para *Microsoft Windows*.

- **Optical Music easy Reader (OMeR)** [39]: este programa, disponible para *Microsoft Windows y Macintosh*, permite escanear una partitura y digitalizarla mediante técnicas OMR. Ha sido desarrollado por la empresa *Myriad*, la misma que desarrolló "*PFDtoMusik*". Este programa no permite la modificación ni reproducción de la partitura por lo que para llevar a cabo estos fines se debe recurrir a otros programas de la misma casa como "*Melody Assistant*" o "*Harmony Assistant*".

2.2.2. OMR no comerciales

A continuación se describirán algunos de los programas de código libre existentes que no necesitan licencia para su uso. La mayor parte de estos surgen como trabajos de investigación en determinados departamentos de instituciones o universidades muchas veces como proyectos para resolver determinados problemas que no son capaces de solventar correctamente los productos comerciales. El principal problema de estos sistemas es que no suelen contar con un grado de madurez equiparable al de un sistema comercial, por lo que es común que dichos proyectos sean mejoras de un sistema más desarrollado.

- **Gamera** [40]: este programa constituye un proyecto de la Universidad *Johns Hopkins* realizado por Ichiro Fujinaga, Michael Droettboom y Kart MacMillan. Funciona en *Linux* y en sistemas como *Microsoft Windows y Macintosh*. Se trata de una utilidad, de código abierto, para reconocimiento general que se puede entrenar para crear sistemas de reconocimiento específicos, del tipo OCR u OMR. Mediante el entrenamiento por parte del usuario el sistema es capaz de crear determinados motores de reconocimiento, para cualquier ámbito, que consiguen un elevado grado de eficacia al reconocer símbolos.

  Escrito en lenguaje *Phyton*, su arquitectura cuenta con una estructura modular que alberga cinco fases:

  - **Pre-Procesado**: se realizan una serie de operaciones iniciales sobre la imagen tales como la eliminación de ruido y desenfoque, girar la imagen si esta ligeramente rotada, ajuste del contraste, conversión a binario (blanco y negro) y operaciones morfológicas. Este paso es esencial ya que adapta la imagen para que los sistemas posteriores puedan funcionar con un alto nivel de eficacia.

  - **Segmentación y análisis del documento**: se analiza la estructura general del documento, se segmenta en secciones y se identifican y eliminan algunos elementos (por ejemplo líneas del pentagrama para el caso de partituras musicales). Esto permite aislar los símbolos para su correcta identificación.

  - **Segmentación y clasificación de los símbolos**: esta etapa es el núcleo del sistema. El programa permite ejecutar distintos segmentadores y clasificadores. El sistema utiliza un formato XML genérico y flexible para

almacenar los datos de la clasificación. Se ha implementado el algoritmo k-NN (*Nearest Neighbor*) y se ha utilizado un algoritmo genético para optimizar los pesos de algoritmo anterior.

- **Análisis sintáctico o estructural**: este proceso reconstruye el documento en una representación semántica de los símbolos individuales.

- **Salida**: se almacenan los resultados en formato XML

- **Audiveris Music Scanner** [41]: este programa es la primera herramienta OMR de código abierto escrita en Java. Básicamente este programa es un módulo capaz de convertir una partitura impresa en un fichero de formato MusicXML. Al ser escrito en *Java*, es multi-plataforma y por tanto soporta multitud de sistemas operativos (*Microsoft Windows, Macintosh, Linux, Solaris*…). Actualmente, esta herramienta sólo admite el reconocimiento de partituras hechas a máquina, obteniendo malos resultados en las partituras hechas a mano.

  Para llevar a cabos sus objetivos, el programa hace uso de algunas técnicas de procesado tales como: *LAG* - Grafos de Líneas Adyacentes - , etiquetas, y redes neuronales.

Existen otros programas como "**OpenOMR**" [42], que no se detallarán porque la lista de programas existentes es muy extensa.

2.3. Técnicas utilizadas

En este apartado se detallarán algunos de los procedimientos descritos en apartados anteriores, que se utilizan en el proyecto desarrollado, para su mejor comprensión.

2.3.1. Codificación RLE

La codificación RLE (*Run Length Encoding*) se utiliza en muchos de los algoritmos anteriormente descritos para detectar el grosor de las líneas del pentagrama y la altura del espacio entre dichas líneas.

Este algoritmo es uno de los algoritmos de codificación más simples que existen y se aplica generalmente para cadenas de datos. El algoritmo se encarga de contar cadenas sucesivas del mismo valor, almacenando el valor y la cantidad por pares.

Para entender el algoritmo se recurrirá a un sencillo ejemplo. Teniendo la siguiente cadena de valores binarios (donde los píxeles negros toman el valor 0 y los blancos toman el valor de 1):

1,1,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0

Se puede codificar mediante el algoritmo RLE, representándolo de la siguiente manera:

(1,2)(0,4)(1,2)(0,2)(1,4)(0,4)(1,6)(0,3)

Como se puede observar se trata de un algoritmo de codificación sin pérdidas, ya que permite reconstruir la imagen original de forma exacta, sin variación alguna.

2.3.2. Proyecciones X e Y

Se ha mencionado en multitud de algoritmos OMR el uso de proyecciones X e Y para detectar diferentes elementos, como pentagramas. Estas proyecciones construyen un histograma que nos muestra en un determinado eje la cantidad de píxeles negros existentes a lo largo de la dirección perpendicular al eje.

La proyección X resulta de la proyección de la imagen binaria en el eje X. El resultado es un vector cuyos valores de cada elemento se obtienen a partir de la suma de píxeles de cada columna de la imagen. Por otro lado, la proyección Y se obtiene a partir de la suma de píxeles de cada fila de la imagen.

En la Figura 7 se puede ver un ejemplo gráfico de las proyecciones X e Y de un pentagrama:

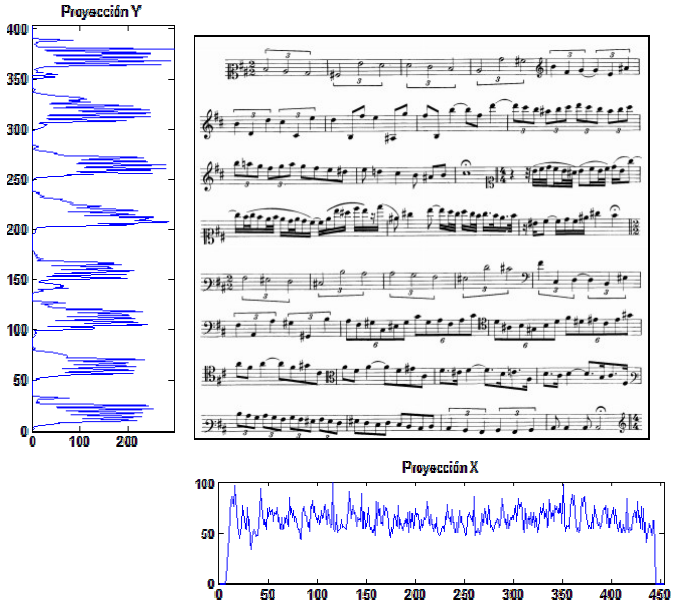


Figura 7 - Proyecciones X e Y de una partitura

Gracias a estas proyecciones se puede detectar en la figura anterior los distintos pentagramas que forman la partitura. Utilizando un umbral sobre la proyección Y podremos determinar que puntos máximos de la proyección indican la presencia de una línea de pentagrama. En este ejemplo el uso de la proyección X es trivial, ya que no nos aporta ningún tipo de información relevante. Si dicha proyección la utilizásemos sobre un pentagrama aislado, podíamos utilizar el histograma resultante para determinar en qué puntos existe un elemento musical (ver Figura 8)

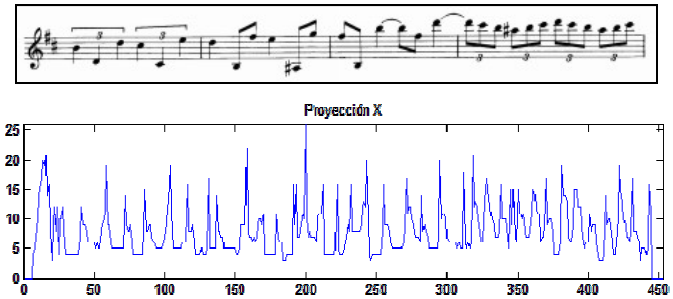


Figura 8 - Proyección X de un único pentagrama

Como se verá en apartados sucesivos, las distorsiones por rotación dificultarán en gran medida la tarea de detección de pentagramas mediante proyecciones.

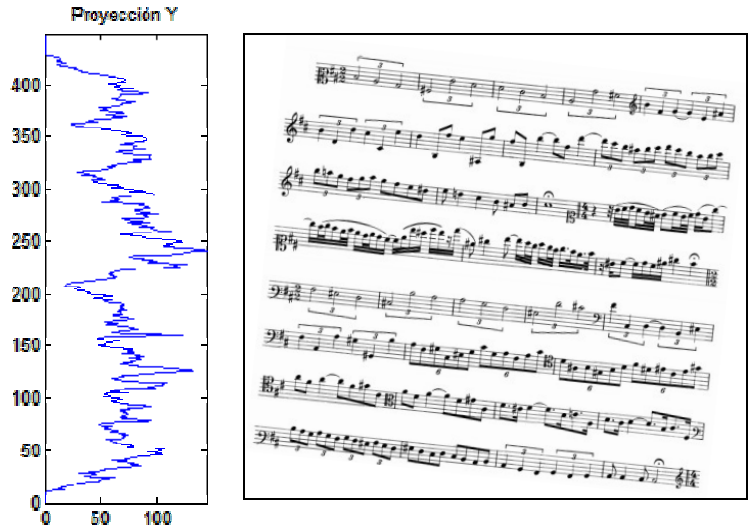


Figura 9 - Proyección Y de una partitura con 5º de rotación

2.3.3. Transformada Discreta de Fourier

Un problema que se presenta a la hora de realizar las proyecciones de las partituras es el hecho de que la imagen fuente tenga algún tipo de distorsión relacionado con la rotación de la página. En el caso de que exista un ángulo, aunque sea mínimo, con respecto a la horizontal, se obtendrán muy malos resultados. Este fenómeno lo podemos observar en la Figura 9.

Para solucionar este problema se puede recurrir a algún algoritmo que sea capaz de detectar la rotación para corregirla. Para implementar esta utilidad se puede recurrir a la transformada discreta bidimensional de *Fourier*. En la Figura 10 se pueden ver los resultados obtenidos al aplicarla sobre un ejemplo:

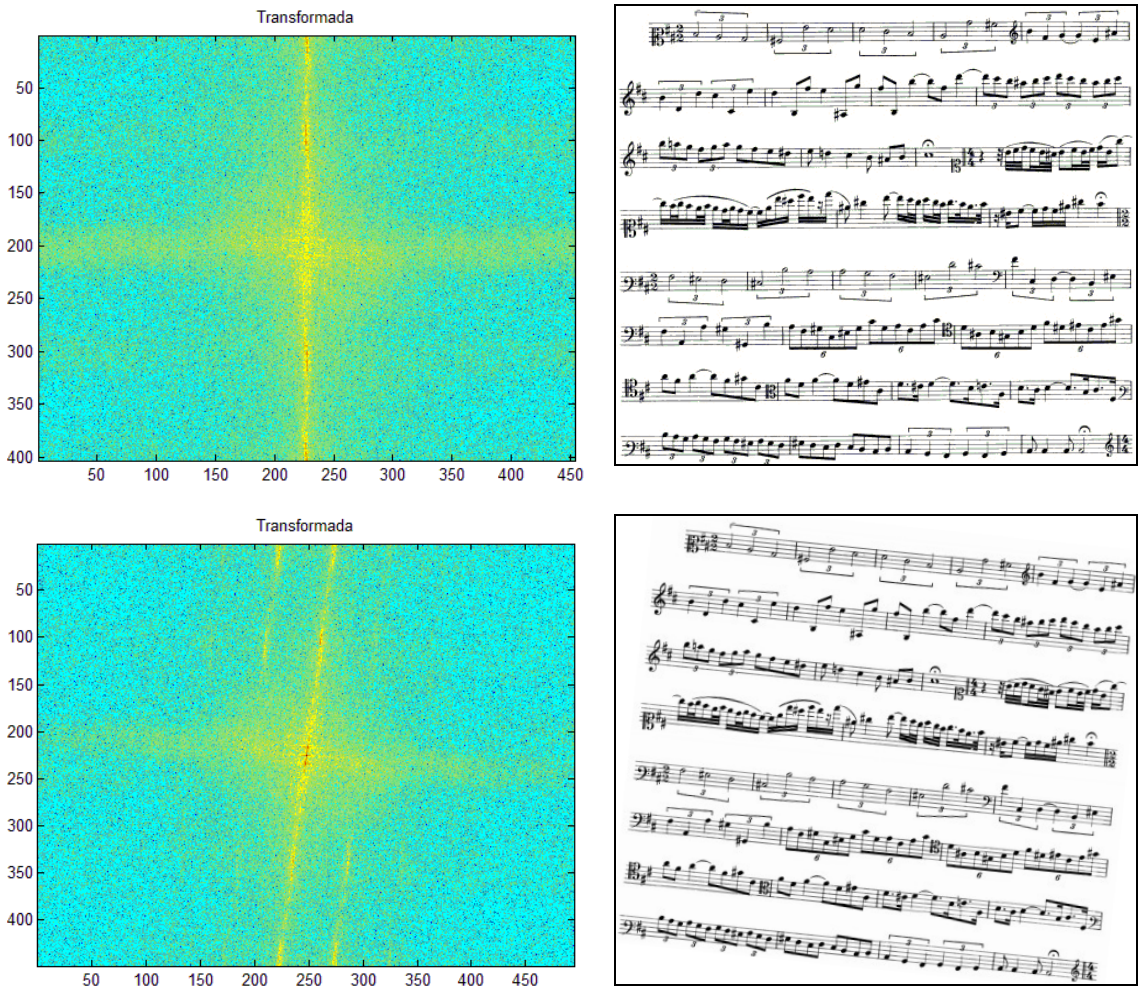


Figura 10 - Arriba: transformada de Fourier de una partitura; Abajo: transformada de la misma partitura rotada 5º

Se puede ver que la transformada de la partitura inclinada se encuentra rotada el mismo número de grados que la partitura. Determinando el ángulo de la transformada se podrá rotar la partitura distorsionada.

La transformada discreta de *Fourier* (DFT) para una imagen M x N y su inversa son definidas de la siguiente forma:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{\left(-j u \frac{2\pi}{M} x - j v \frac{2\pi}{N} y\right)}$$

donde $j = \sqrt{-1}$

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{\left(j u \frac{2\pi}{M} x + j v \frac{2\pi}{N} y\right)}$$

La función f(x,y) representa el píxel de la imagen en la coordenada (x,y) y toma valor real positivo. Por otra parte, F(u,v) es una función que representa la transformada de *Fourier* discreta, mediante números complejos. Estos los representaremos mediante su módulo y la fase en lugar de hacerlo mediante la parte real e imaginaria.

El módulo y la fase de un número complejo vienen definido según las expresiones:

$$Modulo(F) = \sqrt{real(F)^2 + imaginaria(F)^2}$$

$$Fase(F) = \arctan\left(\frac{imaginaria(F)}{real(F)}\right)$$

El módulo indica cuántas componentes de una cierta frecuencia están presentes, mientras que la fase indica la localización de las componentes frecuenciales.

La DFT suele representarse mediante el módulo, centrando el origen de coordenadas (0,0) en el centro de la imagen, de modo que las componentes de baja frecuencia queden centradas y las altas frecuencias en los extremos de la imagen.

Para implementar dicha transformada se utilizará la transformada rápida de *Fourier* (FFT), un algoritmo que obtiene el mismo resultado que la DFT con sólo n·log(n) operaciones aritméticas (en lugar de n^2 operaciones mediante el uso de la transformada directa), lo que se traduce en una reducción del tiempo de procesamiento.

Sin embargo, la detección de inclinación no será la única tarea que pueda ejecutar la transformada descrita. La versión unidimensional permite extraer los descriptores de una imagen para definir su morfología.

La transformada unidimensional discreta de *Fourier* viene dada por la expresión:

$$X(k) = \sum_{j=1}^{N} x[j] \cdot \omega_N^{(j-1)(k-1)}$$, donde $\omega_N = e^{(-2\pi \cdot i)/N}$

Mientras que la transformada inversa:

$$x(j) = \frac{1}{N} \sum_{k=1}^{N} X[k] \cdot \omega_N^{-(j-1)(k-1)}$$

Si se determina el contorno de una imagen, se expresan los pares de coordenadas como números complejos y se realiza su transformada se obtendrán los descriptores definidos. Éstos son capaces de caracterizar de forma cuantitativa la morfología de la imagen. Prueba de ello es que, mediante los primeros descriptores, utilizando la transformada inversa, podremos recuperar la imagen de forma tosca, aumentando en detalle según vamos utilizando más descriptores.

Este concepto es muy importante en el campo de reconocimiento, puesto que, gracias a estos descriptores, podemos comparar varias imágenes de forma cuantitativa comparando sus vectores descriptores.

2.3.4. Etiquetado por componentes conectadas [44]

La técnica de etiquetado por componentes conectadas escanea una imagen y agrupa los píxeles, en distintas componentes, basándose en la conectividad de estos. Esto significa que todos los píxeles que forman parte de un mismo componente comparten valores similares de intensidad y están, de alguna forma, conectados unos con otros. Una vez todos los grupos se han determinado, cada píxel se etiqueta dentro de un grupo, en concordancia al componente al que fue asignado.

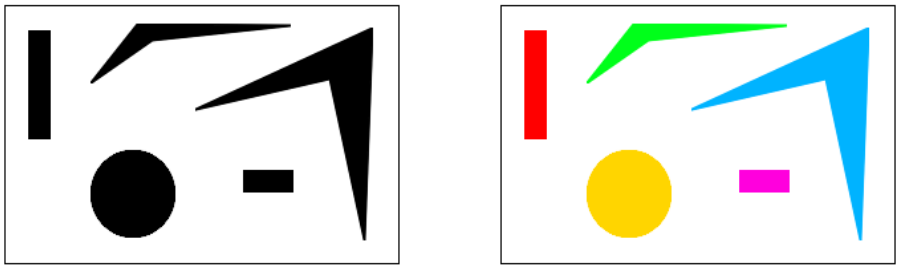


Figura 11 - Etiquetado de las figuras de una imagen

Este algoritmo puede trabajar en imágenes binarias y en imágenes con formato de escala de grises etiquetando mediante distintos tipos de conectividades. A continuación se hablará de algunos de los conceptos relacionados con este algoritmo.

2.3.4.1. Vecinos de un píxel

Se denominan vecinos de un píxel a los píxeles que le rodean. Siendo “p” un píxel con coordenadas (x,y) existen dos vecinos horizontales y dos verticales cuyas coordenadas son:

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$$

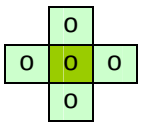


Figura 12 - 4 vecinos de un píxel

Además de estos cuatro vecinos de “p”, tenemos otros cuatro diagonales:

$$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$$

Estos píxeles junto a los cuatro anteriores conforman los conocidos como ocho vecinos de “p”. Cada uno de ellos se encuentra a una unidad de distancia de “p”. Conociendo estos datos, podemos hablar de las distintas relaciones de conectividad.

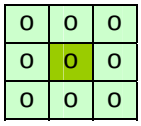


Figura 13 - 8 vecinos de un píxel

2.3.4.2. Conectividad

Este concepto es fundamental para poder establecer los bordes de objetos y componentes de las regiones de una imagen. Para saber si dos píxeles están conectados, únicamente debe conocerse si pertenecen a algunos de los conjuntos de píxeles 4-vecinos y 8-vecinos del otro píxel. De la misma forma, para establecer distintas regiones, es necesario determinar si los niveles de grises cumplen un determinado criterio de similitud. Definiremos dos tipos de conectividad:

- **4-conectividad**: un píxel "a" con valor similar de intensidad a otro píxel "b" será 4-conectivo si está entre los píxeles verticales y horizontales de b

- **8-conectividad**: análogo a la 4-conectividad, pero teniendo en cuenta también los píxeles diagonales

2.3.5. k-Nearest Neighbors (k-NN) [45][46]

El algoritmo k-NN es un método de clasificación de objetos dentro del reconocimiento de patrones basado en el entrenamiento mediante ejemplos similares en un espacio dado. Fue publicado por primera vez por los investigadores E. Fix y J. Hodges en su escrito "*Discriminatory Análisis: Nonparametric Discrimination: Consistency Properties*" [46] en 1951. Este algoritmo se encuentra entre los métodos más simples existentes de aprendizaje máquina (pero no por ello, peor). La idea en la que se basa consiste en clasificar un objeto según la clase a la que pertenezcan los "*k*" vecinos más cercanos. Dichos vecinos se toman a partir de una serie de elementos para los cuales se conoce el valor correcto de la clasificación. Esto se puede tomar como la etapa de entrenamiento del algoritmo, aunque no se requiere de una etapa como tal de forma explícita.

Para identificar los objetos, estos se representan mediante vectores p-dimensionales de características en un determinado espacio multidimensional y se comparan con los vectores de los vecinos conocidos mediante distancia Euclídea (Figura 14) u otras medidas, como la distancia Manhattan. Esto permite dividir las regiones del espacio en distintas clases de objetos.

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p} (x_{ir} - x_{ir})^2}$$

Figura 14 - Distancia Euclídea

La fase de entrenamiento del algoritmo consiste en almacenar los vectores de características y las etiquetas de las clases de los ejemplos de entrenamiento. En la siguiente etapa, clasificación, se presenta la evaluación de un ejemplo en el espacio de características. Para ello, se calcula la distancia entre los valores almacenados y el nuevo vector, seleccionando los "k" ejemplos más cercanos al vector. El nuevo ejemplo se clasifica con la clase que más se repite entre los vectores seleccionados. En la Figura 15 podemos ver un ejemplo gráfico de lo propuesto. Siendo un espacio de tres clases distintas, debemos clasificar el elemento amarillo. Si K = 3, primer circulo, clasificaremos el ejemplo desconocido como un triángulo, dado que hay 2 triángulos frente a un único círculo. En cambio, si K = 9, se clasificará como un círculo.

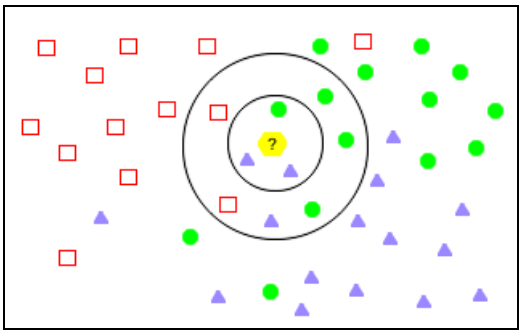


Figura 15 - Clasificación del elemento amarillo utilizando el algoritmo k-NN con k = 3 y 9

Este método supone que los vecinos más cercanos nos dan la mejor clasificación, pero ésta puede ser una mala suposición si existen muchos atributos irrelevantes dentro de los vectores de características. Es pos eso que es de gran importancia la realización de una buena extracción de características que represente de forma adecuada las propiedades de cada elemento, permitiendo la fácil distinción entre ellos.

Sin embargo, muchas veces no es posible realizar una buena extracción de parámetros que permita discernir adecuadamente entre los distintos elementos. Para corregir este fenómeno podemos recurrir a la asignación de pesos a la distancia de cada atributo, dando así mayor importancia a los más relevantes. También se pueden obtener estos pesos ajustando los parámetros mediante ejemplos de entrenamiento conocidos.

Un factor a tener en cuenta en la implementación del algoritmo es la elección del valor de “k”. Este depende fundamentalmente de los datos. Por regla general, valores grandes de “k” reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas. Un buen “k” puede ser seleccionado mediante una optimización de uso. El caso especial en que la clase elegida es la clase más cercana al ejemplo de entrenamiento, cuando k = 1, se llama “*Nearest Neighbor Algorithm*”, algoritmo del vecino más cercano.

Aunque puede parecer que lo más lógico es clasificar el elemento a evaluar como el símbolo más cercano (hablando en términos de distancias entre vectores de características, el símbolo más cercano es el que cuenta con menor distancia al vector del símbolo evaluado, que no tiene porqué corresponderse con el símbolo más parecido), la realidad es que la utilización del vecino más cercano no es una buena elección en todos los casos. Un valor “k” distinto de uno nos asegura que se evaluarán distintos símbolos y que se etiquetará el más frecuente dentro del grupo de los “k” vecinos más cercanos, lo que nos permite rechazar aquellas muestras no representativas (*outliers*) en el caso de que sean las que más de acerquen a nuestro símbolo a etiquetar.

Existen algunas variantes del algoritmo básico, una de las más interesantes consiste en ponderar la contribución de cada vecino de acuerdo a la distancia entre él y el ejemplar a ser clasificado, dando mayor peso a los vecinos más cercanos. Por ejemplo, se puede asignar el peso de cada vecino de acuerdo al cuadrado inverso de sus distancias.

2.3.6. Transformada Radon

La Transformada Radon bidimensional se encarga de realizar las proyecciones de una imagen a lo largo de unas direcciones específicas, codificando la imagen como un conjunto de proyecciones. Para entender este concepto de forma intuitiva, la transformada realiza

proyecciones en función de un eje imaginario que gira alrededor de la imagen, según los ángulos que se le describan, como se puede observar en la figura siguiente:

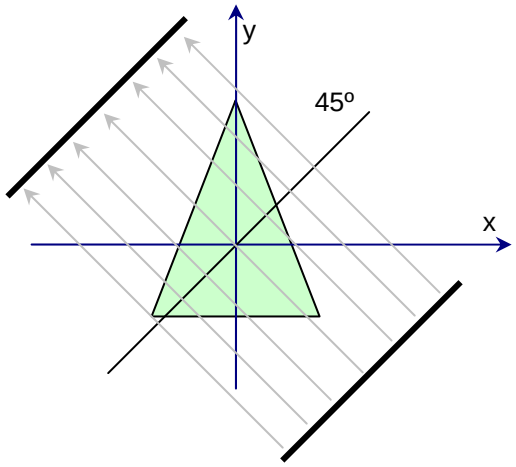


Figura 16 - Transformada Radon para 45º

La transformada Radon se define mediante la expresión:

$$R_\theta(x') = \int_{-\infty}^{\infty} f(x'\cos\theta - y'\sin\theta, x'\sin\theta + y'\cos\theta)\cdot dy'$$

$$\text{donde } \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Las utilidades de esta transformada, de cara al reconocimiento de partituras, se pueden resumir en dos ventajas. La primera es que permite caracterizar la morfología de las notas mediante todas sus proyecciones. La segunda es que permite determinar muy fácilmente el ángulo que inclinación de la partitura.

Para ello, supongamos el caso en el que no exista inclinación. Es lógico pensar que si se realiza la proyección Y se obtendrán una serie de picos máximos en grupos de cinco (por pentagrama). Si inclinamos la imagen y volvemos a realizar la proyección, dichos picos tomarán un valor menor más uniforme. Sabiendo esto, la solución para determinar el ángulo de inclinación es sencilla. Basta con realizar las proyecciones Y girando en cada iteración la imagen el número de grados que queramos que tenga el algoritmo como precisión y determinar el ángulo para el cual las proyecciones obtienen unos valores de pico máximos.

Sin embargo, esta tarea es muy costosa computacionalmente. Girar la imagen en cada iteración es una tarea que consume muchas operaciones. En su lugar, podemos pensar en girar el eje y realizar las proyecciones en lugar de girar la imagen, obteniendo el mismo resultado. Este proceso se realiza mediante la transformada Radon.

2.4. Representación digital de partituras

Existen multitud de paquetes de grabado de partituras de libre distribución para utilizar de forma conjunta con los sistemas de reconocimiento de partituras. Y es que el fin de los sistemas OMR es obtener una salida que represente la partitura original en formato simbólico,

pero de cara al usuario es muy buena idea representar dicha salida de forma gráfica, de igual forma que se representaría en el papel, en lugar de utilizar otras notaciones (independientemente que dichos datos simbólicos se guarden en un fichero)

También sería adecuado que el sistema utilizado para la representación gráfica de los resultados obtenidos pudiera interactuar con el usuario, dándole la posibilidad de modificar notas y parámetros que hubieran sido mal reconocidos por el motor OMR.

En este apartado se estudiaran de forma general algunos de los sistemas de grabado de partituras que podemos encontrar.

2.4.1. LilyPond [47]

Este programa permite a los músicos realizar partituras utilizando un fichero bajo notación *ASCII* como parámetro de entrada. El programa interpreta dicha notación y genera una partitura musical en diferentes formatos, incluyendo los formatos *PostScript* y PFD. Como añadido, el programa también es capaz de leer formatos de ficheros, como *MusicXML* y MIDI.

La documentación del programa es muy buena y se incluyen numerosos ejemplos. Además, no requiere la instalación de otros paquetes y puede funcionar el cualquier plataforma (*Linux, Microsoft Windows, Macintosh*…)

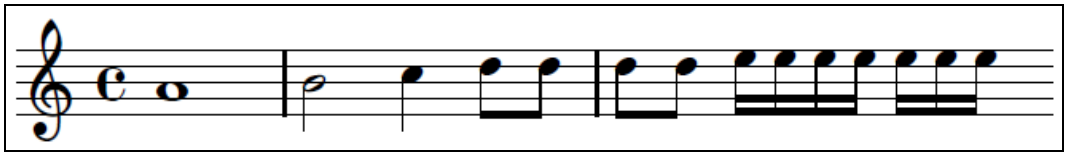


Figura 17 - Ejemplo de un pentagrama realizado con LilyPond

El código utilizado para generar el pentagrama de la Figura 17 es el siguiente:

```
\relative c''
{
   {
      \time 4/4
      \clef treble
      a1 b2 c4 d8 d8 d8 d8 e16 e16 e16 e16 e16 e16 e16 e16 s16_" "
   }
}
```

2.4.2. MusiXTEX [48]

Creado por Daniel Taupin, Ross Mitchell y Andreas Egler, "*MusiXTeX*" es un conjunto de macros y fuentes que permiten que el sistema tipográfico TeX sea capaz de escribir música. Para usarlo, requiere que se encuentren instalados tanto los paquetes de TeX como de MusiXTeX. Recordar que TeX es un sistema de tipografía escrito, muy popular en el ambiente académico, especialmente entre las comunidades de matemáticos, físicos e informáticos y que mediante paquetes de macros, permite realizar diversas tareas de composición (fórmulas matemáticas, formato de texto…).

MusiXTEX se trata de un sistema de grabado de partituras complicado, poco adecuado para un usuario novel. Como añadido, la instalación del paquete es complicada, por lo que puede ser

una opción a descartar si no estamos familiarizados con el formato TeX. No obstante, si se llega a utilizar de forma adecuada, puede obtener unos resultados excelentes.

2.5. Metodología de evaluación de los sistemas OMR

Dado el gran número de sistemas de reconocimiento musical óptico que existen, es necesario realizar una evaluación de los mismos para poder comparar las prestaciones de cada uno de forma subjetiva. Para ello, se debe implantar un proceso de evaluación estándar que se aplique de igual forma a cada sistema, ya que, si cada autor evalúa su sistema con diferentes partituras, será difícil realizar comparaciones objetivas. Además, cada autor suele evaluar su sistema con un conjunto de partituras que su sistema realiza muy bien para ensalzar las capacidades del mismo.

Estos sistemas OMR podrían ser evaluados a partir de la medida del porcentaje total de símbolos identificados correctamente, pero a pesar de que parece lógico utilizar este método, más adelante se verá que el buen funcionamiento del reconocedor no sólo se refleja en este factor.

A pesar de la necesidad de una metodología común para evaluar los sistemas, los distintos autores han utilizado diversas técnicas para la evaluación, sin llegar a un consenso sobre cuál sería la metodología óptima a seguir para poder comparar los sistemas de la forma más objetiva posible. Sin embargo, hay autores que han dedicado trabajo a investigar qué parámetros podrían ser los más influyentes a la hora de determinar la eficacia de un sistema OMR, definiendo distintas metodologías. Una de las metodologías más completas que existe fue descrita por Pierfrancesco Bellini, Ivan Bruno y Paolo Nesi en "*Assessing Optical Music Recognition Tools*" [49].

En el documento, el sistema OMR se evalúa mediante el uso de diferentes categorías en la cual cada una tiene un peso diferente asociado. Los pesos son determinados a partir de los resultados de una encuesta que fue realizada en una conferencia por un grupo de expertos y usuarios de sistemas OMR. Las categorías y pesos se muestran en la Figura 18.

Nota con tono y duración	10
Notas con marcas de accidente	7
Símbolos temporales	10
Símbolos sobre o debajo de las notas	5
Ligaduras y uniones	7
Claves	10
Número de medidas	10
Silencios	10
Grupos de notas	10
Accidentes y escalas	10
Notas de gracia	5
Puntillos	10
Grupos irregulares de notas	10
Numero de pentagramas	10

Figura 18 - Categorías y pesos asociados considerados en las encuestas

La evaluación y los resultados del sistema OMR se obtienen mediante una serie de métricas que se definen para tal efecto. Dicha métricas son:

- El número total de símbolos completos esperados
- Símbolos correctos (N)
- Símbolos añadidos (n1)
- Símbolos mal identificados (nf)
- Símbolos perdidos (nm)

Después de obtener estos parámetros, se puede realizar una tabla a partir de cada categoría y cada métrica. Mediante este procedimiento se podrá comparar de una forma más objetiva cómo funciona cada reconocedor, pudiendo conocer en qué aspectos cada reconocedor hace más hincapié.

3. Gestión del proyecto

3.1. Introducción

Este proyecto surge como una idea propuesta por el departamento de Ingeniería Telemática de la escuela politécnica de la Universidad Carlos III de Madrid. En un principió se ideó como el módulo reconocedor de un sistema de dictado (ver Figura 1) capaz de reconocer una serie de símbolos musicales proporcionados por el usuario, mediante un dispositivo de entrada como una tableta digitalizadora o un *Tablet PC*, para elaborar una partitura y reproducirla. Posteriormente, el sistema de dictado pasó a un segundo plano, diseñando y creando un programa que reconociera símbolos de partituras escaneadas, manuscritas y no manuscritas, mediante un sistema OMR (*Optical Music Recognition*)

El primer paso llevado a cabo fue la definición de unos objetivos del proyecto, los cuales se detallaron en apartados anteriores. Después, hubo que buscar y analizar gran cantidad de documentación para tener una idea global del trabajo realizado por otros autores en el campo del reconocimiento automático de música y conocer algunas de las técnicas habituales de uso. Toda esta información fue obtenida a través de distintas fuentes, de las que cabe destacar la biblioteca de la escuela politécnica de la Universidad Carlos III de Madrid, la base de datos del IEEE y la amplísima documentación proporcionada por Internet. Toda la bibliografía utilizada para redactar este proyecto, así como alguna que otra referencia interesante, se encuentra al final del documento.

Una vez se comprendió realmente la dificultad del problema y los pasos a seguir para construir un sistema como el propuesto, se redactó el punto dos de este documento, el estado del arte. Este apartado es fundamental para poder entender y realizar el proyecto, dado que establece las bases esenciales sobre las que se sustenta un sistema OMR.

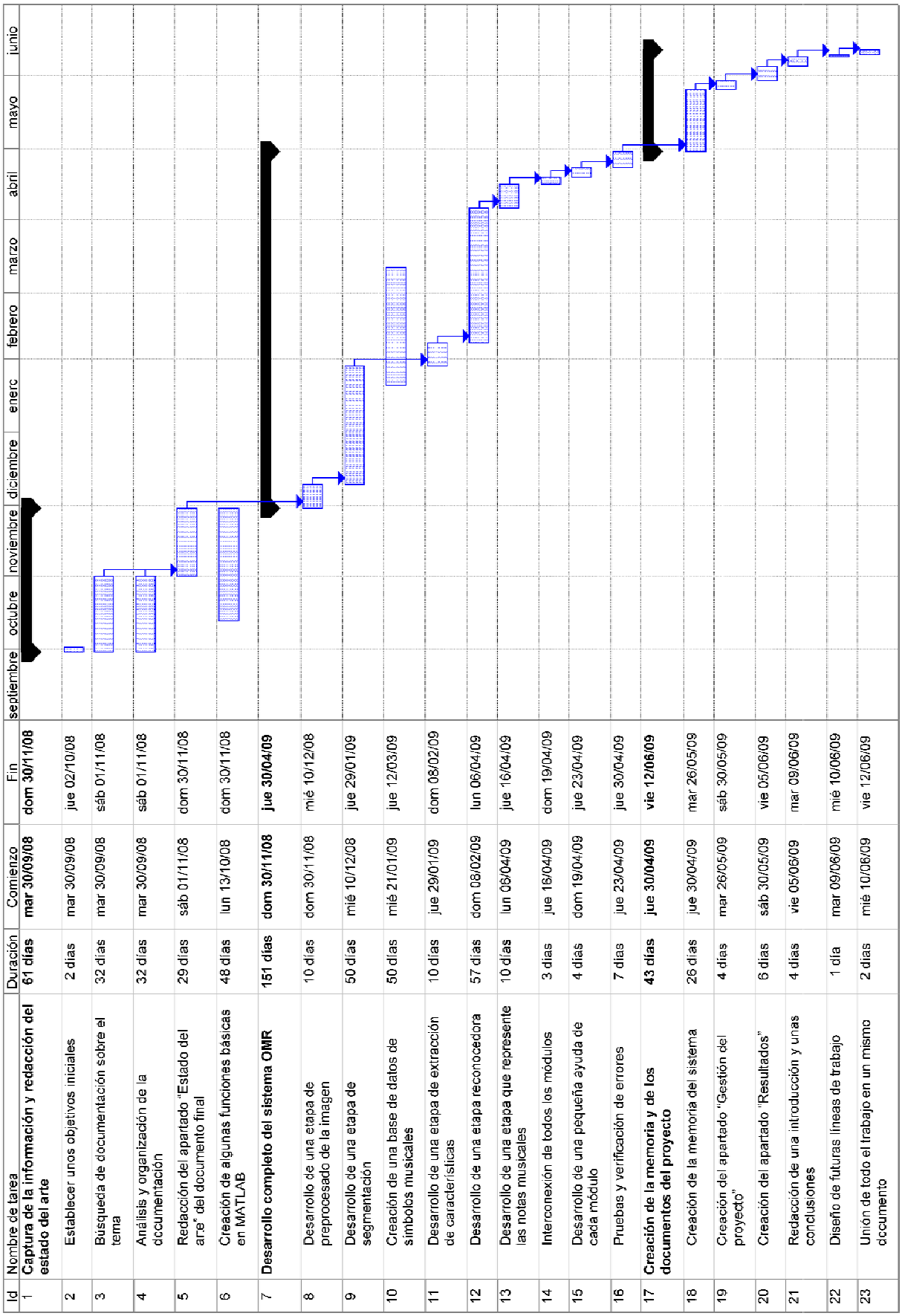
Mientras que se llevaba a cabo la redacción de dicho apartado, se implementaron algunos de los algoritmos propuestos en lenguaje MATLAB, para realizar diversas pruebas y tener código disponible a la hora de implementar el sistema completo.

Una vez se concluyó el apartado dedicado al estado del arte, se comenzó a desarrollar el sistema OMR, etapa sobre la cual más tiempo y esfuerzo se ha dedicado. Una vez concluida, se sometió al sistema a distintas pruebas y se obtuvieron una serie de resultados y conclusiones que se comentarán en los apartados oportunos.

Finalmente, se escribió la descripción del trabajo realizado, añadiendo los apartados adecuados para concluir el documento que resume este proyecto.

3.2. Tareas y duraciones

El proyecto se comenzó a desarrollar el 30 de Septiembre de 2008, finalizándolo el 12 de Junio de 2009. En total ha requerido de, aproximadamente, 8 meses de trabajo en los que se ha creado el sistema propuesto y la memoria que describe el trabajo realizado. Aunque en un principio no se definieron fases, ni tiempos, según se han desarrollado los acontecimientos se ha podido comprobar que se han ejecutado tres fases fundamentales a lo largo del desarrollo del proyecto (ver Figura 19):



Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Captura de la información y redacción del estado del arte**	**61 días**	**mar 30/09/08**	**dom 30/11/08**
2	Establecer unos objetivos iniciales	2 días	mar 30/09/08	jue 02/10/08
3	Búsqueda de documentación sobre el tema	32 días	mar 30/09/08	sáb 01/11/08
4	Análisis y organización de la documentación	32 días	mar 30/09/08	sáb 01/11/08
5	Redacción del apartado "Estado del arte" del documento final	29 días	sáb 01/11/08	dom 30/11/08
6	Creación de algunas funciones básicas en MATLAB	48 días	lun 13/10/08	dom 30/11/08
7	**Desarrollo completo del sistema OMR**	**151 días**	**dom 30/11/08**	**jue 30/04/09**
8	Desarrollo de una etapa de preprocesado de la imagen	10 días	dom 30/11/08	mié 10/12/08
9	Desarrollo de una etapa de segmentación	50 días	mié 10/12/08	jue 29/01/09
10	Creación de una base de datos de símbolos musicales	50 días	mié 21/01/09	jue 12/03/09
11	Desarrollo de una etapa de extracción de características	10 días	jue 29/01/09	dom 08/02/09
12	Desarrollo de una etapa reconocedora	57 días	dom 08/02/09	lun 06/04/09
13	Desarrollo de una etapa que represente las notas musicales	10 días	lun 06/04/09	jue 16/04/09
14	Interconexión de todos los módulos	3 días	jue 16/04/09	dom 19/04/09
15	Desarrollo de una pequeña ayuda de cada módulo	4 días	dom 19/04/09	jue 23/04/09
16	Pruebas y verificación de errores	7 días	jue 23/04/09	jue 30/04/09
17	**Creación de la memoria y de los documentos del proyecto**	**43 días**	**jue 30/04/09**	**vie 12/06/09**
18	Creación de la memoria del sistema	26 días	jue 30/04/09	mar 26/05/09
19	Creación del apartado "Gestión del proyecto"	4 días	mar 26/05/09	sáb 30/05/09
20	Creación del apartado "Resultados"	6 días	sáb 30/05/09	vie 05/06/09
21	Redacción de una introducción y unas conclusiones	4 días	vie 05/06/09	mar 09/06/09
22	Diseño de futuras líneas de trabajo	1 día	mar 09/06/09	mié 10/06/09
23	Unión de todo el trabajo en un mismo documento	2 días	mié 10/06/09	vie 12/06/09

septiembre octubre noviembre diciembre enero febrero marzo abril mayo junio

Figura 19 - Diagrama Gantt del proyecto realizado

1. Captura de la información y estudio del estado del arte
2. Desarrollo completo del sistema OMR
3. Desarrollo de la memoria

En la primera fase, "**Captura de la información y estudio del estado del arte**", se comenzó a idear el proyecto, buscando información sobre el trabajo realizado de otros autores y técnicas involucradas en los sistemas OMR. Se pueden distinguir las siguientes tareas:

- Establecimiento de unos objetivos iniciales
- Búsqueda de documentación sobre el tema
- Análisis y organización de la documentación
- Redacción del apartado "Estado del arte" de la memoria
- Creación de algunas funciones básicas en MATLAB

Esta fase, que duró dos meses, pone las bases necesarias para que se pueda desarrollar un sistema OMR. Para ello, en primer lugar se deben establecer unos objetivos iniciales aún desconociendo la dificultad del problema que supone la realización del sistema reconocedor. De no realizar esto se corre el riesgo de "perder el rumbo" una vez iniciada la tarea; desviar la atención a otros objetivos secundarios y olvidar el objetivo principal.

El siguiente paso es buscar la información necesaria que permita conocer los detalles de un sistema reconocedor de música. Para ello, se recurrió a las fuentes antes mencionadas (biblioteca de la escuela politécnica de la Universidad Carlos III de Madrid, base de datos del IEEE e Internet) de las cuales se pudieron extraer ciertos documentos que describían los trabajos realizados en el campo OMR por otros autores. Este punto es fundamental para entender el verdadero problema que presenta el reconocimiento de notas musicales y para establecer las bases de una arquitectura sobre la cual se desarrolle el programa. De no hacerlo así, y empezar directamente a elaborar el programa sin tener ciertas bases, puede ocurrir que se llegue a un punto en el desarrollo en el que no se sepa como realizar el resto del programa o que el código no sirva para nada y se tenga que volver a desarrollarlo desde cero, alargando el tiempo total de realización del proyecto.

También es conveniente organizar la información obtenida e ir apuntando la bibliografía en un documento para que al realizar la memoria final no haya que volver a releer todos los documentos para determinar cuáles han servido para desarrollar el trabajo.

Una vez hecho esto, se podría haber comenzado con la programación del código pero se prefirió comenzar con la escritura del apartado dedicado al estado de arte. Es interesante realizarlo antes de comenzar a programar por dos motivos. En primer lugar porque es trabajo que se puede completar sin tener que esperar a que se haya desarrollado todo el sistema, como ocurre con la gran mayoría del resto de los capítulos. En segundo lugar, porque la realización de este capítulo permite empaparnos a fondo del trabajo realizado por otros autores y de las técnicas empleadas, siendo mucho más fácil el desarrollo del código a continuación.

De forma paralela, mientras se escribía el apartado oportuno también se desarrollaron algunas funciones básicas en MATLAB para probar algunos de los conceptos descritos por los autores estudiados.

Una vez se terminó de escribir el apartado descrito, se procedió a la creación del código en la siguiente fase "**Desarrollo completo del sistema OMR**". En esta fase se tuvo que crear módulo por módulo, verificando su correcto funcionamiento y que la interconexión entre ellos permitía la correcta ejecución del sistema OMR completo. Se pueden distinguir una serie de tareas y sub-tareas:

- Desarrollo de un bloque de preprocesado de la imagen
- Desarrollo de un bloque de segmentación
    - Extracción y verificación de pentagramas
    - Extracción de las notas
    - Eliminación de las líneas del pentagrama
- Creación de una base de datos de símbolos musicales
- Desarrollo de una etapa de extracción de características
    - Extracción de descriptores de *Fourier*
    - Extracción de otras características
- Desarrollo de un bloque reconocedor
- Desarrollo de una etapa que represente las notas musicales
- Interconexión de todos los módulos
- Desarrollo de una pequeña ayuda de cada módulo
- Pruebas y verificación de errores

Esta fase ha sido la que más tiempo y esfuerzo ha requerido, con un total de cinco meses de trabajo.

Para la etapa de preprocesado se crearon un par de módulos que realizaban determinadas operaciones sobre la partitura de entrada para adaptarla adecuadamente para su posterior procesado al extraer los elementos. Dichos elementos debían ser analizados para determinar si eran o no pentagramas. Posteriormente se diseñó un módulo capaz de extraer las notas de un pentagrama.

De forma paralela, se comenzó el diseño de una base de datos compuesta por 41 categorías. Esta base de datos es necesaria para el funcionamiento de la etapa reconocedora.

Posteriormente, se desarrollaron unos módulos encargados de extraer las características de las notas, mediante descriptores de *Fourier* y otros elementos. Estos permiten comparar de forma cuantitativa los símbolos, pudiendo medir el grado de semejanza entre ellos.

El siguiente paso, la creación del sistema reconocedor encargado de clasificar los símbolos, se realizó variando sucesivamente numerosos parámetros para conseguir los mejores resultados posibles. Además, de cara a la detección de errores, se diseño un módulo encargado de analizar los datos reconocidos en busca de nuevos símbolos compuestos o posibles fallos.

Después, se estudió el lenguaje *Lilypond* para implementar un módulo que realizase la codificación automática de los datos, para un pentagrama.

Una vez funcionaban todos los módulos por separado, se interconectaron y se modificaron algunos elementos para que el programa fuera capaz de analizar tanto pentagramas aislados como partituras completas.

Además, para facilitar la tarea del usuario y de futuros desarrolladores, se realizó una pequeña ayuda con los parámetros de entrada / salida de cada módulo, y se comentó el código para su fácil comprensión.

Finalmente se realizaron una serie de pruebas con símbolos manuscritos y no manuscritos, para comprobar el correcto funcionamiento del programa y arreglar cualquier fallo que apareciera.

Concluida esta fase, se pasó a llevar a cabo la siguiente: "**Desarrollo de la memoria**". Esta comprende la etapa en la que se escribe sobre el sistema que se ha diseñado, indicando la descripción de sus módulos, su funcionamiento, etc. Distinguimos varias etapas:

- Creación de la memoria del sistema
- Creación del apartado "Gestión del proyecto"
- Creación del apartado "Resultados"
- Redacción de una introducción y unas conclusiones
- Diseño de futuras líneas de trabajo
- Unión de todo el trabajo en un mismo documento

Esta fase ha sido la más corta, con una duración de un mes aproximadamente. En los documentos descritos se han intentado explicar los conceptos oportunos de la forma más clara posible, evitando lenguaje muy técnico.

Todos los documentos resumidos en este proyecto se han desarrollado utilizando las mismas directrices, mismo formato, etc. Además, se ha generado un índice de contenido y de figuras, facilitando la búsqueda de los capítulos. Como añadido, se ha elaborado un anexo explicando las bases del lenguaje *Lilypond*, a modo de guía rápida, y se han incluido algunas de las imágenes utilizadas para crear ciertas bases de datos manuscritas del proyecto. También se ha desarrollado un manual de usuario y un manual de referencia.

Hay que destacar que los primeros 4 meses de trabajo del proyecto se realizaron de forma paralela al estudio de una asignatura de la carrera, por lo que no se pudo dedicar todo el tiempo que se hubiera deseado al proyecto. Es posible que, de haber tenido más tiempo para el proyecto en esa época, el trabajo de 4 meses se pudiera haber realizado en la mitad.

Se ha realizado un diagrama Gantt para ilustrar la ejecución de cada fase según lo comentado. Dicho diagrama se puede observar en la Figura 19.

3.3. Presupuesto y coste del proyecto

3.3.1. Introducción

El presupuesto se ha realizado teniendo en cuenta las horas dedicadas al proyecto y el precio de los elementos que se han utilizado. Se diferenciarán dos gastos principales: gastos personales y gastos materiales.

La duración total del proyecto ha sido de ocho meses y diez días. Aunque no se ha seguido un horario estricto a rajatabla y algunos días se ha trabajado más que otros, de forma general se podría establecer un horario de trabajo de seis horas diarias. Teniendo en cuenta la semana laboral, de lunes a viernes, obtendríamos 30 horas semanales y contando que un mes tiene cuatro semanas, 120 horas mensuales. En total el proyecto se traduciría en 1020 horas de trabajo. En ese trabajo se incluiría el desarrollo de la memoria del proyecto.

Para los gastos personales se tendrán en cuenta los honorarios de un Ingeniero Técnico de Telecomunicación. Hasta hace poco, la Junta General del Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT [50]) ofrecía una cantidad a modo de ejemplo para los libres ejercientes de la profesión, de forma que pudieran disponer de una referencia de los honorarios. Dicha cantidad definía unos honorarios de 65 € / hora. Hoy por hoy, el Ministerio de Economía y Hacienda ha remitido a todos los colegios profesionales una nota [51] en la que se indica que no se debe, ni siquiera, publicar un baremo con los honorarios ya que, éstos son libres y responden al libre acuerdo entre el profesional y el cliente.

Dada la situación, los honorarios deben definirse en función de una serie de factores: costes del ingeniero, desplazamientos, mecanografía, volumen de la actividad… Teniendo en cuenta estos elementos y que, por lo general, un ingeniero técnico no suele cobrar menos de 40 € / hora, ni más de 70 € / hora, se definirán unos honorarios de 50 € / hora.

Al número de horas de trabajo se deben añadir las horas correspondientes a las tutorías. En total, se han tenido unas 15 tutorías, de una duración aproximada de una hora y media cada una. Un total de 23 horas. También se deben contar las horas que ha dedicado el tutor del proyecto a las tutorías, sumando otras 23 horas. Dichas horas se cobran de igual forma que las horas de trabajo de proyecto.

Los gastos materiales abarcan todos los elementos necesarios para desarrollar el proyecto, desde el ordenador hasta la conexión ADSL necesaria para obtener los datos de la red. Se presupone que dichos gastos cuentan con el IVA en el precio mostrado.

No se ha requerido coste alguno para la consulta de la bibliografía. Los documentos obtenidos se han consultado a través de Internet, con el consentimiento de los autores oportunos para la publicación de su artículo en la red, y a través de la biblioteca de la Universidad Carlos III de Madrid.

3.3.2. Gastos materiales

Para la realización del proyecto se ha utilizado un ordenador portátil *Acer Aspire 5920G*, con procesador *Intel Core 2 Duo* (T8300) a 2.4 Ghz, con licencia de *Microsoft Windows Vista Home Premium* incluida. Se han requerido licencias de *Microsoft Office 2003*, *Microsoft Project 2003* y

SmartDraw 2008 para la realización de la memoria. Además, para la elaboración del programa se ha necesitado una licencia de MATLAB 7.3.0 (R2006b).

También debemos tener en cuenta los costes de los viajes realizados para llevar a cabo las tutorías. Los viajes se han efectuado entre Talavera de la Reina (Toledo), ciudad donde se ha realizado el proyecto y Leganés (Madrid), donde se encuentra la escuela politécnica de la Universidad Carlos III de Madrid. A razón de 14 € aproximadamente el billete de tren, contando dos viajes (ida y vuelta) con descuento por carné joven.

En resumen, los gastos por materiales se detallan en la tabla a continuación:

Concepto	Coste / Cantidad	Cantidad	Total
Portátil Acer	900 €	1 unidad	900 €
Microsoft Office 2003 Professional	709 €	1 unidad	709 €
Microsoft Project 2003 Professional	1299 €	1 unidad	1299 €
SmartDraw 2008	137,78 €	1 unidad	137,78 €
MATLAB 7.3.0 (R2006b)	500 €	1 unidad	500 €
Signal Processing Toolbox MATLAB	200 €	1 unidad	200 €
Image Processing Toolbox MATLAB	200 €	1 unidad	200 €
Desplazamientos	14 €	15 viajes	210 €
Conexión ADSL	30 €	8 meses	240 €
		TOTAL	**4395.78 €**

Figura 20 - Gastos materiales

3.3.3. Gastos personales

A continuación se mostrará la tabla que detalla el presupuesto de los gastos personales con los datos descritos anteriormente:

Concepto	Coste / Cantidad	Cantidad	Total
Trabajo proyecto	50 € / hora	1020 horas	51000 €
Horas tutoría	50 € / hora	23 horas	1150 €
Salario tutor	50 € / hora	23 horas	1150 €
		TOTAL (SIN IVA)	**53300 €**
		IVA 16 %	**8528 €**
		TOTAL (CON IVA)	**61828 €**

Figura 21 - Gastos personales

3.3.4. Total presupuesto

En total, teniendo en cuenta los gastos materiales y los gastos personales, obtenemos el presupuesto final:

Concepto	Coste
Gastos materiales	**4325,78 €**
Gastos personales	**61828 €**
TOTAL	**66153,78 €**

Figura 22 - Total presupuesto

El presupuesto total de este proyecto asciende a SESENTA Y SEIS MIL CIENTO CINCUENTA Y TRES EUROS CON SETENTA Y OCHO CENTIMOS.

Fdo: David Carretero de la Rocha
Ingeniero Técnico de Telecomunicación especializado en Sonido e Imagen

4. Descripción del sistema

Una vez definido el problema del reconocimiento OMR y establecidas las bases y algunos métodos para poder resolverlo, en este capítulo, se describirá el programa realizado de forma detallada, analizando su arquitectura, las decisiones de diseño e implementación y los módulos desarrollados en profundidad.

En primer lugar se hablará de la funcionalidad del programa. Después se analizará la arquitectura definida para el proyecto. Este último apartado describirá los bloques que conforman el programa.

Finalmente, se describirá de forma detallada cada módulo explicando, en cada uno, su funcionamiento, los parámetros de entrada/salida y, por supuesto, para qué sirve. Además, este capítulo no sólo profundizará en los módulos de mayor nivel sino que detallará los módulos de menor nivel que se utilizan.

4.1. Funcionalidad

El sistema desarrollado es capaz de analizar partituras manuscritas y no manuscritas para digitalizar sus elementos. Dado que constituye un proyecto fín de carrera no cuenta con un grado de desarrollo equivalente al de un sistema comercial. Esto implica que cuente con ciertas limitaciones. A continuación se describirán los elementos que se pueden reconocer y las acciones que se pueden realizar:

- Partituras de uno o varios pentagramas
- Partituras cuyo grosor de línea de pentagrama sea de un píxel
- Partituras de líneas de pentagramas azules
- Símbolos manuscritos y no manuscritos
- Anacrusas
- Posibilidad de cambio de clave y de tiempo durante el transcurso de la partitura
- Notas musicales: cuádruple, doble, redonda, blanca, negra, corchea, semicorchea, fusa, semifusa
- Silencios
- Clave de Sol y Fa
- Tiempos de compás
- Capacidad de agregar nuevas clases de símbolos para la detección
- Capacidad para modificar el sistema de forma fácil

En contraposición, algunas de las limitaciones existentes son:

- No se permite el reconocimiento de notas unidas (corcheas, semicorcheas…)
- No se permite el uso de música polifónica
- No se permite el uso de partituras de piano
- No se permite el uso de notaciones de música distintas a la notación moderna

- No se permite el uso de tablaturas de guitarra
- No se permite el uso de acordes
- No se permite el uso de varias voces
- No se permite el uso de grosores de línea distinto de un píxel
- Se rechazan los elementos que no sean pentagramas (letras, titulos…)

A todas las ventajas del sistema desarrollado se debe añadir que se ha contruido de forma que permita la fácil interconexión con otros sistemas. En la Figura 23 se muestran algunos de los posibles usos del sistema reconocedor de forma conjunta con una etapa de representación gráfica/acústica (*Lilypond*) y un sistema inteligente de dictado musical.

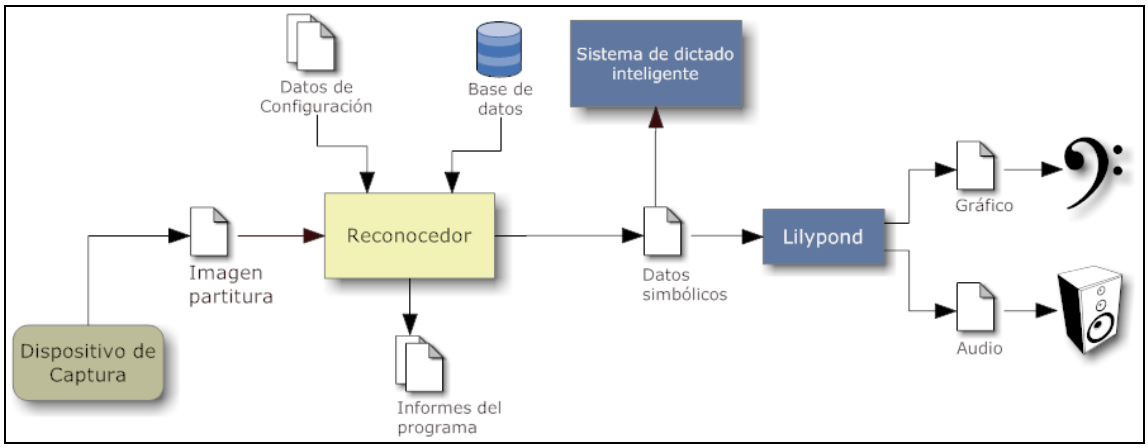


Figura 23 - Interconexión del sistema a desarrollar con otros posibles sistemas

4.1.1. Datos de configuración

El programa requiere de determinados parámetros para ejecutarse correctamente. Estos se proporcionan al escribir la expresión que ejecuta el programa y durante su ejecución mediante cuadros de diálogo. Los datos de configuración sirven para indicar al programa cómo tiene que realizar su trabajo y qué parámetros debe utilizar.

Para ejecutar el programa se deben definir dos variables de configuración: el tipo de líneas que utilizará el pentagrama y el número de vecinos evaluados para la clasificación. El tipo de líneas debe definirse para elegir el algoritmo adecuado que pueda realizar el borrado de estas. En el proyecto se han definido dos tipos principalmente:

- **Lineas normales:** se denominan así a las líneas de color negro de cualquier partitura. Por limitaciones del programa el algoritmo desarollado sólo funciona para el caso en el que el grosor del pentagrama no supere el píxel de grosor.

- **Líneas azules:** una alternativa para poder utilizar pentagras de grosor distinto de un píxel es la utilización de partituras cuyas líneas del pentagrama sean de otro color para diferenciar líneas de símbolos. En este caso se ha implementado la utilización de partituras de azul puro. Dichas partituras se tendrán que dibujar a mano o deberan disponer de un pentagrama azul anteriormente diseñado.

Durante la ejecución del programa se solicitarán una serie de datos que son necesarios para ejecutar el proceso completo:

1. **Imagen de la partitura** para la cual se quiere realizar el proceso de reconocimiento. Los formatos admitidos son BMP, PNG y TIFF.

2. **Base de datos.** Por cuadro de diálogo se requiere el archivo con los vectores de características de la base de datos. Hay que tener en cuenta que dicha base debe ser adecuada al contenido de la partitura. Si se utiliza música manuscrita se debe utilizar una base de datos de símbolos musicales propia de la escritura del autor. En caso de música hecha a máquina, bastará con una base de datos no manuscrita.

3. **Ruta destino.** Se solicita el nombre del fichero codificado (con los datos de la partitura digitalizada) y la ruta. Este fichero albergará los datos simbólicos, extraídos en el proceso del programa, codificados en un formato adecuado para su posterior representación. Se ha optado por la utilización del lenguaje *Lilypond* para tal efecto.

A continuación se describirán las características con las que deben contar la imagen de la partitura y la base de datos.

4.1.1.1. Imagen de entrada

Se ha supuesto que la imagen de la partitura se ha escaneado o se ha obtenido mediante otras fuentes ya digitalizada. Para el caso de que se quieran utilizar símbolos aislados (caso a tener en cuenta si se desea capturar la información mediante una tableta digitalizadora o un *tablet pc*) se deben cargar las imágenes directamente en el bloque clasificador y realizar una pequeña modificación del sistema, como se estudiará en el apartado 11.5.

Es conveniente que la partitura a utilizar cuente con una resolución de entre 75 y 200 dpp (*dots per pixel*). Cuanta más resolución, mejor funcionará el bloque de segmentación. Además, la partitura que se utilice debe estar limpia, sin elementos de ruido o impurezas (polvo, arañazos…) y sin bordes alrededor (Figura 24). Además, se recomienda que el formato sea blanco y negro (salvo en el caso de que las líneas del pentagrama sean azules para la extracción de las notas, como se comentará posteriormente). No importa que la imagen este rotada.

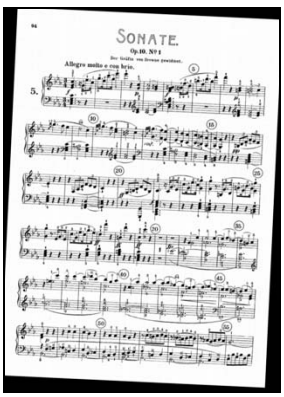


Figura 24 - Bordes de la imagen que pueden aparecer en el proceso de escaneo (extraído de la sonata No. 5 en Do menor, Op. 10 No. 1 de Beethoven)

4.1.1.2. Base de datos

La base de datos constituye un conjunto ordenador de símbolos que utiliza el clasificador para llevar a cabo su tarea. Se divide en carpetas, que representan determinados símbolos musicales, y, a su vez, cada carpeta contiene un determinado conjunto de imágenes de símbolos que caracterízan la clase que representa la carpeta.

Los pasos a seguir para construir una base de datos y el nombre de los directorios de las clases se definen en el apartado 11.1.

La estructura de datos que maneja el sistema para utilizar la base de datos no se corresponde con las imágenes que alberga la base de datos. En su lugar, se utilizan los vectores de características que se encargan de caracterizar la morfología de cada ejemplo. Para crear dicha estructura se debe hacer uso del bloque dedicado a tal efecto.

4.2. Arquitectura de la aplicación

Se pueden diferenciar fundamentalmente cinco bloques en la arquitectura, como se observa en la Figura 25. Estos son:

1. **Procesado de la imagen**: se encarga de adaptar las características locales de la imagen de la partitura para el funcionamiento de los siguientes bloques.

2. **Segmentación pentagramas y notas**: este bloque extrae los pentagramas existentes en la imagen de la partitura y las notas de cada uno.

3. **Clasificación elementos**: en este bloque se realiza el proceso de reconocimiento de los símbolos, dentro de unas clases definidas en una base de datos de símbolos musicales.

4. **Verificación elementos y detección de tono**: se encarga de detectar el tono de las notas y analizar la partitura mediante teoría musical, en busca de posibles errores.

5. **Traductor de formato**: los resultados se codifican en un formato adecuado para su posterior representación. En este caso se ha utilizado el lenguaje *Lilypond* para representar la partitura de forma gráfica y acústica.

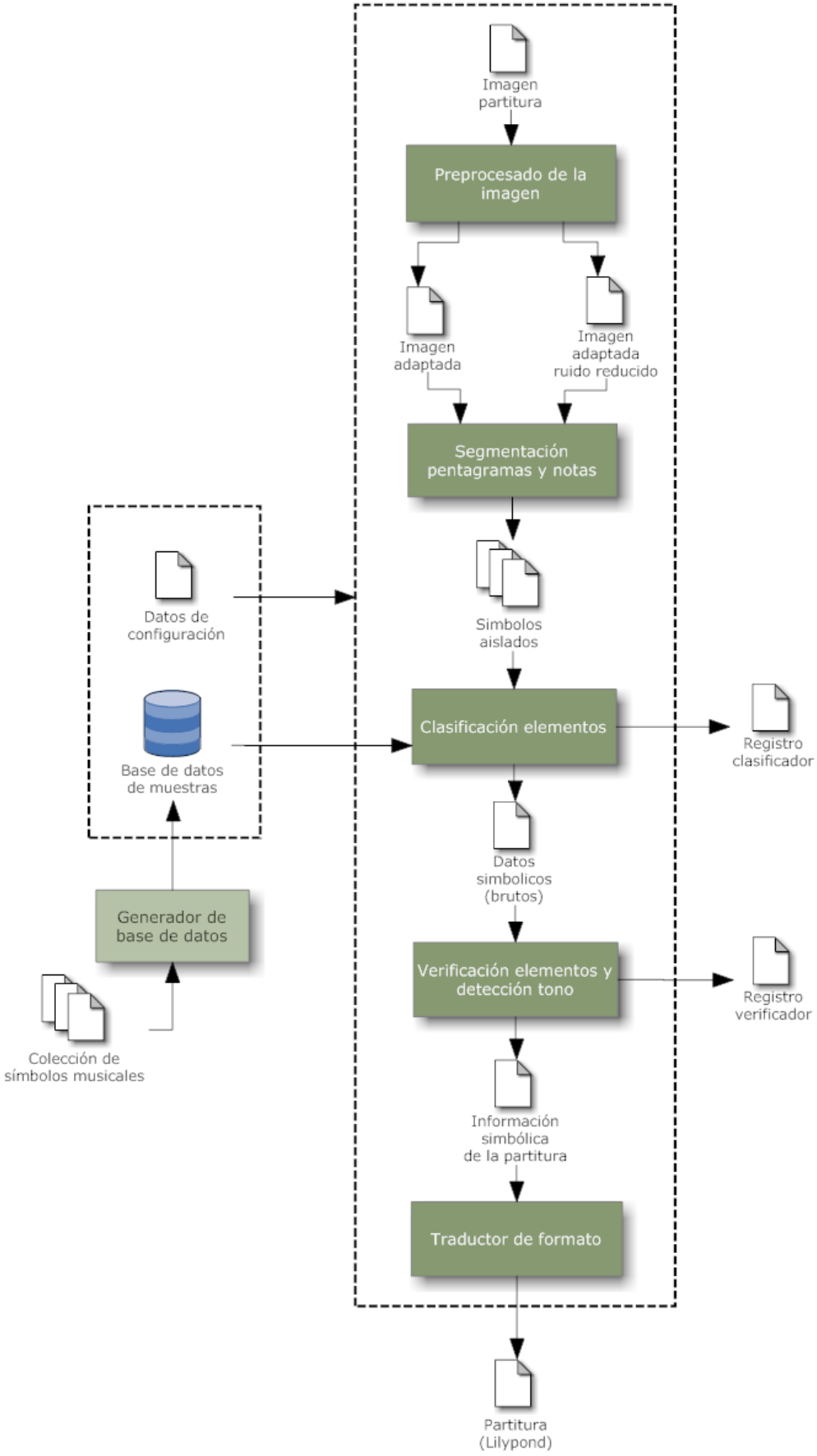


Figura 25 - Arquitectura del sistema

Además, hay que destacar dos bloques que participan en el proceso. El funcionamiento de estos será detallado en apartados posteriores.

1. **Interfaz de usuario:** se encarga de interaccionar con el usuario mostrándole información sobre el proceso y solicitándole determinados parámetros.

2. **Creación de bases de datos:** este bloque se encarga de crear las estructuras que definen las muestras de las que dispone el programa para realizar la clasificación.

La aplicación se ha desarrollado de forma modular. Cada bloque mencionado esta compuesto por una serie de módulos que ejecutan determinadas etapas. Esto permite que cada componente pueda ser ejecutado de forma independiente con respecto a otros módulos, siempre que los parámetros de entrada sean los adecuados.

Para evitar que el usuario tenga que ejecutar un módulo tras otro sucesivamente por línea de comandos, se ha desarrollado un módulo de arranque encargado de interconectar y ejecutar todos los módulos pidiendo al usuario los datos oportunos mediante cuadros de diálogos y mensajes. En caso de que el usuario desee indagar dentro del contenido de cada módulo o ejecutar uno de forma independiente puede acudir a un pequeño apartado de ayuda existente en cada fichero que describe su función y sus parámetros de entrada y de salida.

El lenguaje utilizado para desarrollar la aplicación ha sido MATLAB. La elección de este lenguaje no fue trivial. MATLAB incorpora numerosas funciones predefinidas y un potente "*toolbox*" de procesado digital de imagen que puede ayudar en gran medida en la tarea. Además, es muy fácil realizar pruebas al instante mediante la programación en la línea de comandos, sin necesidad de compilador. Y no solo eso, sino que la ayuda del programa y los mensajes de error proporcionan gran cantidad de información a la hora de corregir fallos del código realizado.

Hay que tener en cuenta que, según la arquitectura desarrollada, para iniciar el programa es necesario determinar una serie de datos de configuración al ejecutar el módulo de arranque. Dichos datos son el tipo de algoritmo de extracción de líneas de pentagrama que se utilizará en la partitura y el número de ejemplos más parecidos que se tendrán en cuenta en el clasificador a la hora de reconocer los símbolos. En la descripción detallada del módulo de arranque se explicarán detenidamente estos conceptos.

4.3. Preprocesado de la imagen

El bloque de preprocesado de la imagen se encarga de adaptar las características locales de la imagen de la partitura para asegurar el correcto funcionamiento del resto de los bloques de la arquitectura. En la Figura 26 se puede visualizar la arquitectura de este bloque:

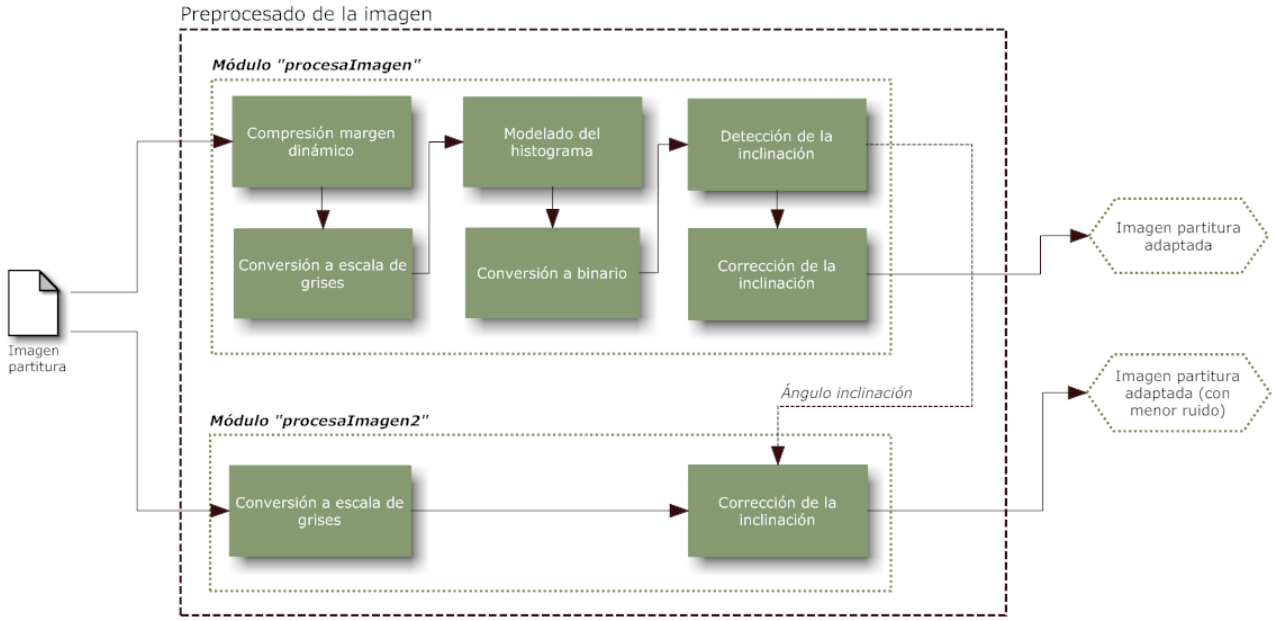


Figura 26 - Diagrama de bloques de bloque de preprocesado de la imagen

Como se puede observar, la imagen de la partitura que constituye el parámetro de entrada debe pasar por dos sub-bloques encargados de realizar un preprocesado. El primero de ellos se ejecuta en el módulo "*procesaImagen*" y se encarga de adaptar las características locales de la imagen para asegurar que la partitura se encuentra en formato binario y que no cuenta con inclinación. El segundo se ejecuta en el módulo "*procesaImagen2*" y realiza un preprocesado menos agresivo para evitar la inclusión de mucho ruido en la imagen.

El principal motivo por lo que se realiza así es porque la primera etapa puede generar mucho ruido durante la ejecución de los pasos de conversión a binario (debido a los pasos de reducción del margen dinámico y modelado del histograma), ruido que será nocivo de cara a la clasificación del símbolo, por lo que se define una nueva etapa que procese la imagen de forma menos agresiva de la que se extraerán los símbolos.

Las etapas del segundo sub-bloque ("Conversión a escala de grises" y "Corrección de la inclinación") son análogas a las mismas del primero. La corrección de inclinación se ejecuta teniendo en cuenta la información de la inclinación proporcionada por el sub-bloque anterior. Conociendo esto, únicamente se describirán las etapas del primer sub-bloque:

4.3.1. Compresión margen dinámico

Este procedimiento sirve para visualizar los bajos niveles de intensidad con un mayor margen dinámico. Esto puede ser útil para partituras antiguas en las cuales la hoja manuscrita no tenga color blanco o para partituras cuyo color se ha deteriorado, como se puede observar en la Figura 27. En el programa se ha utilizado la siguiente expresión [59] para llevar a cabo esta tarea:

$$v(x,y) = \sqrt[3]{u(x,y)}$$

Donde “u(x,y)” corresponde al valor del píxel actual y “v(x,y)” el valor del píxel transformado.

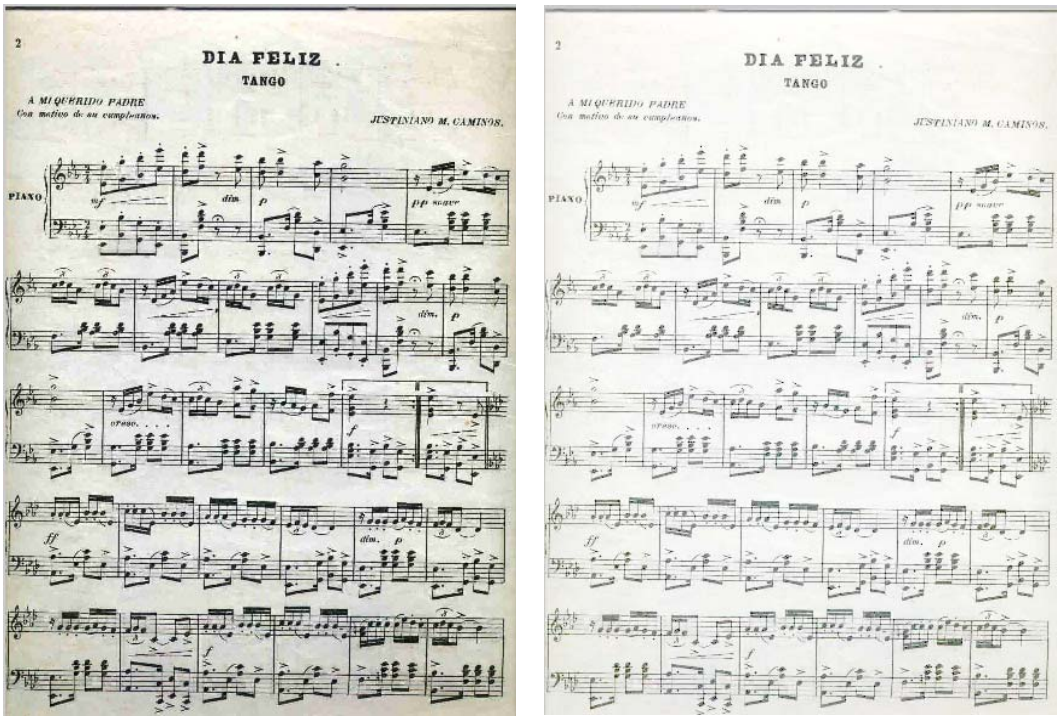


Figura 27 - Compresión del margen dinámico. Derecha: imagen original; izquierda: imagen con margen dinámico comprimido

4.3.2. Conversión a escala de grises

En el caso de que la imagen disponga de formato RGB (color) se debe realizar una pequeña transformación para convertir el formato de la imagen a escala de grises (necesario para la posterior conversión a binario). Para ello, y sabiendo que una imagen en color está formada por tres capas monocromáticas, se podría recurrir a dos métodos. El método más simple consiste en elegir únicamente una imagen de las tres capas monocromáticas, en lugar de realizar cualquier operación. Este método puede ocultar parte de la información de la imagen original, como se puede apreciar si se eligiera cualquiera de las tres capas de la Figura 28.

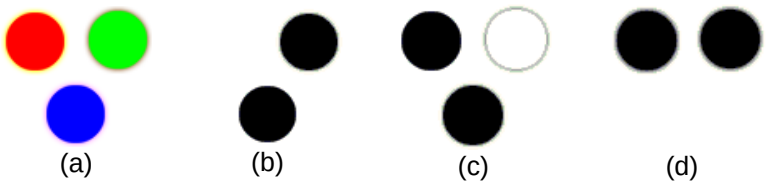


Figura 28 - Capas de una imagen RGB. (a) Imagen original. (b) Componente roja. (c) Componente verde. (d) Componente azul

El otro método, más efectivo, consiste en obtener la luminancia de la imagen mediante una combinación lineal de las tres capas de la imagen. Este parámetro viene dado por la siguiente expresión [58][59]:

$$I(x,y) = 0.30 \cdot R(x,y) + 0.59 \cdot G(x,y) + 0.11 \cdot B(x,y)$$

Siendo R la componente roja del píxel de coordenada (x,y), G la componente verde, B la componente azul e Y el píxel de luminancia resultante en dicha coordenada. Aplicando dicha expresión se obtendría la imagen de la
Figura 29.

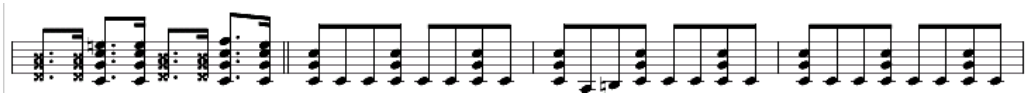
Figura 29 - Imagen RGB y componente de luminancia de dicha imagen

4.3.3. Modelado del histograma

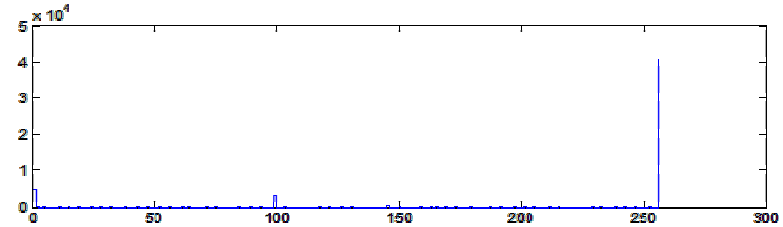
Para ello, se utiliza el histograma de un pentagrama limpio. Al hablar de este término, se hace referencia a un pentagrama en el que únicamente predominan los colores blancos y negros, con poca o ninguna variedad de grises. El histograma de una imagen monocromática permite conocer la frecuencia de aparición de los posibles valores de grises. Mediante su modelado se puede variar la frecuencia de determinados tonos de grises para no haya tonalidades intermedias. Esto permitirá limpiar completamente la imagen, convirtiendo a tono blanco aquellos píxeles más claros y a tono negro los más oscuros.

Para realizar esta operación se utilizan dos funciones implementadas en MATLAB. La primera "*imhist*" proporciona el histograma de una imagen dada, en este caso el histograma de la imagen patrón, y la segunda "*histeq*" modela el histograma de una imagen utilizando el histograma de otra.

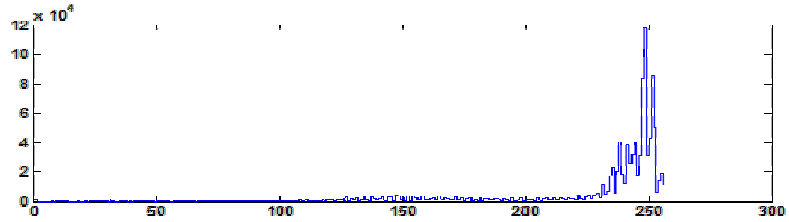
(a)



(b)



(c)



(d)

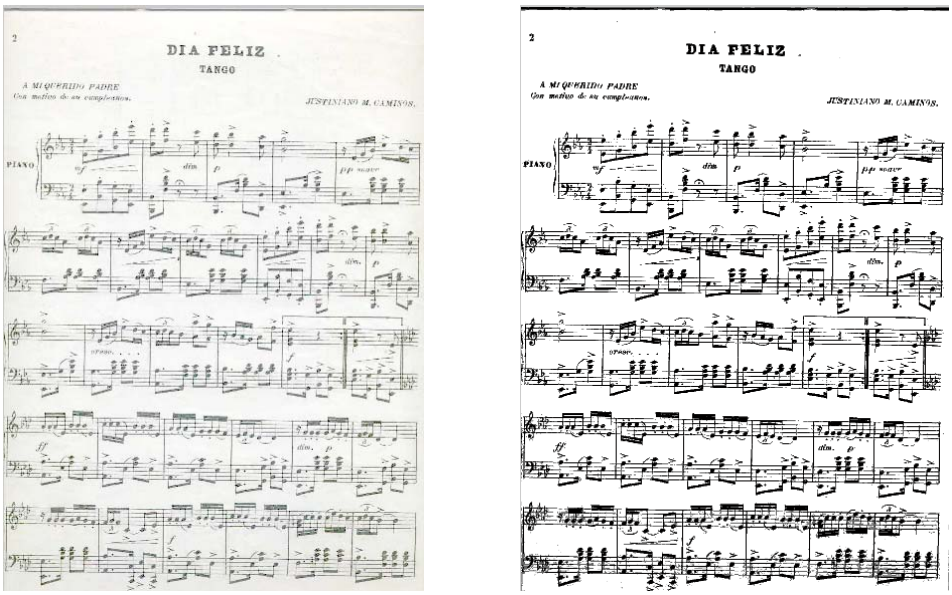


Figura 30 - (a) Pentagrama utilizado para modelar el histograma. (b) Histograma modelo. (c) Histograma de la imagen original. (d) Versión de la partitura antes y después del modelado del histograma

4.3.4. Conversión a binario

Este paso asigna un umbral sobre el cual los píxeles toman valor de cero (negro) por debajo y valor de uno (blanco) por encima. La elección de este umbral es crucial, pues determinará la forma de la imagen final y, por tanto, la eficacia de las etapas sucesivas. Dado que no se puede establecer un valor estándar para todas las partituras (dada la gran variedad existente), se recurrirá a la utilización de un umbral óptimo calculado a partir de cada imagen. Para ello, se acudirá al método de *Otsu* [43]. Este método calcula el histograma de valores de grises, calculando la media y la varianza de forma iterativa. Va estableciendo umbrales que dividen el histograma en dos partes para las cuales se vuelve a repetir el proceso y a establecer nuevos umbrales hasta conseguir el óptimo. En la Figura 31 se puede observar el resultado al ejecutar el método de *Otsu* sobre la imagen en escala de grises.

Figura 31 - Imagen en escala de grises y su versión en blanco y negro

4.3.5. Detección de la inclinación

Se debe analizar la imagen para detectar una posible rotación de la partitura con respecto a la original, debido al proceso de escaneo. Existen diferentes algoritmos para realizar esta tarea. En este proyecto se han llegado a implementar hasta cuatro:

- **Proyecciones Y**

  Se puede realizar la proyección Y de la imagen inclinada para detectar con qué ángulo de rotación se obtiene el mejor resultado. Para ello, dada la imagen original, se debe inclinar sucesivamente la imagen un número determinado de grados analizando la proyección Y

en cada caso para detectar con qué inclinación se acentúan más los picos correspondientes a los pentagramas en la proyección. Este método, aunque es óptimo en el sentido de que proporciona la inclinación que da el mejor resultado, conlleva una gran carga computacional que aumenta cuanta más resolución tenga la imagen original, por lo que no es una opción viable.

- **Transformada de *Hough***

  La versión de la transformada para detección de líneas [59] permite localizar líneas dentro de la imagen y conocer, por lo tanto, su inclinación. Para ello, se realiza la transformación y se escoge una de las líneas de cualquier pentagrama. Se debe tener cuidado a la hora de elegir el segmento a evaluar puesto que pueden existir otros segmentos dentro de la partitura que no sean líneas del pentagrama y, por lo tanto, no tengan la misma inclinación. Una vez extraída se analizará, mediante trigonometría, la inclinación para obtener la solución deseada. Este método es efectivo y no es muy costoso computacionalmente. Sin embargo, no proporciona seguridad ya que siempre cabe la posibilidad de que el algoritmo escoja para evaluar un segmento no apropiado que de otra inclinación distinta a la buscada (de hecho, en la implementación desarrollada era lo más frecuente). Para acelerar el proceso se puede realizar la transformada únicamente sobre una parte de la partitura, por ejemplo, en un único pentagrama.

- **Transformada de *Fourier***

  Mediante este método se puede visualizar de forma gráfica la inclinación de la partitura. Si se realiza la transformada de *Fourier* bidimensional sobre una partitura cualquiera y se visualiza su módulo, se podrá comprobar gráficamente que el gráfico muestra una especie de cruz cuya inclinación corresponde con la inclinación de la partitura. El problema ahora es detectar la inclinación de dicha cruz, ya que ésta no está formada por dos líneas perpendiculares sino por una nube de puntos. Para llevar a cabo la detección se tendrán en cuenta una serie de factores que agilizarán el proceso de búsqueda. En primer lugar si se ordenan las frecuencias del módulo de la transformada, la cruz es simétrica de forma horizontal y vertical, por lo que para conocer la inclinación únicamente bastará con analizar uno de los cuadrantes de la transformada, por ejemplo el superior derecho (Figura 32). En segundo lugar no es necesario que se busque una línea como tal. Los valores de la transformada que definen la línea corresponden a los píxeles con un mayor valor. Se puede aprovechar esta característica, junto al uso de ranuras (Figura 33). Estas son imágenes binarias que utilizadas de forma adecuada nos permiten conocer la dirección de la imagen. Dado que el fondo de la ranura es negro (valor cero) y la ranura es blanca (valor uno) al realizar el producto entre la imagen y la ranura se obtendrá una imagen que mostrará únicamente la parte de la transformada que entre dentro de la ranura.

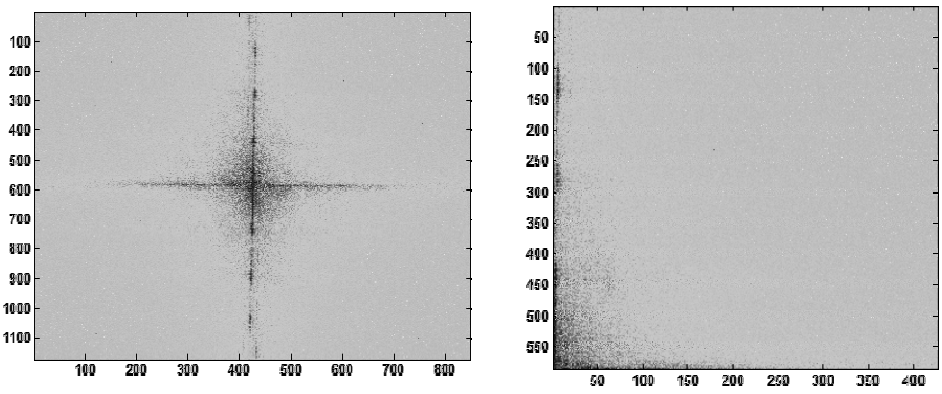


Figura 32 - Izquierda: módulo de la transformada de la partitura, derecha: cuadrante superior derecho del módulo

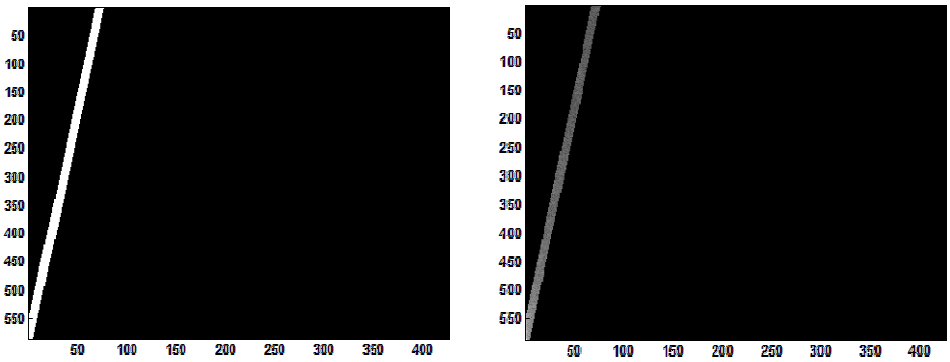


Figura 33 - Izquierda: ranura angular, derecha: cuadrante enmascarado de la transformada

Si se realiza esta operación, variando la inclinación de la ranura en cada iteración, y se obtiene la suma de los píxeles de la imagen enmascarada resultante se podrá comprobar que existirá una inclinación en la que la suma será máxima. Esto indica que los píxeles con mayor valor, es decir los que se encuentran dentro de la cruz, están dentro de la ranura, o de otras palabras, que la inclinación de la ranura coincide con la de la cruz. Este proceso de detección se puede acelerar si en lugar de la transformada de Fourier utilizamos su versión rápida FFT (*Fast Fourier Transform*). Finalmente, este algoritmo se desestimó porque en algunos casos no proporcionaba la inclinación adecuadamente o la proporcionaba de forma inexacta.

- **Transformada *Radon***

  Como alternativa al método descrito por las proyecciones Y, en lugar de rotar la imagen para realizar la proyección, se puede rotar el eje de proyección sobre la imagen. Mediante este procedimiento se obtendrán los mismos resultados y se evitará la sobrecarga computacional. Para implementar este algoritmo se puede hacer uso de la transformada de *Radon*, la cual computa las proyecciones de una imagen a lo largo de determinadas direcciones, como ya se definió anteriormente. Este algoritmo será el que finalmente se utilice puesto que, según se ha diseñado, es capaz de proporcionar la inclinación de la partitura con 0.1º de precisión, de -90 a 90º.

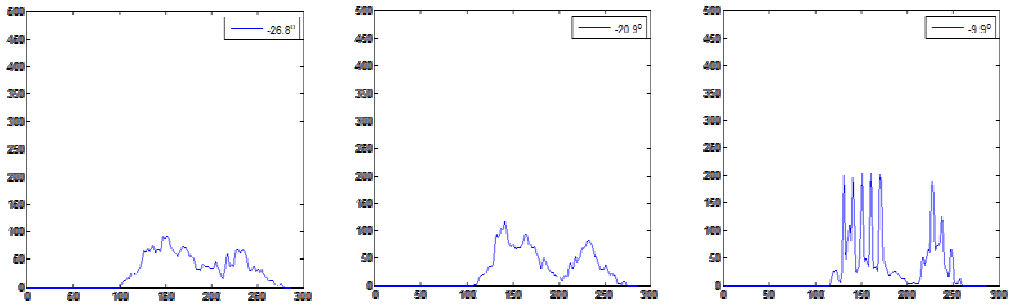


Figura 34 - Diferentes valores de la proyección realizada por la transformada Radon

La implementación de la última transformada descrita en el sistema final no utiliza la partitura completa para realizar el análisis de la imagen, sino una porción de ésta de tamaño 200 x 200 con centro en el centro de la partitura. Esto se realiza para reducir el número de operaciones necesarias al ejecutar la transformada, disminuyendo el tiempo de procesamiento. El único

requisito necesario para que funcione bien el algoritmo es que en dicha porción de imagen exista al menos una línea de pentagrama de la que se pueda extraer la inclinación.

Una vez se obtenga la transformada se deben procesar los datos para obtener los picos deseados de la proyección. En primer lugar se redondearán los valores de la transformada a la baja para obtener valores enteros. Posteriormente, se definirá un umbral dado por la mitad del ancho de la parte de la imagen que se analiza. Todos los valores que estén por debajo de dicho umbral se desecharán. Este umbral se basa en la suposición de que la longitud de las líneas del pentagrama, en la parte de la imagen que se evalúa, es más grande que la mitad del ancho de dicha parte.

Posteriormente, para detectar el ángulo adecuado, se debe buscar aquel ángulo que nos de, en general, un mayor nivel en la proyección debido a la suma de los píxeles de un pentagrama. Para ello, se realizará una especie de picómetro para las proyecciones que almacenará el valor máximo para cada posición de la proyección y el ángulo que lo da. Realizando la moda de dichos ángulos podemos descubrir el ángulo que nos ofrece el mejor resultado en la proyección.

Dado que existen numerosos valores en la transformada Radon con valor máximo cero, que no aportan información, los rechazamos y les asignamos a cada uno un valor negativo trivial. Esto permite que la moda de los máximos no sea cero.

El ángulo que nos ofrece mejor resultado viene dado por la moda del vector de máximos. El resultado de la moda lo dividimos entre 10 para obtenerlo en grados (ya que de -90 a 90º con un paso de 0.1º hay 1800 elementos) y se lo restamos a 90º (ya que esta es la referencia que utilizamos), obteniendo finalmente el ángulo de inclinación.

4.3.6. Corrección de la inclinación

En este paso se gira la imagen el número de grados determinado por el paso anterior. Para ello, se hace uso de la función "*imrotate*" del "*toolbox*" de procesado de imagen de MATLAB. Dado que dicha función deja un fondo negro al rotar la imagen y se desea blanco, antes de corregir la inclinación se realiza el negativo la imagen para rotarla con el fondo negro y después volver a realizar el negativo para obtener la imagen corregida con el fondo blanco.

El resultado final es la partitura adaptada, en formato binario y sin inclinación.

4.4. Segmentación pentagramas y símbolos

Este bloque constituye uno de los elementos fundamentales en cualquier sistema de reconocimiento óptico de música. Se encarga de analizar la imagen y extraer los pentagramas y sus símbolos en el formato adecuado para el buen funcionamiento de los bloques posteriores.

Como se puede ver en la Figura 35, el bloque recibe como parámetros de entrada la imagen de la partitura procesada y la imagen de la partitura procesada con menos ruido (ver bloque anterior) y como parámetro de salida, una variable con los símbolos segmentadas de la partitura.

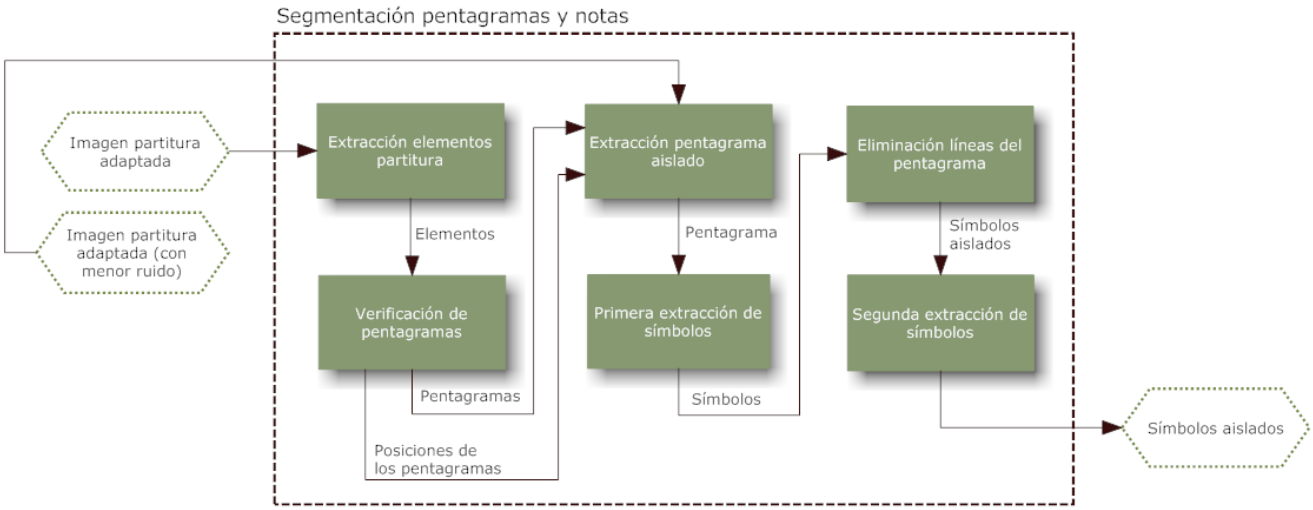


Figura 35 - Diagrama de bloques de la segmentación

Las etapas que conforman este bloque son las siguientes:

4.4.1. Extracción elementos partitura

Se analiza la imagen de la partitura procesada para extraer los pentagramas existentes. Realmente no se buscan pentagramas como tal, sino elementos que puedan ser pentagramas.

Para ello, basándonos en la suposición de que el fondo es blanco, se analiza la existencia de filas, dentro de la partitura, donde la existencia de píxeles negros (que formen parte de algún pentagrama) no supere un umbral determinado. Dichas filas se almacenan como candidatos para cortar la partitura en esa posición. Una vez se tienen todos los candidatos, se extraen los elementos definidos entre dos cortes obteniendo el mismo resultado que si cortásemos una partitura real de lado a lado, mediante cortes horizontales. Este algoritmo no permite distinguir entre elementos, por lo que de analizar una partitura real obtendríamos como elementos el título de la obra, el autor, la letra de la canción, algunas anotaciones y otros elementos, aparte de los pentagramas como se puede observar en la Figura 36.

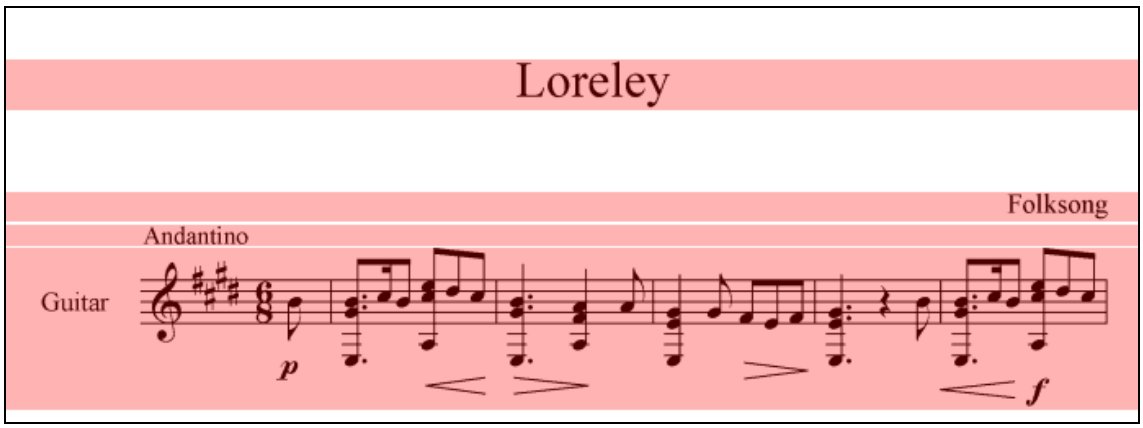


Figura 36 - Elementos extraídos de una partitura

El umbral mencionado anteriormente no se comporta como tal. Realmente, la proyección Y de la partitura se adapta mediante la disminución de un valor igual a la mitad del valor más frecuente. Esto provoca que gran parte de los valores cercanos a las posiciones de la separación entre los pentagramas tomen valor cero, indicador de ausencia de píxeles negros.

4.4.2. Verificación de pentagramas

La siguiente etapa se encarga de determinar si los elementos extraídos en la etapa anterior realmente son pentagramas o no. Para ello, se realizan las proyecciones de los elementos y se buscan cinco picos en la proyección Y que correspondan a las líneas del pentagrama.

Para determinar dichos picos se ha establecido un umbral de 2/3 el valor máximo de dicha proyección. Este umbral intenta representar la longitud de las líneas del pentagrama. Toda la información que se encuentre por debajo de ese umbral será rechazada.

Si realmente el elemento evaluado se trata de un pentagrama, la proyección Y debería presentar claramente cinco picos que se deben aislar para conocer el máximo de cada uno. Dichos máximos nos darán la posición exacta de cada línea.

Dichos elementos se almacenarán como pentagramas reales, junto a los atributos de cada uno (cortes del pentagrama con respecto a la partitura, proyecciones, posiciones de las líneas del pentagrama…), y el resto se desechará.

4.4.3. Extracción pentagrama aislado

Este paso se encarga de seleccionar uno de los pentagramas anteriormente definidos para proceder a su análisis y a la extracción de notas en las siguientes etapas. Para ello, utilizando los datos relativos a las posiciones de cada pentagrama se extrae un pentagrama determinado a partir de la imagen de la partitura con menor ruido. De esta forma se asegura que las siguientes etapas dispongan de la partitura en las mejores condiciones posibles. Este proceso se repetirá para cada uno de los pentagramas extraidos de la partitura.

4.4.4. Primera extracción de símbolos

Esta etapa se encarga de realizar una primera extracción de elementos del pentagrama. De forma análoga a como se definieron los cortes a la hora de analizar y extraer los pentagramas de la partitura, se concretan los cortes para extraer los símbolos dentro del pentagrama. Si para extraer los pentagramas de una partitura real se realizaban cortes horizontales, para extraer los símbolos de la partitura se realizarán cortes verticales. Antes, los cortes se definían al encontrar filas vacías de píxeles negros. Ahora, se realiza la proyección X de la partitura y se establece un valor umbral por debajo del cual se marca un punto de corte. Este umbral, variable para cada caso, se establece como cinco veces el grosor de las líneas del pentagrama, lo que implica definir puntos de corte donde no haya símbolos, sino solo las cinco líneas. De ahí la importancia de la existencia de dichas líneas para el correcto funcionamiento del programa.

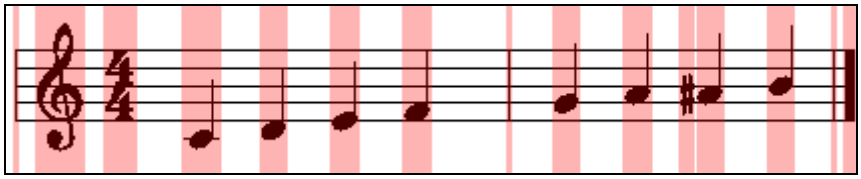


Figura 37 - Extracción de notas de un pentagrama

El valor del grosor de las líneas del pentagrama se obtiene a partir de una modificación del proceso de codificación RLE (*Run Length Encoding*), descrito en el apartado 2.3.1. Dicho

método codifica por pares el tipo de valor del píxel inicial seguido de los píxeles consecutivos con el mismo valor que se repitan. La modificación hecha realiza esta codificación pero agrupando en un vector la codificación para los píxeles blancos y en otro los píxeles negros. Hallando la moda del vector de píxeles negros se obtendrá el grosor de las líneas del pentagrama y realizando la de blancos se obtendrá el espacio entre líneas.

El umbral definido funciona bastante bien para la mayor parte de los casos, no obstante se definió de forma alternativa otro umbral para los casos problemáticos (casos en los que no funcione bien el umbral definido). Este umbral, por defecto desactivado (a menos que se active mediante la modificación del código), consta de varias fases para su obtención. En primer lugar se desechan los valores cero de la proyección X y posteriormente se determina el valor mínimo de la proyección resultante. Dicho valor se restará a todos los valores de la proyección y finalmente se definirá el umbral como:

$$umbral = \left[\frac{media(proyección)}{2} + valorMinimo \right]$$

Siendo *media(proyección)* la media aritmética de los valores obtenidos y "*valorMinimo*" el mínimo de la proyección.

4.4.5. Eliminación líneas del pentagrama

Para cada símbolo se ejecuta un proceso que elimina las líneas del pentagrama. En etapas anteriores, las líneas del pentagrama fueron útiles para extraer determinados datos de la partitura, pero de cara al funcionamiento de etapas posteriores la existencia de dichas líneas dificulta el proceso de clasificación.

La tarea de eliminar las líneas no es trivial. En un principio se estudió el desarrollo de un módulo para poder extraer líneas de cualquier tipo de grosor, pero el principal problema es que es muy difícil crear un algoritmo que sea capaz de diferenciar entre píxeles de líneas del pentagrama y píxeles de símbolos musicales. Tal como se explica en el capitulo 2, se han estudiado numerosas técnicas [52][53] pero ninguna consigue unos resultados realmente buenos, por lo que los mismos autores que realizaron dichos estudios recomiendan probar diferentes algoritmos para diferentes tipos de partitura.

Para resolver el problema de la eliminación de las líneas se han definidos dos técnicas a elegir a la hora de ejecutar el programa. La primera de ellas se encarga de buscar líneas de grosor de un píxel, mientras que la segunda se encarga de buscar líneas de pentagrama azules.

El algoritmo utilizado para el primer caso es sencillo. Se buscan píxeles negros cuyo píxel inmediatamente superior e inferior sea blanco. En el caso de que se cumpla esta condición, se considerará dicho píxel como parte de una línea del pentagrama, por lo que se marcará para ser eliminado.

Para el segundo caso, la realización de estas líneas en color azul nos permite identificar exactamente qué píxeles pertenecen al pentagrama pudiendo eliminarlo fácilmente. El azul que se extrae es un azul puro, según componentes RGB de R = 0, G = 0 y B = 255. Se podría haber implementado la posibilidad de tolerar un mayor rango de azules, de cara al trabajo con partitura escaneadas de pentagramas azules, pero no se llegó a desarrollar por considerarse fuera del campo de los objetivos propuestos. Utilizar este método permite la posibilidad de

utilizar cualquier grosor de pentagrama a diferencia del otro, ya que permite aislar completamente los símbolos de las líneas, facilitando la tarea en gran medida.

Esta etapa se podría haber desarrollado de múltiples formas. Uno de los métodos ideados en un principio consistía en eliminar las líneas directamente del pentagrama para después extraer los símbolos mediante alguna técnica de detección de objetos aislados. El principal problema que surgió a la hora de resolverlo es que las líneas por lo general cuentan con ciertas irregularidades y deformaciones, sobretodo si se trata de líneas de partitura escaneadas, por lo que se prefirió simplificar el problema y tratarlo a nivel de símbolo.

4.4.6. Segunda extracción de símbolos

Esta etapa se encarga de realizar una segunda segmentación en busca de posibles elementos que no se hayan aislado por la naturaleza de los cortes realizados en el pentagrama (ver Figura 38). Un ejemplo muy claro es el tiempo del pentagrama. Utilizando los módulos anteriores se obtendrá el tiempo unido, es decir, numerador y denominador en el mismo símbolo. Sin embargo, a la hora de comparar símbolos para clasificar de forma adecuada esto puede conducir a error, por lo que es conveniente separar el numerador y el denominador. Mediante un análisis por componentes conectados (utilizando la función "*bwlabel*" de MATLAB) usando conectividad de 8 vecinos para cada símbolo, se consiguen aislar todos los elementos que se encuentren. Así mismo, no todos se almacenan, sino sólo los que superen un determinado umbral establecido por el área media para cada símbolo. Esto permite eliminar posibles impurezas que pudieran aparecer y que dieran lugar a símbolos inexistentes.

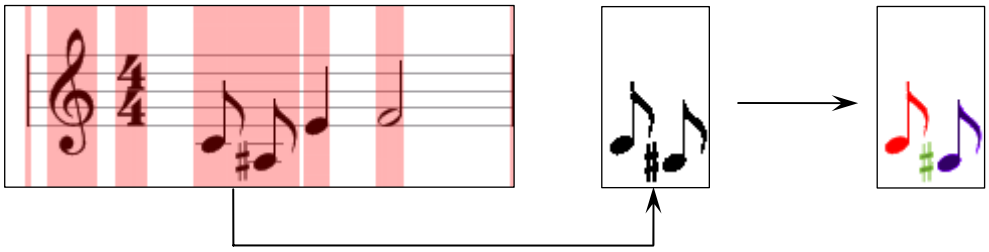


Figura 38 - Segmentación por etiquetado de componentes conectadas

Para evitar falsos símbolos extraídos, resultado de las impurezas de los símbolos originales, se ha definido un algoritmo que los descarta. Dicho algoritmo determina el área de cada objeto encontrado en la imagen (incluyendo las impurezas) y posteriormente realiza la media de dichas valores. Los objetos cuya área no superen un umbral definido por la tercera parte del área media se consideran impurezas y se rechazan.

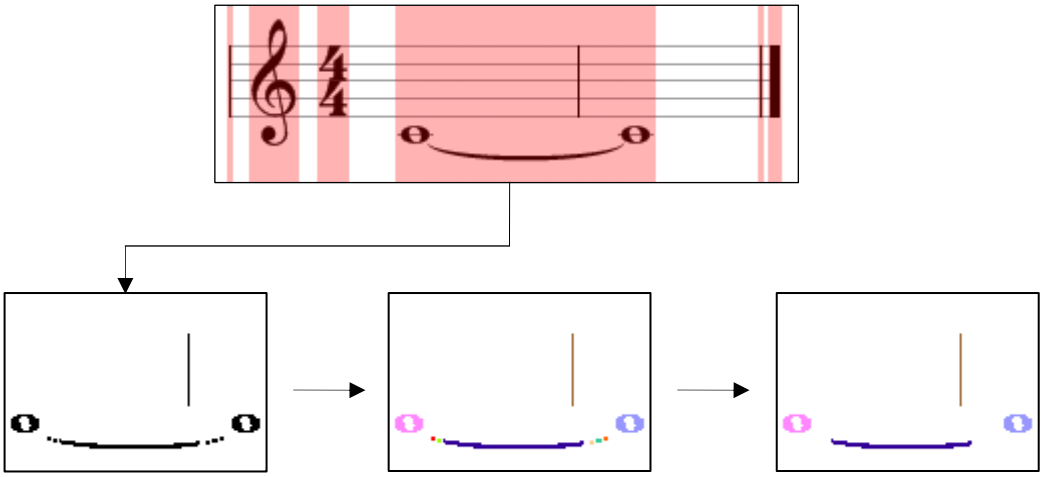


Figura 39 - Descarte de las impurezas de la imagen

4.5. Clasificación elementos

Este bloque se encarga de la clasificación de los símbolos dentro del abanico de posibilidades definido por la base de datos. En la Figura 40 se puede ver el diseño de este bloque.

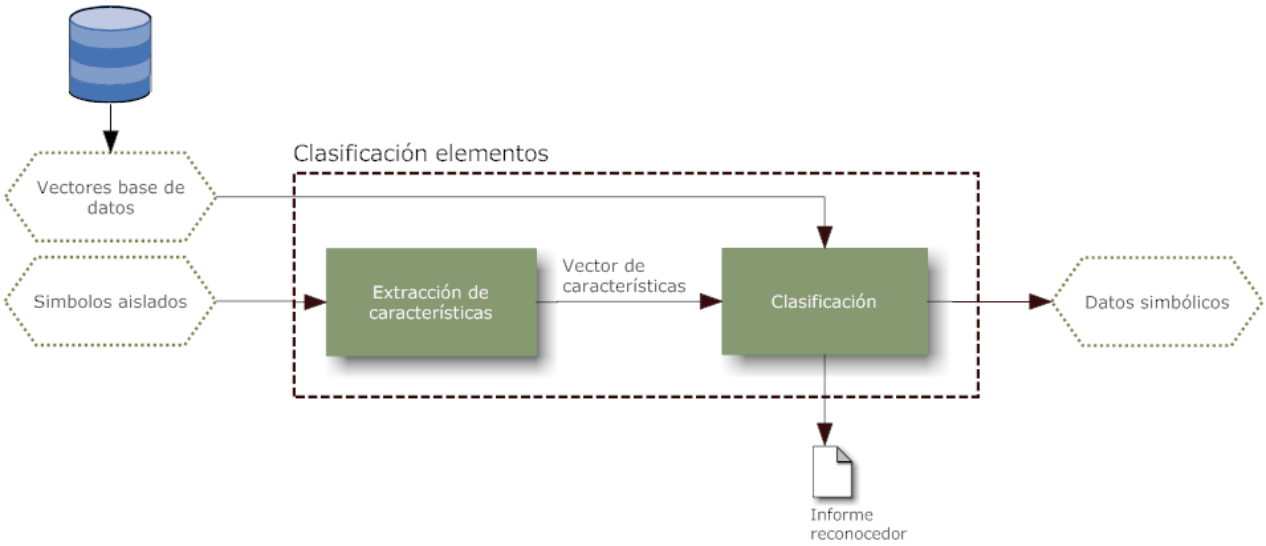


Figura 40 - Diagrama de bloques de la clasificación de los elementos

Las etapas que conforman el bloque se encargan de extraer el vector de características de los elementos a evaluar, cargar los vectores de características de la base de datos y realizar la clasificación mediante la comparación de los vectores. De forma más específica, la tarea desarrollada por cada etapa se define a continuación:

4.5.1. Extracción de características

Para caracterizar el elemento musical de forma adecuada se ha optado por utilizar una serie de características que definan la morfología de la imagen. Para ello, se han utilizado los 30 primeros descriptores de *Fourier* y algunas propiedades más: solidez, número de discontinuidades y centroide

Para la extracción de características se debe normalizar la imagen de forma que se puedan comparar de forma objetiva todos los elementos bajo el mismo contexto. Esta normalización depende del tipo de característica a extraer, por lo que se diferencian dos tipos (Figura 41). La primera comprende la eliminación del espacio sobrante alrededor del símbolo, lo que se traduce en determinar el rectángulo mínimo que puede contener al elemento (*bounding box*), y el reescalado a un tamaño de 50 x 50 píxeles. Este tipo de normalización se utiliza para la extracción de las características de número de saltos abruptos, centroide y solidez. Por otra parte, la segunda, además de realizar el algoritmo *bounding box*, reescala el símbolo para cubrir de forma máxima un cuadro de 100 x 100 píxeles sin perder la relación de aspecto original y obtiene el contorno mediante un filtro que implementa el método de *Roberts*. Este último tipo de normalización se requiere para la extracción de los descriptores de *Fourier*, los cuales también sufren un proceso de normalización para hacerlos invariantes frente al escalado y la rotación.

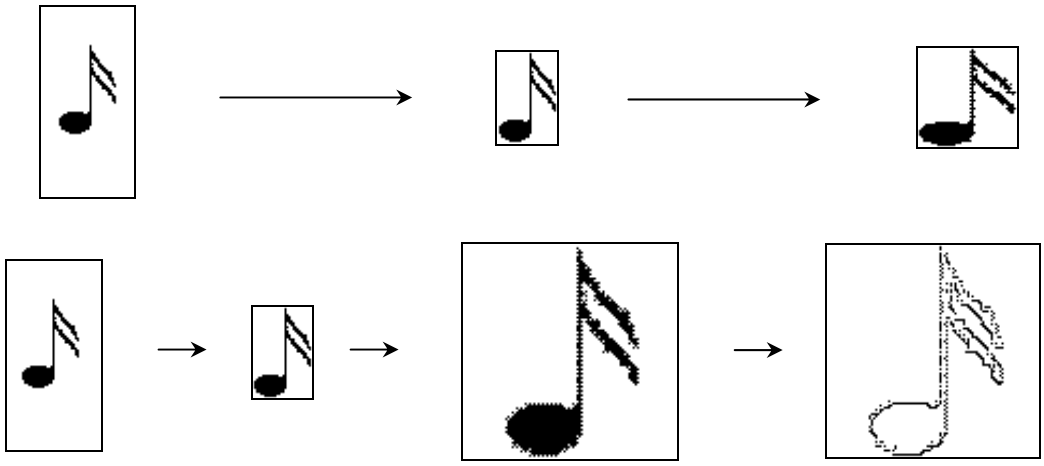


Figura 41 - Métodos de normalización implementados. Arriba: primer método. Abajo: segundo método

Dentro de este apartado de extracción de características se desarrolló un pequeño algoritmo encargado de realizar un filtrado paso bajo variable según el tamaño de la imagen, después de la realización del algoritmo *bounding box*, para difuminar el contorno del símbolo (a más tamaño, más difuminado, de forma proporcional para que todos tengan el mismo grado de difuminado en el caso de que todos tengan el mismo tamaño). En los casos que exista ruido en la imagen este algoritmo puede ser útil para suavizar el borde evitando posibles “picos” debido al ruido, adaptando la morfología de la imagen para facilitar la tarea del reconocedor y obtener mejores resultados. Sin embargo, aunque teóricamente este algoritmo debería ser beneficioso, se pudo comprobar de forma práctica que no lo era, principalmente porque, aunque suaviza el contorno no corrige correctamente la morfología de la imagen, quedando por ejemplo, una línea recta con ruido ligeramente curvada en las zonas de ruido.

El filtro paso bajo implementado crea una máscara del tipo:

$$mascara = \frac{1}{m^2}\begin{bmatrix} 1 & \dots & 1 \\ \dots & \dots & \dots \\ 1 & \dots & 1 \end{bmatrix}$$

Donde “m” es un valor que caracteriza el grado de desenfoque de la imagen que viene dado por la expresión

```
m = alto_imagen*0.060
```

De esta forma cuánto más grande sea la imagen mayor será el grado de desenfoque de forma proporcional, para que todas las imágenes tengan el mismo grado de desenfoque en el caso de que posteriormente fueran redimensionadas.

Existen numerosos descriptores a implementar, no obstante, se deben utilizar aquellos que definan bien las características de nuestras imágenes. Por ejemplo, no sirve utilizar como posibles descriptores la anchura y la altura de la imagen si todas ellas cuentan con el mismo tamaño. Tras probar numerosos descriptores se decidió elegir cuatro fundamentales:

- **Número de saltos abruptos entre píxeles contiguos**. Este algoritmo realiza las proyecciones X e Y del símbolo a evaluar y comprueba para cada proyección si la diferencia de cualquier par de píxeles consecutivos supera el umbral de la mitad del tamaño de fila (si se trata de proyección X) o la mitad del tamaño de columna (para la proyección Y). En caso de superarlo se puede afirmar que existe un cambio brusco de valor entre dicho para de píxeles y lo contabilizamos.

- **Centro de masas o centroide**. Este concepto geométrico define un centro sobre el que recaería todo el peso del cuerpo. Para conseguirlo en una imagen digital se deben almacenar todas las posiciones X-Y de los píxeles y realizar la media de las componentes X y de las componentes Y. El resultado de cada operación dará las coordenadas del centroide

- **Solidez**. Este concepto se define como el cociente del área de la imagen entre el área convexa, siendo el área de la imagen el número de píxeles negros y el área convexa el número de píxeles contenidos dentro de la envolvente convexa. Dicha envolvente trata de encerrar al objeto utilizando como esquinas los bordes exteriores del símbolo. De forma intuitiva se puede entender la envolvente convexa como la envolvente descrita por una goma elástica que encierra cualquier tipo de objeto. La envolvente convexa de un círculo sería el mismo círculo, mientras que la de una estrella de cinco puntas sería un pentágono.

- **Descriptores de Fourier**. Mediante la transformada de Fourier se puede caracterizar la morfología de la imagen como ya se comentó anteriormente. En este proyecto únicamente se utilizarán los 30 primeros para acelerar el proceso de ejecución y caracterizar la imagen mediante una aproximación de la forma, evitando detalles que puedan afectar al proceso de reconocimiento. El proceso de adquisición de los descriptores se detalla a continuación:

    - ***Obtención del contorno del símbolo:*** se utiliza el algoritmo de *Roberts* que encuentra bordes usando la aproximación de *Roberts* a la derivada. Devuelve contornos en aquellos puntos donde el gradiente de la imagen es máximo.

    - ***Conversión al plano complejo:*** se transforma el contorno de la imagen de ejes X e Y al plano complejo para caracterizarlo mediante un único número y trabajar mejor con los datos.

- ***Normalización de los datos***: se resta a los valores su media para conseguir coeficientes invariantes respecto a la posición. Después, se ordenan según la fase de cada número complejo de menor a mayor.

- ***Realización de la transformada de Fourier:*** para ello se utiliza la transformada rápida de *Fourier* unidimensional sobre los datos del contorno en el plano complejo.

- ***Procesado de los datos:*** posteriormente se debe realizar un procesado de los datos obtenidos para que los descriptores de *Fourier* sean invariantes ante el escalado y la rotación, de forma que la misma imagen original rotada o escala nos proporcione los mismos coeficientes que la original. Para ello, se determinan los módulos de cada número complejo obtenido con la transformada y se dividen por el mayor módulo. Dichos resultados conformarán los descriptores de *Fourier*.

- ***Selección de los descriptores:*** como ya se ha comentado, no se toman todos los descriptores de *Fourier*, sino los 30 primeros. De no haber tantos, los que falten para llegar a dicho valor tomarán el valor cero.

4.5.2. Clasificación

Esta etapa compara los vectores de características de la base de datos con los vectores de los símbolos a evaluar. Para ello, se utiliza un algoritmo de clasificación por distancia mínima, el k-NN (*k-Nearest Neighbours*), que se encarga de comparar distancias entre el vector de características del símbolo evaluado y los vectores de características de los símbolos de la base de datos y elegir la clase más frecuente entre los "*k*" vecinos más cercanos (entendiéndose por vecino, otro símbolo con otro vector de características distinto) para realizar la clasificación. Como medida de distancia, se ha utilizado la distancia Euclídea que se define mediante la expresión:

$$dist = \sqrt{\sum_{i=1}^{n} \left(Xm_i - Xt_i \right)^2}$$

Donde X_{mi} se corresponde con la muestra de la base de datos y X_{ti} se refiere a la muestra a evaluar, ambas en la posición "*i*".

Si existen varias clases igual de frecuentes que optan a ser candidatas para clasificar el símbolo (ya que proporcionan el mismo valor para la moda) dentro del grupo de los "k" vecinos más cercanos, se seleccionará la clase cuya distancia media proporcione un valor menor.

En el programa se recomienda un valor de cinco, aunque es un parámetro que puede tomar cualquier valor entero positivo, par o impar.

Los resultados obtenidos al ejecutar este algoritmo se muestran por línea de comandos y paralelamente se crea un informe detallado que almacena datos como la cantidad de ejemplos de la base de datos, los vecinos más cercanos y las distancias de cada uno para el símbolo evaluado.
Este sistema reconoce los símbolos analizando la morfología (es decir, la forma de la imagen) a diferencia de otros sistemas OMR desarrollados que se encargan de descomponer el símbolo

en elementos principales. Esto tiene ciertas ventajas e inconvenientes. Por un lado, el descomponer el símbolo en sus elementos fundamentales (en el caso de una corchea serían la cabeza de la nota, la plica y el corchete) obliga a realizar un algoritmo que se encargue de buscar dichos elementos y los extraiga del símbolo. Después, se debe realizar el reconocimiento de los elementos separados para poder clasificar la nota y asignar el tono adecuado mediante otros algoritmos.

La principal razón por la que no se desarrolló este método desde el principio fue porque se pensó que sería interesante apostar por otro distinto al que se utiliza normalmente, sobretodo sabiendo que el implementado reduce las posibilidades de fallo al tener en cuenta menos variables que puedan fallar durante el proceso.

4.6. Verificación elementos y detección de tono

Este bloque se encarga de extraer los atributos de los símbolos y realizar determinadas comprobaciones sobre la partitura, tales como la identificación de símbolos compuestos, la verificación de compases y la búsqueda de posibles errores de los datos clasificados. La disposición de las etapas en dicho bloque se puede observar en la Figura 42.

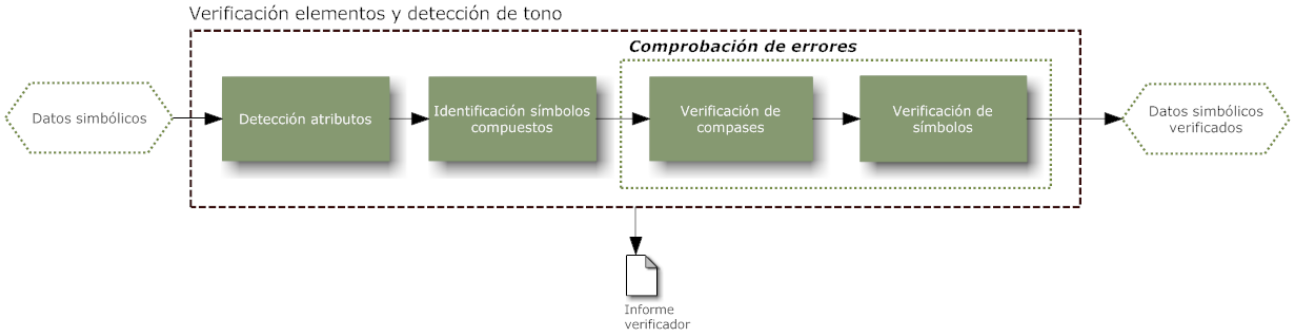


Figura 42 - Diagrama de bloques de la verificación de elementos.

Las etapas de la que consta el bloque son:

4.6.1. Detección atributos

Esta etapa se encarga de detectar y extraer los atributos adecuados para cada símbolo musical. Para el caso de una clave, el atributo vendrá definido por los tonos correspondientes a las cinco líneas del pentagrama, tonos que varían en función del tipo de clave. Para una nota, el atributo será el tono de dicha nota. Los atributos que podemos asignar varían en función del parámetro, de hecho hay algunos símbolos que no cuentan con atributos. En la tabla siguiente se pueden observar los atributos definidos para los símbolos que sí cuentan con ellos:

Símbolo/s	Atributo
Notas musicales	Tono
Clave	Posiciones de los tonos
Tiempo	Valor numérico del tiempo
Silencio de Redonda / Blanca	Tipo (Redonda o blanca)

Figura 43 - Relación entre los símbolos y sus atributos

A continuación se exponen los métodos desarrollados para extracción de los atributos:

4.6.1.1. Tono de las notas

Para detectar el tono de las notas del pentagrama se necesitan conocer dos parámetros fundamentales: la posición de las líneas del pentagrama y la clave del pentagrama. De no existir alguno de estos dos parámetros no se podrá llevar a cabo el proceso. Para ello, se debe buscar, al igual que lo haría cualquier persona en la realidad, la cabeza de la nota y ver en qué posición del pentagrama se encuentra.
La dificultad del algoritmo consiste en determinar la posición exacta de la cabeza de la nota. Dada la naturaleza de cada una y la varianza entre símbolos que puede existir en el caso de notas manuscritas, no podemos definir una posición fija dentro de cada símbolo para la cabeza de la nota. Es por eso que se define una máscara de un tamaño aproximado a la cabeza de la nota para recorrer la imagen y determinar la posición. Suponemos que si se recorre la imagen con dicha máscara obtendremos un área mayor de máscara en el punto donde se encuentre la cabeza.

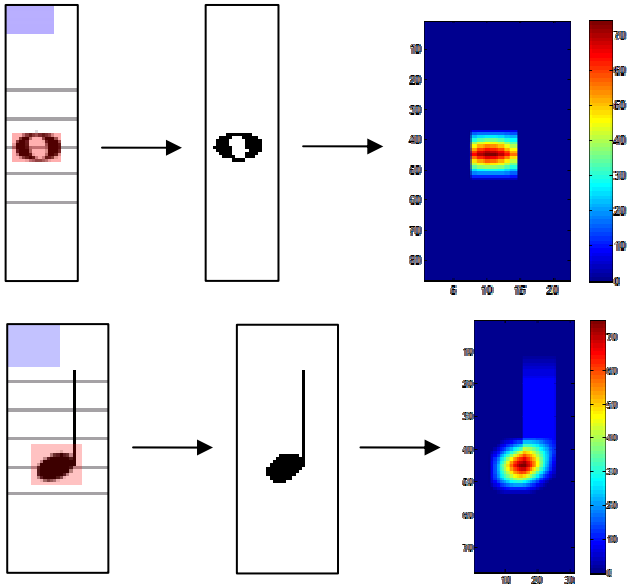


Figura 44 - Búsqueda de la cabeza de la nota. Izquierda: nota y la máscara definida que recorre la imagen. Centro: símbolo evaluado. Derecha: área de la máscara para cada píxel.

Se han implementado dos algoritmos utilizados en función de si la nota de la cual queremos conocer el tono se trata de una redonda o doble, u otra cualquiera (es decir, de si sólo tiene cabeza o tiene plica). En el primer algoritmo, conociendo el tamaño exacto de la nota redonda o doble mediante su "*bounding box*" se define una máscara del mismo tamaño. Dicha máscara se desplazará a lo largo de la imagen del símbolo original (con los espacios superiores e inferiores de haber extraído el símbolo del pentagrama) sin salir fuera del borde. Para cada posición se determinará el área de la imagen que encierra la máscara y se almacenará dicho valor en la posición del píxel correspondiente al centro de la máscara. Una vez se hayan evaluado todos los puntos se determinará el píxel que dé un área mayor, obteniendo la posición de la nota con respecto al símbolo extraído del pentagrama (ver Figura 44). Conociendo ese dato y las posiciones de las líneas del pentagrama se podrá determinar su posición absoluta y por tanto, el tono definido por la clave. En el caso de que se trate de otra nota distinta, se realizará el mismo proceso definiendo una máscara según las expresiones:

$$alto = round(0.257 \cdot altura)$$
$$l\arg o = round(0.30 \cdot altura)$$

Dichas expresiones se han obtenido de forma empírica y nos permiten obtener una máscara de tamaño la cabeza de la nota a partir del tamaño de la plica de la misma.

Conociendo la posición vertical de la cabeza de la nota se puede comparar con las posiciones de los tonos del pentagrama para determinar qué tono es más cercano al píxel. Esta operación se realiza mediante la diferencia en valor absoluto para cada posición del tono. La diferencia que dé un valor mínimo será la posición del tono de la nota.

4.6.1.2. Tipo de silencio de Redonda / Blanca

Los silencios de blanca y de redonda son iguales desde el punto de vista morfológico. Esto es un conveniente a la hora de clasificar los símbolos ya que el reconocedor no podrá ser capaz de distinguir el tipo de silencio si le proporcionamos el símbolo aislado. Para detectar de qué silencio se trata debemos tener en cuenta la información de contexto, en este caso, las líneas del pentagrama.

Ambos silencios se distinguen únicamente por su posición respecto a las líneas del pentagrama. En el caso de que el símbolo se encuentre tocando la cuarta línea, por debajo de ésta, se tratará de una redonda, mientras que si se encuentra tocando la tercera línea por encima de ésta se trata de una blanca. La referencia la tomamos desde la línea inferior del pentagrama, considerada la primera línea.

Para ello, se analiza la proyección Y del silencio extraído del pentagrama con el espacio en blanco superior e inferior. De ahí se obtienen las posiciones del inicio y final del símbolo, de forma vertical. Después se mide la distancia absoluta desde el inicio del símbolo a la cuarta línea y desde el final del símbolo hasta la tercera línea. La menor distancia decidirá el tipo de símbolo. De ser el primer caso será redonda, en otro caso blanca.

4.6.2. Identificación símbolos compuestos

Existen determinados símbolos que se pueden formar a partir de otros. En esta etapa se buscan las relaciones entre esos símbolos para definir los nuevos. En la siguiente tabla se pueden ver las relaciones diseñadas:

Primer símbolo	Segundo símbolo	Símbolo compuesto
Barra divisoria	Barra divisoria	Doble Barra (Barra de final)
Puntillo	Puntillo	Doble punto
Cuatro	Cuatro	4 / 4
Dos	Cuatro	2 / 4
Tres	Cuatro	3 / 4
Dos	Dos	2 / 2
Tres	Dos	3 / 2
Seis	Ocho	6 / 8
Doble barra	Doble punto	Inicio de barra de repetición
Doble punto	Doble barra	Final de barra de repetición

Figura 45 - Tabla con las relaciones de símbolos compuestos

4.6.3. Verificación de compases

Esta etapa conforma el primer elemento del grupo dedicado a la verificación de errores de los datos simbólicos. Su finalidad es la de comprobar que en cada pentagrama la suma de las duraciones de las notas coincide con el valor de duración impuesto por el tiempo del compás. Para entender este concepto se presupone que cada nota musical "suma" una duración determinada al compás vacío de valor cero. La negra suma 1, la blanca 2, la redonda 4, la doble 8, la cuádruple 16, la corchea 0.5, la semicorchea 0.25, la fusa 0.125 y la semifusa 0,0625. De esta forma, si se entiende un compás de 4/4 como una base de cuatro negras obtendríamos una duración de compás de 4. En el caso de 3/2 (tres blancas) un valor de 6. Y así sucesivamente con todos los casos.

En caso de que aparezcan errores, se mostrará por línea de comandos un mensaje indicando la posición del error, así como la métrica esperada y la obtenida. Además, se almacenarán los datos en un informe.

El rango de tiempos definido en el proyecto es limitado. Podemos trabajar con tiempos de 2/4, 2/2, 4/4, 3/4, 6/8 y 3/2.

4.6.4. Verificación de símbolos

Esta etapa se encarga de comprobar que los elementos cumplan con las reglas de teoría musical. En caso de incumplirlas es muy probable que se deba porque se ha realizado una mala clasificación de los símbolos. Entre los puntos a tener en cuenta se encuentran:

- **Contabilización de las claves de Sol o de Fa existentes en el pentagrama**. Útil para casos en los que se repiten varias claves iguales por errores de clasificación.

- **Comprobación de que detrás de un accidente existe una nota a la que se modifica el tono**. De no ser así es muy probable que exista un símbolo mal clasificado.

- **Comprobación de que si nos encontramos dos puntillos nos encontraremos con una barra doble** que definirá una barra de inicio o final de repetición.

- **Búsqueda de números aislados:** los tiempos se definen por pares de números, por lo que si encontramos un número aislado o un número impar de números, se habrá producido algún error de clasificación en las etapas anteriores.

- **Búsqueda de anacrusas** en el caso de que el primer compás tenga una duración distinta a la definida.

Para el caso de las anacrusas, debido a algunas limitaciones con el lenguaje de representación musical *Lilypond*, sólo se han podido definir una serie de valores posibles de duración: 7, 6, 4, 3.5, 3, 2, 1.75, 1.5, 1, 0.875, 0.75 y 0.5. Para evitar errores por encontrar una anacrusa distinta a alguno de los tamaños definidos, se establecerá como valor real aquel valor que se acerque más a la duración real.

Un punto a tener en cuenta es que este bloque define unos valores por defecto en caso de no encontrar algunos datos esenciales para la partitura. En concreto, define una clave de Sol, un tiempo de 4/4 y un compás inicial (para el caso de que haya varios pentagramas en una misma partitura). Teóricamente estos deberían ser extraídos de la partitura, pero en la práctica pueden aparecer errores que hagan que los símbolos no sean segmentados correctamente o puede ocurrir que el reconocedor clasifique dichos símbolos de forma errónea. En estos casos, de no contar con los datos por defecto, el programa sería incapaz de representar la partitura aunque el resto de los símbolos musicales estuvieran bien clasificados.

Además, de encontrar nuevas claves de pentagrama o tiempos se cambiarán los valores oportunos (ya sean los valores por defecto, unos valores definidos por el usuario o valores extraídos de símbolos del pentagrama) a los que definan los últimos símbolos encontrados.

En el caso de que la partitura cuente con ligaduras, hay que tener en cuenta que según se ha diseñado el programa se permiten tanto ligaduras de unión como de expresión, siempre y cuando las notas involucradas sean adyacentes (obviando barras divisorias o accidentes que les separen). Se ha tomado esta decisión después de comprobar que, por cuestiones de la segmentación, es muy difícil determinar correctamente las notas involucradas en la ligadura. Generalmente la etapa de segmentación acorta la ligadura eliminando parte de los extremos como impurezas, por lo que, para solventar el problema, se debería rediseñar en gran parte dicha etapa, tarea que queda fuera del alcance del proyecto.

El bloque genera un informe con los datos analizados e información de interés, como los posibles errores, y guarda en una variable los datos simbólicos finales del pentagrama, incluyendo los datos de los nuevos símbolos compuestos encontrados.

Hay que tener en cuenta que, dentro del informe generado, se podrán ver dos tipos de errores: avisos y errores críticos. Mientras que el primero avisa de un problema, el segundo da información de que la partitura puede no generarse correctamente (refiriéndose al trabajo realizado por el módulo encargado de representar la partitura).

4.7. Traductor de formato

Este bloque toma los datos simbólicos de la partitura y los traduce a un formato adecuado para su posterior tratamiento. En este caso se ha optado por el lenguaje *Lilypond* para poder representar la partitura de forma gráfica y acústica, aunque se podría haber utilizado cualquier otro formato.

Para realizar este proceso, se analiza cada dato simbólico y se codifica, según un modelo que se definirá en el apartado 9 al hablar de la estructura del lenguaje *Lilypond*, hasta finalizar la partitura entera, momento en el que se incluye todo lo obtenido dentro de una estructura de texto que define otros parámetros como la versión del programa utilizada (2.12.2) para codificar o la velocidad de reproducción de la partitura en MIDI (90 negras por minuto)

Este módulo generará un fichero de extensión “ly” (formato *Lilypond*) que tendrá que ejecutarse para obtener la partitura en PDF, PostScript y MIDI. Es necesario tener instalado el programa Lilypond para poder ejecutar el código, de otra forma no funcionará.

Es posible que en determinados casos al ejecutar el código generado únicamente se obtenga un informe (*log*). En estos casos la partitura no se ha definido correctamente, por lo que existe

un error en el código y *Lilypond* lo rechaza. Aunque en la mayor parte de los casos no ocurre, hay excepciones que no generan salida. Es importante tener en cuenta los mensajes generados por el bloque anterior, sobretodo si se produce algún error del tipo "*puede que la partitura no se genere correctamente*" puesto que si esto ocurre, existe alguna probabilidad de que *Lilypond* no sea capaz de interpretar el código realizado.

4.8. Otros bloques

Existen otros bloques, además de los mencionados, que participan en el proceso de ejecución del sistema:

4.8.1. Interfaz

El bloque de la interfaz es el encargado de la comunicación entre el usuario y el programa, permitiendo la adquisición de determinados datos necesarios para la ejecución mediante cuadros de diálogo.

Este bloque no está formado por un conjunto de módulos sino que forma parte del módulo de arranque, encargado de interconectar todos los bloques. Se ha optado por este diseño porque la interfaz debe ejecutarse durante el transcurso del programa, mostrando mensajes por línea de comandos o solicitando datos para la continuación del proceso general.

No se ha desarrollado una interfaz gráfica completa (GUI), ya que no era uno de los objetivos fundamentales del proyecto, sino una basada en mensajes por línea de comandos y cuadros de diálogo. Aunque dicha interfaz es válida al cumplir correctamente su objetivo, la creación de una nueva se propone como posible mejora en el apartado 7.5

4.8.2. Creación de bases de datos

Existe un bloque dedicado a la creación de las estructuras que representan las bases de datos. Para ello, se debe disponer de una base de datos de símbolos musicales adecuada que se adapte convenientemente a las características de las imágenes de las partituras que se vayan a utilizar. No constituye un bloque más de la arquitectura, sino un bloque independiente que se puede ejecutar antes de iniciar el programa principal.

Dicho bloque, básicamente, se encarga de leer las imágenes clasificadas de los símbolos musicales y extraer los vectores de características de forma análoga a como realizaba el bloque clasificador. Dichos vectores se almacenarán en una estructura de datos que etiquetará para cada vector la categoría de símbolo a la que pertenece, en la ruta que el usuario defina.

Es muy importante tener ejemplos que describan muy bien la morfología y las características del símbolo que representen, puesto que estos parámetros determinarán la eficacia del clasificador y serán uno de los pocos elementos que podamos variar para mejorar los resultados.

Dado el carácter extensible del proyecto, se ha definido que la base de datos pueda albergar hasta 99 categorías distintas (por cuestiones de implementación), suficientes para describir la mayoría de piezas musicales. Además, en el caso de que un usuario avanzado modificase el código e implementara más descriptores para definir los símbolos, la base de datos toleraría estos cambios de forma adecuada, adaptándose y guardando todos los elementos, incluidos

los nuevos descriptores, correctamente. Esto se consigue gracias a que antes de crear la matriz de características, se comprueban los descriptores que se extraen al realizar la extracción de características, mediante un ejemplo almacenado en la base de datos (*"test.bmp").* Para evitar problemas al cargar las imágenes, cargamos las carpetas sin tener en cuenta el archivo de prueba mencionado y el fichero oculto *"thumbs.db"* del sistema operativo *Microsoft Windows*.

La creación de una base de datos puede llegar a ser una tarea pesada, más si tenemos en cuenta que debe contar con una determinada estructura para el correcto funcionamiento del programa. Los detalles del proceso de elaboración de la base de datos se encontrarán en el manual de referencia del proyecto, presuponiendo que únicamente generaran dichas bases los desarrolladores del programa.

Para demostrar el funcionamiento del programa se han creado una serie de bases de datos que se pueden utilizar al ejecutar el sistema completo.

4.8.3. Bloque de arranque

El bloque de arranque se encarga de ejecutar toda la arquitectura descrita hasta ahora. Es la puerta de entrada del sistema.

El módulo ("*programaPFC*") que conforma este bloque necesita de unos datos de configuración que se deben concretar para arrancar el programa. Dichos datos se definen al invocar la ejecución del programa y mediante la comunicación con el usuario a través la interfaz.

Hay que tener en cuenta que se han implementado pausas durante la ejecución, para que el usuario pueda leer la información proporcionada en línea de comandos según se ejecuta cada módulo, así como avisos u errores. Además, se habilita la monitorización de los resultados que se van obteniendo en cada módulo: pentagramas y notas extraídas, proyecciones, etc. De producirse algún error, se detiene la ejecución del programa y se lanza un mensaje de error por línea de comandos explicando el fallo.

Los detalles de este módulo se explicarán en apartados posteriores.

4.8.4. Bloque de segmentación de símbolos aislados

Este bloque se encarga de extraer los elementos existentes en una hoja en blanco. Realmente se trata de una modificación del módulo "*extraeNotas2*", descrito en el apartado 4.9.13, con la finalidad de ayudar en la tarea de crear una base de datos.

Si se desea crear una base de datos de símbolos se deberá crear las carpetas oportunas según las reglas definidas en el proyecto e incluir ejemplos de cada clase para completar la base de datos. Para ello, se debe crear o copiar cada ejemplo, guardándolo en formato binario y almacenándolo. Esta tarea puede llegar a ser muy pesada para una base de datos normal.

Para ayudar en la tarea, se puede realizar un único ejemplo por clase, que contenga numerosos símbolos. Imaginemos que dicho ejemplo es una hoja en blanco en el que se han decidido dibujar 100 claves de Sol. Si en lugar de repetir la operación de dibujar, adaptar y

guardar 100 veces, se dibuja 100 veces y se adapta y guarda un único fichero la tarea es más ligera.

La función descrita tiene el objetivo de cargar dicho ejemplo, aislar los dibujos que se encuentren, mediante la técnica de etiquetado por componentes conectados, y guardarlos en archivos independientes de forma inmediata.

4.8.5. Bloque de carga aislada de símbolos

Este bloque permite cargar una colección de símbolos almacenados en una carpeta en archivos independientes para obtener las variables que permitan ejecutar el bloque clasificador de forma separada, evitando la ejecución del programa.

Este bloque es muy útil para realizar comprobaciones y evaluación de la eficacia del bloque clasificador.

4.9. Implementación

En este apartado se estudiará cada uno de los módulos implementados de forma detallada, indicando el funcionamiento, los parámetros de entrada / salida y los procedimientos utilizados. Todos estos módulos se interconectan según aparece en el diagrama de flujo de la Figura 46.

Como ya se comentó, los módulos conforman los bloques de la arquitectura del sistema. Este capítulo se podría haber desarrollado detallando para cada bloque los módulos implicados. Sin embargo, existen módulos que se ejecutan en varios bloques por lo que se prefirió detallar cada uno de ellos de forma alfabética, ayudando al lector en la búsqueda de un módulo concreto.

No obstante, para mantener la estructura mencionada y tener una idea del contenido de cada bloque, se enunciarán de forma breve los módulos que participan en cada bloque:

1. **Preprocesado de la imagen:** cada sub-bloque se ejecuta mediante un módulo principal, "*procesaiImagen*" y "*procesaImagen2*". En este bloque se utilizan otras funciones como "*imagenABinario*" y "*rotarImagenRadon*" de más bajo nivel.

2. **Segmentación pentagramas y notas:** para cada una de las seis etapas se ejecuta un módulo determinado: "*extraePentagramas*", "*adaptaPentagramas*", "*extraePentagramasBinarios*", "*extraeNotas*", "*extraeLineasNotas*" y "*extraeNotas2*" respectivamente.

3. **Clasificación elementos:** el módulo principal que conforma este bloque es "*reconocedor*". Dicho modulo hace uso del módulo de menor nivel "extraeCaract", que a su vez hace uso de otros: "*BoundingBox*", "*negativo*", "*imagenABinario*".

4. **Verificación elementos y detección tono:** el módulo principal que conforma este bloque es "*detectaTono*". Dicho módulo hace uso de otros módulos como "*defineEscalaVacia*", "*defineEscalaSol*", "*defineEscalaFa*", "*reconoceTonoNota*, y "*insertarFila*".

5. **Traductor de formato:** el módulo dedicado a este fin es "representaPartitura"

Además, la gran mayoría de los módulos que conforman estos bloques utilizan algunos de módulos de más bajo nivel que se enumeran: "*proyecciones*", "*negativo*", "*imagenABinario*", "*BoundingBox*".

También hay que mencionar los módulos que forman parte de otros bloques del programa:

1. **Bloque de arranque:** se ejecuta mediante el módulo "*programaPFC*". Utiliza gran parte de los módulos desarrollados en este proyecto.

2. **Creación de bases de datos:** el módulo encargado de tal tarea es "*cargaBD*".

3. **Bloque de segmentación de símbolos aislados:** se ejecuta mediante "*segmentaNotas*".

4. **Bloque de carga aislada de símbolos:** el módulo encargado de tal tarea es "*cargaEjemplos*"

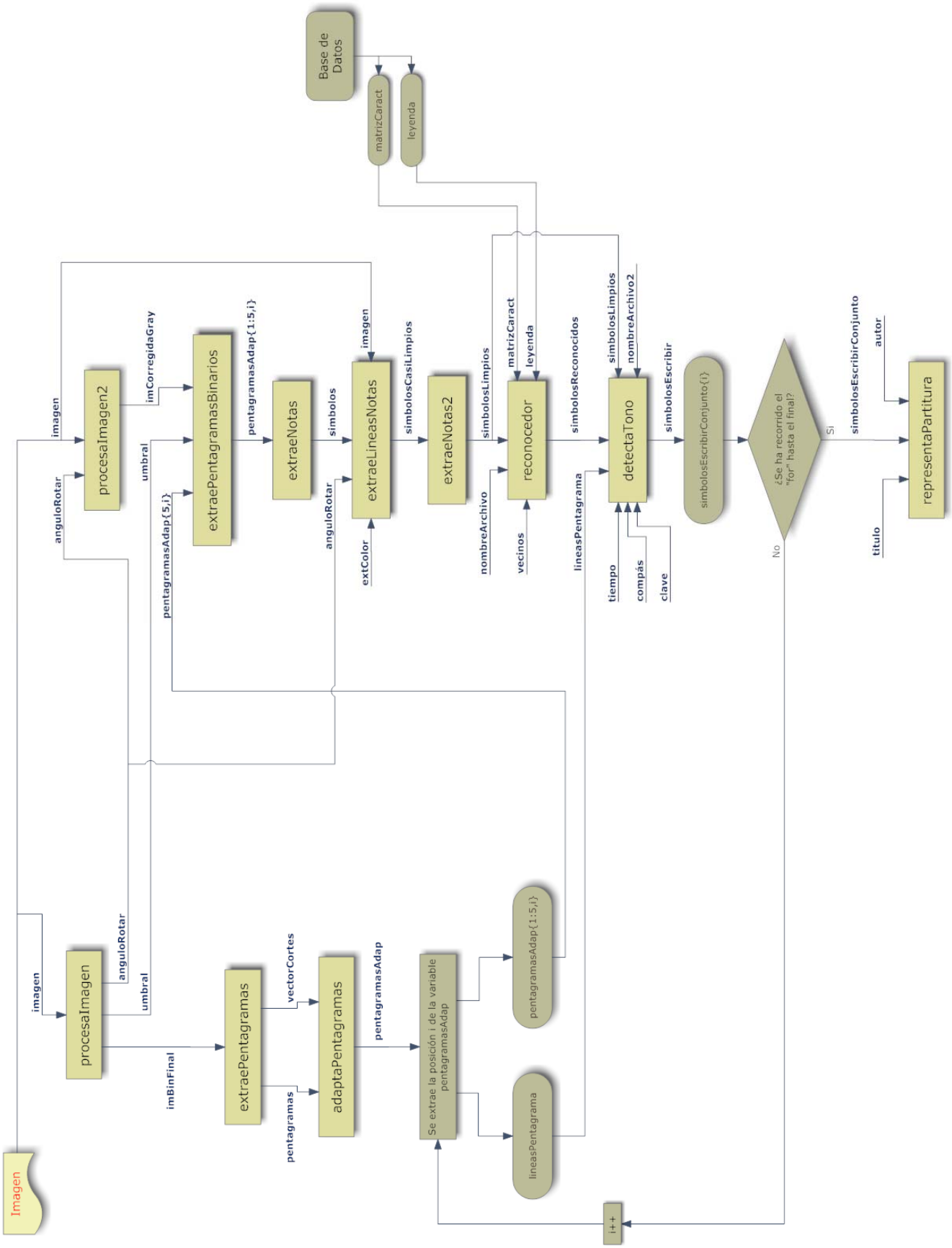


Figura 46 - Diagrama de flujo del sistema realizado

4.9.1. adaptaPentagramas

```
[pentagramasAdap] = adaptaPentagramas(pentagramas,vectorCortes)
```

Este módulo (o función) analiza los datos proporcionados por el módulo "*extraePentagramas*" para determinar si los elementos encontrados son realmente o no pentagramas, desechando los que no lo sean. Los pentagramas reales se almacenan en una estructura junto a otros datos de interés.

Los parámetros de entrada utilizados son:

- **`pentagramas:`** *cell array* (5 x i) con los elementos extraídos en la etapa "*extraePentagramas*", donde "i" representa el número de elementos encontrados en la partitura.

- **`vectorCortes:`** vector con la posición de los cortes horizontales que habría que hacer con respecto a la imagen original para obtener los pentagramas. Los cortes se almacenan por pares para cada partitura.

El único parámetro de salida "**`pentagramasAdap`**" es un *cell array* de tamaño (5 x i) que almacena los "*i*" pentagramas reales encontrados en la variable de entrada, entre otros datos que se detallan:

- **Columna 1:** imagen del pentagrama extraído
- **Columna 2:** posiciones de las líneas del pentagrama
- **Columna 3:** proyección X
- **Columna 4:** proyección Y
- **Columna 5:** posiciones de los cortes con respecto a la partitura

4.9.2. BoundingBox

```
[simbSNBB] = BoundingBox(simbSN)
```

Este módulo se ejecuta en gran parte del resto de los módulos implementados en el programa principal. Se trata de una función encargada de, dado un símbolo cualquiera en formato binario, extraer el símbolo original sin espacios alrededor (aunque según ha sido programado añade un píxel por cada lado para evitar problemas al extraer los descriptores de *Fourier*, ver apartado 4.9.10)

Los parámetros de entrada de éste módulo son:

- **`simbSN:`** imagen en formato binario de la que queremos extraer el rectángulo mínimo.

Los parámetros de salida son:

- **`simbSNBB:`** imagen original recortada según su *bounding box* con un margen de un píxel alrededor.

La técnica *bounding box* trata de obtener el rectángulo mínimo que encierra al objeto, proporcionando las dimensiones de este último y el objeto aislado sin espacios (ver Figura 47). El algoritmo a seguir es sencillo: para cada lado de la imagen nos desplazamos hacia el extremo opuesto realizando un barrido que busque píxeles negros. En el punto en el que se encuentre uno, se almacena dicha posición y se realiza la misma operación para el resto de los lados. Una vez completo, se tendrán 4 píxeles que definirán las coordenadas del rectángulo, con la posibilidad de extraer el símbolo y determinar su tamaño.

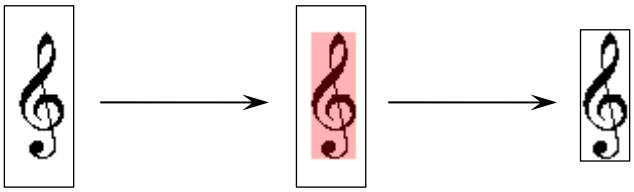


Figura 47 - Ejecución del algoritmo bounding box

La razón por la que se añade un píxel de margen para las coordenadas, aumentando así en un píxel el grosor de la imagen es porque pueden aparecer errores a la hora de extraer los descriptores de *Fourier* en los símbolos cuya morfología rellene el rectángulo mínimo, como ocurre con los silencios de blanca y de redonda.

4.9.3. cargaBD

```
[matrizCaract,leyenda] = cargaBD()
```

Este módulo es el encargado de crear las variables oportunas para describir una base de datos de símbolo. Esta base de datos estará compuesta por un conjunto de símbolos musicales debidamente ordenado en carpetas, bajo unas directrices que se detallarán en apartados posteriores. Las variables creadas serán utilizadas por el módulo reconocedor para clasificar los símbolos.

Su funcionamiento es simple. Una vez definido el directorio de la base de datos, el programa analiza las carpetas existentes y recorre cada una cargando las imágenes y extrayendo sus características, almacenando los resultados en una matriz.

Los datos necesarios, concretamente el directorio de la base de datos, el nombre y la ruta del archivo con los datos generados, se pedirán durante la ejecución del programa mediante interfaz gráfica.

A diferencia de la gran mayoría de los módulos desarrollados, este no se ejecuta en el programa principal por defecto, sino que se presupone que ya ha sido ejecutado con anterioridad para generar las variables de la base de datos (concretamente los vectores de características de cada elemento) que se utilizan en el programa.

Las variables de salida generadas son:

- **`matrizCaract:`** variable formada por un conjunto de vectores de características, donde cada columna representa el vector de una imagen de la base de datos. La última fila de cada columna indica la categoría a la que pertenece la imagen de dicho vector de características.

- **`leyenda:`** variable *cell array* que relaciona los nombres de las categorías con la posición que ocupa dicha categoría en la base de datos.

Además, el programa crea un archivo que define dicha base de datos, el cual se debe especificar al ejecutar el programa principal. Este archivo no es más que la recopilación de las dos variables de salida del programa, ambas necesarias para la ejecución del módulo reconocedor.

En caso de que no se proporcione alguno de los parámetros solicitados por interfaz gráfica, se mostrarán mensajes de errores terminando con la ejecución del módulo.

Durante la ejecución, se ha implementado la posibilidad de que por línea de comandos se informe al usuario del progreso de carga de la base de datos, mediante porcentajes. De ésta forma se podrá comprobar que el programa no se ha detenido ni se ejecuta indefinidamente para bases de datos muy grandes.

4.9.4. cargaEjemplos

```
[simbolosLimpios] = cargaEjemplos()
```

Este programa permite cargar una serie de símbolos musicales pudiendo ejecutar las tareas propias del reconocedor, sin necesidad de cargar una partitura y realizar todo el proceso de segmentación. Para ello, definido un directorio, carga una a una las imágenes y las convierte a binario, almacenándolas en un *cell array* (1 x i) donde cada posición representa una de las imágenes.

No se requieren parámetros de entrada para ejecutar el programa. Únicamente se necesitará definir el directorio donde se encuentren las imágenes que se deseen cargar, tarea que se realiza mediante interfaz gráfica.

El único parámetro de salida, "**`simbolosLimpios`**", es un *cell array* (1 x n) que almacena las imágenes en formato binario encontradas en la carpeta, donde "n" es el número de imágenes. Para facilitar el uso y la interacción con el resto de los módulos del programa se diseña la variable de salida como la salida del módulo "*extraeNotas2*".

Parece lógico pensar que en muy pocas ocasiones se van a disponer de notas musicales aisladas. Las razones que explican el porqué del diseño de este módulo son variadas. En primer lugar, puede utilizarse para evaluar la eficacia del reconocedor, dada una nueva base de datos. Se podría cargar una serie de símbolos de ejemplo y ver qué símbolos han sido reconocidos correctamente. Por otra parte, se puede utilizar para reconocer partituras escaneadas sin líneas del pentagrama. En este caso será necesario un preprocesado previo para segmentar cada símbolo y almacenarlo por separado. Para dicha tarea se puede recurrir al módulo "*segmentanotas*", almacenado en la carpeta "*segmentaBaseDatos*". Otra razón de peso, de cara al futuro, es el reconocimiento de símbolos aislados generados mediante una tableta digitalizadora. Se puede crear un programa que permita escribir símbolos mediante un puntero o un lápiz óptico y que almacene dichos símbolos en imágenes para pasárselo al módulo descrito, ejecutando el resto de los módulos posteriormente para crear el pentagrama o partitura dibujado por el usuario.

4.9.5. defineEscalaFa

```
[posicionesLineasFinal] = defineEscalaFa(posicionesLineas)
```

Este módulo se encarga de asignar un tono determinado a cada una de las posiciones de los tonos definidos en el módulo "*defineEscalaVacia*", según la clave de Fa. Ésta clave asigna a la cuarta línea del pentagrama (contando desde la línea inferior) el tono Fa inferior inmediatamente más cercano al Do central. En base a dicha asignación se realizan las demás asignaciones de tono.

El parámetro de entrada, "**`posicionesLineas`**", corresponde con el parámetro de salida del módulo anterior, "*defineEscalaVacia*", siendo este un *cell array* (i x 2) con las posiciones de los tonos vacíos y un identificador.

El parámetro de salida, "**`posicionesLineasFinal`**", es el resultado de sustituir el identificador de cada posición por la nota adecuada, obteniendo un *cell array* de las mismas dimensiones que el anterior (i x 2)

Hay que destacar que para designar las notas se utiliza la nomenclatura española, haciendo referencia al Do como al Do central. El resto de notas se escribirán añadiendo signos `"+"` o `"-"`, que expresarán una octava por encima respecto a la escala del Do central o por debajo. Cuantos más signos unamos, más octavas se expresarán.

4.9.6. defineEscalaSol

```
[posicionesLineasFinal] = defineEscalaSol(posicionesLineas)
```

Este módulo realiza la misma función que el módulo "*defineEscalaFa*", con la diferencia de que la asignación de notas se realiza utilizando la clave de Sol, que define en la segunda línea del pentagrama el Sol de la escala central.

Los parámetros de entrada y de salida son los mismos que en el módulo antes mencionado, por lo que no se describirán en este apartado.

4.9.7. defineEscalaVacia

```
[posicionesLineas] = defineEscalaVacia(lineasPentagrama)
```

Esta función, utilizada por el módulo "*detectaTono*", se encarga de definir las posibles posiciones de los tonos de la partitura. Utilizando las posiciones de las cinco líneas del pentagrama se determinan qué tonos podrían caber en la imagen.

Para ello, se asume que la posición del tono comprendido entre dos líneas del pentagrama vendrá definido por la mitad de la distancia entre dos líneas adjuntas. Utilizando esta distancia

se podrá definir los tonos por encima y por debajo de las cinco líneas del pentagrama para los casos en los que las notas se encuentren fuera de dichas líneas.

Para los tonos más agudos la tarea es más fácil puesto que una vez se llegue a la posición superior de la imagen no se seguirán buscando posibles tonos superiores (ya que no existen píxeles con valor menor de 1 y los píxeles de la posición superior cuentan con la coordenada fila de valor uno). Sin embargo, para los inferiores no se puede utilizar esta ventaja. Es por eso que se debe definir un límite de tonos por debajo. La implementación realizada define dicho umbral en 7 repeticiones (siendo cada repetición 2 tonos, en total 14 tonos) De esta forma aseguramos de que ningún símbolo pueda quedar fuera del rango.

El único parámetro de entrada necesario es "**`lineasPentagramas`**", vector que contiene la posición de las cinco líneas del pentagrama con respecto a la imagen original. Este vector se genera automáticamente en el módulo "*adaptaPentagramas*", formando parte del *cell array* generado.

El parámetro de salida, "**`posicionesLineas`**", almacena mediante una estructura *cell aray* (i x 2) las posiciones definidas para cada uno de los tonos detectados y un identificador para cada posición, siendo "*i*" el número de posiciones. Estos aún no están definidos, únicamente almacenan la posición del tono vacío. Para definir el tono se debe ejecutar el módulo "*defineEscalaSol*" o "*defineEscalaFa*" para definir una clave de Sol o de Fa respectivamente.

4.9.8. detectaTono

```
[simbolosEscribir,tiempo,compas,clave] =
etectaTono(simbolosLimpios,simbolosReconocidos,lineasPentagrama,tiempo,compas,…
clave,nombreArchivo)
```

Este módulo es una de las piezas clave del programa puesto que se encarga de extraer los atributos propios de cada símbolo (tonos, tiempos…) y realizar algunas comprobaciones sobre los datos reconocidos en busca de posibles errores de tiempo y símbolos mal reconocidos.

Posiblemente este sea el módulo sobre el cual más tiempo se ha dedicado.

Entre otras variables, se encarga de definir aquella que contenga los datos necesarios para poder representar la partitura correctamente (datos simbólicos). Se podría pensar que ese trabajo debería formar parte de la tarea del reconocedor, pero mientras que el primero se encarga de reconocer símbolos, éste módulo se encarga de comprobar errores y encontrar nuevos elementos resultado de las relaciones entre los símbolos reconocidos. A modo de ejemplo, si el reconocedor detectase dos líneas divisorias consecutivas, el programa no debería interpretarlas como tal, sino como barra de final de partitura. De igual modo si después encontráramos dos puntillos esos cuatro símbolos se corresponderían con el inicio de una barra de repetición.

Además, para facilitar la tarea del programa, se permite introducir manualmente los datos del tiempo, compás y clave al ejecutar la función mediante línea de comandos (independientemente de que se establezcan valores por defecto sin no se incluyen estos datos ni se encuentran en la partitura). Hay que destacar que si se introducen los datos y después el programa encuentra nuevos símbolos que definan esos datos, en el momento que los encuentre, el valor de los datos cambiará por el último encontrado.

Hay que tener en cuenta que, en el caso de las claves, estas quedan definidas mediante las funciones "*defineEscalaVacia* "y "*defineEscalaSol*" o "*defineEscalaFa*", que proporcionan el tono concreto para cada posible posición de los tonos. Estos datos son necesarios para la correcta extracción del tono de las notas.

Para el reconocimiento del tono de las notas se utiliza el módulo "*reconoceTonoNota*", el cual se describirá en el apartado correspondiente.

Los parámetros de entrada:

- **`simbolosLimpios:`** *cell array* (1 x i) que contiene los símbolos limpios, sin líneas del pentagrama de por medio, en formato binario, donde "*i*" corresponde al número de símbolos. Esta variable se puede obtener mediante la ejecución de los módulos "*cargaEjemplos*" o "*extraeNotas2*".

- **`simbolosReconocidos:`** *cell array* (i x 2) que contiene la posición dentro de la base de datos del símbolo reconocido y la clase a la que pertenece. Esta variable se obtiene ejecutando el módulo reconocedor y el número de símbolos que se le pasa como parámetro se corresponde con el valor de "*i*".

- **`lineasPentagrama:`** vector que almacena la posición de las cinco líneas del pentagrama con respecto a la imagen del pentagrama. Dicha variable se obtiene como parte del *cell array* de salida resultante al ejecutar el módulo "*adaptaPentagramas*".

- **`tiempo:`** valor numérico entero que indica el tiempo que se establece en la partitura para contabilizar los tiempos de los elementos y analizar si existen errores. En el caso de *tiempo = 0* se define un tiempo de 4/4 por defecto, equivalente a *tiempo = 4*. Si se encuentra en la partitura otro tiempo, se cambia al que establezca este último, aunque se haya definido anteriormente otro.

- **`compas:`** valor numérico que indica el compás inicial del pentagrama. Únicamente tiene sentido en el caso de existir varios pentagramas en la misma partitura. En el caso de *compas = 0* se define por defecto el compás 1 como primer compás, equivalente a *compas = 1*.

- **`clave:`** cadena de caracteres que establece la clave del pentagrama. Puede tomar el calor de *clave = 'Sol'* o *clave = 'Fa'*. En el caso de *clave = 'vacio'* se define por defecto la clave de Sol. Si en la partitura se encuentra una clave distinta a la definida se define la nueva a partir del símbolo que la represente.

- **`nombreArchivo:`** nombre del informe que se genera con los posibles errores y algunos avisos que se lanzan durante la ejecución.

Los parámetros de salida son:

- **`simbolosEscribir:`** *cell array* (i x 2) que almacena en la primera columna la clase a la cual pertenece cada símbolo definido para escribir en la partitura final y en la segunda algunos atributos relacionados con dicho elemento (tono, partitura, duración anacrusa, tiempos…)

- **`tiempo, compas y clave`**: estas variables son las mismas que se han definido en los parámetros de entrada por lo que no se describen en este apartado. Se utilizan para realimentar el módulo en el próximo pentagrama a evaluar.

Aunque no se define como parámetro de salida, el módulo genera un informe detallado que resume los problemas encontrados, avisos sobre los datos existentes o errores que puedan impedir la representación de la partitura. Además, por línea de comandos se muestra parte de ese informe. Éste queda almacenado en la carpeta “Informes” de la carpeta raíz. En el caso de no existir se creará automáticamente y de haber otro informe ya elaborado, el nuevo reemplazará el antiguo.

4.9.9. extraeCaract

```
[vectorCaract] = extraeCaract(simbSBB,filtrar)
```

Este módulo se encarga de extraer unas determinadas características de la imagen dada que permitan caracterizarla y diferenciarla de otra. Dichas características se denominan descriptores y permiten representar cuantitativamente determinados parámetros de la imagen como su morfología.

Estas características se utilizan principalmente para diferenciar símbolos y compararlos. Dos símbolos con una forma similar tendrán unos descriptores similares, aunque cuenten con otras diferencias. Este concepto es fundamental en el proyecto porque es sobre el que se basa la clasificación de los símbolos. Los símbolos que tengan los descriptores más parecidos al símbolo a evaluar serán los que definan la clase a la que pertenece el símbolo desconocido.

Las características implementadas, definidas anteriormente, se corresponden con:

- **Número de saltos abruptos entre píxeles contiguos**. Se implementa como una función interna dentro del módulo:

  ```
  [ numDiscontX numDiscontY] = numPicos(símbolo)
  ```

  Tomando como parámetro de entrada el símbolo en formato binario y como salida el número de saltos abruptos en la proyección X y en la proyección Y.

- **Centro de masas o centroide**. La función se incluye como una función interna de la forma:

  ```
  [x y] = centroide(símbolo)
  ```

  Donde el parámetro de entrada es el símbolo del que queremos averiguar el centroide y los parámetros de salida con las coordenadas del mismo.

- **Solidez**. Dado que la solidez es un cociente, no existe una función dedicada como tal, sin embargo, para extraer el área y el área convexa sí que se han desarrollado funciones:

  ```
  [Ao] = area2(simbolo)
  [areaConvex] = areaConvexa(simbolo)
  ```

Para determinar la envolvente convexa de la imagen se ha utilizado el algoritmo *QuickHull*, ya implementado en MATLAB mediante la función "**`convhull`**".

- **Descriptores de Fourier.** Para extraer dichos descriptores se utilizará el módulo "*extraeDescriptores*" que será descrito posteriormente. Actualmente el diseño realizado extrae los 30 primeros descriptores de *Fourier*.

Una vez descrito el módulo se detallan los parámetros de entrada:

- **`simbSNN`**: imagen en formato binario que se quiere caracterizar mediante sus descriptores

- **`filtrar`**: indica si se quiere realizar o no el filtrado paso bajo para suavizar el contorno de la imagen. Puede tomar valor "*true*" o "*false*".

Para los parámetros de salida tenemos:

- **`vectorCaract:`** vector que describe cuantitativamente la imagen mediante los descriptores antes mencionados. Dicho vector es de tamaño i x 1, donde "*i*" se corresponde con el número de descriptores de la imagen.

4.9.10. extraeDescriptores

```
[descriptores] = extraeDescriptores(imOriginal)
```

Este módulo se encarga de extraer los descriptores de *Fourier* de una imagen dada según el procedimiento descrito en apartados anteriores.

Para ello, el parámetro de entrada definido es:

- **`imOriginal`**: imagen en formato binario o en escala de grises sobre la cual se deseen extraer los descriptores de *Fourier*.

Se obtendrá a la salida el siguiente parámetro:

- **`descriptores:`** vector (1 x 30) que almacena los descriptores de *Fourier* extraídos.

4.9.11. extraeLineasNotas

```
[simbolosLimpios] = extraeLineasNotas(simbolos,color,imagen,anguloRotar)
```

Este módulo es el encargado de eliminar las líneas del pentagrama de los símbolos segmentados.

Los parámetros de entrada del módulo son:

- **`símbolos:`** *cell array* (5 x i) que contiene cada uno de los símbolos extraídos en la etapa anterior, "*extraeNotas*".

- **`color:`** variable que indica si realizamos el algoritmo de las líneas de grosor 1 píxel o el del pentagrama azul. Puede tomar el valor '*normal'* o '*azul'*, según el método deseado. Para el correcto funcionamiento es imprescindible que a la imagen original se le puedan aplicar los criterios definidos para la identificación de las líneas del pentagrama.

- **`imagen:`** imagen RGB o en escala de grises original que contiene la partitura o el pentagrama

- **`anguloRotar:`** valor numérico que indica la inclinación, en grados, de la partitura original. Necesario para corregir la inclinación de la imagen original.

Para los parámetros de salida se tiene:

- **`simbolosLimpios:`** *cell array* (1 x i) que contiene los símbolos de entrada sin líneas del pentagrama.

4.9.12. extraeNotas

```
[simbolos] = extraeNotas(pentagramas)
```

Este módulo se encarga de extraer las notas de un pentagrama, mediante la utilización de un umbral que separa los símbolos de forma análoga a si se realizaran cortes verticales a un pentagrama en papel.

Los parámetros necesarios para ejecutar este módulo son:

- **`pentagramas:`** *cell array* (5 x 1) del pentagrama del que se extraerán las notas. Debe contener el pentagrama, las posiciones de las líneas del pentagrama, las proyecciones X e Y, y las posiciones de los cortes con respecto a la imagen original. Esta variable de obtiene mediante la salida de "*adaptaPentagramas*".

Los parámetros de salida obtenidos son:

- **`simbolos:`** *cell array* (5 x i) que almacena los símbolos extraídos y otros datos de utilidad, donde "*i*" se corresponde con el número de símbolos. Los datos que se almacenan en la estructura son:

    - **Columna 1**: símbolo extraído en formato binario
    - **Columna 2**: proyección X del símbolo extraído
    - **Columna 3**: proyección Y del símbolo extraído
    - **Columna 4**: posición de las líneas del pentagrama
    - **Columna 5**: posición de los cortes sobre el pentagrama de entrada para obtener el símbolo actual

4.9.13. extraeNotas2

```
[simbolosLimpios] = extraeNotas2(simbolosCasiLimpios)
```

Este módulo se encarga de realizar una segunda segmentación sobre los datos extraídos para asegurar que no existen símbolos mezclados con otros, utilizando la técnica de etiquetado por componentes conectados. Dicha técnica, comentada en apartados anteriores, permite extraer símbolos, dentro de una imagen, que no se encuentren solapados.

Los parámetros definidos para la entrada del programa son:

- **`simbolosCasiLimpios:`** *cell array* (1 x i) que contiene los "*i*" símbolos sin líneas del pentagrama. Esta variable se corresponde con la salida del módulo "*extraeLineasNotas*".

Los parámetros de salida son:

- **`simbolosLimpios:`** *cell array* (1 x i) que contiene los símbolos extraídos después de realizar el análisis por componentes conectados, donde "*i*" se corresponde con el número de símbolos encontrados.

4.9.14. extraePentagramas

```
[pentagramas,vectorCortes] = extraePentagramas(imBinFinal)
```

Este módulo se encarga de extraer los elementos de una partitura (sean o no pentagramas), almacenándolos en variables distintas.

Para determinar si realmente el objeto se trata o no de un pentagrama se debe recurrir al módulo "*adaptaPentagramas*".

El parámetro de entrada se corresponde con la variable "**`imBinFinal`**" que representa la partitura en formato binario sin inclinación. Esta variable se puede obtener tras ejecutar el módulo "*procesaImagen*". Los parámetros de salida son:

- **`pentagramas:`** *cell array* (1 x i) con los pentagramas extraídos, siendo "*i*" el número de elementos.

- **`vectorCortes:`** vector con la posición de los cortes horizontales que habría que hacer con respecto a la imagen original para obtener los pentagramas extraídos. En caso de haber varios pentagramas los cortes se almacenan por pares.

4.9.15. extraePentagramasBinarios

```
[imBin3] = extraePentagramasBinarios(imCorregidaGray,cortes,umbral)
```

Este módulo sirve para realizar la misma función que el módulo "*extraePentagramas*" con la salvedad de que no se realiza ningún tipo de procesado sobre la imagen, únicamente se utilizan las posiciones de los cortes definidas en el módulo anterior para realizar el nuevo corte.

La principal función de éste módulo es la de extraer el pentagrama con menos ruido en comparación con el módulo anterior, utilizando la imagen preprocesada de forma más suave.

Los parámetros de entrada del módulo son:

- **`imCorregidaGray`**: partitura o pentagrama en escala de grises, obtenido como salida del módulo "*procesaImagen2*".
- **`cortes:`** posiciones de los cortes del pentagrama con respecto a la partitura original.
- **`umbral:`** umbral utilizado anteriormente para convertir la imagen a formato binario.

Los parámetros de salida:

- **`imBin3:`** pentagrama extraído con menor nivel de ruido.

4.9.16. imagenABinario

```
[imBin,umbral] = imagenABinario(im,umbral)
```

Esta función se encarga de convertir la imagen original RGB o en escala de grises a formato binario. Para ello, se debe establecer un umbral mediante el cual los valores de la imagen superiores al umbral se conviertan en blanco y los inferiores en negro.

Este umbral se define de 0 a 255 y puede ajustarse manualmente o automáticamente. Para ajustarlo de forma automática se utiliza el método de Otsu, descrito en apartados anteriores, que analiza la imagen y define un umbral óptimo conforme a ella. Para ello, se ha utilizado la función "*graythresh*" de MATLAB que devuelve el umbral óptimo.

Los parámetros de entrada del módulo son:

- **`im`**: imagen a convertir, en formato RGB o escala de grises
- **`umbral:`** umbral para realizar la conversión. Puede tomar valores enteros de 0 a 255. En el caso de tomar el valor *umbral* = *-1*, se aplicará el umbral de *Otsu*.

Los parámetros de salida son:

- **`imBin:`** imagen convertida a formato binario
- **`umbral:`** umbral utilizado en el proceso

4.9.17. insertarFila

```
[cellArray2] = insertarFila(cellArray,pos,fila)
```

Esta función sirve para insertar una fila de datos de tipo *cell array* de tamaño (1 x 2) en otro *cell array* dado de tamaño (i x 2). Su funcionamiento es muy simple, dada la fila a insertar, la posición donde se quiere insertar y la estructura de datos original, se crea una nueva que copia los datos de la original hasta la posición indicada, después copia la fila a insertar y por último agrega el resto.

Los parámetros de entrada que utiliza son los siguientes:

- **`cellArray:`** *cell array* de tamaño (i x 2) sobre el cual queremos insertar la fila
- **`pos:`** variable numérica de valor entero comprendido entre 1 e "*i*", que indica la posición donde queremos insertar la fila *cell array*.
- **`fila:`** *cell array* de tamaño (1 x 2) que queremos insertar

La salida obtenida "**`cellArray2`**" será un *cell array* de tamaño (i+1 x 2) idéntico al original con la fila agregada en la posición definida.

4.9.18. negativo

```
[imagenNeg] = negativo(im)
```

Esta función se encarga de realizar el negativo de cualquier imagen. El negativo de una imagen en escala de grises o en formato binario se define como una transformación lineal de la forma:

$$v(x,y) = u_{\max} - u(x,y)$$

Siendo "u(x,y)" el valor de la imagen en un punto determinado, "u_{max}" el valor máximo de la imagen y "v(x,y)" el resultado del negativo en dicho punto.

Para el caso de trabajar con imágenes RGB se deben realizar estas operaciones de forma independiente en cada componente de color.

El parámetro de entrada utilizado es "**`im`**" que representa una imagen en cualquier formato de color. La salida se corresponde con "**`imagenNeg`**", el negativo de la imagen en el mismo formato de color que la imagen de entrada.

4.9.19. procesaImagen

```
[imCorregidaGray,anguloRotar,umbral] = procesaImagen(imagen)
```

Este módulo se encarga de realizar un preprocesado de la imagen para adaptarla de manera adecuada para su análisis posterior en otros módulos.

El único parámetro de entrada al módulo es "**imagen**", que representa la imagen de la partitura o pentagrama sobre el cual queremos realizar el preprocesado, en formato RGB o escala de grises. Los parámetros de la salida de detallan a continuación:

- **imCorregidaGray:** imagen preprocesada en formato binario.
- **anguloRotar:** ángulo de inclinación de la partitura, en grados.
- **umbral:** umbral establecido para realizar la conversión a formato binario.

4.9.20. procesaImagen2

```
[imCorregidaGray] = procesaImagen2(imagenEq,anguloRotar)
```

Este módulo se encarga de preprocesar la imagen original de una forma "suave", evitando el preprocesado realizado por el módulo "*procesaImagen*" que introduce demasiado ruido.

Los parámetros de entrada del módulo son "**imagenEq**" siendo ésta una imagen en formato RGB o escala de grises y "**anguloRotar**", el valor de la inclinación de la imagen en grados. El único parámetro de salida es "**imCorregidaGray**" que constituye la imagen preprocesada de forma "suave" en escala de grises.

4.9.21. programaPFC

```
programaPFC(extraccionLineas,vecinos)
```

Este módulo se corresponde con el módulo de arranque, encargado de ejecutar todo el sistema de principio a fin, interconectando los módulos de forma adecuada y proporcionando información para conocer en todo momento qué acción se esta realizando.

Los parámetros que necesita para iniciar su ejecución son el tipo de algoritmo que se va a utilizar para eliminar las líneas del pentagrama de los símbolos y el número de vecinos a evaluar a la hora de realizar la clasificación de cada símbolo. Los valores que pueden tomar son:

- **extraccionLineas**: valor '*normal'* para la eliminación de las líneas de grosor de un píxel o valor '*azul'* para los pentagramas que utilicen pentagramas azules.
- **vecinos**: cualquier número entero positivo excepto el cero

Como información extra a tener en cuenta, el valor del algoritmo para eliminar las líneas debe ir en concordancia con la partitura que se cargue en el programa, ya que de no ser así el programa no realizará bien sus funciones.

Los datos correspondientes a la localización de la imagen, la base de datos a utilizar o dónde deseamos guardar el fichero codificado con la partitura son parámetros que se obtendrán durante la ejecución del programa mediante interfaz gráfica.

Este módulo no genera ninguna variable de salida, sin embargo si que genera un fichero de texto codificado mediante el lenguaje utilizado por *Lilypond*. La ruta de dicho fichero, así como su nombre, son datos que se requerirán mediante interfaz gráfica durante la ejecución del programa.

4.9.22. proyecciones

```
[vectorX, vectorY] = proyecciones(imBinFinal)
```

Este módulo realiza las proyecciones X e Y de una imagen en formato binario. Dichas proyecciones ya fueron definidas en apartados anteriores.

El parámetro de entrada es "**imBinFinal**" que representa la imagen de entrada en formato binario. Las salidas son los vectores de tamaño (1 x columnas) y (1 x filas) para las proyecciones X e Y respectivamente.

4.9.23. reconocedor

```
[simbolosReconocidos] =
reconocedor(simbolos,matrizCaract,leyenda,vecinos,nombreArchivo)
```

El módulo reconocedor es el encargado de clasificar los símbolos según una base de datos determinada, mediante la utilización de un algoritmo k-NN.

Los vectores utilizados para comparar serán los vectores de características que se encargan de describir un símbolo mediante descriptores. En apartados anteriores ya se explicó este concepto.

Los parámetros de entrada necesarios para ejecutar el programa son:

- **símbolos:** *cell array* (1 x i) que contiene los símbolos limpios a reconocer, siendo "*i*" el número de símbolos. Esta variable se obtiene mediante la ejecución del módulo "*extraeNotas2*".
- **matrizCaract:** variable formada por un conjunto de vectores de características, donde cada columna representa el vector de una imagen de la base de datos. La última fila de cada columna indica el grupo al que pertenece la imagen de dicho vector de características.
- **leyenda:** variable *cell array* que relaciona los nombres de las categorías con la posición que ocupa dicha categoría en la base de datos.
- **vecinos:** número de elementos que debemos tener en cuenta para clasificar el símbolo desconocido en función de sus distancias mínimas.
- **nombreArchivo:** variable de cadena de caracteres que especifica el nombre del archivo que proporcionaremos al informe generado con los datos reconocidos.

El único parámetro de salida "**simbolosReconocidos**" es un *cell array* de tamaño (i x 2) donde "*i*" constituye el número de símbolos reconocidos, en el que la primera columna de cada elemento contiene la posición de la clase del símbolo clasificado dentro de la base de datos y la segunda contiene la clase a la que pertenece.

Además, aunque no se genere como variable de salida, se genera un informe dentro de la carpeta "*Informes*" en el directorio raíz en el que se detallan los datos reconocidos para cada símbolo, los vecinos seleccionados con sus distancias, el tamaño de la base de datos, etc. En el caso de no existir dicha carpeta, se creará.

4.9.24. reconoceTonoNota

```
[tono] = reconoceTonoNota(simbolo,posicionesLineasFinal,tipo)
```

Este módulo se encarga de determinar el tono de una nota cualquiera.

Los parámetros de entrada definidos para el módulo son:

- **`símbolo:`** nota musical (en formato binario) de la que se quiere extraer el tono. Es importante que para el buen funcionamiento del módulo no se recorte la imagen original del símbolo extraído del pentagrama, puesto que para detectar el tono se compara la posición de la cabeza de la nota con las posiciones definidas para los tonos. Estas notas se obtienen en el módulo "*extraeNotas2*".

- **`posicionesLineasFinal:`** *cell array* de tamaño (i x 2) que almacena las posiciones de los distintos tonos del pentagrama y los tonos asignados por la clave definida.

- **`tipo:`** variable numérica que representa la posición del grupo al que pertenece la nota dentro de la base de datos.

El único parámetro de salida obtenido es el tono de la nota, en formato de cadena de caracteres según ya se definió anteriormente al hablar de las claves del pentagrama.

4.9.25. representaPartitura

```
representaPartitura(simbolosEscribirConjunto,titulo,autor)
```

Este módulo se encarga de codificar los datos musicales reconocidos en una partitura mediante el lenguaje de representación musical *Lilypond*. Dicho lenguaje se explicará de forma breve en los anexos.

El módulo incluye una serie de subfunciones que se describen a continuación:

- **`[] = insertaTexto(texto)`**
  Función para insertar un texto en el fichero de salida definido en el programa. La variable texto debe ser una cadena de caracteres.

- **`[] = insertaSalto()`**
  Función para insertar un salto de línea en el fichero de salida definido en el programa.

- **`[] = insertaTabulacion()`**
  Función para insertar una tabulación en el fichero de salida definido en el programa.

- **`[partituraF,activar2] = introduceLigadura(partitura,activar2)`**
  Función para introducir una ligadura a la nota adecuada. Se ejecuta cada vez que se evalúa una nota musical. Parámetros de entrada: "**`partitura`**" (nota musical) y "**`activar2`**" (variable que indica si se debe insertar ligadura mediante el valor "*true*" o "*false*"). Devuelve la partitura con la ligadura introducida y el valor de activación anulado.

- **`[partitura,activar] = introduceAccidente(simbolo,activar)`**
  Función que introduce una alteración a una nota. Se ejecuta cada vez que se evalúa una nota musical. El valor de activación varía de uno a ocho según el tipo de alteración (1 = bemol, 2 = doble bemol, 3 = sostenido, 4 = doble sostenido, 5 = medio sostenido, 6 = medio bemol, 7 = sostenido y medio, 8 = bemol y medio)

- **`[notaTraducida] = dameNota(nota)`**
  Función que devuelve el tono de la nota codificado.

Los parámetros de entrada del programa son:

- **`simbolosEscribirConjunto:`** *cell array* (1 x i) que almacena en cada posición un *cell array* "**`simbolosEscribir`**", resultado del análisis y reconocimiento de cada pentagrama. Esta variable recoge todos estos datos para poder escribir todos los elementos de los pentagramas que conforman la partitura completa. Dicha variable se obtiene a partir del módulo "*detectaTono*".

- **`titulo:`** variable de cadena de caracteres que indica el titulo de la partitura. En caso de *titulo = 0* se omite.

- **`autor:`** variable de cadena de caracteres que indica el autor de la partitura. En caso de *valor = 0* se omite.

No se genera ninguna variable de salida, aunque sí se genera un fichero de extensión *Lilypond* (.ly) que permite imprimir la partitura mediante PDF y PostScript y escucharla mediante MIDI. Los datos necesarios para guardar este archivo se pedirán durante la ejecución del programa.

4.9.26. rotarImagenRadon

`[imCorregidaGray,anguloRotar] = rotarImagenRadon(imBin3)`

Este módulo se encarga de detectar y corregir la inclinación de la partitura mediante la utilización de la transformada *Radon*. Para ello, se utiliza la implementación de MATLAB, mediante la función "*radon*".

El programa implementa la capacidad de generar un video en formato AVI con las proyecciones realizadas en función del ángulo de proyección, útil para detectar errores en casos conflictivos. Por defecto, esta opción esta desactivada aunque se puede activar descomentando el código oportuno. El vídeo se almacena en la carpeta raíz donde se ejecute el módulo.

El único parámetro de entrada del módulo es "**imBin3**" que representa la imagen inclinada en formato binario. Los parámetros de salida son:

- **imCorregidaGray:** imagen en formato binario sin inclinación
- **anguloRotar:** ángulo que indica la inclinación de la partitura o pentagrama, en grados.

4.9.27. segmentaNotas

```
[simbolosLimpios] = segmentaNotas()
```

Este módulo se encarga de extraer los símbolos de ejemplo de una hoja para ayudar en la tarea de la creación de la base de datos.

El funcionamiento interno del programa no se describirá porque es muy similar al descrito en el módulo "*extraeNotas2*". De igual forma realiza un etiquetado por componentes conectadas y desecha posibles impurezas.

Como diferencias a tener en cuenta con el módulo mencionado, destaca la implementación de la carga de la imagen mediante interfaz gráfica, así como la especificación de la ruta para guardar las imágenes. Las imágenes se guardan en formato binario, sin bordes alrededor utilizando el método *bounding box*.

El programa no cuenta con parámetros de entrada; la información necesaria se pide mediante interfaz gráfica. En cuanto a los parámetros de salida, se genera la variable "**simbolosLimpios**", *cell array* de tamaño (1 x n) siendo "*n*" el número de elementos extraídos. Esta variable de salida se corresponde con la obtenida con el módulo "*extraeNotas2*".

Las funciones necesarias para su ejecución, así como el modulo en concreto se encuentran almacenadas en el directorio "*SegmentaBaseDatos*", dentro del directorio raíz.

5. Resultados

5.1. Introducción

La meta de este capitulo es analizar los resultados obtenidos para determinar si se cumplen los objetivos propuestos y evaluar la eficacia de cada una de las etapas que conforman el sistema reconocedor de partituras.

Los objetivos se definieron en apartados anteriores. Para cumplirlos, cada bloque debe realizar correctamente la tarea para la que ha sido creado de forma que los bloques posteriores puedan ejecutarse correctamente. Sin embargo, de cara al usuario el principal objetivo es que el programa funcione bien, sin distinción de objetivos entre bloques ni módulos. Este factor es determinante a la hora de que un usuario elija un programa u otro creado para el mismo fin.

En este caso, el principal objetivo será que el programa obtenga la misma partitura que la analizada, en formato simbólico, para posteriormente representarla en formato gráfico y acústico. Como se verá más adelante, este objetivo depende de numerosos factores cuyos resultados se deberán analizar de forma separada. Para ello, se evaluarán dos elementos principalmente: segmentación de la partitura y clasificación de los elementos.

5.2. Conjunto de imágenes de entrenamiento

El sistema reconocedor de partituras necesita de un conjunto de imágenes anteriormente clasificadas para poder realizar su tarea correctamente. Dicho conjunto, organizado previamente por categorías, debe caracterizar la morfología de la clase a la que pertenece cada elemento musical de forma adecuada.

Para ello, los elementos de dicha colección se presuponen de un tamaño normal (ni muy grande ni muy pequeño, con una resolución entre 75 y 200 dpp) y de un nivel de ruido bajo. Este último factor será un importante elemento a tener en cuenta ya que afecta directamente sobre la eficacia del reconocedor al modificar la morfología de la imagen mediante ruido.

Los ejemplos se han extraído de fuentes musicales digitalizadas, para el caso de símbolos no manuscritos, y de ejemplos escaneados o dibujados en el ordenador, para el caso manuscrito.

Los elementos mencionados no serán partituras ni pentagramas, sino símbolos musicales aislados. En la Figura 48 se pueden observar algunos de los símbolos utilizados para describir la clase dedicada a la corchea, en el caso no manuscrito.

De igual forma, en el caso manuscrito se deben obtener las imágenes de entrenamiento. Sin embargo, en este caso se debe puntualizar que conviene que se diferencien conjuntos de imágenes de entrenamiento para la escritura de cada usuario. El principal inconveniente de la escritura manuscrita es que cada autor proporciona al símbolo un carácter propio, diferenciándolo claramente de otros símbolos de la misma categoría realizados por otros autores. De ahí que lo más conveniente para el buen funcionamiento del sistema sea la creación de determinadas colecciones de símbolos para cada usuario que se correspondan con los símbolos de las partituras que haya creado. En la Figura 49 se pueden comprobar las variaciones de algunos de los símbolos de la clase dedicada a la corchea para dos usuarios.

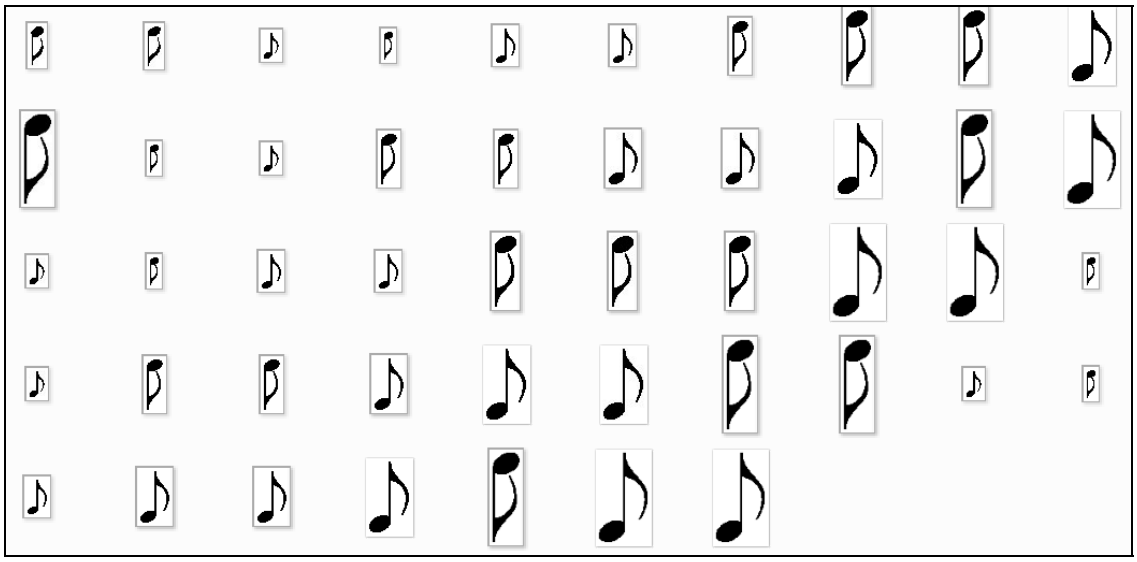


Figura 48 - Colección de imágenes no manuscritas para la corchea

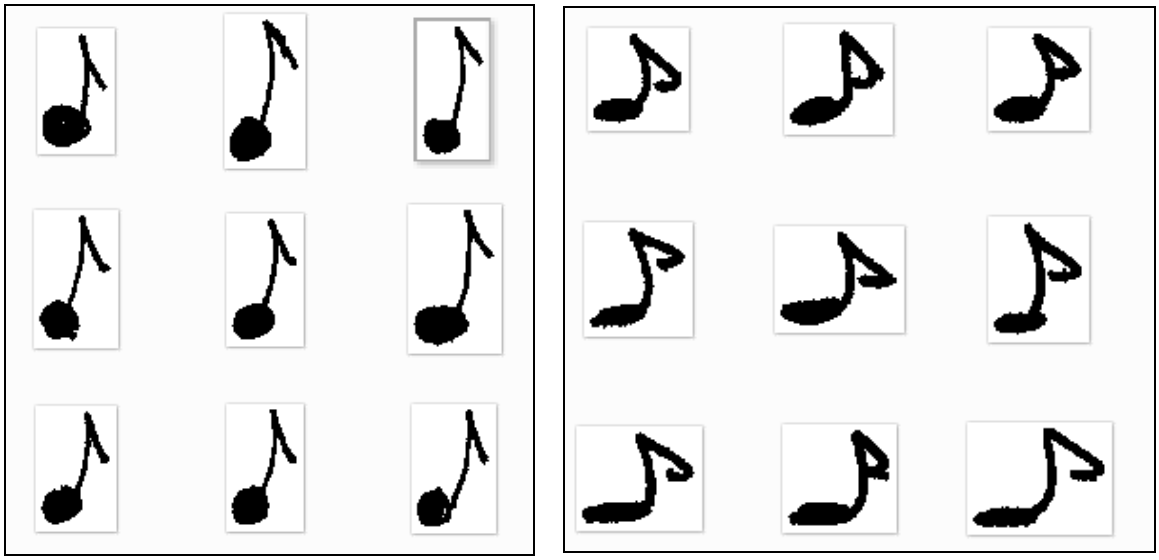


Figura 49 - Colección de símbolos manuscritos de corchea para distintos usuarios

Todos estos datos se almacenarán en una base de datos de símbolos que albergará los descriptores de características de cada elemento. El contenido de dicho vector ya se detalló en apartados anteriores y permite caracterizar la morfología de los símbolos.

En la realización del proyecto se han definido 7 bases de datos que albergan determinados conjuntos de clases de símbolos:

- **`baseDatos:`** dicha colección de imágenes de símbolos musicales esta formada por un amplio conjunto de símbolos no manuscritos con 41 categorías. Se trata de la base de datos más extensa de las 7 realizadas y cuenta con 1397 símbolos. Los datos de dicha base de datos (los vectores de características de los elementos) se almacenan en el fichero "*baseDatos_noManuscritos.mat*".

- **baseDatos_Reducida:** versión reducida de la colección anterior. Cuenta con 9 categorías y 353 símbolos. Los datos se almacenan en el fichero "*baseDatos_noManuscritos_REDUCIDA.mat*".

- **baseDatos_Manuscritas_Raquel:** colección reducida que alberga un conjunto de símbolos manuscritos de un usuario concreto. Cuenta con 7 categorías y 76 símbolos. Los datos de la base de datos se almacenan en el fichero "*baseDatosManuscritos_Raquel.mat*".

- **baseDatos_Manuscritas_Rocio:** colección manuscrita de símbolos musicales que alberga 12 categorías y 410 símbolos. Los datos de dicha base de encuentran en el fichero "*baseDatosManuscritos_Rocio.mat*". Hay que destacar que esta base de datos se ha generado a partir de símbolos escaneados, no dibujados directamente en el ordenador.

- **baseDatos_Manuscritas_David:** colección manuscrita de símbolos musicales que alberga 9 categorías y 696 símbolos. Los datos de dicha base de encuentran en el fichero "*baseDatosManuscritos_David.mat*".

- **baseDatos_Manuscritas_AMPLIADA_David:** amplia colección de símbolos musicales manuscritos que cuenta con 31 categorías y 922 símbolos. Los datos de la base se encuentran en el fichero "*baseDatosManuscritas_AMPLIADA_David.mat*". Hay que destacar que esta base de datos se ha generado a partir de símbolos escaneados, no dibujados directamente en el ordenador.

- **baseDatos_Manuscritas_REDUCIDA_David:** colección manuscrita de símbolos musicales que cuenta con 8 categorías y 235 símbolos. Los datos de dicha base de encuentran en el fichero "*baseDatosManuscritas_REDUCIDA_David.mat*". Esta base de datos se corresponde con una versión simplificada de la base de datos anterior.

5.3. Conjunto de imágenes de test

El conjunto de imágenes de *test* sirve para evaluar el sistema y poder establecer unos porcentajes de acierto y error. Dada la naturaleza de los símbolos utilizados en este proyecto se utilizarán imágenes de *test* para los casos de partituras manuscritas y otras para las no manuscritas.

En ambos tipos de podrán diferenciar dos conjuntos de imágenes. Por un lado se dispone de notas aisladas para comprobar el funcionamiento del bloque clasificador y por otro se dispone de partituras o pentagramas para comprobar el funcionamiento del sistema reconocedor íntegro.

Par el caso de los elementos no manuscritos, se dispone de:

1. **Colección de símbolos agrupados por categorías**, de 20 muestras cada una. En algunas de las 41 categorías se han utilizado únicamente 10 muestras debido a la dificultad para encontrar símbolos de dicho grupo. Dichos símbolos se han extraído de fuentes distintas a las utilizadas para crear el conjunto de entrenamiento.

2. **Colección de 10 partituras**, de las cuales, algunas sólo cuentan con un único pentagrama. Las partituras se han elaborado mediante un programa de edición musical con el fin de obtener grosores de línea de pentagrama de un píxel.

Para el caso de los elementos manuscritos, se dispone de varias clases de prueba:

1. **Colección de notas aisladas**. Se disponen de cuatro colecciones de notas aisladas, pertenecientes a las notas de cuatro autores distintos. De forma detallada:

    a. **`Notas_Aisladas_Raquel:`** conjunto de 32 símbolos de *test* obtenido mediante el dibujo de los símbolos en un *tablet PC.*

    b. **`Notas_Aisladas_Rocio:`** conjunto de 52 símbolos de *test*, obtenido mediante el escaneo de la hora original donde se habían dibujado.

    c. **`Notas_Aisladas_David:`** conjunto de 177 símbolos de *test* obtenido mediante el dibujo de los símbolos en un *tablet PC.*

    d. **`Notas_Aisladas2_David:`** conjunto de 606 símbolos de *test*. Se trata del conjunto de *test* más amplio. Será el que se utilice para medir las tasas de símbolos correctamente clasificados ya que cuenta con una colección organizada por carpetas de 31 categorías con 20 elementos en cada una. Dichos símbolos se han dibujado en papel y posteriormente se han digitalizado mediante un escáner. Algunos de los símbolos realizados, concretamente 14 símbolos, han sido desechados porque se habían dibujado mal o porque aparecían quebrados.

2. **Colección de pentagramas**. Se dispone de dos colecciones de pentagramas:

    a. **`Pentagramas_Raquel:`** colección de 9 pentagramas manuscritos cuyas líneas son azules. Estos se han dibujado en un *tablet PC*.

    b. **`Pentagramas_David:`** colección de 10 pentagramas manuscritos de características similares a la colección anterior, con grosor más grande de línea de pentagrama

5.4. Evaluación de los resultados

A continuación se analizarán las distintas etapas del sistema para evaluar su funcionamiento. Para ello, se definirán una serie de casos de prueba y se utilizarán las imágenes de *test* descritas en el apartado anterior.

5.4.1. Evaluación de la corrección de inclinación

El bloque de preprocesado de imagen, además de adaptar la partitura convenientemente para su conversión a formato binario, debe detectar la inclinación de la imagen y corregirla. Como ya se mencionó en apartados anteriores, se han desarrollado hasta cuatro algoritmos para tal efecto, escogiendo finalmente la transformada Radón.

A continuación se muestran algunos de los resultados obtenidos con partituras distintas a las partituras de *test*:

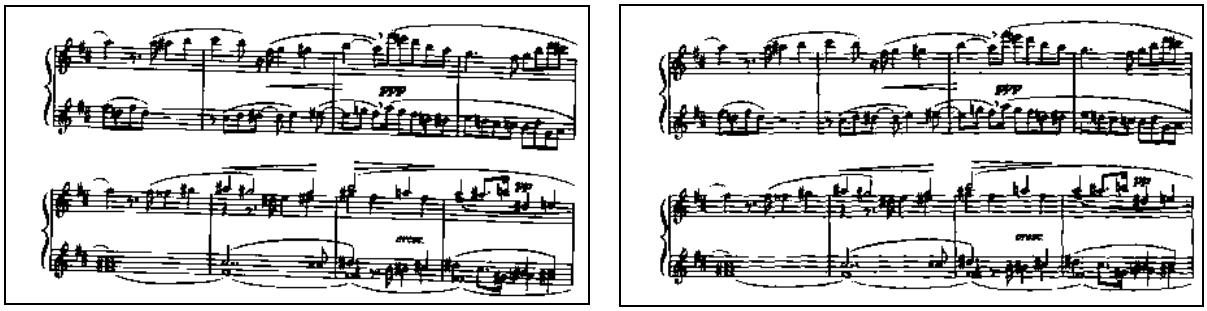


Figura 50 - Izquierda: partitura original. Derecha: partitura corregida (1.4º) (extraído de "Aída" de Verdi)

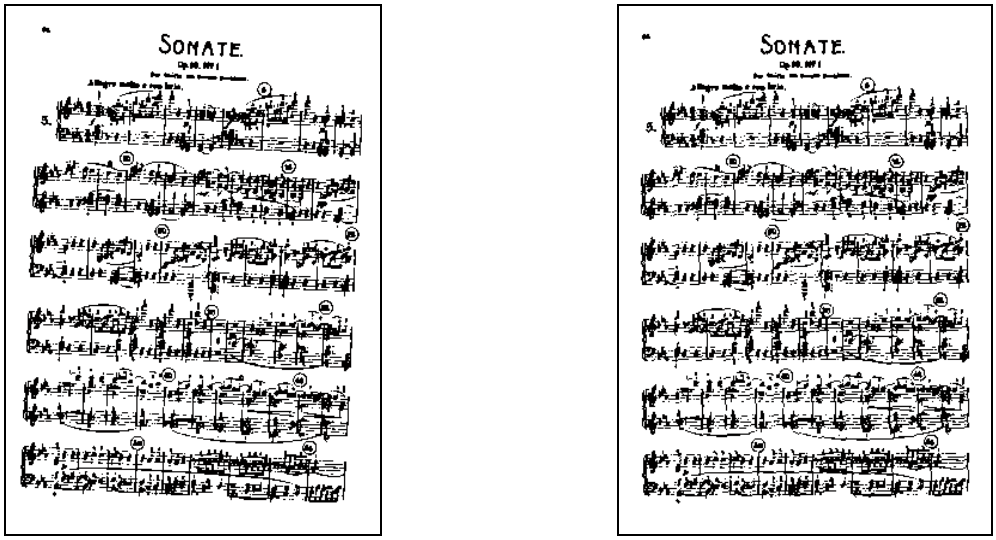


Figura 51 - Izquierda: partitura original. Derecha: partitura corregida (2.9º) (extraído de la sonata No. 5 en Do menor, Op. 10 No. 1 de Beethoven)

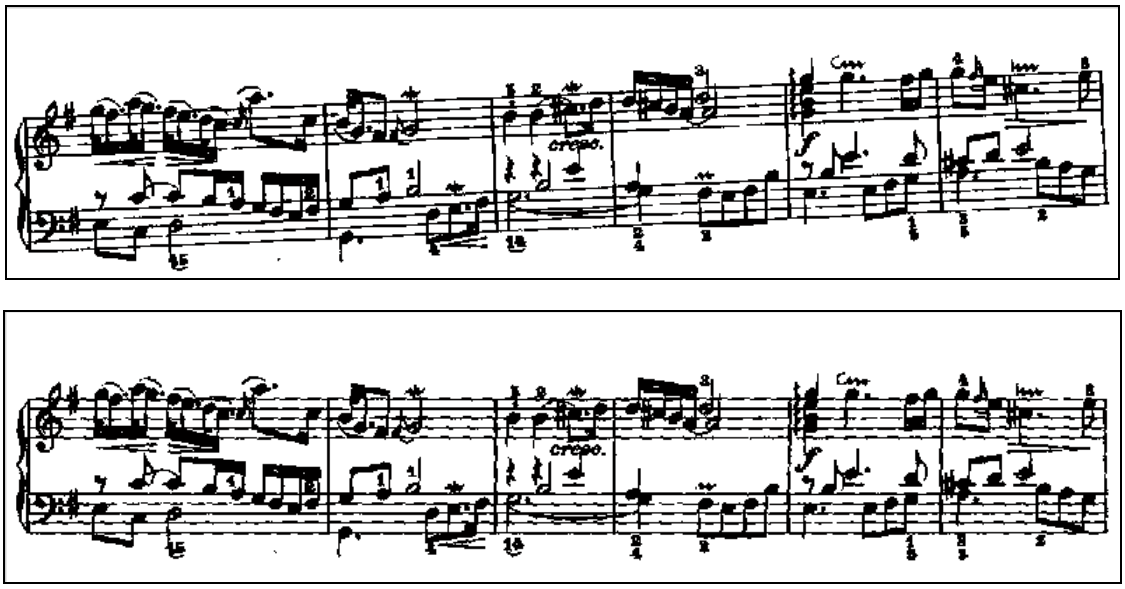


Figura 52 - Izquierda: pentagrama original. Derecha: pentagrama corregido (-2.5º) (extraído de "Goldberg Variations, BWV 988" de Bach)

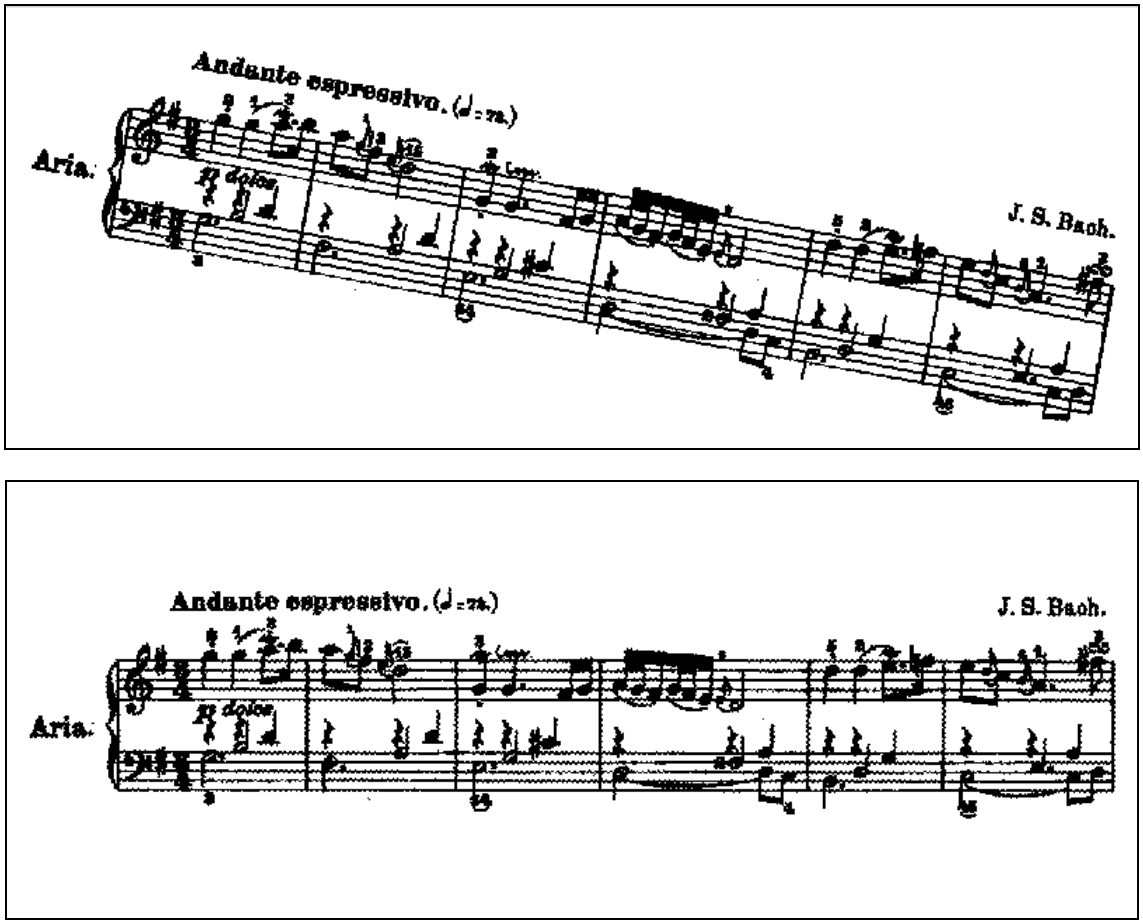


Figura 53 - Izquierda: pentagrama original. Derecha: pentagrama corregido (10.1º) (extraído de "Goldberg Variations, BWV 988" de Bach)

Un punto que no se ha tenido en cuenta es la distorsión de la partitura, en el sentido de que los pentagramas no sean perfectamente rectos (considerar el caso de una partitura escaneada de un libro, en la cual es muy probable que en el pliegue del libro la partitura aparezca encorvada). Si esto ocurre, es probable que la corrección de la inclinación se realice mal y, de producirse bien, es posible que el bloque reconocedor no reconozca adecuadamente el tono de las notas en dichas posiciones. Se supone que la partitura esta en perfectas condiciones y que no existen elementos nocivos en la imagen.

Ejecutando el proceso sobre el conjunto de las imágenes de *test* para el caso de símbolos manuscritos y no manuscrito se han obtenido los siguientes resultados.

Imagen de *test*	Rotación real	Rotación detectada
Manuscritos (David)		
1	Sin inclinación	-0.2º
2	Sin inclinación	Sin inclinación
3	Sin inclinación	Sin inclinación
4	Sin inclinación	-0.2º
5	1.6º	1.6º
6	Sin inclinación	Sin inclinación
7	Sin inclinación	Sin inclinación
8	Sin inclinación	Sin inclinación
9	Sin inclinación	0.1º
10	Sin inclinación	-0.2º
Manuscritos (Raquel)		
1	Sin inclinación	Sin inclinación
2	Sin inclinación	Sin inclinación
3	Sin inclinación	Sin inclinación
4	Sin inclinación	Sin inclinación
5	Sin inclinación	-0.2º
6	Sin inclinación	Sin inclinación
7	Sin inclinación	Sin inclinación
8	Sin inclinación	Sin inclinación
9	Sin inclinación	Sin inclinación
No manuscritos		
1	Sin inclinación	Sin inclinación
2	Sin inclinación	Sin inclinación
3	Sin inclinación	Sin inclinación
4	Sin inclinación	Sin inclinación
5	Sin inclinación	Sin inclinación
6	Sin inclinación	Sin inclinación
7	Sin inclinación	Sin inclinación
8	Sin inclinación	Sin inclinación
9	Sin inclinación	Sin inclinación
10	Sin inclinación	Sin inclinación

Figura 54 - Resultados del algoritmo de detección de inclinación

Como se puede ver, en la mayor parte de los casos el sistema detecta correctamente la inclinación, y en los casos que no (5 de 29, un 17 %), el fallo es mínimo por lo que no supone un obstáculo para el funcionamiento de los bloques posteriores. La tarea del módulo de rotación cumple con las expectativas de forma satisfactoria aunque fuera una de las etapas más difíciles de resolver.

El tiempo requerido para ejecutar este proceso es corto. En general, es de unos segundos, aunque este factor depende en gran parte del tamaño de la imagen. Y es que, aunque el tamaño para realizar la transformada sea siempre el mismo (recordar que se utiliza una porción de la imagen original de 200 x 200), el módulo rota la imagen y representa las versiones antes y después de la corrección mediante las funciones de MATLAB "*imrotate*" e "*imshow*", respectivamente, las cuales requieren más tiempo de cómputo a mayor tamaño de imagen. En la siguiente tabla se pueden ver alguno de los tiempos de cómputo del módulo "*rotarImagenRadon*" en el equipo utilizado para desarrollar el proyecto (ver apartado "Gestión del proyecto"):

Figura	Tamaño de la imagen	Tiempo con representación
Figura 50	480 x 940	4.990410
Figura 51	1825 x 1342	71.107108
Figura 52	294 x 1200	2.674209
Figura 53	474 x 1218	7.193295

Figura 55 - Tabla con los tiempos de cómputo de "rotarImagenRadon"

Si en lugar de este, se atiende a los tiempos empleados por la primera etapa del bloque de preprocesado, correspondiente al módulo "*procesaImagen*", que se encarga de ejecutar el módulo descrito anteriormente y otros métodos se obtendrá:

Figura	Tamaño de la imagen	Tiempo con representación
Figura 50	480 x 940	5.038791
Figura 51	1825 x 1342	71.118979
Figura 52	294 x 1200	2.684884
Figura 53	474 x 1218	7.207632

Figura 56 - Tabla con los tiempos de cómputo de "procesaImagen"

Observando ambas tablas se puede comprobar que la mayor parte del tiempo de cómputo de "*procesaImagen*" se dedica a la detección de la inclinación. En resumen, se puede decir que el tiempo empleado para cualquier imagen es de aproximadamente 15 µs / píxel.

5.4.2. Evaluación de la segmentación

El buen funcionamiento de este bloque va determinado por las características de la imagen original de la partitura y el funcionamiento del bloque de preprocesado. A continuación se detallarán algunos casos de prueba:

5.4.2.1. Extracción de pentagramas

La extracción de pentagramas, como ya se comentó en apartados anteriores, comprende las etapas de extracción de elementos de la partitura y verificación de pentagramas.

Dicho proceso en su conjunto se ha evaluado con las imágenes de *test*, obteniendo buenos resultados. Todos los pentagramas de cada partitura fueron extraídos correctamente. Sin embargo, la mayor parte de las imágenes de prueba únicamente cuentan con un único pentagrama, por lo que, para asegurar el buen funcionamiento del proceso, se han utilizado, de forma complementaria, otras partituras más complejas obteniendo los siguientes resultados:

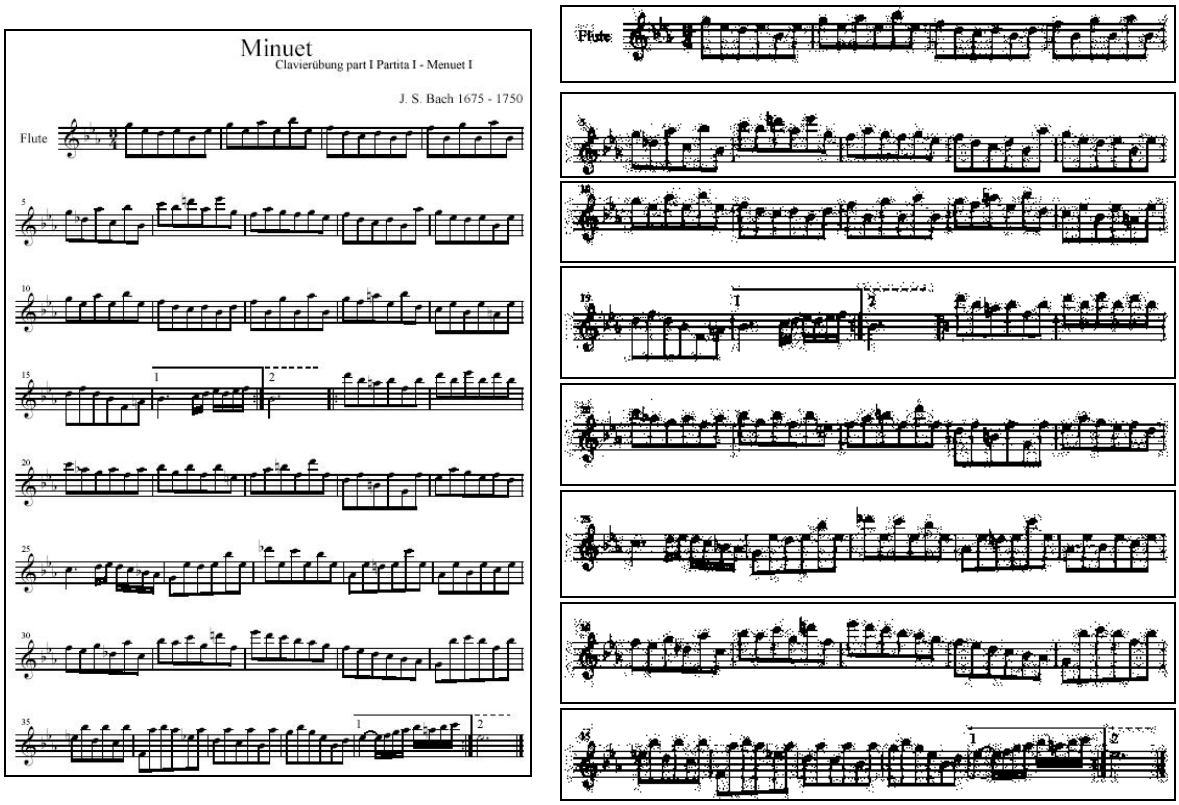


Figura 57 - Ejemplo 1. (Extraído de "Minuet" de J.S. Bach)

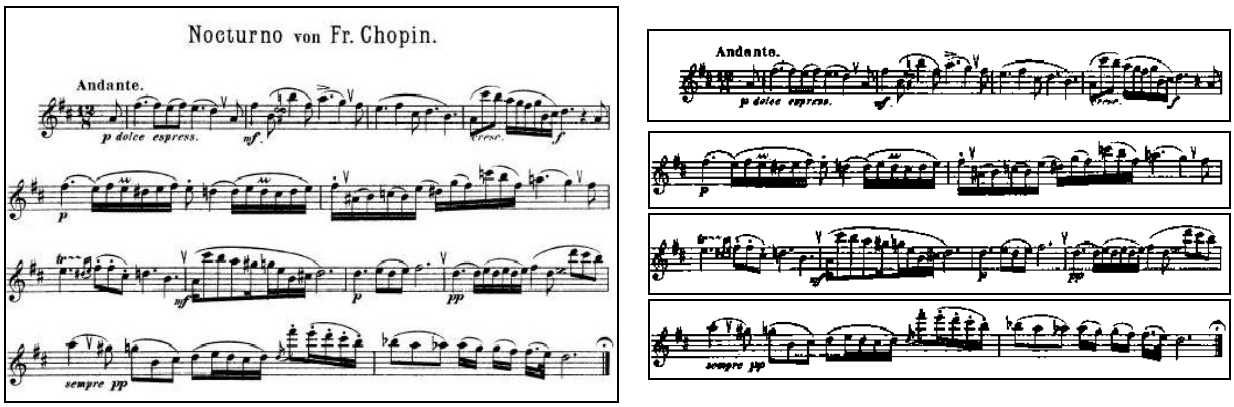


Figura 58 - Ejemplo 2. (Extraído de "Nocturno" de Chopin)

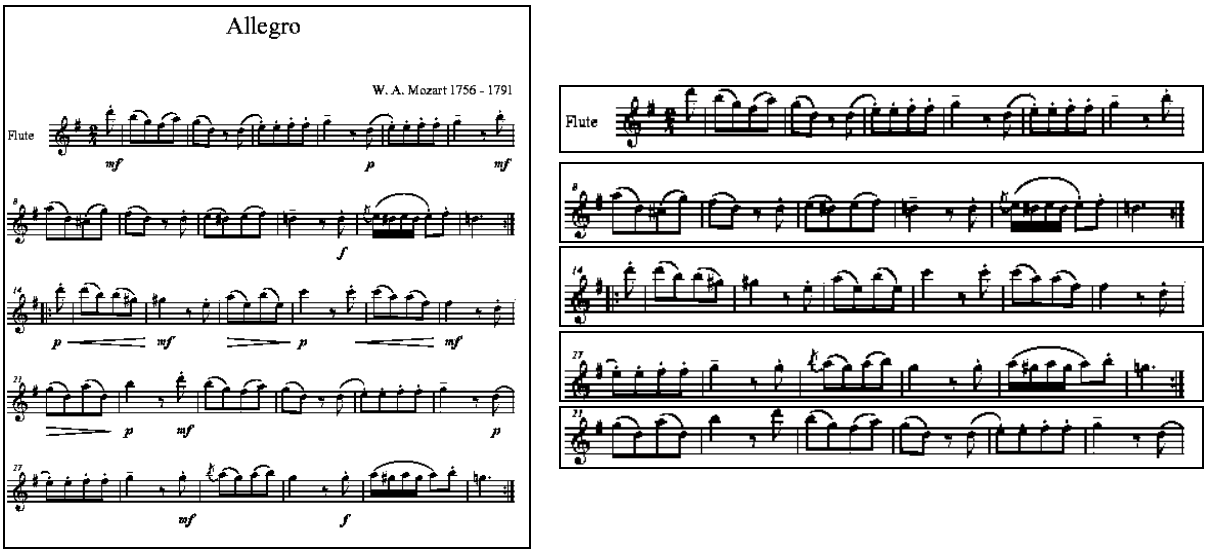


Figura 59 - Ejemplo 3. (Extraído de "Allegro" de Mozart)

Se puede observar que el proceso realiza la tarea correctamente, como ya se indicó al ejecutar las imágenes de prueba. El tiempo de cómputo requerido es pequeño, en torno a un par de segundos.

Puede ocurrir que, en ciertas partituras, al ejecutar la etapa encargada de separar los elementos ("*extraePentagramas*") se separe algún símbolo musical del pentagrama. Esto es poco frecuente, pero puede ocurrir (ver Figura 60). Si el símbolo esta completamente separado de la partitura, el módulo extraerá ambos elementos. No se ha contemplado una solución, suponiendo que la gran mayoría de los pentagramas incluyen dentro de él todos los símbolos musicales.

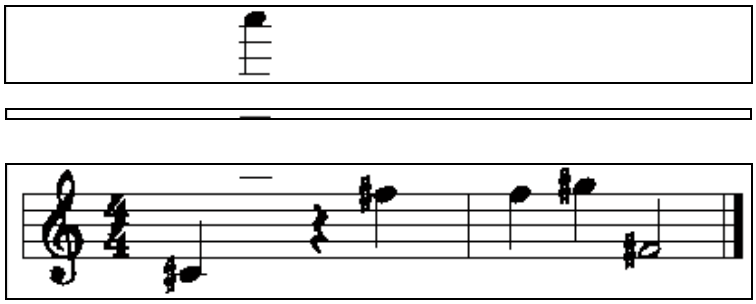


Figura 60 - Caso particular de la extracción de elementos de la partitura

Una vez se han extraído los elementos, deben ser analizados para determinar si realmente son pentagramas o no mediante el módulo "*adaptaPentagramas*". En general, este módulo cumple con su cometido. Sin embargo, pueden ocurrir casos en los que se detecte un elemento como pentagrama no siéndolo. Si se ejecuta el módulo con los resultados obtenidos en la Figura 60 se puede comprobar que se extraen como pentagramas el primer elemento (la nota aislada) y el último (el pentagrama real).

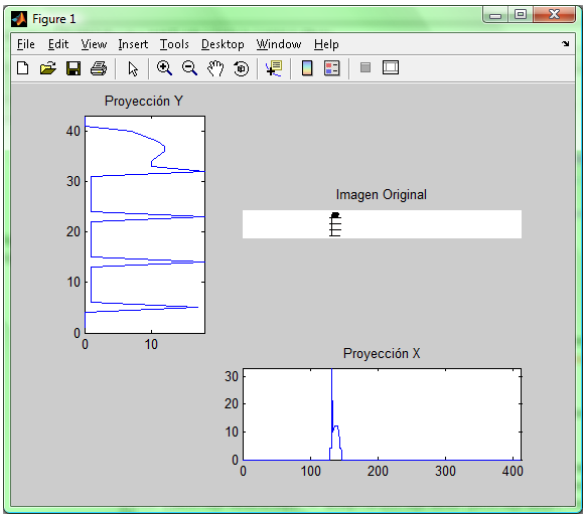


Figura 61 - Proyecciones de la nota aislada

Si se observa la proyección Y de la nota, en la Figura 61, se podrá ver un conjunto de valores en forma de montaña y cuatro picos bien definidos correspondientes a las líneas. La forma que tiene el módulo de detectar la existencia de los cinco picos es mediante al análisis de la proyección, en busca de valores que superen el umbral definido por 2/3 del valor máximo de la proyección. Gracias a esto se pueden detectar pentagramas cuyas líneas estén quebradas o con cierta distorsión, ya que no se aplica un umbral estricto que defina la longitud de la línea como tal. Sin embargo, en este caso, los cuatro picos se corresponden con las líneas adicionales y algunos de los valores en forma de montaña superan el umbral definido, engañando al programa con un quinto pico ficticio.

Se podría solventar este problema mediante la imposición de un umbral mayor, pero se correría el riesgo de no identificar adecuadamente los pentagramas de determinadas partituras cuyas líneas no estén bien definidas.

5.4.2.2. Extracción de notas

El bloque de extracción de notas se encarga de segmentar los símbolos de un pentagrama dado. Para ello, realiza tres procedimientos: primera extracción de elementos, borrado de líneas del pentagrama y segunda extracción de elementos.

1. **Primera extracción de elementos**

   El módulo encargado de la primera extracción de elementos, "*extraeNotas*", segmenta los símbolos del pentagrama realizando cortes verticales en aquellos puntos donde únicamente existen las cinco líneas del pentagrama. Este umbral funciona bastante bien, no obstante, se pueden dar casos en los que no se ejecute correctamente.

   Dicho algoritmo utiliza el grosor de cada línea para determinar el umbral. De partida, se presupone que todas las líneas cuentan con el mismo grosor y éste se determina mediante la moda del vector de los píxeles negros codificados según el método *RLE*. Si se utilizase una partitura como la de la Figura 62 sería probable que se produjese un error al determinar el grosor de las líneas del pentagrama, confundiendo este valor con el grosor de las uniones de las semifusas.

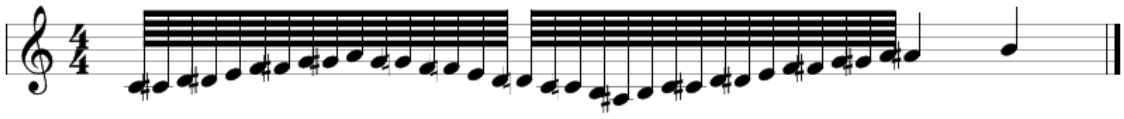


Figura 62 - Pentagrama con multitud de uniones entre semifusas

Para estos casos se puede utilizar un umbral alternativo, definido en apartados anteriores, que no utiliza la información del grosor de las líneas del pentagrama sino datos estadísticos de la proyección sobre el eje X.

El tiempo requerido para ejecutar tal módulo es muy pequeño. La operación que más tiempo de cómputo requiere es la representación de los símbolos aislados.

2. Borrado de líneas del pentagrama

Posteriormente, se debe ejecutar el módulo encargado de eliminar las líneas del pentagrama a cada símbolo: "*extraeLineasNotas*". Dicho módulo no constituye un módulo de segmentación como tal, pero su correcto funcionamiento es esencial para la ejecución del siguiente módulo "*extraeNotas2*" que sí realiza un proceso de segmentación.

El módulo "*extraeLineasNotas*" puede ejecutar uno de los dos métodos definidos para eliminar las líneas del pentagrama. En el caso de utilizar líneas azules no habrá problema con la partitura (siempre que el azul que utilicemos sea puro, con valores Rojo: 0, Verde: 0, Azul: 255), ya que permite distinguir claramente los símbolos de las líneas por la diferencia de color, sin embargo, si se utiliza el método normal, que elimina las líneas del pentagrama de un píxel de grosor, pueden haber determinados problemas.

El más frecuente es que el proceso de eliminación rompa o separe alguna de las notas. Esto se produce porque suele haber partes horizontales de un píxel de grosor localizadas en los símbolos que se confunden con las líneas del pentagrama. Si esto ocurre, las notas quedarán separadas y el siguiente módulo las separará como notas distintas generando un error en el sistema final. En la Figura 63 se pueden ver algunos ejemplos de este fenómeno:

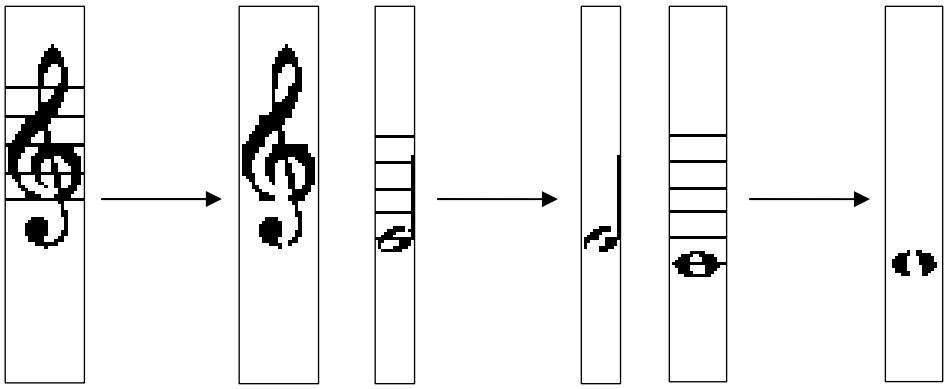


Figura 63 - Columnas impares: símbolos originales. Columnas pares: símbolos quebrados después del proceso de limpieza de líneas

Para solventar este problema sería necesario utilizar símbolos más grandes en la partitura, sin embargo, este tamaño es el más común cuando se utilizan partituras de tamaño de línea del pentagrama de un píxel. La solución ideal sería desarrollar un algoritmo, el cual se plantea como trabajo futuro, que analice de una forma más extensa los píxeles de la imagen para determinar cuáles forman parte del pentagrama, habilitando también la capacidad para utilizar partituras de grosores de línea de más de un píxel.

3. Segunda extracción de elementos

El último módulo encargado de la segmentación "*extraeNotas2*", se encarga de analizar cada uno de los símbolos extraídos en busca de nuevas notas que no se hayan separado anteriormente mediante cortes verticales, tal y como ya se comentó anteriormente en profundidad. El módulo además incluye un procedimiento para detectar si algunos de los elementos extraídos de un símbolo forman parte de impurezas (píxeles aislados, restos del pentagrama…)

En general, este procedimiento funciona adecuadamente, aislando los símbolos adecuadamente y almacenando los correctos. Sin embargo, pueden darse casos en los que falle, como ocurre con el primer elemento de la Figura 64.

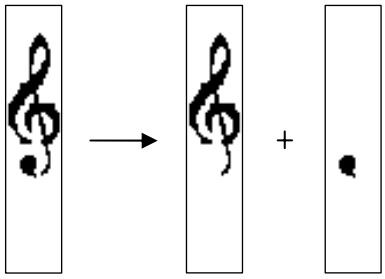


Figura 64 - Fallo de la segunda etapa segmentadora

Esto se produce porque el área de la clave de Sol es mucho mayor en comparación con el área de la bola, detectándose esta última como impureza, siendo rechazada. La raíz del problema no se encuentra en el método de segmentación utilizado ahora, sino en la imagen del símbolo proporcionado por la etapa encargada de eliminar las líneas. Es raro que la etapa de segmentación realice mal su trabajo, a no ser que haya problemas en la imagen a causa del procesado de otros módulos.

Se espera que cuando se mejore el módulo "*extraeLineasNotas*" y se consigan resultados realmente buenos para cualquier grosor, el resultado de la etapa de segmentación en su conjunto sea casi perfecto.

Al igual que las etapas anteriores, el módulo "*extraeNotas2*" realiza su tarea en poco tiempo. No se puede contabilizar dicho tiempo ya que depende de dos factores en cada caso: el tamaño de los símbolos que se analicen y la cantidad. Además, hay que tener en cuenta que dichos símbolos se representan por pantalla posteriormente, lo que implica más tiempo de cómputo total.

La evaluación de estos tres procedimientos no se ha hecho de forma separada sino de forma conjunta, ya que lo que interesa no es el funcionamiento de cada proceso sino el resultado de la segmentación de las notas en su conjunto. Para ello, se han utilizado las imágenes de test

anteriormente descritas, determinando en cada caso en número de notas reales de cada pentagrama y el número de notas obtenidas. Los resultados se muestran en la Figura 65.

Imagen de *test*	Notas reales	Notas detectadas
Manuscritos (David)		
1	18	18
2	21	21
3	21	21
4	24	24
5	24	24
6	19	19
7	16	16
8	16	16
9	18	18
10	17	17
Manuscritos (Raquel)		
1	15	15
2	16	16
3	13	13
4	11	11
5	13	13
6	13	13
7	15	16
8	13	13
9	11	11
No manuscritos		
1	28 (P1) + 11(P2)	28 (P1) + 11 (P2)
2	32 (P1) + 33 (P2) + 26 (P3)	32 (P1) + 33 (P2) + 26 (P3)
3	42 (P1) + 26 (P2)	38 (P1) + 23 (P2)
4	29	28
5	23	23
6	26	27
7	36	34
8	24	24
9	35	35
10	43	43

Figura 65 - Resultados de la extracción de notas

Se puede ver que los resultados obtenidos por lo general son buenos, salvo determinados casos problemáticos. Para el caso de notas manuscritas únicamente se ha fallado en una partitura de las 19 existentes. En contraposición, las notas no manuscritas obtuvieron peores resultados con 4 fallos de 10 partituras.

Dichos resultados se corresponden con los resultados esperados de forma teórica. Para el caso de las notas manuscritas es difícil que el proceso cometa un error dado que, por lo general, cuando una persona escribe en el pentagrama tiende a separar los elementos de forma que no se solapen entre ellos. Además, el utilizar líneas de pentagrama azules permite diferenciar perfectamente notas de líneas de pentagrama, por lo que el proceso conjunto de la segmentación se ejecuta correctamente.

Para el caso de las notas no manuscritas, es difícil distinguir las notas de las líneas del pentagrama, lo que puede provocar roturas de elementos al eliminar las líneas. Además, en este tipo de escrituras es común que determinados elementos, como las alteraciones o los números de los tiempos, estén muy próximos, llegando al punto que el programa los trate como un único símbolo, generando un error. Mientras el primer factor mencionado obtiene más símbolos de los esperados, el segundo consigue menos.

5.4.3. Evaluación del clasificador

El reconocimiento de los símbolos, es decir, la clasificación en las categorías descritas en la base de datos se realiza en el módulo "*reconocedor*". Este implementa el algoritmo k-NN utilizando la distancia Euclídea para determinar el grado de similitud entre elementos.

Antes de entrar en detalle sobre las tasas de reconocimiento del programa, hay que tener en cuenta que dichos resultados vendrán definidos por una serie de parámetros:

1. **Elementos de la base de datos**: es esencial que las imágenes que representen cada símbolo realmente caractericen la forma de dicho elemento y, además, se correspondan con la clase de símbolos utilizada en la partitura a evaluar.

2. **Vecinos a evaluar**: es importante definir un número de vecinos a evaluar adecuado a la hora de ejecutar el algoritmo k-NN, ya que la clase de la muestra más parecida al símbolo evaluado no tiene porqué ser la clase correcta. En el proyecto se ha definido un valor de $k = 5$ como recomendable, aunque dicho parámetro se puede variar.

3. **Descriptores del elemento**: en el proyecto se han utilizado los descriptores de *Fourier*, más tres características de la imagen para caracterizar la morfología de la imagen del símbolo. Es muy importante que dicho vector de características represente bien la forma de la imagen puesto que será el elemento que se utilice para comparar entre imágenes y decidir el grado de similitud.

A continuación se describirán las tasas de reconocimiento de determinadas notas aisladas, para símbolos manuscritos y no manuscritos.

Con un total de 922 símbolos y 31 clases en la base de datos, se ha cargado cada una de las categorías de prueba de símbolos manuscritos, mediante el módulo "*cargaEjemplos*", y se han clasificado mediante el módulo "*reconocedor*", obteniendo los siguientes resultados:

Categoría	Aciertos	Porcentaje
1 – Semicorchea	1 / 20	5 %
2 – Barra Divisoria	7 / 20	35 %
5 – Bemol	15 / 20	75 %
6 – Compasillo Cortado	10 / 20	50 %
7 – Compasillo	17 / 20	85 %
8 – Sostenido y medio	13 / 20	65 %
9 – Sostenido	13 / 20	65 %
10 – Silencio Semifusa	16 / 17	94 %
11 – Silencio Semicorchea	7 / 18	39 %
12 – Silencio Redonda / Blanca	16 / 20	80 %
13 – Becuadro	16 / 19	84 %
14 – Silencio Negra	9 / 20	45 %
15 – Silencio Fusa	15 / 20	75 %
16 – Blanca	18 / 20	90 %
17 – Medio Sostenido	8 / 20	40 %
18 – Silencio Corchea	6 / 19	32 %
19 – Cuatro	20 / 20	100 %
20 – Semifusa	9 / 20	45 %
24 – Clave de Fa	20 / 20	100 %
25 – Dos	6 / 20	30 %
26 – Clave de Sol	9 / 20	45 %
27 – Corchea	15 / 20	75 %
32 – Doble Sostenido	13 / 20	65 %
33 – Tres	17 / 20	85 %
35 – Fusa	5 / 15	33 %
38 – Ligadura	20 / 20	100 %
41 – Seis	11 / 20	55 %
42 – Negra	17 / 20	85 %
43 – Ocho	6 / 20	30 %
45 – Puntillo	4 / 20	20 %
46 – Redonda	10 / 20	50 %

Figura 66 - Porcentajes de acierto en la clasificación de partituras manuscritas para un conjunto amplio de símbolos

Como se puede observar, los porcentajes de acierto para las partituras manuscritas varían considerablemente en función del tipo de símbolo, consiguiendo desde porcentajes muy altos, como ocurre con las ligaduras (100 %), las claves de Fa (100 %), los silencio de semifusas (94 %) o las notas negras (85 %) y blancas (90 %), a porcentajes muy bajos, tomando como ejemplo las semicorchea (5%), los silencios de corchea (32 %), las barras divisorias (35%), el silencio de negra (45 %)… Esta variedad de porcentajes de aciertos depende de muchos factores: la forma de escribir del usuario, la varianza que introduzca cada usuario al escribir un mismo símbolo, la frecuencia con la que se escriben los símbolos (se supone que los símbolos más habituales se realizan de una forma más uniforme, disminuyendo la varianza entre símbolos del mismo genero), etc.

Realizando la media de los valores se obtiene el porcentaje de acierto general, un 60 % de acierto sobre la partitura (o, en otros términos, un 40 % de error global). Este valor de acierto se considera bastante bajo, no obstante, hay que tener en cuenta que se ha analizado cada elemento dentro de un abanico de 31 clases posibles, con 922 símbolos a comparar. Lógicamente, a mayor número de clases y elementos, mayor será la posibilidad de error.

Normalmente, en una partitura manuscrita no se utilizan tantos símbolos, sobretodo si la partitura forma parte de un sistema de dictado, como en un principio se definió el proyecto. Si únicamente se tiene en cuenta un número reducido de clases (utilizando 235 elementos en la base de datos) la probabilidad de realizar una mala clasificación será menor, aumentando la tasa de reconocimiento, como se puede observar en la siguiente tabla y en la matriz de confusión:

Categoría	Aciertos	Porcentaje
2 – Barra Divisoria	6 / 20	30 %
12 – Silencio Blanca	20 / 20	100 %
14 – Silencio Negra	11 / 20	55 %
16 – Blanca	19 / 20	95 %
18 – Silencio Corchea	19 / 20	95 %
26 – Clave de Sol	15 / 20	75 %
27 – Corchea	20 / 20	100 %
42 – Negra	17 / 20	85 %

Figura 67 - Porcentajes de acierto en la clasificación de partituras manuscritas para un conjunto reducido

	Barra	Silencio Blanca	Silencio Negra	Blanca	Silencio Corchea	Clave de Sol	Negra	Corchea	E.T.
Barra	**30 %**	0 %	10 %	5 %	20 %	30 %	0 %	5 %	**70 %**
Silencio Blanca	0 %	**100 %**	0 %	0 %	0 %	0 %	0 %	0 %	**0 %**
Silencio Negra	5 %	0 %	**55 %**	0 %	0 %	40 %	0 %	0 %	**45 %**
Blanca	0 %	0 %	5 %	**95 %**	0 %	0 %	0 %	0 %	**5 %**
Silencio Corchea	0 %	0 %	5 %	0 %	**95 %**	0 %	0 %	0 %	**5 %**
Clave de Sol	10 %	0 %	15 %	0 %	0 %	**75 %**	0 %	0 %	**25 %**
Negra	0 %	0 %	0 %	15 %	0 %	0 %	**85 %**	0 %	**15 %**
Corchea	0 %	0 %	0 %	0 %	0 %	0 %	0 %	**100 %**	**0 %**

Figura 68 - Matriz de confusión de la clasificación de partituras manuscritas para un conjunto reducido

Se observa un gran incremento del porcentaje de acierto, reduciendo el error global a un 20 %, es decir, la mitad. Variando las clases a utilizar se pueden llegar a conseguir tasas de reconocimiento más altas.

En la matriz de confusión se puede observar que los símbolos de las claves de Sol y las barras de compás se confunden frecuentemente. De hecho, el 30 % de las barras se clasificaron como claves de Sol y el 10% de las barras como claves. Para explicar este fenómeno se debe acudir a los descriptores de *Fourier*. Es probable que los 30 primeros descriptores de la clave de Sol dibujen una especie de rectángulo que vaya tomando forma según se añaden más descriptores. Esta forma coincidirá con la forma de la barra de compás, impidiendo la correcta clasificación. Lo mismo ocurre con los símbolos de los silencios de negra. Debido a la forma tosca que se genera con los primeros descriptores, se obtiene un error confundiendo el elemento con barras (5%) o claves de Sol (40%).

Para solucionar este fenómeno podemos hacer uso de otros descriptores distintos o, ya que los elementos problemáticos son principalmente las barras y las claves de Sol, no tener en cuenta dichas clases y suponer que estos datos los proporcionará el usuario mediante una interfaz gráfica.

Para el caso de símbolos no manuscritos, el procedimiento para evaluar el sistema ha sido similar. Con un total de 1397 elementos en 41 categorías en la base de datos, se han obteniendo los siguientes porcentajes:

Categoría	Aciertos	Porcentaje
1 – Semicorchea	9 / 20	45 %
2 – Barra Divisoria	20 / 20	100 %
3 – 4 por 4	15 / 20	75 %
4 – 2 por 4	10 / 10	100 %
5 – Bemol	17 / 20	85 %
6 – Compasillo Cortado	12 / 20	60 %
7 – Compasillo	16 / 20	80 %
8 – Sostenido y medio	10 / 10	100 %
9 – Sostenido	6 / 15	40 %
10 – Silencio Semifusa	17 / 20	85 %
11 – Silencio Semicorchea	16 / 20	80 %
12 – Silencio Redonda / Blanca	10 / 10	100 %
13 – Becuadro	6 / 10	60 %
14 – Silencio Negra	5 / 20	25 %
15 – Silencio Fusa	10 / 20	50 %
16 – Blanca	20 / 20	100 %
17 – Medio Sostenido	10 / 10	100 %
18 – Silencio Corchea	17 / 20	85 %
19 – Cuatro	8 / 10	80 %
20 – Semifusa	6 / 10	60 %
24 – Clave de Fa	20 / 20	100 %
25 – Dos	7 / 10	70 %
26 – Clave de Sol	16 / 20	80 %
27 – Corchea	13 / 20	65 %
29 – Cuádruple	9 / 10	90 %
31 – Doble	7 / 10	70 %
32 – Doble sostenido	5 / 10	50 %
33 – Tres	5 / 10	50 %
34 – 2 por 2	7 / 10	70 %
35 – Fusa	3 / 10	30 %
36 – 3 por 2	7 / 10	70 %
38 – Ligadura	8/ 10	80 %
39 – Medio Bemol	10 / 10	100 %
40 – 6 por 8	8 / 10	80 %
41 – Seis	8 / 10	80 %
42 – Negra	20 / 20	100 %
43 – Ocho	10 /10	100 %
45 – Puntillo	8 / 10	80 %
46 – Redonda	10 / 10	100 %
48 – Doble bemol	10 / 10	100 %
49 – 3 por 4	4 / 10	40 %

Figura 69 - Porcentajes de acierto en la clasificación de partituras no manuscritas para un conjunto extenso

En este caso se puede comprobar que la mayor parte de los resultados (concretamente el 70%) se sitúan entre un 70% – 100% de acierto. Esto es lógico, ya que los símbolos hechos a máquina cuentan con muy poca varianza entre símbolos de la misma clase, por lo que las

tasas de reconocimiento son mayores que en el otro ejemplo. Si se determina el porcentaje de acierto global se obtiene un 76 % aproximadamente, lo que equivale a un 24 % de error global.

El valor de error global se puede considerar algo elevado si tenemos en cuenta que los símbolos no manuscritos son muy parecidos entre elementos de la misma clase. La mayor parte de este error viene dado por la morfología de determinados símbolos de prueba, ya que, como ya se comentó, los símbolos se han extraído de fuentes distintas a las utilizadas para la creación de la base de datos.

De la misma forma que en el caso manuscrito, si se tiene en cuenta un conjunto reducido de símbolos la tasa de error global disminuirá notablemente:

Categoría	Aciertos	Porcentaje
2 – Barra Divisoria	20 / 20	100 %
12 – Silencio Blanca	10 / 10	100 %
14 – Silencio Negra	7 / 20	35 %
16 – Blanca	20 / 20	100 %
18 – Silencio Corchea	18 / 20	90 %
26 – Clave de Sol	20 / 20	100 %
27 – Corchea	20 / 20	100 %
42 – Negra	20 / 20	100 %

Figura 70 - Porcentajes de acierto en la clasificación de partituras no manuscritas para un conjunto reducido

	Barra	Silencio Blanca	Silencio Negra	Blanca	Silencio Corchea	Clave de Sol	Negra	Corchea	E.T.
Barra	**100 %**	0 %	0 %	0 %	0 %	0 %	0 %	0 %	**0 %**
Silencio Blanca	0 %	**100 %**	0 %	0 %	0 %	0 %	0 %	0 %	**0 %**
Silencio Negra	0 %	0 %	**35 %**	0 %	0 %	65 %	0 %	0 %	**65 %**
Blanca	0 %	0 %	0 %	**100 %**	0 %	0 %	0 %	0 %	**0 %**
Silencio Corchea	0 %	0 %	0 %	5 %	**90 %**	5 %	0 %	0 %	**10 %**
Clave de Sol	0 %	0 %	0 %	0 %	0 %	**100 %**	0 %	0 %	**0 %**
Negra	0 %	0 %	0 %	0 %	0 %	0 %	**100 %**	0 %	**0 %**
Corchea	0 %	0 %	0 %	0 %	0 %	0 %	0 %	**100 %**	**0 %**

Figura 71 - Matriz de confusión de la clasificación de partituras no manuscritas para un conjunto reducido

Se puede ver, al igual que en el ejemplo anterior, que la tasa de error global disminuye, en este caso hasta el 9 % aproximadamente. Si se conoce a priori el conjunto de símbolos más común en la partitura a utilizar, se puede adaptar la base de datos para obtener mejores resultados. *(NOTA: se ha escogido para la evaluación el mismo conjunto de símbolos que para el caso manuscrito para poder comparar los valores de reconocimiento bajo las mismas circunstancias)*

Al igual que ocurre en el caso manuscrito, en este caso también hay símbolos que se confunden frecuentemente con otros. Destaca el caso del silencio de negra, el cual se clasifica como clave de sol el 65 % de las veces. La explicación de este fenómeno es la misma que se definió en el caso manuscrito.

Tanto para el caso de símbolos manuscritos como para el de símbolos no manuscritos, la presencia de ruido en los elementos afecta en gran medida la eficacia del clasificador. Es muy importante que la partitura de entrada sea lo más grande posible (refiriéndonos a la resolución, dentro de los límites impuestos por los algoritmos de eliminación de líneas del pentagrama) para que el efecto del ruido que aparece al realizar el preprocesado de la imagen sea mínimo. A más ruido, más posibilidades hay de que falle el clasificador.

El tiempo de cómputo necesario para realizar la clasificación suele ser del orden de un par de segundos, sino menos. La operación que más tiempo requiere es el cálculo de las distancias entre elementos, a partir de los vectores de características.

5.4.4. Evaluación integrada

Este apartado evaluará, de forma general, las prestaciones del programa de cara al usuario, teniendo en cuenta todas las etapas que conforman el proyecto.

Como ya se comentó, el objetivo principal del usuario es que el programa funcione bien, en nuestro caso, que las partituras se analicen adecuadamente y se representen tal y como son en la realidad mediante datos simbólicos que se podrán representar de forma gráfica o acústica. Esta tarea no es trivial, existen numerosos factores que pueden llevar a que el programa realice mal su tarea en algún punto. Generalmente éstos suelen ser provocados por las etapas de segmentación, por notas quebradas resultado de la eliminación de las líneas del pentagrama, no obstante, como ya se ha visto, también existen errores de clasificación que se podrían mejorar con el diseño de un clasificador más potente.

Debemos prestar especial atención al ruido de la imagen. Este elemento es determinante a la hora de que el programa realice bien su tarea. Si la imagen original ya cuenta con ruido o la imagen no es nítida (los elementos están ligeramente difuminados) al realizar la conversión a formato binario se creará una gran cantidad de ruido, muy evidente en las líneas del pentagrama. Además, si la partitura cuenta con cierta inclinación, el proceso de rotación de la imagen para la corrección generará más ruido, debido a la aproximación que se realiza al recolocar todos los píxeles en sus respectivas posiciones nuevas (cosa que no ocurriría si la imagen en lugar de ser una señal discreta fuera continúa).

En cuanto al manejo del programa se considera que se ha diseñado un sistema fácil de manejar, intuitivo, el cual dispone de una gran documentación y gran cantidad de mensajes durante la ejecución.

1. Partituras manuscritas

A continuación se mostrarán algunos de los resultados obtenidos en la ejecución del programa para determinados casos de partituras manuscritas de los ejemplos de *test*:

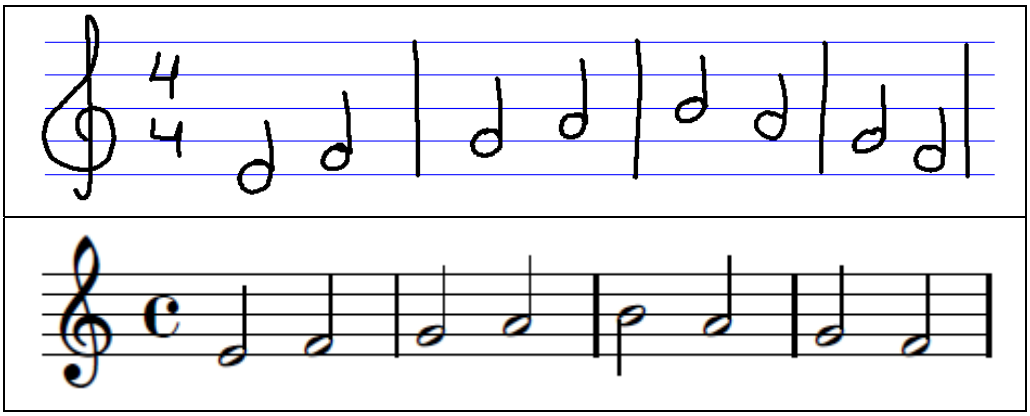


Figura 72 - Ejemplo 1. (Pentagrama 1 de “Pentagramas_Raquel”)

En la Figura 72 se puede observar un pentagrama reconocido perfectamente. Por defecto, el tiempo se representa en *Lilypond* como compasillo si se trata de una partitura de 4/4 o como tiempo binario si se trata de un 2/2.

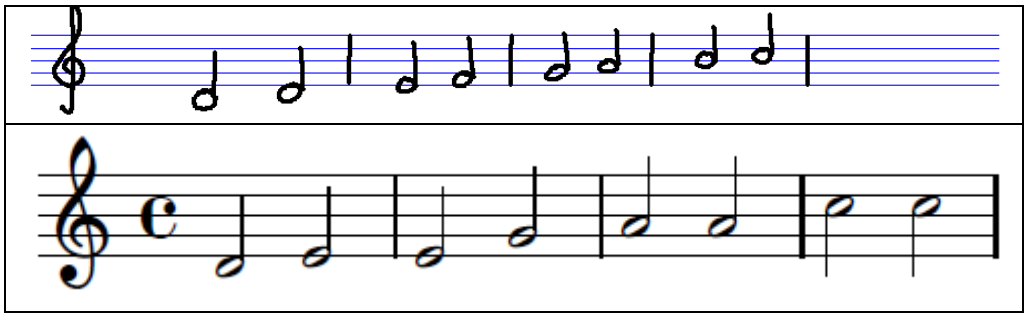


Figura 73 - Ejemplo 2. (Pentagrama 3 de “Pentagramas_Raquel”)

En la Figura 73 se observa como el pentagrama original no dispone de tiempo de compás definido. Como ya se comentó, por defecto se establece el compasillo (tiempo de 4/4) en caso de no encontrar tiempo en el pentagrama. En cuanto al resultado obtenido, podemos apreciar que hay 4 blancas que no cuentan con el tono correcto. Esto se debe a que el algoritmo dedicado a la detección del tono utiliza la altura del símbolo para definir una máscara, a partir de unos valores empíricos, que busque la cabeza de la nota en la imagen del pentagrama para asignar el tono. Si la plica es más grande de lo normal o más pequeña, estos valores empíricos no proporcionarán unos valores adecuados y fallará el algoritmo.

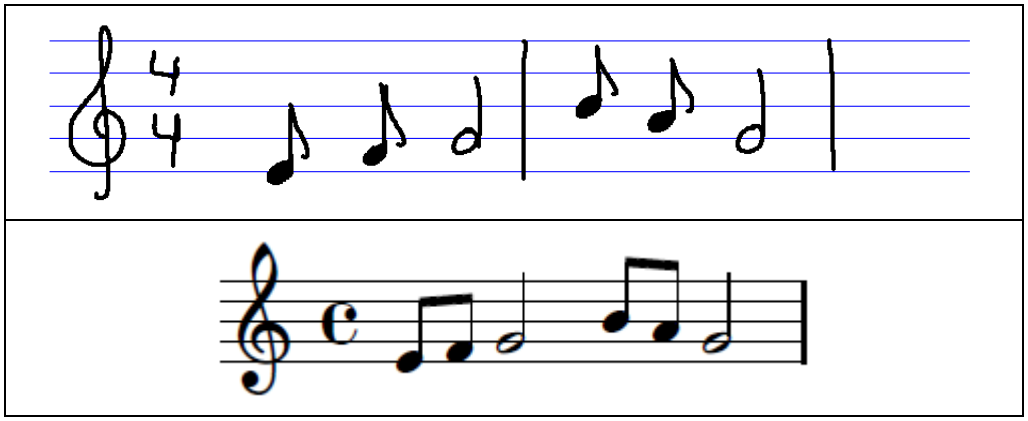


Figura 74 - Ejemplo 3. (Pentagrama 4 de “Pentagramas_Raquel”)

El ejemplo 3 muestra un pentagrama manuscrito perfectamente reconocido. Cabe mencionar que las corcheas no aparecen separadas como en la partitura original, sino unidas. Este fenómeno lo genera el programa de representación *Lilypond* de forma automática. Es posible desactivarlo, como se puede comprobar en el anexo de este proyecto, pero finalmente no se ha implementado la opción por ser una mera cuestión estética.

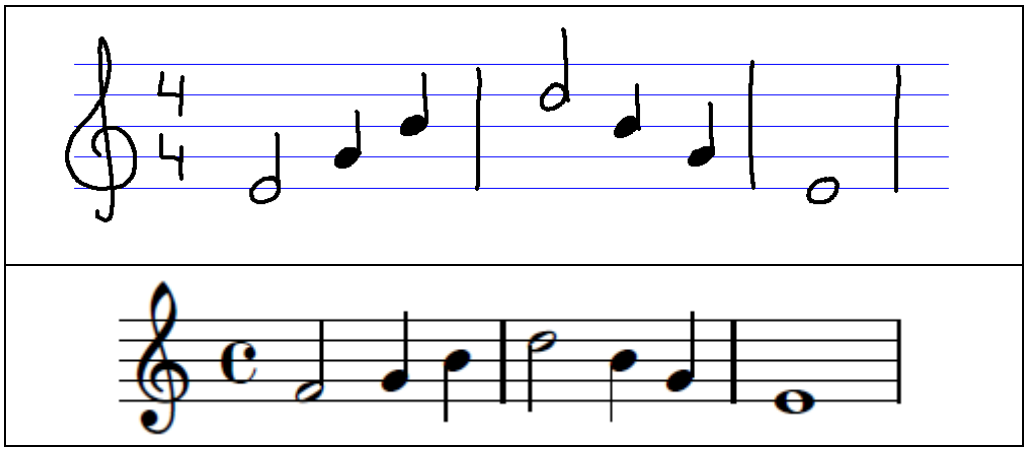


Figura 75 - Ejemplo 4. (Pentagrama 5 de "Pentagramas_Raquel")

En la Figura 75 se puede observar otro pentagrama reconocido adecuadamente. Este ejemplo sirve para ilustrar que las plicas de las notas se colocan de forma automática en el sistema de representación *Lilypond*, aunque la partitura original cuente con las plicas sin colocar.

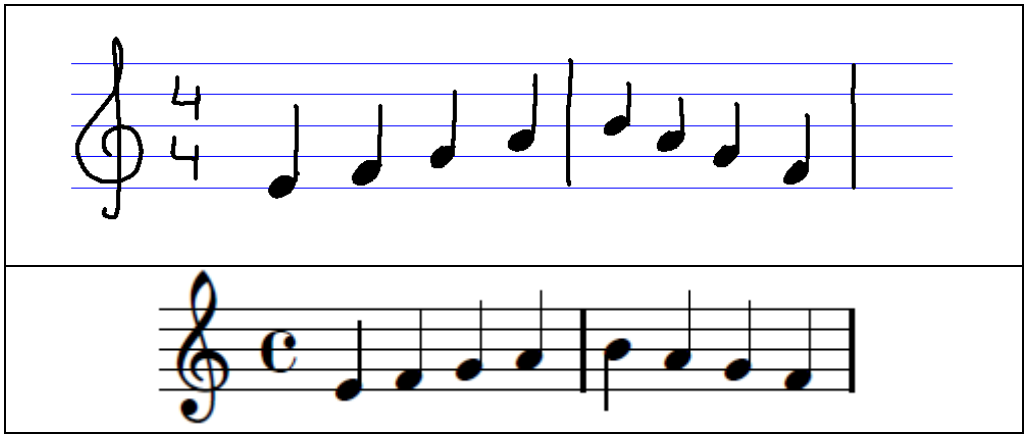


Figura 76 - Ejemplo 5. (Pentagrama 6 de "Pentagramas_Raquel")

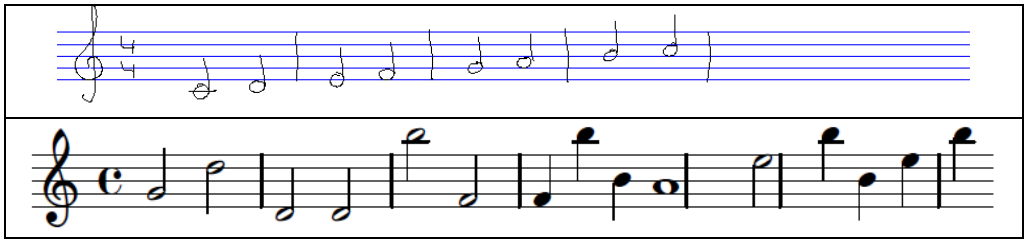


Figura 77 - Ejemplo 6. (Pentagrama 7 de "Pentagramas_Raquel")

El pentagrama de la Figura 77 sirve como ejemplo para ilustrar que los símbolos de la partitura que se dibujen deben tener un grosor mínimo para su correcta interpretación. En este caso, los

símbolos cuentan con un grosor tan pequeño que el reconocedor es incapaz de clasificarlos correctamente. Además, este ejemplo también sirve para avisar de que las componentes de los símbolos deben solaparte y no encontrarse separados. Si se comprueba la sexta blanca, se puede ver que la cabeza de la nota y la plica están separadas, siendo considerados como dos símbolos independientes por la etapa de segmentación, y generando dos errores de clasificación.

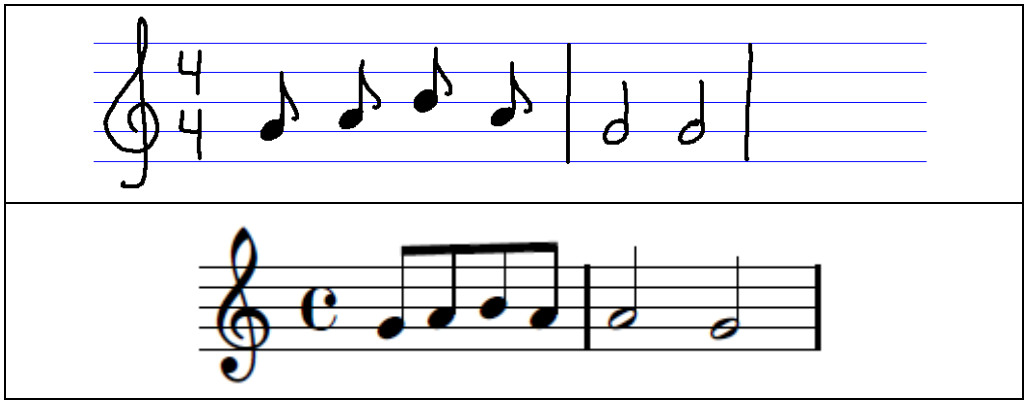


Figura 78 - Ejemplo 7. (Pentagrama 9 de "Pentagramas_Raquel")

En el ejemplo 7 se puede observar un pentagrama detectado correctamente con un único fallo de detección de tono en la primera nota blanca. Sin embargo, la finalidad de este ejemplo es demostrar que el programa también soporta anacrusas. En este caso, se cuenta con una anacrusa de valor dos, cuando el valor del compás es de cuatro. Este valor es uno de los posibles valores válidos definidos para las anacrusas dentro del programa, por las limitaciónes de *Lilypond*.

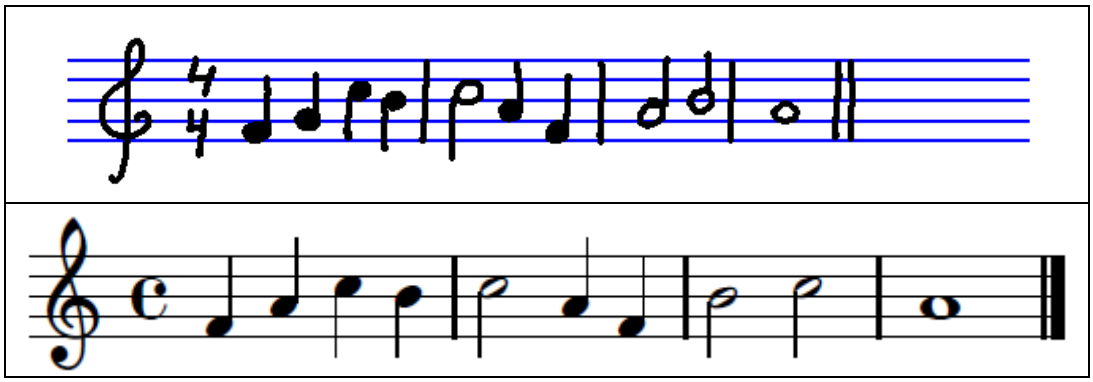


Figura 79 - Ejemplo 8. (Pentagrama 1 de "Pentagramas_David")

En la figura anterior se puede ver como existe un pequeño error de detección de tono en la penúltima nota, posiblemente producido por la corta longitud de la plica.

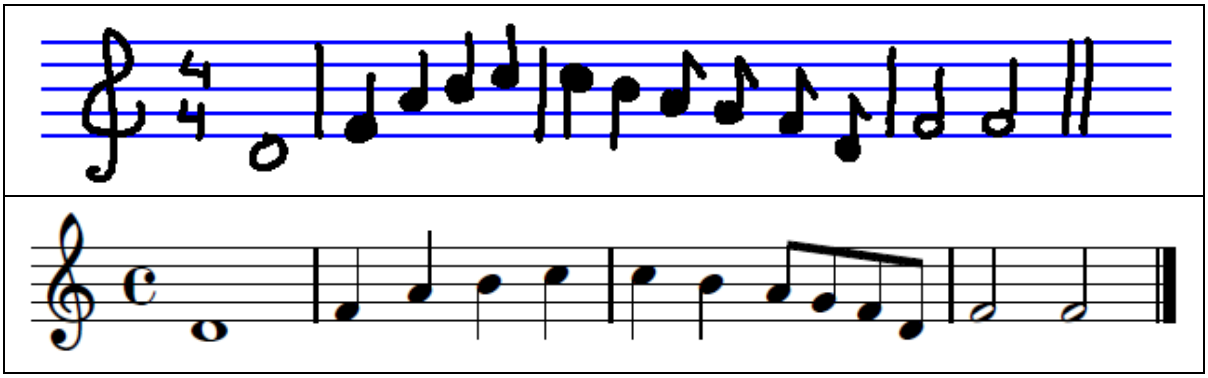


Figura 80 - Ejemplo 9. (Pentagrama 2 de “Pentagramas_David”)

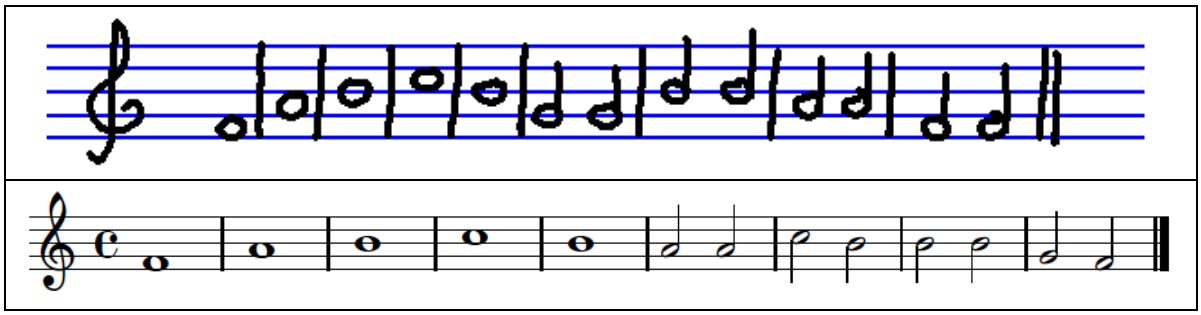


Figura 81 - Ejemplo 10. (Pentagrama 4 de “Pentagramas_David”)

En este ejemplo también se producen algunos fallos de detección de tono en las notas blancas. Este fenómeno es muy difícil de arreglar en este tipo de partituras puesto que el tamaño de la plica varía frecuentemente en las notas manuscritas.

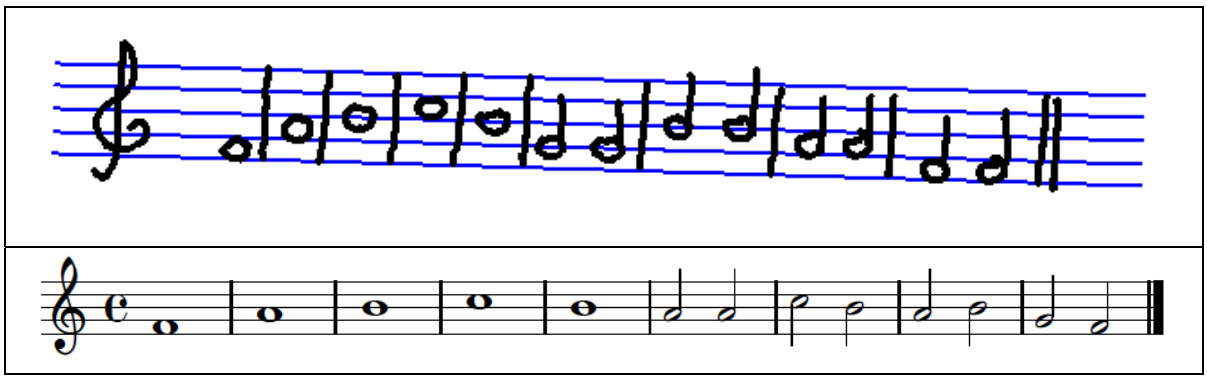


Figura 82 - Ejemplo 11. (Pentagrama 5 de “Pentagramas_David”)

Este ejemplo se corresponde con una versión inclinada de la figura anterior. Como se puede observar, el programa consigue el mismo resultado gracias al proceso de corrección de la inclinación desarrollado. A pesar de que se introduce ruido por la corrección, gracias a que los símbolos son de tamaño grande se puede realizar una buena clasificación (sin poder evitar los fallos de detección del tono de las notas blancas, comentado en el ejemplo 10)

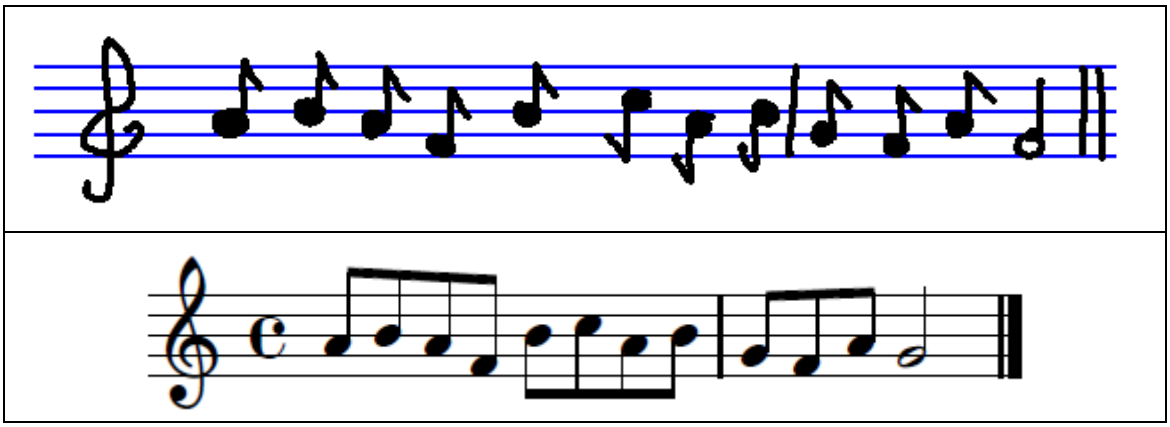


Figura 83 - Ejemplo 12. (Pentagrama 6 de "Pentagramas_David")

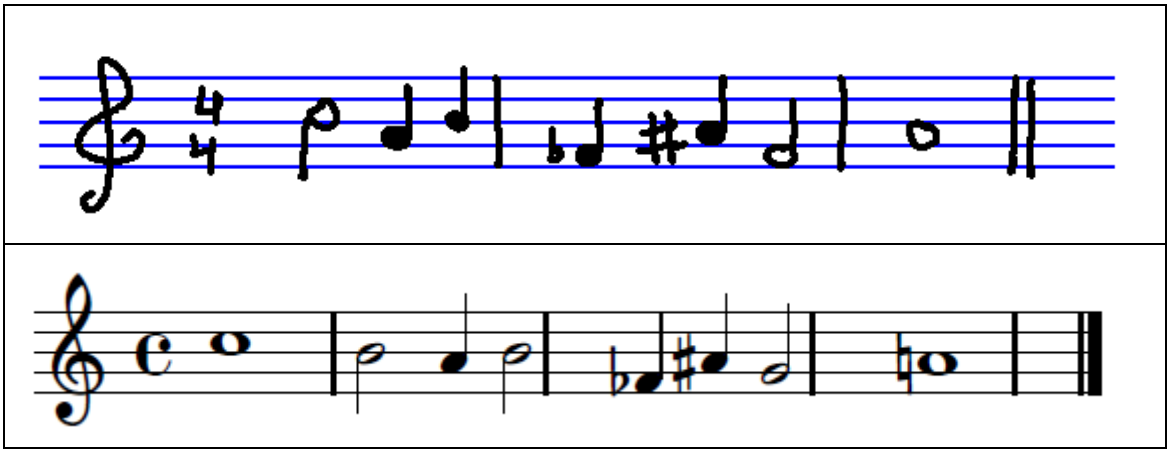


Figura 84 - Ejemplo 13. (Pentagrama 7 de "Pentagramas_David")

Se puede observar en la Figura 84 que se ha incluido una redonda como primer símbolo y el cuarto símbolo se ha clasificado como blanca cuando en realidad es una negra. La aparición de la redonda se produce por la mala clasificación de uno de los cuatros que representan el tiempo de 4/4. Uno de ellos se habrá clasificado como tal, mientras el otro se habrá definido como redonda. Al existir un único número aislado, no se toma en cuenta. Si no fuera porque se generan valores por defecto (tiempo de 4/4 y clave de Sol), los pentagramas que no cuenten con elementos que definan el tiempo y la clave (ya sea porque no existen o se han reconocido mal), no se habrían podido representar correctamente.

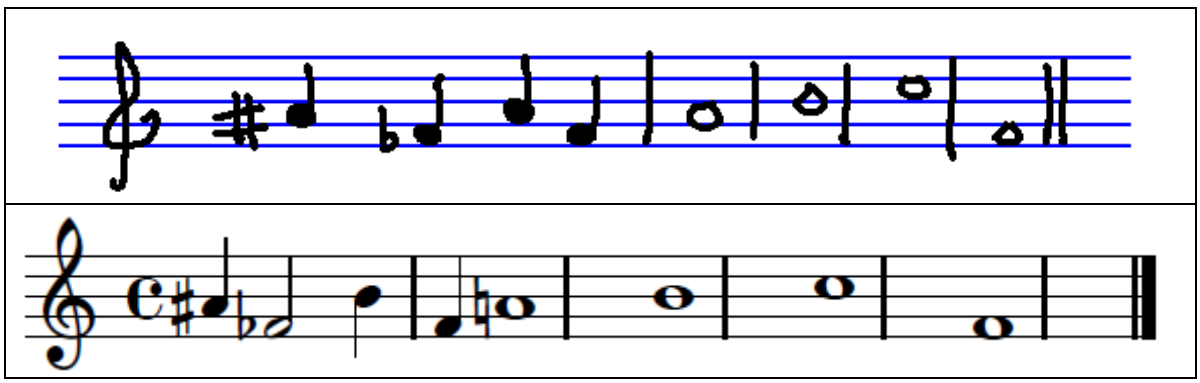


Figura 85 - Ejemplo 14. (Pentagrama 9 de "Pentagramas_David")

En este ejemplo se ha clasificado mal una negra como una blanca y no se han detectado correctamente el tono de otras dos negras. Este error ha provocado que la colocación de las barras de separación de compás por parte del programa *Lilypond* se haya realizado mal.

2. Partituras no manuscritas

De la misma forma que en el caso anterior, se mostrarán algunos de los resultados obtenidos en la ejecución del programa para determinados casos de partituras no manuscritas de los ejemplos de *test*:

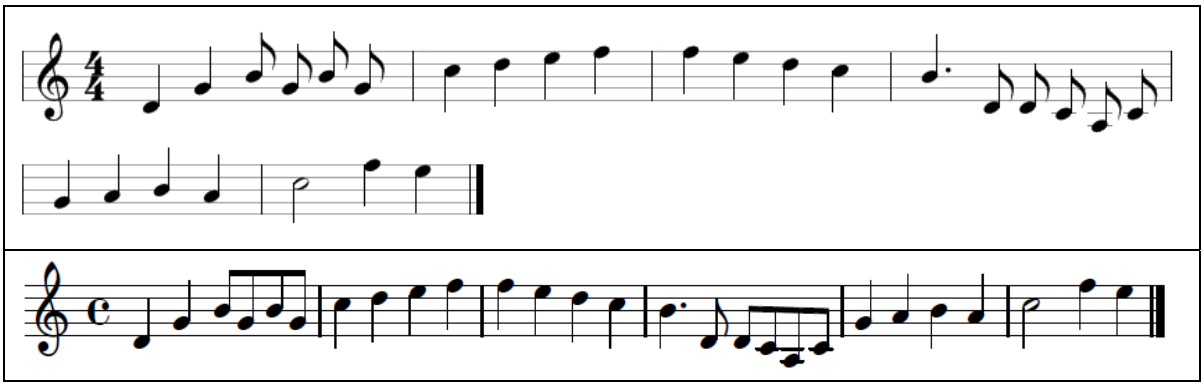


Figura 86 - Ejemplo 15. (Ejemplo 1 no manuscrito)

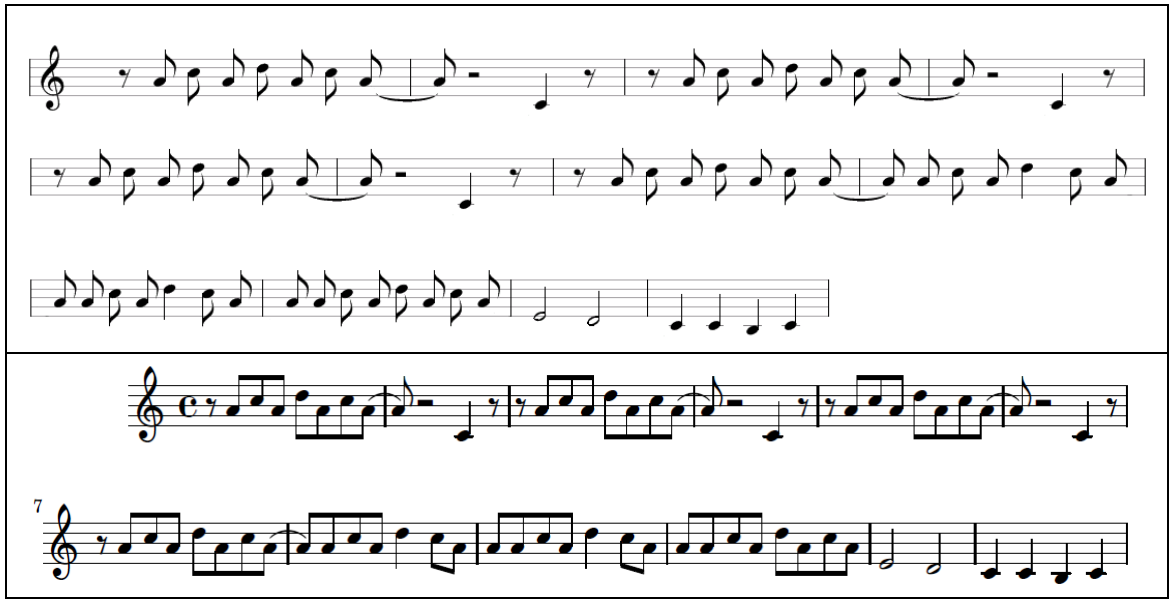


Figura 87 - Ejemplo 16. (Ejemplo 2 no manuscrito)

En los ejemplos 15 y 16 se pueden observar dos partituras no manuscritas, de dos y tres pentagramas respectivamente, que han sido detectadas correctamente. Estos ejemplos demuestran que el programa también es capaz de trabajar con más de un pentagrama por imagen a analizar.

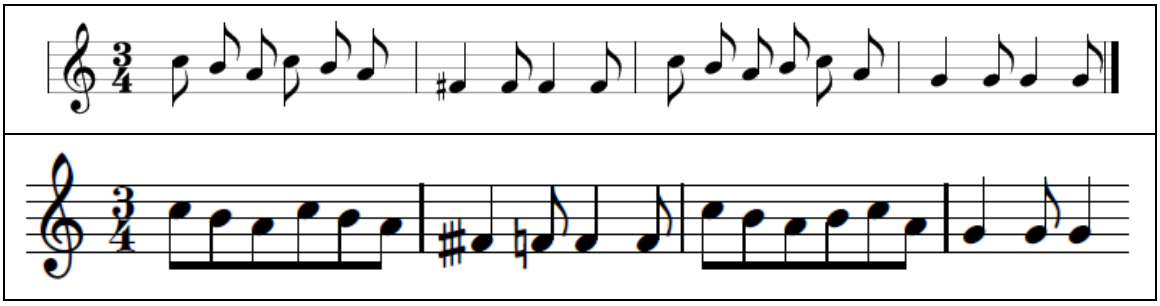


Figura 88 - Ejemplo 17. (Ejemplo 4 no manuscrito)

En la figura Figura 88, se ha clasificado mal el sostenido de la partitura, confundiéndolo con un sostenido y medio. Además, la última corchea del pentagrama esta tan cerca de la barra final que, al realizar el proceso de binarización de la imagen, ambos elementos se solapan y pasan a formar parte de un mismo elemento, produciéndose una mala clasificación al no segmentarse.

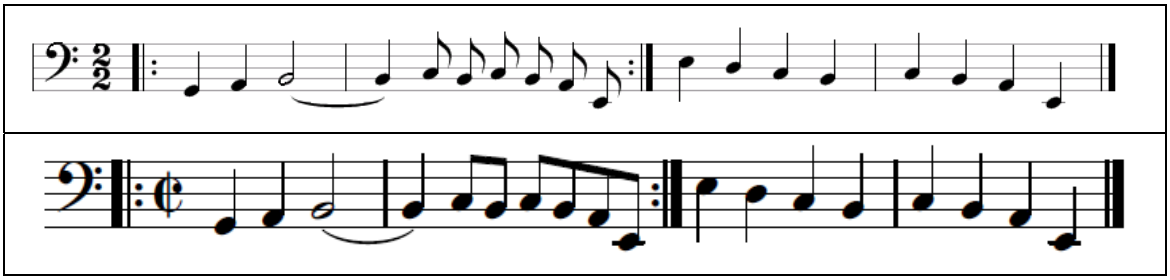


Figura 89 - Ejemplo 18. (Ejemplo 7 no manuscrito)

En el ejemplo 18, se puede observar un pentagrama clasificado correctamente. El compás binario de la figura se ha definido antes del inicio de la barra de repetición, pero por cuestiones del programa de representación *Lilypond*, aparece representado después de dicha barra.

3. Resumen

Después de mostrar y comentar algunos de los resultados obtenidos para ambos tipos de partituras, se resumirán los resultados generales para todas las imágenes de *test* almacenadas:

Para el caso manuscrito se tiene:

Caso	Cantidad	Porcentaje
Ejemplos reconocidos correctamente	6	32 %
Ejemplos reconocidos correctamente con, al menos, un fallo de tono de nota	5	26 %
Ejemplos con, al menos, un símbolo mal clasificado y el resto de las notas sin fallos de tono	1	5 %
Ejemplos con símbolos mal clasificados y fallos de detección de tono	7	37 %
Ejemplos que no se han podido representar con *Lilypond* por problemas de sintaxis	0	0 %
Total ejemplos	**19**	**100 %**

Figura 90 - Resultados caso manuscrito

Para el caso no manuscrito:

Caso	Cantidad	Porcentaje
Ejemplos reconocidos correctamente	4	40 %
Ejemplos reconocidos correctamente con, al menos, un fallo de tono de nota	0	0 %
Ejemplos con, al menos, un símbolo mal clasificado y el resto de las notas sin fallos de tono	3	30 %
Ejemplos con símbolos mal clasificados y fallos de detección de tono	2	20 %
Ejemplos que no se han podido representar con *Lilypond* por problemas de sintaxis	1	10 %
Total ejemplos	**10**	**100 %**

Figura 91 - Resultados caso no manuscrito

Se puede comprobar que para el caso de los símbolos manuscritos el porcentaje de partituras reconocidas sin fallos es menor (32 %) que para el de símbolos no manuscritos (40 %). Sin embargo se observan algunas peculiaridades. Mientras para el caso manuscrito existe un 26 % de partituras reconocidas correctamente con fallos en la detección del tono, en el otro caso no existen partituras que cumplan esta condición. Este fenómeno ocurre porque, como ya se explicó, para la detección del tono es fundamental que la plica del símbolo cuente con un determinado tamaño en función de la cabeza de la nota, y de variar su tamaño, cosa que no ocurre en el caso no manuscrito, es posible que falle.

Para los casos de errores de clasificación, podemos comprobar que en ambos casos los resultados son similares. Aunque teóricamente se deben obtener mejores resultados de clasificación con las partituras no manuscritas, existe el hecho de que se introducen errores frecuentemente por la eliminación de las líneas del pentagrama de grosor un píxel, que puede fragmentar las notas. Esto aumenta el error global hasta conseguir una tasa superior a la del caso de los símbolos manuscritos.

En estos casos de mala clasificación, se puede ver, para el caso de las partituras no manuscritas, que los símbolos que sí se han clasificado correctamente cuentan con el tono correcto, verificando el buen funcionamiento del algoritmo de detección de tono para estos casos.

Por último, se puede ver que son muy pocos los casos en los que el sistema de representación *Lilypond* ha sido incapaz de representar los resultados. Si se produce un caso de este estilo suele ser porque se han realizado una mala clasificación de los elementos y se han representado de forma contigua símbolos que no se pueden representar juntos (por ejemplo, un sostenido antes que una clave de Sol)

5.5. Conclusiones

Se ha podido comprobar que, si se define una base de datos adecuada y la partitura de entrada cuenta con unas características determinadas, se pueden obtener unos resultados satisfactorios, consiguiendo una versión de la partitura digitalizada en formato PDF, Postscript y MIDI.

Además, la implementación del programa permite ejecutar la tarea completa (procesado de la imagen, segmentación, clasificación y representación) en muy poco tiempo, en torno al medio minuto, en contraposición con otros sistemas OMR desarrollados que necesitan de varios minutos para conseguir unos resultados. Además, hay que tener en cuenta que todas las tareas realizadas se monitorizan mediante gráficos y mensajes por línea de comandos, y que dichas tareas consumen parte del tiempo del programa. Si no ejecutáramos dicha monitorización, la tarea del programa se ejecutaría de forma inmediata en menos tiempo.

No obstante, aunque el resultado obtenido es satisfactorio, todavía queda mucha tarea por realizar en este campo puesto que hay multitud de factores a tener en cuenta que pueden dar lugar a errores y se considera que el reconocedor aún no es lo suficientemente robusto como para ejecutarlo con fines comerciales.

6. Conclusiones

El desarrollo de un sistema OMR supuso un gran reto al comienzo. Poco a poco, mediante el estudio de los trabajos de otros autores en el campo del reconocimiento óptico musical, se fue simplificando el problema hasta desarrollar unos bloques que conformaron la arquitectura del proyecto.

En general, se han cumplido todos los objetivos. Se ha conseguido crear un sistema funcional capaz de traducir una partitura en papel a una partitura digitalizada.

Dicho programa cuenta con una serie de módulos que realizan cada una de las tareas definidas por los objetivos iniciales. Se ha conseguido implementar un método eficaz para adaptar la imagen adecuadamente, se ha resuelto el problema de la corrección de la inclinación de la partitura, se ha conseguido crear módulos que extraigan adecuadamente los pentagramas y las notas...

El resultado final obtenido es satisfactorio y se considera más que suficiente para el trabajo realizado por una única persona. Aunque para según qué casos pueden presentarse errores, se ha realizado un trabajo en el que se han tenido en cuenta numerosísimos factores que competen dentro del sistema OMR y aún así se ha conseguido implementar un sistema completamente funcional de principio a fin.

Sin embargo, debido a las características del sistema desarrollado, se considera especialmente adecuada una metodología colaboratíva, particularmente en la fase de diseño, para desarrollar un trabajo de estas carácterísticas. Se considera que de esta forma, aparte de concluir el proyecto antes, se habrían conseguido resultados aún mejores que los obtenidos, gracias al trabajo en grupo y a las ideas propuestas por varias mentes pensantes. Dado el carácter de un proyecto fín de carrera, que por definición es individual, no se pudo disponer de dicha metodología.

Algunos de los objetivos propuestos han presentado determinadas complicaciones y se ha requerido una dedicación especial para su resolución. Cabe destacar el trabajo realizado para el desarrollo de:

- Método eficaz para la detección y corrección de la inclinación de la partitura
- Módulo clasificador
- Módulo encargado de la detección del tono de las notas y de comprobar los posibles errores de la partitura

Para el primer elemento se llegaron a implementar hasta cuatro algoritmos distintos hasta que los resultados del último fueron lo suficientemente buenos como para considerar cumplido el objetivo. Para el segundo, se estudiaron en profundidad las características de los símbolos a utilizar para poder definir unos descriptores que pudieran representar de forma adecuada las imágenes. Tras comprobar los resultados con multitud de ellos, se utilizaron principalmente los descriptores de *Fourier* con otros elementos citados en apartados anteriores. Para el tercero, fue necesario diseñar un algoritmo de detección del tono de las notas e implementar un amplio conjunto de reglas basadas en determinados puntos de la teoría musical para comprobar la

métrica y la existencia de posibles errores en los datos reconocidos. El módulo encargado de dicha tarea cuenta con más de 1000 líneas de código.

Como conclusión final, se considera que el trabajo realizado ha obtenido unos buenos resultados. Se ha creado un sistema flexible, útil y eficaz, que permite la modificación de ciertos parámetros internos gracias a la gran documentación disponible y la extensión del programa por su carácter modular. Además, aparte de haber cumplido los objetivos propuestos, facilita el cumplimiento de otros objetivos posteriores como se describe en las líneas de trabajos futuras.

En resumen, el proyecto realizado cumple con los objetivos planteados satisfactoriamente, reconociendo las partituras manuscritas y no manuscritas para la mayoría de los casos descritos con una pequeña tasa de error asumible.

7. Líneas de trabajos futuros

Aunque se han conseguido los objetivos marcados desde un principio y se ha desarrollado un sistema OMR funcional que obtiene los resultados en muy poco tiempo respecto a otros sistemas (ver capítulo 2), aún se pueden realizar una serie de mejoras que perfeccionen el funcionamiento del programa.

El sistema desarrollado aún no es lo suficientemente maduro como para utilizarlo como un sistema comercial. El actual uso de este programa se debería limitar al uso académico y experimental. Esto se debe a que todavía existen demasiadas variables a tener en cuenta para poder analizar una partitura correctamente sin errores. Además, como ya se comentó, la generación de una buena base de datos es imprescindible para el buen funcionamiento del sistema y actualmente la implementación realizada no facilita lo suficiente esta tarea. Sería adecuado que en futuras versiones el programa fuera capaz de proporcionar al usuario una fase de entrenamiento donde el sistema pudiera aprender la grafía del usuario y crear su propia base de datos, en el caso de partituras manuscritas.

Para mejorar el sistema y poder plantearlo como un programa comercial debemos centrarnos, principalmente, en la reducción drástica del error global de clasificación, así como en la reducción de problemas de segmentación. De cara al usuario el programa no debe generar apenas errores y no debe requerir un gran esfuerzo por su parte para el entrenamiento. A continuación se detallan algunos de los objetivos que se consideran elementales para la mejora del sistema en futuras versiones del programa:

7.1. Creación de una base de datos genérica

Uno de los principales problema del programa es que requiere de una base de datos de símbolos musicales con unas características determinadas que caractericen adecuadamente los símbolos. La creación de dicha base de datos suele ser, por normal general, una tarea un poco pesada y lo ideal sería disponer de una fase de entrenamiento por parte del sistema para obtener los datos de los símbolos que se utilizaran en la partitura, para el caso manuscrito. Para el no manuscrito, dicha fase de entrenamiento no supone una fase tan crítica dado que se puede tener una base genérica para todos los casos.

Además, se podría implementar la posibilidad de que el sistema se realimentara automáticamente con los símbolos clasificados correctamente. Esto permitiría el continuo desarrollo de la base de datos sin la intervención del usuario para adaptarse mejor a determinadas partituras nuevas. Sin embargo, esto puede provocar dos problemas. En primer lugar, un incremento de número de elementos de la base de datos conllevaría un incremento del tiempo de cómputo para comparar todos los símbolos al clasificar un elemento dado. En segundo lugar, no se puede realizar una validación de la clasificación de forma automática (de hecho, si se realiza la clasificación a una categoría es porque el algoritmo supone que es el resultado adecuado), por lo que sería necesario un operador humano que se encargase de aceptar o no las clasificaciones realizadas por el programa y, de igual forma, de corregir aquellos errores que se produjeran.

7.2. Reconocimiento de símbolos por componentes

En la implementación del proyecto realizado no se utiliza esta técnica, sino que se comparan las notas enteras, sin añadir variables que puedan dar problemas en el proceso global de reconocimiento de partitura.

Sin embargo, si se implementase y funciona adecuadamente, este método superaría con creces el desarrollado en el proyecto. Y es que permite reconocer los símbolos de una manera más parecida a como reconocemos las personas. Dada una nota, buscamos la cabeza para detectar el tono, buscamos el número de corchetes para determinar que tipo de nota de trata, si dispone de plica… La base de datos inicial de símbolos musicales se transformaría en una base de datos reducida de elementos básicos (con cabezas de notas, plicas, corchetes, números, etc.) que permitiría reconocer una gama más amplia de símbolos según las especificaciones que se implementen. Mediante esta técnica se podría solucionar el problema del no reconocimiento de símbolos unidos (corcheas, semicorcheas, fusas…) y mejorar las prestaciones del sistema.

7.3. Mejora del algoritmo de clasificación

Se ha implementado el algoritmo k-NN para la clasificación de los símbolos segmentados en la partitura. Dicho algoritmo constituye un sencillo procedimiento para clasificar los símbolos de una forma eficaz. No obstante, es posible que la utilización de otro algoritmo, o una modificación del realizado, pueda obtener mejores resultados.

Una posible modificación del algoritmo k-NN sería la asignación de pesos a cada uno de los descriptores de los símbolos para poder ajustar los parámetros convenientemente según el caso. Este procedimiento ya se empleaba, utilizando el algoritmo genético para calcular el valor de dichos pesos, en el trabajo realizado por Fujinaga (ver capitulo 2 del proyecto), creando un sistema OMR adaptativo.

Como alternativa, sería de gran interés trabajar en algoritmos con capacidad de aprendizaje, que ajustaran de una forma más detallada la morfología de un elemento cuantos más ejemplos dispongan. De entre dichos algoritmos se debe destacar la creación de una red neuronal para la clasificación.

7.4. Creación de método eficaz de eliminado de líneas de pentagrama

En el proyecto se han implementado dos métodos de borrado de líneas, pero no se ha conseguido un resultado óptimo general. El objetivo es que se puedan eliminar las líneas del pentagrama de cualquier partitura, sean dichas líneas del grosor que sean y en las condiciones en las que se encuentren.

Esta tarea es por lo general muy compleja y, salvo casos excepcionales, no se obtienen unos resultados satisfactorios para todos los casos. Otros autores [52] [53] han desarrollado multitud de algoritmos con el fin de buscar un método genérico de borrado de líneas de pentagrama, sin éxito. La mayor parte de ellos consiguen buenos resultados para determinadas partituras, pero no consiguen un método genérico.

Aparte del grosor de las líneas del pentagrama, debemos contar con que puede que, debido al proceso de conversión de la imagen a formato binario, éstas se rompan o pierdan la continuidad. Además, las líneas no tienen porqué estar completamente derechas, puede darse el caso de que existan deformaciones en la partitura debido al proceso de escaneo.

Y a todo esto debemos añadir que el algoritmo debería ser capaz de diferenciar elementos musicales de líneas de pentagrama, tarea muy difícil de realizar si ambos elementos se encuentran solapados.

Todo esto hace que dicho problema se considere un proyecto en sí mismo por lo que no se abarca de una forma muy extensa y se propone como trabajo futuro.

7.5. Mejora de la interfaz del usuario

Por cuestiones de tiempo para la finalización del proyecto no se pudo investigar la realización de interfaces gráficas mediante MATLAB. En su lugar, para facilitar el manejo del programa y la tarea del usuario se desarrolló una interfaz mediante línea de comandos, gráficos y cuadros de diálogos. Aunque se considera que dicha interfaz es perfectamente válida para la ejecución del programa, se podría desarrollar una distinta, completamente, gráfica utilizando las herramientas de MATLAB.

Este objetivo no mejoraría en absoluto el rendimiento del programa, ni aumentaría la eficacia del sistema OMR, pero permitiría que el usuario inexperto considerase la aplicación como una aplicación fácil de utilizar, de entorno amigable, incluso intuitivo. Y es que, hoy en día, la gran mayoría de las aplicaciones disponen de una interfaz gráfica y la ejecución de programas mediante línea de comandos suele ser motivo de pánico para ciertos usuarios.

7.6. Creación de un módulo de interacción con dispositivos de entrada

En el proyecto se ha dado por hecho que las partituras de entrada se han obtenido mediante la digitalización de la partitura física o mediante el dibujo de las notas en un sistema de entrada (tableta digitalizadora, *tablet PC*…). Sin embargo, en ningún momento se ha definido un método que obtenga, a través del programa principal, los datos de entrada.

Lo que se propone es una modificación de la arquitectura del programa que obtenga el símbolo mediante un dispositivo de entrada en lugar de mediante una imagen. Para ello, se debe tener en cuenta que MATLAB dispone de elementos para interactuar con los dispositivos periféricos.

Si el dispositivo a interconectar es un escáner, el proceso no variará significativamente ya que se obtendrá una imagen similar a la que se obtiene actualmente para ejecutar el programa. En el caso de un dispositivo de escritura por pantalla deberán realizarse algunos cambios para la detección del tono y el reconocimiento directo sin pasar por las etapas de extracción de pentagramas, notas, etc. Además, convendría implementar una interfaz que mostrara por pantalla un pentagrama donde se fueran escribiendo los símbolos que se realizasen sobre el dispositivo.

7.7. Traducción del código a Java

Este objetivo se plantea como una forma de reducción del presupuesto del proyecto. MATLAB es una aplicación propietaria cuya licencia es cara para un usuario común. Se escogió dicho programa porque dispone de un potente "*toolkit*" de procesado de imagen y porque permite realizar operaciones mediante línea de comandos sin necesidad de compilar código para probar posibles algoritmos a implementar.

Sin embargo, la realización del código a formato Java, lenguaje de código libre que permite ejecutar programas en cualquier plataforma, disminuiría el presupuesto del proyecto en una cantidad considerable al eliminar la licencia de MATLAB. Java también dispone de algunas librerías de procesado de imagen pero cuenta con el inconveniente de que debemos compilar todo el programa para ejecutarlo.

De traducir el programa, se deberían implementar todas las funciones a formato Java y desarrollar (o buscar en las librerías de Java) funciones que realicen las operaciones que ejecutaban determinados métodos del "*toolkit*" de MATLAB.

7.8. Implementación de otros módulos de traducción de formato

El bloque de traducción de formato se encarga de mostrar al usuario los datos extraídos de la partitura. Para ello, se requieren los datos simbólicos que extrae el programa con la información de la partitura. En este caso se ha planteado que lo más conveniente es representar la partitura de forma gráfica y acústica y para ello se ha utilizado la herramienta de representación musical *Lilypond*. No obstante, se podría haber utilizado cualquier otro método de representación.

Si se ha utilizado *Lilypond* ha sido por cuestiones de facilidad de manejo del código. Se podrían implementar otros módulos que tradujeran los datos simbólicos a otros formatos, como MusicXML. La representación de los datos puede ser muy variada, por lo que de no estar conformes con los resultados se puede desarrollar un módulo propio. Gracias al diseño modular, se puede sustituir o incorporar nuevos módulos para la representación de dicha información en notaciones alternativas. Para más información sobre los pasos a seguir y las variables a tener en cuenta se recomienda consultar el capítulo de manual de referencia y el capítulo que describe cada módulo del programa en profundidad.

8. Bibliografía

[1] **Adaptive Optical Music Recognition**
I. Fujinaga
Faculty of Music, McGill University
Fecha de publicación: Junio de 1996
Descripción: tesis doctoral que estudia la implementación de un sistema de reconocimiento óptico musical adaptativo

[2] **Automatic Recognition of Sheet Music**
D. Pruslin
Sc. D. Thesis, MIT
Fecha de publicación: 1966
Descripción: uno de los primeros trabajos sobre el reconocimiento de partituras mediante sistemas OMR

[3] **Computer Pattern Recognition of Standard Engraved Music Notation**
D.S. Prerau
Ph. D. Thesis, MIT
Fecha de publicación: 1970
Descripción: uno de los primeros trabajos sobre el reconocimiento de partituras

[4] **Computing in Musicology**
URL: http://www.ccarh.org/publications/books/cm/
Fecha de revisión: 2 de Junio de 2009
Descripción: números de la revista "Computing in Musicology" para descargar en formato PDF

[5] **Automated High Speed Recognition of Printed Music (WABOT – 2 Vision System)**
T. Matsushima
Proceeding of the 1985 International Conference on Advance Robotics, pp 477-82, Japan Industrial Robot Asociation (JIRA)
Fecha de publicación: 1985
Descripción: desarrollo del sistema de lectura automática de partituras WABOT-2

[6] **WABOT – 2**
URL: http://es.youtube.com/watch?v=ZHMQuo_DsNU
Fecha de revisión: 2 de Junio de 2009
Descripción: video del robot Wabot-2 en funcionamiento

[7] **Automatic printed – music – to – braile translation system**
T. Matsushima
Journal of Information Processing 11(4): 249-57
Fecha de publicación: 1988
Descripción: reconocedor con capacidad de traducir partituras a braile

[8] **Input method of [musical] note and realization of folk music data-base**
Y. Nakamura, M. Shindo y S. Inokuchi
Institute of Electronics and Communications Engineers of Japan (IECE) TG PRL78-73: 41-50
Fecha de publicación: 1978
Descripción: desarrollo de un sistema de lectura de partituras para crear una base de datos con canciones del folclore japonés

[9] **Automatic recognition of music score**
H.Aoyama y A.Tojo
Electronic Image Conference Journal. 11(5): 427-35
Fecha de publicación: 1982
Descripción: sistema de lectura de partituras para crear bases de datos

[10] **Non-physics measurements on the PEPR System: Seismograms and music scores**
G.E. Wittlich
Report to the Oxford Conference on Computer Scanning. Oxford
Nuclear Physics Laboratory: 487-9
Fecha de publicación: 1974

[11] **On automatic pattern recognition and acquisition of printed music**
A. Andronico y A. Ciampa
Proceedings of the International Computer Music Conference. 245-78.
Fecha de publicación: 1982

[12] **SYMFONI: System for note coding**
S. Tonnesland
Institute of Informatics
Fecha de publicación: 1986

[13] **Towards computer recognition of the printed musical store**
N.G. Martin
B. Sc. project report. Thames Polytechnic, London
Fecha de publicación: 1987

[14] **Using domain knowledge in low-level visual processing to interpret handwritten music: An experiment**
J.W. Roach y J.E. Tatum
Pattern Recognition. 21(1): 333-44
Fecha de publicación: 1988

[15] **An essay toward specification of a music-reading machine**
M. Kassler
Musicology and the computer, ed. B.S. Brook. New York: The City University of
New York Press. 151-75.
Fecha de publicación: 1970
Descripción: definición de los componentes básicos de un sistema OMR y de lo elementos que debería ser capaz de reconocer la máquina

[16] **Optical character recognition of printed music: A review of two dissertations**
M. Kassler
Perspectives of New Music. 11: 250-4
Fecha de publicación: 1972
Descripción: documento en el que se discute sobre los trabajos realizados por Pruslin y Preau en el campo del reconocimiento OMR

[17] **Recognition of music using the special image-input-device enabling to scan the staff of music as the supporting system for the blind**
K. Maenaka y Y. Tadokoro
PRL83-60. 37-45
Fecha de publicación: 1983
Descripción: creación de un sistema OMR que utiliza una cámara de TV como dispositivo de entrada

[18] **Recognition system for a printed music store**
W.J Kim, M.J. Chung, y Z. Bien
Proceedings of TENCON 87: 1987 IEEE Region 10 Conference 'Computers and Communications Technology Toward 2000.' 1380: 573-7
Fecha de publicación: 1987
Descripción: sistema de lectura de partituras que utiliza una cámara de TV como dispositivo de entrada y un robot que interpreta las melodías

[19] **The acquisition, representation and reconstruction of printed music by computer: A survey**
N.P. Carter, R.A. Bacon, y T. Messenger
Computers and the Humanities. 22: 117-36
Fecha de publicación: 1988
Descripción: documento en el que se estudian algunos puntos de los sistemas OMR

[20] **Automatic recognition of printed music in the context of electronic publishing**
N. P. Carter, G.R.S.M. (Hons.), L.R.A.M.
Depts. of Physics and Music, University of Surrey
URL: http://www.npcimaging.com/thesis/thesis.html
Fecha de publicación: Febrero de 1989
Descripción: tesis doctoral sobre el reconocimiento automático de música impresa

[21] **The optical scanning of medieval music**
W.F. McGee y P. Merkley
Computers and the Humanities. 25(1): 47-53
Fecha de publicación: 1991
Descripción: sistema de lectura de partituras de notación medieval (notas cuadradas)

[22] **Symbol recognition for printed piano scores based on the musical knowledge**
H. Miyao, T. Ejima, M. Miyahara y K. Kotani
Transactions of the Institute of Electronics, Information and Communication Engineers D-II, J75D-II(11): 1848-55
Fecha de publicación: 1992
Descripción: sistema OMR que incluye una gramática de notación musical para ayudar al reconocimiento

[23] **MUSER-a prototype musical recognition system using mathematical morphology**
B.R. Modayur, V. Ramesh, R.M. Haralick y L.G. Shapiro
Intelligent Systems Laboratory, EE Dept, FT-IO. University of Washington
Fecha de publicación: 1992
Descripción: sistema de un sistema OMR

[24] **An approach to recognition of printed music**
M. Roth
Technical Report 210. Department of Computer Science, Swiss Federal Institute of Technology
Fecha de publicación: 1994
Descripción: sistema de lectura de partituras cuya salida se utiliza de forma conjunta con el editor de notación musical Lipsia

[25] **Exemplar-based learning in adaptive optical music recognition system**
I. Fujinaga
Conservatorio de Música Peabody, Universidad Johns Hopkins
Fecha de publicación: 1996
Descripción: descripción del proceso de aprendizaje de un sistema AOMR

[26] **An extensible Optical Music Recognition system**
D. Bainbridge, T. Bell
Nineteenth Australasian Computer Science Conference
URL: http://www.cs.waikato.ac.nz/~davidb/publications/acsc96/
Fecha de publicación: Septiembre de 1997
Descripción: proyecto que implementa un sistema OMR extensible

[27] **Fast capture of sheet music for an agile digital music library**
R. Lobb, T. Bell, D. Bainbridge
URL: http://ismir2005.ismir.net/proceedings/1018.pdf
Fecha de publicación: 2005
Descripción: implementación de un sistema OMR rápido que identifique partituras capturadas mediante una cámara digital o un escáner barato.

[28] **Optical Music Recognition: a Generalised Approach**
D. Bainbridge
Departmen of Computer Science, University of canterbury, Christchurch
URL: http://www.cs.waikato.ac.nz/~davidb/publications/nzcsgc96/
Fecha de publicación: Septiembre de 1997
Descripción: estudio sobre los sistemas OMR y descripción del lenguaje Primera

[29] **Preliminary experiments in musical score recognition**
D. Bainbridge
B.Sc. Thesis. University of Edinburgh
Fecha de publicación: 1991
Descripción: tesis doctoral donde se puede encontrar la definición de las técnicas "slicing" propuestas por el autor en otros documentos

[30] **Optical Music Sheet Segmentation**
P. Bellini, I. Bruno, P. Nesi
Dep. of Systems and Informatic University of Florence
Fecha de publicación: Noviembre de 2001
Descripción: investigación que define el módulo O^3MR

[31] **Musitek**
URL: http://www.musitek.com/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial de la empresa Musitek

[32] **Neuratron**
URL: http://www.neuratron.com/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial de la empresa Neuratron

[33] **Acorn Computers**
URL: http://en.wikipedia.org/wiki/Acorn_Computers
Última actualización: 28 de Febrero de 2009
Descripción: artículo de la Wikipedia sobre antigua empresa Acorn

[34] **Acorn Computers Mobile Computing**
URL: http://acorncomputers.co.uk/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial de la nueva empresa Acorn; ahora fabricante y vendedor de ordenadores portátiles

[35] **Capella – Scan**
URL: http://www.capella-software.com/capscan.htm
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial del programa Capella - Scan

[36] **SharpEye**
URL: http://www.visiv.co.uk/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial del programa SharpEye

[37] **PDFtoMusic PRO**
URL: http://www.myriad-online.com/en/products/pdftomusicpro.htm
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial del programa PDFtoMusic PRO

[38] **VivaldiStudio**
URL: http://www.vivaldistudio.com/Eng/VivaldiScan.asp
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial del programa VivaldiScan

[39] **OMeR**
URL: http://www.myriad-online.com/en/products/omer.htm
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web oficial del programa OMeR

[40] **Gamera**
URL: http://ldp.library.jhu.edu/projects/gamera/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web del proyecto Gamera

[41] **Audiveris**
URL: https://audiveris.dev.java.net/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio web del programa Audiveris

[42] **OpenOMR**
URL: http://sourceforge.net/projects/openomr/
Fecha de revisión: 2 de Junio de 2009
Descripción: página de descarga del programa OpenOMR

[43] **A Threshold Selection Method from Gray-Level Histograms**
N. Otsu
IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, No. 1, pag. 62-66
Fecha de publicación: 1979
Descripción: ensayo que describe un método para seleccionar un umbral de forma automática para convertir las imágenes a binario

[44] **Connected Component Labeling - Wikipedia, the free enciclopedia**
URL: http://en.wikipedia.org/wiki/Connected_Component_Labeling
Última actualización: 15 de Abril de 2009
Descripción: definición del método de etiquetado por componentes

[45] **K-Nearest Neighbor algorithm – Wikipedia, the free enciclopedia**
URL: http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm
Última actualización: 26 de Mayo de 2009
Descripción: definición del algoritmo k-NN

[46] **Discriminatory Analisys – Nonparametric Discrimination: Consistency Properties**
E. Fix y J.L. Hodgens, Jr
University of California, Berkeley. Report nº4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph, Texas
Fecha de publicación: Febrero de 1951
Descripción: documento original donde se define por primera vez el algoritmo de clasificación k-NN

[47] **LilyPond**
URL: http://lilypond.org/web/
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio oficial del programa LilyPond

[48] **MusiXTEX**
URL: http://icking-music-archive.org/software/indexmt6.html
Fecha de revisión: 2 de Junio de 2009
Descripción: sitio oficial del programa MusiXTEX

[49] **Assessing Optical Music Recognition Tools**
P. Bellini, I. Bruno y P. Nesi
Computer Music Journal Spring 2007, Vol. 31, Nº1: 68-93
Fecha de publicación: Verano 2007
Descripción: documento en el que se realiza un estudio sobre la forma de evaluar los distintos sistemas OMR de forma general

[50] **Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT)**
URL: http://www.coitt.es/
Fecha de revisión: 02 de Junio de 2009
Descripción: web oficial del COITT

[51] **Honorarios profesionales**
URL: http://www.coitt.es/res/libredocs/Honorarios.pdf
Fecha de revisión: 02 de Junio de 2009
Descripción: comunicado del COITT sobre los honorarios a percibir de los ingenieros técnicos de telecomunicación en el ejercicio libre de la profesión

[52] **A Comparative Study of Staff Removal Algorithms**
C. Dalitz, M. Droettboom, B. Pranzas y I.Fujinaga
IEEE TPAMI, VOL. 30, NO. 5
Fecha de publicación: Mayo de 2008
Descripción: estudio sobre la eliminación de las líneas del pentagrama en los sistemas OMR

[53] **Staff line detection and removal with stable paths**
A. Capela, A. Rebelo, J.S. Cardoso y C. Guedes
SIGMAP 2008 - International Conference on Signal Processing and Multimedia Applications
Fecha de publicación: 2008
Descripción: estudio sobre la eliminación de las líneas del pentagrama en los sistemas OMR

[54] **Documentación de LilyPond versión 2.12.2**
URL: http://lilypond.org/doc/v2.12/Documentation/
Última Actualización: 20 de Enero de 2009
Descripción: documentos y manuales oficiales para aprender a utilizar el lenguaje Lilypond

[55] **Reconocimiento por regiones de caracteres manuscritos**
N. Rangel, I. Delgado, B. Zúñiga
Facultad de Ingeniería, UNAM (Universidad Nacional Autónoma de México)
URL: http://www.cio.mx/3_enc_mujer/files/extensos/Sesion%204/S4-ING12.doc
Fecha de revisión: 2 de Junio de 2009
Descripción: breve descripción de un algoritmo de reconocimiento por regiones de caracteres manuscritos

[56] **Music Notation – Optical Music Recognition (OMR)**
G. Castan
URL: http://www.music-notation.info/en/compmus/omr.html
Fecha de revisión: 2 de Junio de 2009
Descripción: página web que recoge gran parte del software disponible hoy en día para reconocimiento de música impresa

[57] **Reading Sheet Music**
A.F. Desaedeleer
University of London
Fecha de publicación: Septiembre de 2006
Descripción: proyecto que estudia la implementación de un algoritmo OMR

[58] **Tratamiento digital de imágenes**
R.C. González, R. Word
Fecha de publicación: 1996
Descripción: libro que trata distintos métodos de tratamiento digital de imágenes existentes

[59] **IMAGine – Cursos Interactivos de Tratamiento Digital de Imagen**
J. Cid, Universidad Carlos III de Madrid
URL: http://www.tsc.uc3m.es/imagine/index.html
Última actualización: 2007
Descripción: web que recoge cursos sobre procesado digital de la imagen

[60] **Extracción de líneas melódicas a partir de imágenes de partituras musicales**
A. Sánchez, J.J. Pantrigo y J.I. Pérez
Universidad Rey Juan Carlos de Madrid
Fecha de publicación: 2006
Descripción: desarrollo de una herramienta automática para la segmentación de partituras musicales no manuscritas.

[61] **Sistema reconocedor de partituras musicales**
F.A. Zenteno
Universidad de las Américas Puebla, Escuela de Ingeniería, departamento de Ingeniería en Sistemas Computacionales
Fecha de publicación: 2003
Descripción: tesis que estudia e implementa distintas mejoras para el sistema reconocedor de partituras musicales MIDIWORD

9. Anexo A. Guía rápida de representación en Lilypond

Este apartado intentará servir de referencia para aquellos que deseen programar en el lenguaje Lilypond para representar partituras. La guía intentará explicar de forma breve y concisa los elementos más importantes a tener en cuenta a la hora de representar partituras básicas.

Toda la información se ha extraído de la documentación oficial de *Lilypond* [54]. Ahí se encontrara un manual de aprendizaje, un glosario musical, referencia de la notación y del funcionamiento, recursos para desarrolladores, multitud de ejemplos, etc.

9.1. Estructura general

El modelo general que sigue la mayor parte de los ficheros codificados en formato *Lilypond* es el siguiente:

```
\version “2.12.2”
\header { … }
\score { … }
```

El primer elemento indica la versión que se ha utilizado para codificar el fichero, se utiliza para disponer compatibilidad en futuras versiones. El segundo conforma la cabecera de la partitura, donde podemos incluir determinados datos de la obra. En la tercera se incluyen los datos codificados de la partitura.

Algunos de los elementos que podemos incluir en la segunda cabecera son:

```
\title = “Titulo de la obra”
\subtitle = “Subtitulo de la obra”
\composer = “Autor”
\opus = “opus”
```

En la tercera cabecera podemos incluir una estructura para generar un archivo PDF y un MIDI. Dicho fichero MIDI puede ser configurado para definir la velocidad. Según la estructura propuesta, esta configurado en una velocidad de 90 negras (4) por minuto:

```
\layout {
        \midi {…}
        \context {
                    \Score
                    tempoWholesPerMinute = #(ly:make-moment 90 4)
        }
}
```

9.2. Elementos de la cabecera “score”

Se puede obviar la estructura propuesta anteriormente y escribir la partitura sin ningún elemento más. De realizar esto, se generará una partitura en clave de Sol y tiempo de compasillo.

La cabecera "**score**" puede estar formada por un pentagrama o varios. Además, se pueden incorporar letras y cifrado de acordes (aunque dichas tareas no se incluirán en esta guía rápida). Las notas se representan según el sistema de notación musical inglesa, como se define a continuación:

Nota	Do	Re	Mi	Fa	Sol	La	Si
Notación inglesa	C	D	E	F	G	A	B

Figura 92 - Representación de las notas en el sistema de notación musical inglesa.

También podemos representar silencios, utilizando "r" acompañado de la duración, como se explicará posteriormente. También podemos utilizar "s" para un silencio separador (constituyendo un espacio en el pentagrama)

Podemos representar las notas de dos formas distintas, mediante modo absoluto o modo relativo. En modo absoluto definimos las notas en relación al Do absoluto. En el otro modo, se definen las notas tomando como referencia una proporcionada de antemano. Las notas cuyo tono se encuentren más cerca de la proporcionada serán las definidas.

Para crear un pentagrama, basta con escribir una expresión musical encerrada entre corchetes. Si se desea trabajar con el modo relativo (más fácil de usar de cara a programar manualmente) se debe utilizar la expresión "**\relative**" como se indica a continuación:

```
\relative c'
{
        c d e f
}
```

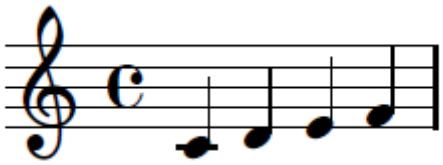


Figura 93 - Ejemplo 1. Modo relativo

En este caso se generan las notas en modo relativo utilizando como referencia el do central (c'). Si se utilizara el modo absoluto se debería escribir:

```
{
        c' d' e' f'
}
```

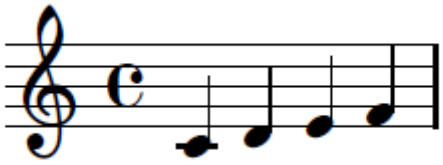


Figura 94 - Ejemplo 2. Modo absoluto

El modo absoluto es muy útil si se quiere crear la partitura de forma automática mediante un programa. Como se puede observar, las notas van acompañadas de un apóstrofe que indica que la nota se ha aumentado en una octava su tono. Se puede subir el tono tantas octavas como apóstrofes se incluyan. De igual forma, si se utiliza una coma se podría disminuir el tono en una octava. En el siguiente ejemplo se pueden apreciar algunas de las octavas de Do:

```
{
        c, c c' c''
}
```

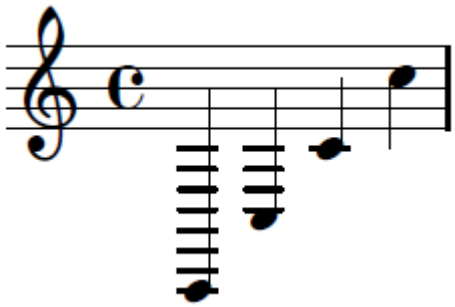


Figura 95 - Ejemplo 3. Octavas de Do

Se puede ver que de forma automática las duraciones de la notas toman el valor de una negra (o de la última nota que tenga valor). Este parámetro se puede variar añadiendo una de las duraciones definidas en el programa para cada nota al final del nombre, después de cualquier signo de aumento/disminución de octava, si lo hubiera:

Nombre	Redonda	Blanca	Negra	Corchea	Semicorchea	Fusa	Semifusa
Valor	1	2	4	8	16	32	64

Figura 96 - Relación del valor de las notas con respecto a la duración

```
{
 c'1 c'2 c'4 c'8
 c'16 c'32 c'64
}
```

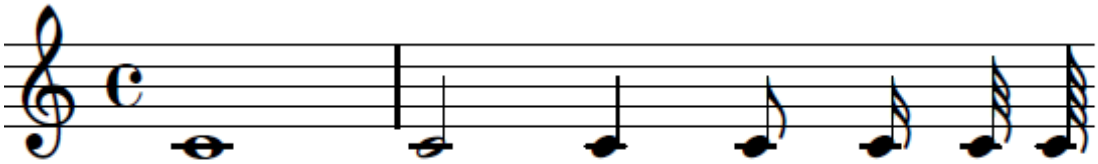


Figura 97 - Ejemplo 4. Duraciones de las notas

Como se puede comprobar, *Lilypond* analiza el tiempo de las notas y coloca barras de compases en aquellos puntos donde finalice un compás. Esta opción se puede deshabilitar (consultar manual oficial). Además, también se pueden poner barras “extra” de separación de pentagrama si así lo deseáramos (aunque no tenga mucho sentido musicalmente), como se estudiará más adelante.

Según la estructura desarrollada, se pueden añadir más elementos a las notas como se indica en la tabla siguiente:

Nombre	Código	Posición	Ejemplo
Bemol	`es`	Colocar después del nombre de la nota, pero antes del signo de aumento/disminución de octava	`ces'1`
Doble bemol	`eses`	Igual que el anterior	`ceses'1`
Sostenido	`is`	Igual que el anterior	`cis'1`
Doble sostenido	`isis`	Igual que el anterior	`cisis'1`
Medio sostenido	`ih`	Igual que el anterior	`cih'1`
Medio bemol	`eh`	Igual que el anterior	`ceh'1`
Sostenido y medio	`isih`	Igual que el anterior	`cisih'1`
Bemol y medio	`eseh`	Igual que el anterior	`ceseh'1`
Puntillo	`.`	Colocar después del signo de la duración de la nota (o después del nombre si no hay tal símbolo)	`c'1.`
Ligadura de unión	`~`	Colocar entre las dos notas consecutivas del mismo tono a ligar	`c'1~ c'2`
Ligadura de expresión	`( , )`	Ambos símbolos se colocan después de la duración de las notas que unen. Se puede formar con cualquier tono y cualquier posición. No se puede realizar una ligadura de expresión dentro de otra (aunque sí otra de otro tipo)	`c'1(  d'2 e'2)`
Fraseo	`\( , \)`	Igual que el anterior	`c'1\(  d'2 e'2\)`

Figura 98 - Algunos símbolos que se pueden añadir a las notas

Si se quiere crear acordes, basta con encerrar las notas mediante signos de mayor y menor que, como se explica a continuación:

```
{
 <b' c'>8 d'2 b'4

}
```

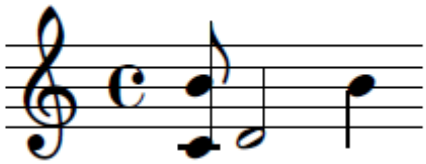


Figura 99 - Ejemplo 5. Creación de acordes

La duración del acorde se incluye después del último signo. De igual forma, se pueden crear polifonías, utilizando la expresión << {voz 1} \\ {voz 2} >>

```
{
      <<{c''4 d''2 e''8 f''8}
      \\
      {f'4 e'4 d'4 c'4}>>
}
```

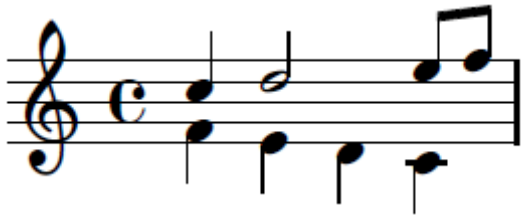


Figura 100 - Ejemplo 6. Creación de polifonías

En el caso de que se quieran tener varios pentagramas simultáneamente, se debe utilizar la estructura **"\new Staff**" que genera un nuevo pentagrama. Este comando se ejecuta de forma automática cada vez que escribimos notas aisladas. Para unirlos, se deben encerrarlos entre las expresiones << y >> al igual que en el ejemplo anterior:

```
{
<<
\new Staff
      {
      c'4 d'2 e'8 e'8
      a'4 g'2 d'8 e'8
      }

      \new Staff
      {
      f'2 g'4 f'8 e'8
      }
>>
}
```

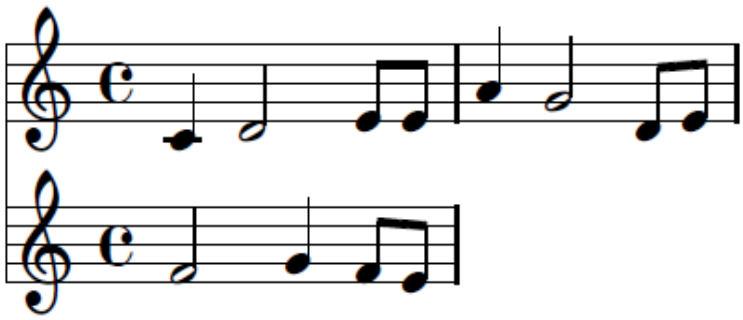


Figura 101 - Ejemplo 7. Creación de varios pentagramas

En el caso de no encerrar las expresiones obtendremos el siguiente resultado:

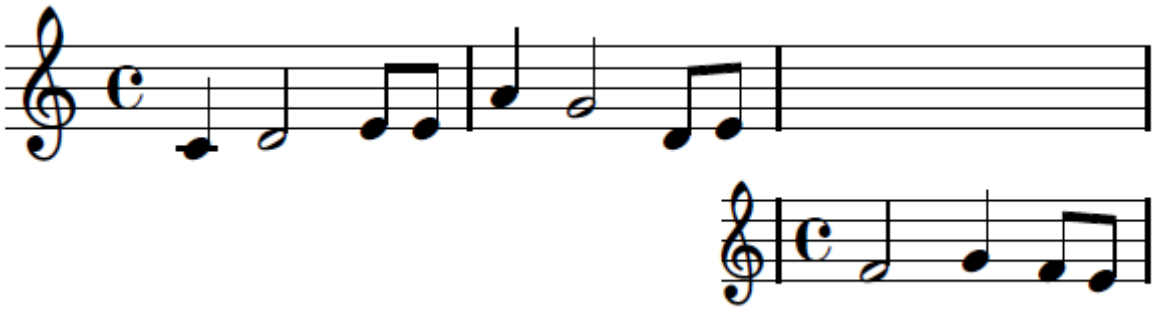


Figura 102 - Ejemplo 8. Creación de varios pentagramas de forma incorrecta

También se pueden crear partituras para piano. En ese caso se debe utilizar la expresión **"\new PianoStaff**":

```
{
\new PianoStaff <<
      \new Staff
      {\clef treble
       c'4 d'2 e'8 e'8
       a'4 g'2 d'8 e'8
      }
      \new Staff
      {\clef bass
       f2 g4 f8 e8
       f2 g4 f8 e8
      } >> }
```

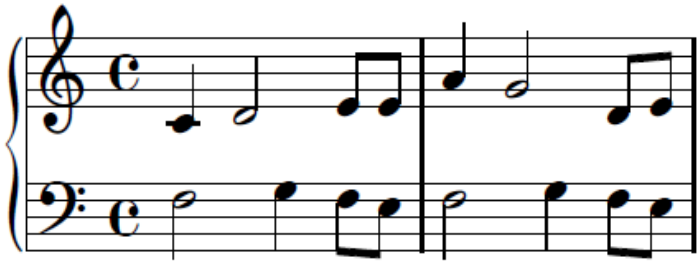


Figura 103 - Ejemplo 9. Creación partituras para piano

Para asegurar que se genera la partitura correctamente, se han incluido las etiquetas **"\clef bass**" y "**\clef treble**" que definen la clave correcta. El código "**clef**" define que se va a incluir una clave y el valor posterior define la clave. Algunos de esos valores son "**treble**" (clave de Sol), "**bass**" (clave de Fa), "**alto**" (clave de Do en tercera línea) y "**tenor**" (clave de Do en cuarta línea)

Si se desea definir el tiempo del pentagrama basta con escribir "**\time x/y**" dentro de la estructura que define el pentagrama, siendo "x" el numerador del tiempo e "y" el denominador:

```
{
      \clef bass
      \time 3/4
            ces4 d8 e8 f4
}
```

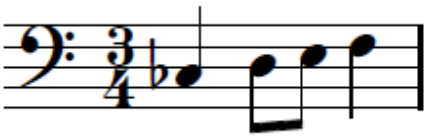


Figura 104 - Ejemplo 10. Definición del tiempo del pentagrama

Algunos de los otros elementos que se pueden incluir en la partitura se incluyen a continuación:

- **\breathe** = respiro
- **\partial N** = anacrusa de duración N
- **\times 2/3 {notas}** = estructura para crear tresillos, donde "notas" conforman las notas del tresillo.
- **\bar "|"** = barra divisoria
- **\bar "|."** = barra de final de pentagrama
- **\bar "|:"** = inicio de barra de repetición
- **\bar ":|"** = final de barra de repetición

Otro elemento interesante a tener en cuenta es que *Lilypond* genera automáticamente las uniones de las notas de corcheas, semicorcheas, etc. Se puede desactivar esta tarea mediante la utilización del código **"\autoBeamOff**" y unir las notas deseadas mediante corchetes:

```
{
      \autoBeamOff
      ces'4 d'8[ e'8 f'8]
      g'8 d'8[ e'8]
}
```

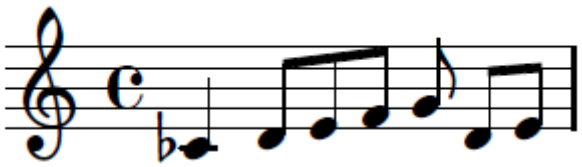


Figura 105 - Ejemplo 11. Desactivación de la unión automática de elementos

10. Anexo B. Manual de usuario

En este apartado se explicará brevemente el funcionamiento y manejo del sistema creado para que cualquier usuario pueda ejecutar la aplicación de forma correcta.

El reconocedor de notas musicales descrito en este proyecto se encarga de extraer los datos de una partitura, analizarlos y representarlos por pantalla y altavoces, mediante la utilización del lenguaje de representación *Lilypond*. Es necesario disponer de dicho programa de distribución gratuita para la obtención de estos elementos, por lo que si no se dispone de él se aconseja consultar el apartado "*Instalar Lilypond*".

10.1. Obtención de la partitura

El programa necesita la imagen de una partitura o pentagrama para poder funcionar. Dicha imagen puede obtenerse de muchas formas: escaneando partituras, imprimiendo sobre una imagen una partitura digitalizada, creándola desde el mismo equipo… Puede ser tanto manuscrita como no manuscrita y del tamaño que queramos.

Se debe tener en cuenta que cuanto más grande sea la imagen de entrada, mejor resultados se podrán tener por el menor efecto del ruido (siempre y cuando la base de datos sea la adecuada), sin embargo, también se debe tener en cuenta que una imagen de tamaño grande conlleva más operaciones de cómputo y por lo tanto más tiempo de ejecución del programa, por lo que para obtener unos buenos resultados de forma rápida se recomienda escoger una imagen de resolución media, entre 72 y 200 dpp.

De igual forma, se debe recordar que una partitura pequeña es muy probable que tenga unos resultados bastante malos.

No se pueden utilizar partituras escaneadas de grosor desconocido. Únicamente funcionarán aquellas cuyas líneas del pentagrama sean de exactamente un píxel de grosor. Este problema se pretende solventar en versiones posteriores.

Para el escaneado de partituras se recomienda, aparte de la resolución de imagen descrita, el no utilizar hojas muy deterioradas en el que los símbolos puedan aparecer rotos o borrosos. Esto puede provocar que el programa reconozca de forma errónea los símbolos. También se sugiere no utilizar partituras cuyo fondo sea distinto al blanco.

Otra opción para adquirir partituras es mediante la búsqueda en Internet o la copia desde un documento, convirtiéndolo a formato de imagen posteriormente. Debemos recordar que la función del programa no es únicamente la de representar partituras, sino la de permitir la reproducción de ellas. Es muy difícil imaginarse como es la melodía de una partitura sólo con verla. Podemos utilizar el programa para reproducir las partituras que nos encontremos.

En cuanto a las partituras manuscritas, las recomendaciones son las mismas para el escaneado. Sin embargo, se ha desarrollado un algoritmo mediante el cual se eliminan de forma muy sencilla los pentagramas que tengan el grosor que tengan siempre que se pinten en azul. Por ello se recomienda la realización de dichos pentagramas de forma digital, en el ordenador, dibujando las líneas azules y las notas negras.

Se debe destacar que este programa está enfocado únicamente a la extracción de notas musicales, desechando cualquier otro elemento (titulo de la obra, letras, signos de expresión…) y cualquier otro tipo de imagen que no sea partituras o pentagramas musicales. Además, se centra en el uso de partituras de música moderna (occidentales) no habiéndose comprobado en lenguajes alternativos (notaciones antiguas, etc.)

Los formatos permitidos para cargar la imagen son: BMP, PNG y TIFF. No se asegura el correcto funcionamiento del programa con otros formatos de imagen que no sean esos, ya que, para la mayor parte de los casos, es muy probable que hubiera que realizar una conversión de formato.

Otro punto que debemos tener en cuenta es que el programa cuenta con determinadas restricciones debido al poco grado de madurez del programa (aún no permite la utilización de partituras de piano, varias voces, acordes, uniones de notas…). La lista de dichas restricciones se puede consultar en el apartado dedicado a la arquitectura de la aplicación y se intentarán solventar en futuras versiones.

10.2. Ejecución del programa

Una vez abierto MATLAB debemos definir la carpeta que contenga los módulos realizados para ejecutar el programa principal. Una vez realizado, desde la línea de comandos escribimos:

```
>>
>> programaPFC('tipoEliminaLineas',vecinos);
```

Donde "**`tipoEliminaLineas`**" puede tomar el valor "*normal*" para la utilización de partituras cuyo grosor de línea del pentagrama sea de un píxel o "azul" para aquellos pentagramas con las líneas azules. La variable "*vecinos*" puede tomar cualquier valor entero distinto de cero e indica el número de elementos más parecidos de la base de datos que se utilizarán para determinar las clases a las que pertenecen cada símbolo.

Es muy importante que el parámetro que indica el tipo de pentagrama que se va a usar en la partitura sea el correcto. Si elegimos un tipo y luego cargamos una imagen de otro tipo, el programa no funcionará correctamente.

Al ejecutar el programa se mostrará por línea de comandos un mensaje indicándonos que se ha iniciado el programa y que se requiere una partitura o pentagrama para comenzar el análisis. Además, se abre un cuadro de diálogo como el de la Figura 106 para que seleccionemos dicha partitura (se insiste que debe ser del mismo tipo definido al ejecutar el programa)

Durante la ejecución del programa se mostrarán, por línea de comandos, mensajes que nos proporcionarán todo tipo de información e indicarán en cada momento el módulo que se va a ejecutar. Además, se establecen pausas para que el usuario pueda visualizar los resultados, reanudando la ejecución al pulsar cualquier tecla. Los resultados que se muestren por pantalla se borrarán según pasemos a la ejecución de módulos superiores por lo que no hace falta que cerremos ventanas manualmente una y otra vez. Para detener la ejecución del programa en cualquier momento debemos pulsar Ctrl + C.

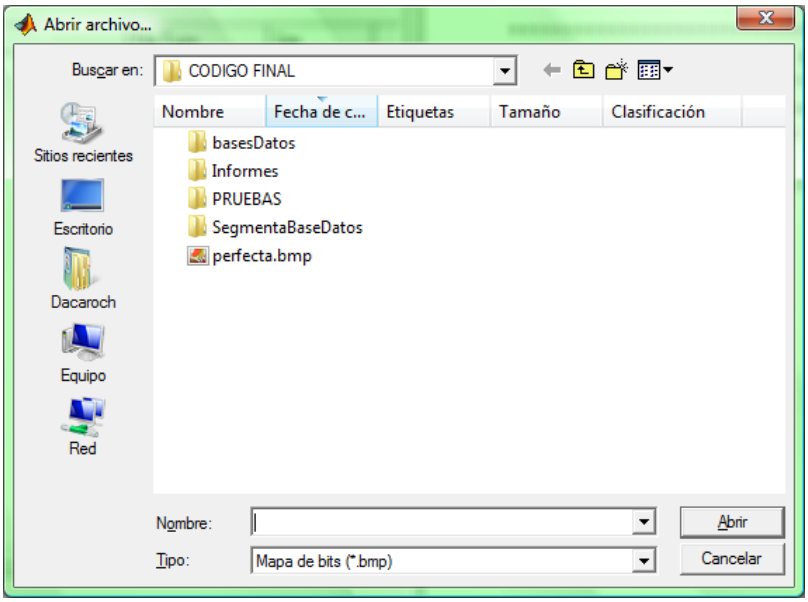


Figura 106 - Cuadro de diálogo para cargar el pentagrama

En la carpeta "PRUEBAS" existen algunos ejemplos de partituras organizadas según sean manuscritas o no para su carga. Si en lugar de éstas queremos utilizar otra, únicamente tendremos que seleccionar el archivo y cargarlo.

Una vez cargado el fichero aparecerá por línea de comandos un mensaje similar al siguiente y se abrirá un nuevo cuadro de diálogo:

```
Cargado:
D:\PFC\Pruebas MATLAB\CÓDIGO\CODIGO
FINAL\PRUEBAS\Manuscritos\Pentagramas_Raquel\1\p1.bmp

 => Por favor, selecciona una base de datos...
```

Se debe seleccionar el archivo de extensión "*mat*" que contenga los datos almacenados de la base de datos que se desee utilizar. Para ello, se puede cargar una de las bases de datos definidas como ejemplos (dentro de la carpeta "*basesDatos*") o una que hayamos creado nosotros mismos mediante el módulo "*cargaBD*" (ver manual de referencia para más información). Una vez realizado este paso aparecerá por pantalla un mensaje similar al siguiente:

```
Base de datos cargada correctamente:
D:\PFC\Pruebas MATLAB\CÓDIGO\CODIGO
FINAL\basesDatos\baseDatosManuscritos_Raquel.mat


 => PREPROCESADO DE LA IMAGEN Y CORRECCIÓN DE LA INCLINACIÓN
A continuación se realizará un preprocesado sobre la imagen para
ajustar algunos parámetros y determinar la inclinación
de la misma para corregirla.
(Para salir en cualquier momento pulse Ctrl + C)

    Pulse cualquier tecla para continuar...
```

Si todos los datos están bien, se mostrará la imagen cargada y comenzará el trabajo del programa con el preprocesado de la imagen y la corrección de la inclinación. Para más

información sobre alguno de los módulos que se ejecuten a continuación, se debe consultar el apartado de descripción a alto nivel de cada módulo

Una vez se pulse cualquier tecla, debe aparecer por pantalla la imagen original preprocesada y la imagen corregida sin inclinación. Por línea de comandos se mostrará un aviso con el ángulo que se ha utilizado para rotar la imagen y corregir la inclinación. En caso de no detectar inclinación, no se modifica la imagen.

El siguiente paso, extracción de pentagramas, obtendrá todos los elementos de interés de la partitura, tanto pentagramas como otros elementos. Por cada elemento se mostrará una ventana mostrando dicho elemento. Además, se mostrará otra ventana con las proyecciones horizontales y verticales de la partitura original.

Pulsando cualquier tecla se cerrarán las ventanas y se ejecutará la siguiente etapa, verificación de pentagramas, encargada de determinar si los elementos anteriormente extraídos son realmente o no pentagramas. Para ello, comprobará si cada elemento cuenta con cinco picos en la proyección vertical, desechando los que no cuenten con ello. Se mostrará por pantalla la proyección de cada elemento (Figura 107), así como los pentagramas reales encontrados por separado.

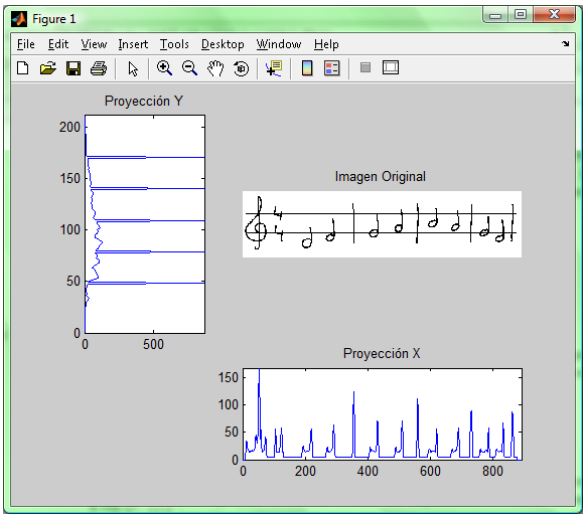


Figura 107 - Proyecciones X e Y de un pentagrama

En caso de no encontrar ningún pentagrama se mostrará el siguiente error por línea de comandos y se detendrá la ejecución del programa:

```
ERROR: No se han encontrado pentagramas válidos
```

El siguiente paso consiste en recorrer los pentagramas encontrados, para realizar una serie de análisis sobre cada uno que determinará los símbolos existentes y almacenará los datos necesarios para su representación.

Sobre línea de comandos se mostrará el número del pentagrama se que va a analizar y se procederá a la tarea. La primera etapa vuelve a realizar un preprocesado sobre el pentagrama en cuestión para extraerlo de la imagen original con menos ruido que la etapa de preprocesado anterior. La siguiente, se encargará de extraer los símbolos del pentagrama con líneas de

pentagrama incluidas. Cada símbolo se mostrará por pantalla en una ventana distinta (ver Figura 108)

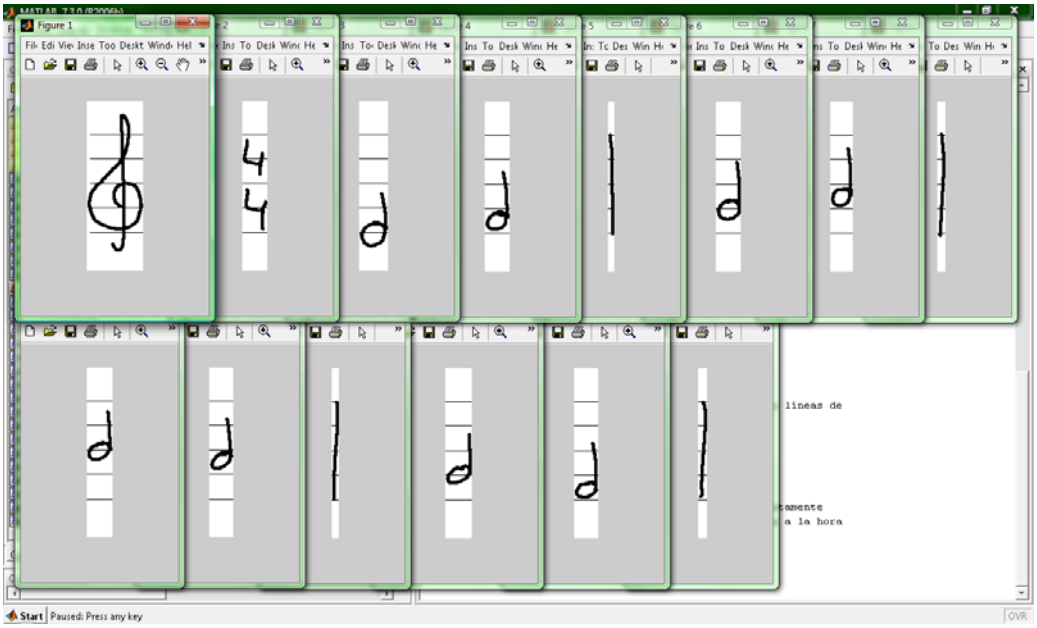


Figura 108 - Símbolos extraídos de un pentagrama

La siguiente etapa se encargará de eliminar las líneas del pentagrama para aislar el símbolo, mostrando los resultados en ventanas independientes. Después se volverá a realizar una segunda segmentación para buscar símbolos que se hayan podido quedar unidos en el mismo elemento.

Finalmente se pasa a la clasificación de los símbolos. Esta es la etapa encargada de reconocer los símbolos utilizando la base de datos, por lo que es de gran importancia que hayamos elegido una base de datos acorde a la partitura para aumentar la eficacia del reconocedor. Una vez ejecutemos dicha etapa, por línea de comandos se mostrarán los resultados del clasificador, relacionando cada símbolo con la clase a la que se ha determinado que pertenece, tal y como aparece a continuación:

```
Símbolo detectado: 26 - claveSol
Símbolo detectado: 27 - corchea
Símbolo detectado: 19 - cuatro
Símbolo detectado: 16 - blanca
Símbolo detectado: 16 - blanca
Símbolo detectado: 2 - barraDiv
Símbolo detectado: 16 - blanca
Símbolo detectado: 16 - blanca
Símbolo detectado: 26 - claveSol
Símbolo detectado: 16 - blanca
Símbolo detectado: 16 - blanca
Símbolo detectado: 26 - claveSol
Símbolo detectado: 16 - blanca
Símbolo detectado: 42 - negra
Símbolo detectado: 26 - claveSol
Elapsed time is 1.012635 seconds.
```

La última línea (“*Elapsed time is … seconds*.”) nos muestra el tiempo necesario para ejecutar todo el proceso del clasificador. Este tiempo variará según la partitura y el equipo utilizado para ejecutar el programa.

Aparte de los resultados mostrados por pantalla, se generará un archivo con los resultados detallados (*reconocedor_n.log*, donde n = número de pentagrama actual) que se podrá visualizar para ver los elementos que se han utilizado para clasificar los símbolos, entre otros datos de interés. Dicho informe se genera dentro de la carpeta “*Informes*” en el directorio raíz (si no existe tal directorio, se crea automáticamente).

Llegados a este punto se ejecutará el módulo encargado de detectar el tono de las notas y comprobar si hay algún que otro fallo en los datos reconocidos, analizando, por ejemplo, los tiempos en los compases. Un fallo típico de tiempo erróneo en el compás puede ser el siguiente:

```
AVISO: problema de compás en la posición: 14
Compás: 2
Tiempo registrado: 11
Tiempo requerido: 4
```

La posición que indica el aviso se corresponde con la posición que ocupa el símbolo dentro de los símbolos reconocidos, el número de compás es aquel que está analizando el programa, el tiempo requerido es el tiempo definido para el pentagrama y el tiempo registrado es el tiempo que se ha encontrado en el compás actual. Lógicamente, si tiempo requerido y registrado no coinciden, existe algún elemento mal reconocido. No obstante, dicho elemento introduce un error pero no evita que se genere la partitura.

Un caso distinto sería aquel en el que el módulo lanzara, aparte de otros mensajes de avisos, un error como el que se muestra a continuación:

```
ERROR: es posible que la partitura no se genere correctamente.
Ver log para más información.
```

Mientras los avisos proporcionan información sobre datos mal reconocidos, los errores nos indican de que, aparte de símbolos mal reconocidos, hay indicios serios de que la partitura final pueda ser mal codificada y no se genere la partitura (en forma gráfica y acústica). En este caso, se lanza dicho error porque el programa encuentra varias claves de Sol (debido a errores de clasificación) que se codificarán seguidas.

Sin embargo, aunque la codificación de los datos pueda estar mal, *Lilypond* es muy flexible y salvo casos extremos, aún generándose error, la partitura se podrá representar sin problema.

Este módulo además elabora un informe detallado (*detectaTono_n.log*, donde n = número de pentagrama actual) con los problemas encontrados para todos los símbolos analizados.

En caso de que el pentagrama evaluado no disponga de clave ni tiempo, se definirán unos valores estándar como son la clave de Sol y el tiempo de 4/4 (siempre que dicho pentagrama no forme parte de una partitura y obtenga estos datos de otro pentagrama)

Llegados a este punto se mostrará el siguiente pentagrama a evaluar y se repetirá todo el proceso hasta ahora mencionado o se mostrará el siguiente mensaje por línea de comandos en caso de que se hayan analizado todos los pentagramas:

```
=> REPRESENTACIÓN EN LILYPOND
Esta etapa se encarga de escribir en un fichero los datos
reconocidos utilizando
el lenguaje de representación musical Lilypond

     Pulse cualquier tecla para continuar...
```

Si se ha llegado a este punto significa que ya se han obtenido los datos de todos los pentagramas encontrados en la partitura, con lo que se procede al almacenamiento del fichero codificado en formato *Lilypond*. Junto al mensaje por línea de comandos se mostrará un cuadro de diálogo para indicar la ruta y el nombre del fichero codificado con la partitura. Si todo se ha realizado correctamente y se guarda el fichero debería mostrarse el siguiente mensaje:

```
Fichero guardado correctamente en:
C:\Users\Usuario\Desktop\ejemploPartitura.ly


 => Fin ejecución
```

En este punto terminaría la ejecución del programa y se devolvería el control de la línea de comandos al usuario. Ahora es cuando debemos hacer uso del programa *Lilypond* para obtener los resultados finales.

Nos dirigimos a la carpeta donde se almacena el fichero y, teniendo instalado *Lilypond* (consultar siguiente apartado en caso contrario), lo ejecutamos. El propio programa se iniciará e interpretará los datos para crear varios archivos en la misma carpeta donde se encuentre el fichero, con distintas extensiones.

Los archivos generados serán:

- **Informe de extensión "log"** que indica el proceso realizado y los posibles errores que puedan aparecer. Si hay problemas que impiden la creación de la partitura, únicamente se generará este archivo.

- **PDF** con la partitura (extensión "pdf")

- **Archivo PostScript** (extensión "ps")

- **Canción de la partitura en formado MIDI** (extensión "mid")

Puede ocurrir que encontremos errores en la partitura final. El proceso realizado por el programa es complejo y hay muchos factores a tener en cuenta para que una partitura se reconozca perfectamente. Generalmente el principal problema que provoca una mala clasificación de símbolos musicales es la falta de ejemplos en la base de datos o la existencia de ejemplos que no definen muy bien la clase a la que pertenecen. También se producen errores de reconocimiento por fallos a la hora de separar los símbolos y elementos (fases de segmentación), aunque esto ocurre en menor medida.

10.3. Instalación de Lilypond

Lilipond es un programa de representación de notación musical que permite generar ficheros PDF, PostScript y MIDI a partir de la codificación del pentagrama en el lenguaje de dicho programa (ver anexo para más información)

Dicho programa genera partituras de gran calidad, basándose en la apariencia de las partituras clásicas. No utiliza interfaces gráficas, sino que toda la partitura de define mediante líneas de texto ASCII, permitiendo que cualquier programa externo pueda implementar la capacidad de escribir el texto de forma automática. Además, cuenta con la gran ventaja de ser software libre, de descarga gratuita, cuyo código fuente está a disposición del usuario y puede ser modificado o copiado, pudiendo mejorar algunos aspectos del programa. Además, se puede ejecutar en las plataformas más populares (*Linux, MacOs X y Microsoft Windows*) y cuenta con un extenso archivo de ayuda y numerosos ejemplos que permiten al usuario novel aprender rápidamente los fundamentos del lenguaje. Todos estos factores han determinado que este programa fuera elegido entre una lista de programas diseñados para tal efecto.

Para su instalación, se debe descargar la versión oportuna del instalador desde la web oficial [47]. Posteriormente, se debe ejecutar el programa siguiendo todas las instrucciones del instalador (pueden variar según el sistema operativo utilizado). Finalmente, si todo se ha realizado de forma correcta, se podrán crear partituras automáticamente. Para probarlo, abriremos un documento de texto vacío y escribiremos:

```
\relative c' {
  c d e f g a b c
}
```

Guardamos el fichero con formato “ly” y lo ejecutamos. Si todo se ha realizado correctamente se deben haber creado tres ficheros: uno PDF, uno LOG y otro PostScript. Si abrimos el fichero PDF podremos ver la partitura que debe tener el siguiente aspecto:

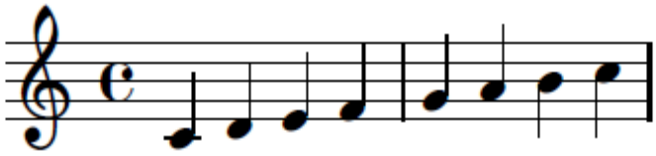


Figura 109 - Partitura realizada con Lilypond

Si se han seguido todos los pasos y se han obtenido los resultados descritos, el programa se habrá instalado adecuadamente y se podrá utilizar junto al programa realizado en este proyecto.

11. Anexo C. Manual de referencia

Este apartado documenta el código implementado para facilitar la continuación de su desarrollo, ampliación o modificación. Para facilitar la tarea, se han desarrollado una serie de labores típicas que se detallarán a continuación.

11.1. Creación de una base de datos

La creación de una base de datos es una tarea relativamente fácil que nos permitirá ajustar la eficacia del módulo reconocedor al utilizarlo con determinadas partituras. No todas las bases de datos funcionan igual de bien para todas las partituras. Para el caso de partituras no manuscritas, dado que los símbolos de una misma clase suelen ser muy parecidos, es probable que no exista el problema de una mala clasificación si utilizamos una base de datos de símbolos no manuscritos con una partitura cualquiera hecha a máquina u ordenador, se haya hecho de la forma que sea. Sin embargo, para el caso de partituras manuscritas los símbolos de una misma clase pueden variar sustancialmente si quien los dibuja es una persona u otra. Es por eso que es recomendable que cada usuario tenga una base de datos independiente a la de otro con ejemplos de sus propios símbolos, para que pueda obtener la máxima eficacia al realizar la clasificación de los notas del pentagrama.

Para ello, se debe partir de la base que, por cuestiones de diseño del programa, se podrán utilizar hasta 99 categorías de símbolos distintos. En los ejemplos del proyecto, la base de datos más grande cuenta con 41 clases de ejemplo, más otro archivo necesario que se detallará más adelante.

El utilizar más clases de las 41 realizadas conlleva un problema: no se representarán los símbolos de las clases nuevas a menos que se modifique adecuadamente el código de "*representaPartitura*". Además, si queremos que dichos símbolos sean analizados en busca de posibles errores, también tendremos que modificar el módulo "*detectaTono*.". Estas tareas se detallarán más adelante.

Si queremos que el programa funcione sin modificar código, utilizando las 41 clases ya predefinidas, debemos crear, dentro de una carpeta que defina nuestra base de datos, las carpetas oportunas que representen los símbolos que queremos identificar. Cuantas más clases, mayor abanico de posibilidades tendrá la clasificación de un símbolo (y por lo tanto, mayor tasa de error). El nombre de la carpeta será del tipo "*Nº - nombreClase*", siendo "*N*" el número de la posición del elemento dentro de la base de datos, y "*nombreClase*" el nombre de la clase. Ambos elementos están separados por un espacio-guión-espacio, también necesario para definir la carpeta. Para el correcto funcionamiento es muy importante que los nombres que se utilicen sean los siguientes:

1 - semicorchea	18 - silencioCorchea	35 - fusa
2 - barraDiv	19 - cuatro	36 - 3por2
3 - 4por4	20 - semifusa	[37]
4 - 2por4	[21]	38 - ligadura
5 - bemol	[22]	39 - medioBemol
6 - compasBinario	[23]	40 - 6por8
7 - compasillo	24 - claveFa	41 - seis
8 - sostenidoymedio	25 - dos	42 - negra
9 - sostenido	26 - claveSol	43 - ocho
10 - silencioSemifusa	27 - corchea	[44]
11 - silencioSemicorchea	[28]	45 - puntillo
12 - silencioRedondaBlanca	29 - cuadruple	46 - redonda
13 - becuadro	[30]	[47]
14 - silencioNegra	31 - doble	48 - dobleBemol
15 - silencioFusa	32 - dobleSostenido	49 - 3por4
16 - blanca	33 - tres	[50]
17 - semiSostenido	34 - 2por2	[51 hasta 99]

Figura 110 - Clases predefinidas del programa

Las posiciones entre corchetes se refieren a posiciones libres a ocupar por determinadas clases. De las 49 primeras posiciones iniciales que de definieron se han eliminado 8 clases pertenecientes a símbolos que indicaban la intensidad del sonido (piano, forte…) por considerar que no eran relevantes para la tarea desarrollada, con lo que se obtienen las 41 clases que se mencionaban anteriormente.

Aparte de las carpetas, se debe incluir un archivo aislado denominado "*test.bmp*" dentro de la carpeta raíz de la base de datos. Dicha imagen, en formato binario, representa una nota (aunque puede representar cualquier cosa) y se usa para determinar el número de descriptores que se utilizarán para definir la base de datos.

Una vez se hayan creado todas las carpetas, se deben incluir ejemplos representativos para las partituras sobre las que vayamos a trabajar. Esto significa que si se van a utilizar partituras hechas a mano, no se guarden símbolos hechos a ordenador en la base de datos, sino símbolos realizados a mano por la persona que ha escrito la partitura, preferentemente.

Se recomienda que cada símbolo sea de un tamaño aproximado de 100 x 100 píxeles, aunque este factor únicamente se dicta a modo de referencia. No importa utilizar símbolos más pequeños o más grandes, sin embargo, como es lógico cuanto más pequeño sea un símbolo menos representativo será de la forma de la clase a la que pertenece dicho elemento y cuanto más grande, más tiempo de cómputo requerirá el proceso para su análisis.

Los símbolos se pueden obtener de muchas formas… Dibujándolos mediante una tableta digitalizadora, escaneándolos… Para el caso de símbolos manuscritos se recomienda utilizar el módulo "*segmentaNotas*" (dentro de la carpeta "*SegmentaBaseDatos*" en el directorio raíz) que extrae los símbolos de una imagen y los almacena en archivos independientes. Dicha imagen puede ser una hoja en blanco digitalizada en la que se hayan dibujado un número determinado de símbolos de una misma clase. Para más información, se debe consultar el apartado dedicado a la descripción a alto nivel de cada módulo

Los únicos requisitos que deben cumplir las imágenes de cada categoría son que se encuentren aislados (sin otros símbolos alrededor, ni líneas de pentagrama), que tengan formato binario y que el símbolo sea de color negro y fondo blanco. No importa el tamaño de la imagen, ni que tenga espacios blancos alrededor de la imagen.

El número de ejemplos que debemos incluir por categoría varía en función del símbolo. Habrá símbolos que por su morfología puedan ser representados con un número reducido de símbolos (clave de fa, negra…), mientras otros más complejos necesiten de más símbolos para ser caracterizados adecuadamente (clave de sol, silencio de negra…). Por regla general se recomienda una gran cantidad de ejemplos por clase ya que de esta forma tendremos más ejemplos sobre los que comparar.

Una vez se hayan incluido todos los ejemplos, se debe codificar la base de datos en un archivo de MATLAB. Para ello, se extraerán los descriptores de cada ejemplo almacenado en la base de datos y lo guardaremos en dicho fichero. Esta tarea se realiza mediante la ejecución del módulo "*cargaBD*" cuyos parámetros se pueden consultar mediante la descripción realizada en el apartado 4.3 o mediante la ayuda del propio fichero escribiendo "*help cargaBD*" en línea de comandos.

Como ya se comentó, de utilizar alguna categoría no incluida en las 41 mencionadas anteriormente tendremos que realizar una serie de modificaciones sobre los módulos "*detectaTono*" y "*representaPartitura*" si queremos que sobre dichas categorías se hagan comprobaciones y se representen en la partitura final los símbolos nuevos. Se van a detallar algunos de los elementos a tener en cuenta dentro de cada módulo para las modificaciones:

- **detectaTono**

  Este módulo encargado de comprobar el tono de las notas y otros elementos de la partitura puede modificarse para que analice las nuevas clases que se incluyan. Obviamente, el análisis que se realice dependerá de cada elemento por lo que se presupone tarea del futuro desarrollador. Sin embargo se deben tener en cuenta algunas variables para tal efecto:

  - **`simbolosReconocidos{i,1}`** = primera columna de dicha variable *cell array.* Describe la posición que ocupa la clase del símbolo dentro de la base de datos. Se utiliza para determinar qué símbolo vamos a evaluar y realizar el procesado oportuno para detectar errores.

  - **`simbolosEscribir`** = variable *cell array* compuesta por dos columnas. En la primera se almacena la clase a la cual pertenece cada símbolo definido para escribir en la partitura final y en la segunda algunos atributos relacionados con dicho elemento (tono, partitura, duración anacrusa, tiempos…). Dicha variable se utiliza para definir los elementos que se escribirán en la etapa "*representaPartitura*". Además, dentro de la clase almacenada en la primera columna se permiten añadir clases extras resultado de las relaciones entre símbolos. Actualmente hay 5 de estas clases definidas (con signos negativos para no interferir con los definidos por la base de datos):

- - 1 = Anacrusa
- - 2 = Doble barra divisoria
- - 3 = Inicio barra de repetición
- - 4 = Final barra de repetición
- - 5 = Doble punto

o **`contador`** = variable numérica que indica la próxima posición sobre la que se escribirá en la variable "**`simbolosEscribir`**". Es necesario que cada vez que queramos hacerlo utilicemos las expresiones:

```
simbolosEscribir{contador,1} = clase;
simbolosEscribir{contador,2} = atributos;
contador = contador + 1;
```

En el módulo se definen dos partes fundamentales. La primera se encarga de analizar los elementos que existen en la variable "**`simbolosReconocidos`**" y de escribir los resultados en "**`simbolosEscribir`**", siguiendo el pseudo-código:

```
for i = 1:size(simbolosReconocidos,1)
      switch simbolosReconocidos{i,1}

            // Se analizan las clases una por una
      case {elemento/s}
            // Se analiza el símbolo determinado
            …

            // Se escriben los datos en simbolosEscribir
            simbolosEscribir{contador,1} = clase;
            simbolosEscribir{contador,2} = atributos;

            // Se aumenta el contador una unidad
            contador = contador + 1;
      end
end
```

En esta primera parte también se pueden buscar relaciones entre elementos llamando a la posición anterior mediante **`simbolosReconocidos{i-1,1}.`** Por ejemplo, en el caso de la doble barra al recorrer el *array* nos encontramos al principio con una barra divisoria y se guarda como tal en la variable "**`simbolosEscribir`**". Al mismo tiempo se ha comprobado si existe la misma barra en la posición anterior (suponemos que al principio no). Una vez realizado, pasamos al siguiente símbolo que también resulta ser una barra divisoria. Antes de etiquetarla como tal, se comprueba si en la posición anterior existe una barra divisoria. En este caso sí, por lo que sobrescribimos el etiquetado anterior y continuamos. En términos de pseudo-código sería:

```
switch simbolosReconocidos{i,1}
      case {id_barra}
      if simbolosReconocidos{i-1,1} == id_barra
            // Se escriben los datos en simbolosEscribir
            simbolosEscribir{contador,1} = id_2barras;
            simbolosEscribir{contador,2} = atributos;
      else
            // Se escriben los datos en simbolosEscribir
            simbolosEscribir{contador,1} = id_barra;
            simbolosEscribir{contador,2} = atributos;
      end
      // Se aumenta el contador una unidad
      contador = contador + 1;
end
```

La segunda parte fundamental se encarga de realizar comprobaciones sobre los datos que se van a escribir en la partitura ("**simbolosEscribir**"), así como de reconocer nuevos símbolos compuestos. En términos de pseudo-código:

```
for i = 1:size(simbolosEscribir,1)
      switch simbolosEscribir{i,1}
            case {id_elemento}
            // Realizamos las comprobaciones
      end
end
```

Si se encuentra un nuevo símbolo compuesto, resultado de la mezcla de otros, y se quiere anular alguno (por ejemplo, se tiene una barra doble seguida de un punto doble, lo que da como resultado el inicio de una barra de repetición y se quiere definir esta última y anular las otras dos), basta con marcar el identificador a cero o sobrescribir el antiguo por el nuevo. La siguiente etapa lo ignorará en el caso de estar a cero.

En este apartado se comprueba el tiempo de las notas cada vez que nos encontramos con una barra divisoria. Dicho tiempo se define por unas determinadas variables:

- **tiempo**: tiempo definido para el pentagrama, utilizando la nomenclatura descrita anteriormente (negra = 1, blanca = 2, redonda = 4, corchea = 0.5…)

- **recuentoTiempo** = variable que registra el tiempo actual para cada nota. Se modifica cada vez que nos encontramos con una nota o un silencio sumando el valor de dicho elemento

El resto de las comprobaciones realizadas no se detallarán, dejando como tarea del desarrollador su estudio.

Si el programador quiere incluir algún mensaje en el informe generado en este módulo, basta con que utilice la expresión:

```
fprintf(archivoLog,texto);
```

Donde "*texto*" representa el texto a incluir en el fichero. Si toma el valor *'\n'* se genera un salto de línea y si toma el valor *'\t'* se incluye una tabulación.

- **representaPartitura**

  Este módulo se encarga de codificar los datos de la variable "**simbolosEscribir**" en un fichero de formato *Lilypond*. El funcionamiento es simple: para cada símbolo de la variable descrita, se genera un código que representa dicho elemento en formato *Lilypond*. En términos de pseudo-código se describiría de la siguiente forma:

```
for p = 1:size(simbolosEscribirConjunto,2)
simbolosEscribir = simbolosEscribirConjunto{p};
      for elemento = 1:size(simbolosEscribir,1)
            switch simbolosEscribir{elemento,1}
                  case {id_elemento}
                  // Se codifica el elemento
                  partitura = [partitura,texto_lilypond];
            end
      end
end
```

  Donde "**simbolosEscribirConjunto**" es un *cell array* (1 x i) que almacena en cada posición un *cell array* "**simbolosEscribir**" que define cada pentagrama de una partitura, "**id_elemento**" se corresponde con el valor en la posición "**elemento**" de "**símbolosEscribir**" y donde "**texto_lilypond**" representa el elemento codificado. Para ver el formato a utilizar, consultar el anexo de este proyecto.

11.2. Modificación de algunos parámetros del reconocedor

El reconocedor es una de las etapas más importantes del proyecto encargada de realizar la clasificación de los símbolos. Para ello realiza una comparación de distancias mediante el algoritmo k-NN utilizando una base de datos de símbolos y el símbolo a evaluar. En este apartado se detallarán algunas de las variables existentes en el programa para futuros cambios en su código:

- **k** = número de vecinos a evaluar en el algoritmo k-NN. Este parámetro toma valor a partir de los parámetros de entrada (variable "**vecinos**")

- **vectorCaract** = vector de descriptores del símbolo a evaluar. Se obtiene como la salida del módulo "*extraeCaract*". La modificación de este módulo para variar el número de descriptores utilizados se detallará posteriormente.

- **simbolosReconocidos** = *cell array* de tamaño (i x 2) que contiene en la primera columna la posición dentro de la base de datos del símbolo reconocido y en la segunda columna la clase a la que pertenece.

- **matrizCaract** = variable formada por un conjunto de vectores de características, donde cada columna representa el vector de una imagen de la base de datos. La última fila de cada columna indica el grupo al que pertenece la imagen de dicho vector de características.

- **leyenda** = variable *cell array* que relaciona los nombres de las categorías con la posición que ocupa dicha categoría en la base de datos.

La estructura en pseudo-código que utiliza dicho módulo es la siguiente:

```
for i = 1:length(simbolos)
      [vectorCaract] = extraeCaract(simbolos{i},false)
      // Se ejecuta el algoritmo clasificador, en nuestro caso el k-NN
      …
      simbolosReconocidos{i,1} = identificador_simbolo_reconocido
      simbolosReconocidos{i,2} = nombre_clase
end
```

Donde la variable "**simbolos**" se corresponde con un *cell array* con los elementos a clasificar. El identificador que se debe incluir en la primera columna de "**simbolosReconocidos**" se corresponde con la posición que ocupa la clase del símbolo en la base de datos. El nombre de la clase que se debe incluir en la segunda columna se obtiene a partir de la variable "**leyenda**".

En este módulo se ha implementado el algoritmo k-NN pero se puede sustituir o utilizar otro distinto si así se prefiere. Basta con implementarlo utilizando la estructura propuesta en pseudo-código, dejando intacto el resto de código del módulo que se encuentre fuera de esa estructura.

Si se quiere modificar la distancia utilizada en el algoritmo k-NN implementado (actualmente la que se utiliza es la distancia Euclídea), únicamente se tiene que modificar la estructura siguiente:

```
      for indice = 1:tobjetos
            d(1,indice) = valor_nuevaDistancia_en_posicion_"indice";
      end
```

Dicha estructura define una variable "**d**" con las distancias para cada elemento. La variable "**tobjetos**" indica el número de elementos en la base de datos. También existe otra variable de interés denominada "**tcaracteristicas**" que indica la longitud de cada vector de características. Se debe recordar que la última posición de dicho vector incluye un identificador que representa la clase a la que pertenece, por lo que si un vector es de longitud "**tcaracterísticas**", los descriptores de dicho vector serán los elementos comprendidos entre el primer elemento y el elemento "**tcaracteristicas**" menos la unidad.

En el caso de que se desee modificar el número de descriptores utilizados, se debe variar, dentro del módulo "*extraeDescriptores*", el valor del parámetro **M** que define dicho número.

11.3. Creación de módulos de extracción de líneas del pentagrama

El módulo encargado de extraer las líneas del pentagrama sobre cada símbolo se puede modificar para utilizar otro algoritmo distinto a los implementados en el proyecto. Para ello, debemos modificar adecuadamente el módulo "*extraeLineasNotas*" y añadir el algoritmo deseado.

La estructura en pseudo-código que sigue el módulo realizado es la siguiente:

```
for i = 1:size(simbolos,2)

      // Se obtienen los datos de la nota "i"
      notaActual = simbolos{1,i};
      proyX = simbolos{2,i};
      proyY = simbolos{3,i};

      // Se crea una estructura del mismo tamaño que la imagen
      para almacenar // los píxeles que se eliminarán
      [filas, columnas] = size(notaActual);
      eliminar = zeros(filas, columnas);

      // Se ejecuta el algoritmo para definir las líneas que se
      eliminarán
       …

      //  Se eliminan los píxeles definidos en la variable
      "eliminar"
       …

      // Se guardan los símbolos
      simbolosLimpios{1,i} = logical(notaActual);

      // Se representa el símbolo
      figure; imshow(simbolosLimpios{1,i});

end
```

El nuevo algoritmo implementado debe incluirse en el apartado dedicado para definir las líneas del pentagrama y debe grabar en la variable "**eliminar**" los píxeles que se deben eliminar. Además, se debe definir un método para ejecutar el algoritmo ya que, actualmente, la variable "**color**" que puede tomar valor "*true/false*" define si se ejecutará el método normal o el método de las líneas azules.

Hay que recalcar que no se debe modificar la variable "**notaActual**" en el algoritmo que se programe, dado que esta tarea se realiza posteriormente en el apartado encargado de eliminar las líneas del pentagrama.

11.4. Codificación de la partitura en otros lenguajes

Para realizar la codificación se utiliza el módulo "*representaPartitura*" que se encarga de crear un archivo de formato *Lilypond* con la partitura. Para ello, se asigna a cada nota un determinado código que se representa en el archivo final.

Si se desea utilizar otro sistema de representación se debe crear otro módulo distinto que utilice la sintaxis adecuada. Para facilitar la tarea, se puede utilizar la estructura del módulo desarrollado, descrito en el apartado C.1.

El parámetro de entrada principal del nuevo módulo creado debe ser la variable "**simbolosEscribirConjunto**". Dicha variable es un *cell array* que recoge la variable "**simbolosEscribir**" de cada partitura. El nuevo módulo debe extraer ésta última variable de cada posición de la otra y codificar los datos adecuadamente para cada pentagrama.

La variable "**simbolosEscribir**" se trata de un *cell array* de tamaño (i x 2) que almacena en la primera columna la clase a la cual pertenece cada símbolo definido para escribir en la

partitura final y en la segunda algunos atributos relacionados con dicho elemento (tono, partitura, duración anacrusa, tiempos…). Debemos estar familiarizados con los valores que pueda tomar dicha variable para poder definir los parámetros a codificar.

A la hora de escribir el contenido de un fichero, serán de gran utilidad las funciones "*fopen*", "*fprintf*" y "*fclose*" implementadas en MATLAB. Éstas permitirán crear un fichero, imprimir los datos en él y guardarlo, respectivamente. Para más información se puede consultar la ayuda de MATLAB o ver cómo se ha utilizado en el módulo diseñado.

11.5. Utilización del reconocedor con otros sistemas de entrada

Actualmente, el reconocedor encargado de realizar la clasificación realiza su tarea utilizando la salida proporcionada por el módulo "*extraeNotas2*". Sin embargo, también es posible utilizarlo mediante ejemplos aislados con el módulo "*cargaEjemplos*".

El primer módulo proporciona una salida que incluye información sobre el tono de las notas (debido al procesamiento realizado en etapas anteriores), sin embargo, el segundo no (recordemos que son notas aisladas). Esto implica que, si se utiliza el segundo módulo, no se podrá representar ninguna partitura.

En este último caso, se tendrá que hacer una modificación de la arquitectura para definir mediante otra vía los tonos de los elementos. Esto se tendrá que hacer en el caso de que se desee utilizar como sistema de entrada un conjunto de símbolos aislados que representen una partitura o un sistema que digitalice una a una las notas de un pentagrama (por ejemplo, mediante una tableta digitalizadora)

No se entrará en más detalle sobre cómo realizar esta tarea, puesto que se presupone tarea del desarrollador y pueden darse multitud de posibilidades. Sin embargo, hay que hacer hincapié de que para que el reconocedor funcione adecuadamente se deben utilizar unos determinados parámetros de entrada. Se recomienda estudiar la estructura de dicho módulo antes de realizar ningún diseño para conocer exactamente qué variables deben generar los nuevos módulos que se desarrollen.

12. Anexo D. Símbolos manuscritos

En este apartado se mostrarán algunos de los símbolos dibujados en papel que se han utilizado para definir determinadas bases de datos de símbolos y ejemplos para evaluar las prestaciones del clasificador.

Dichos ejemplos se pueden clasificar como se detalla a continuación:

- **Test David (2 páginas):** símbolos para crear el conjunto de evaluación "**`Notas_Aisladas2_David`**" que contiene 606 símbolos útiles.

- **David 2 (4 páginas):** símbolos para crear la base de datos "**`baseDatos_Manuscritas_AMPLIADA_David`**" que contiene 923 símbolos útiles.

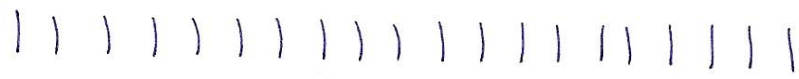
- **David (1 página):** símbolos para crear el conjunto de símbolos aislados "**`Notas_Aisladas_David`**" que contiene 177 símbolos útiles.

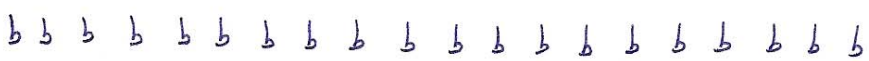
- **Rocío (3 páginas):** símbolos para crear la base de datos "**`baseDatos_Manuscritas_Rocio`**" que contiene 411 símbolos útiles y el conjunto de símbolos aislados "**`Notas_Aisladas_Rocio`"** compuesto por 52 símbolos útiles.

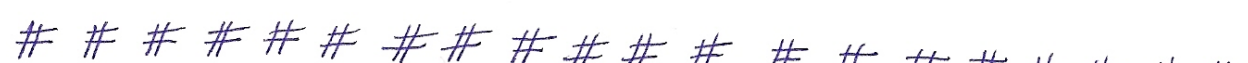
Al hablar de símbolos útiles se hace referencia a aquellos símbolos que son válidos para la base de datos o para el conjunto de símbolos. Hay que tener en cuenta que puede que haya símbolos manuscritos cuyos componentes se hayan dibujado correctamente pero no se solapen. En este caso, el módulo encargado de extraer y guardar los elementos, separará sus componentes y los almacenará como archivos distintos. Esos símbolos se deben desechar para evitar problemas de clasificación.

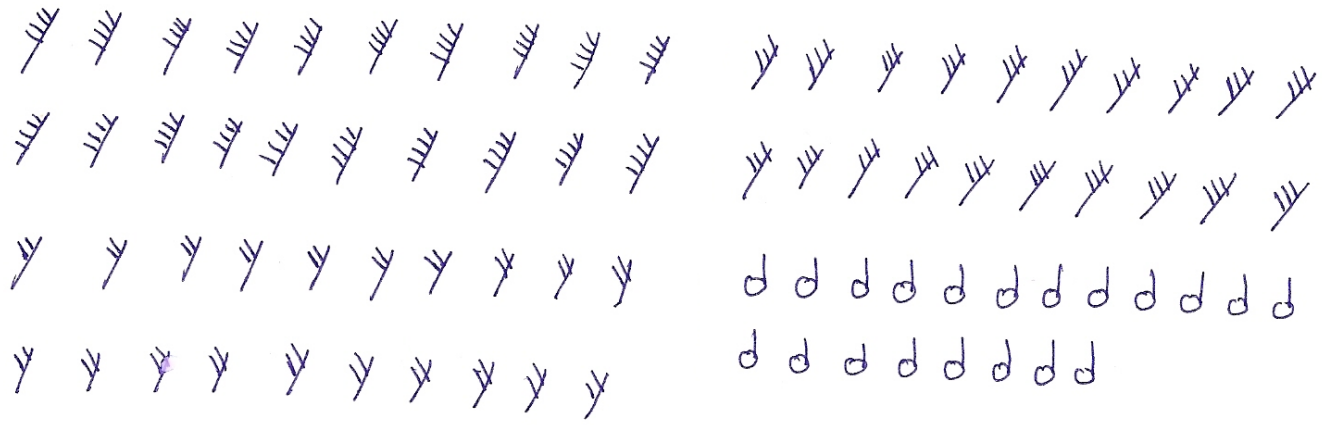
Test
David

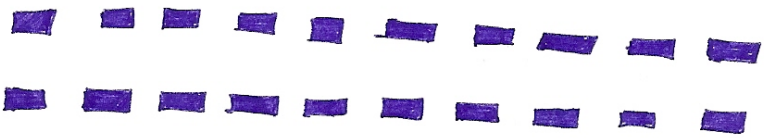
1/2

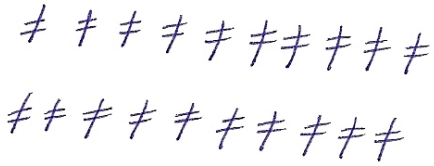


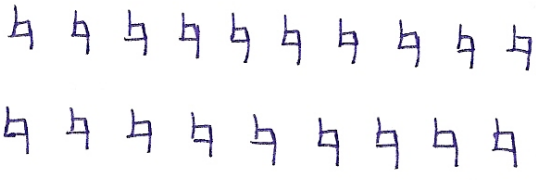


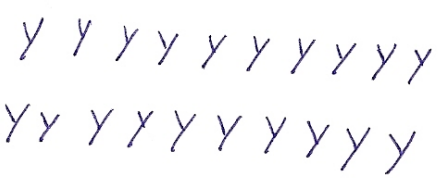


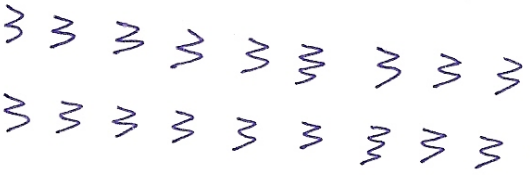


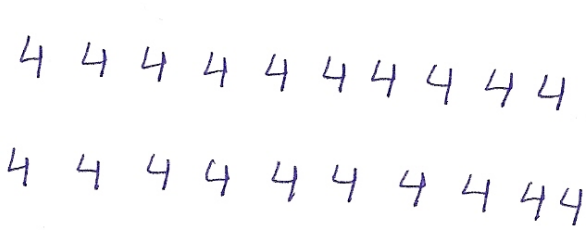


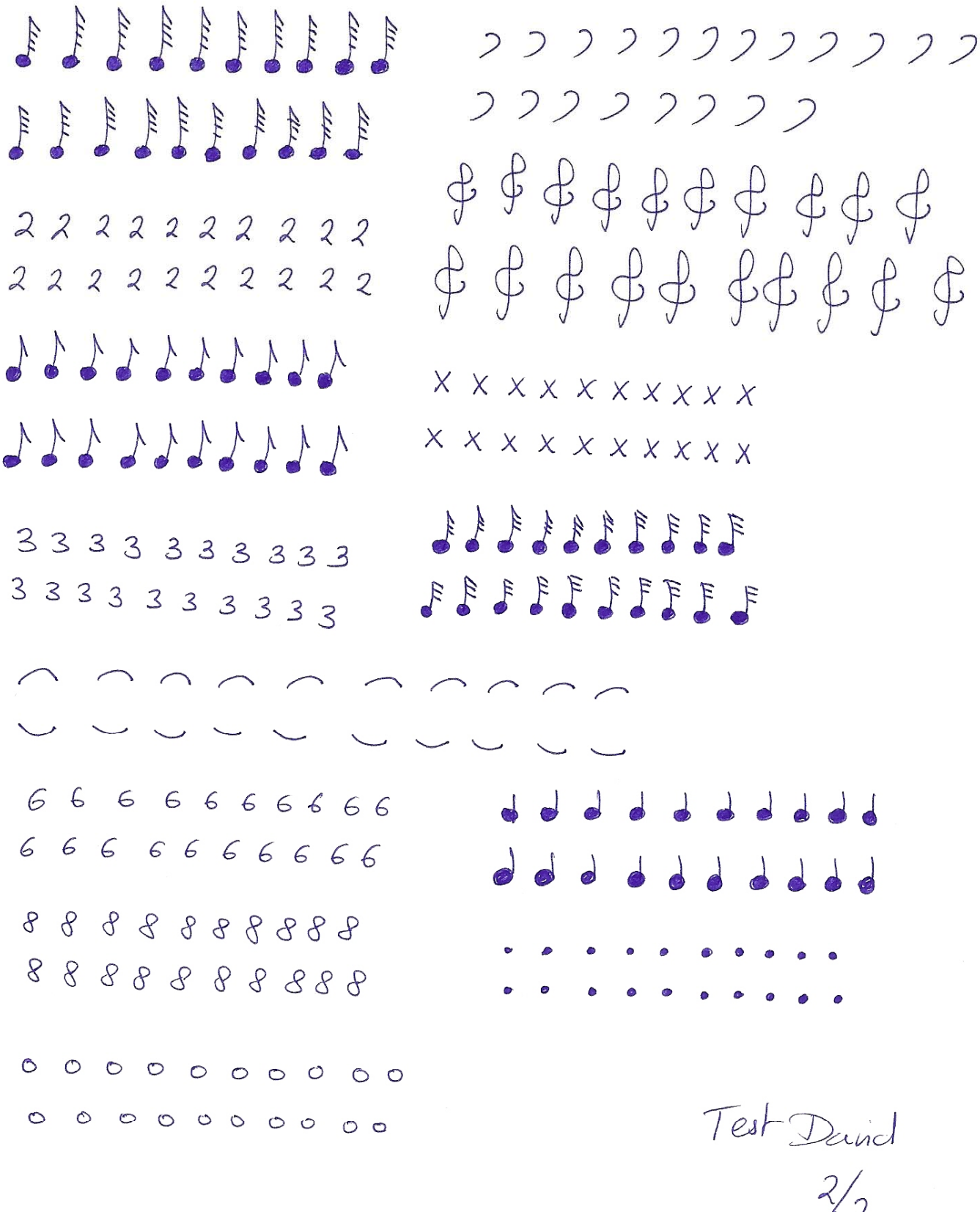




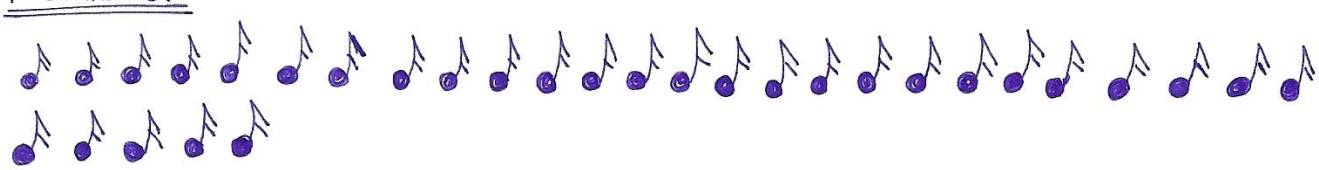




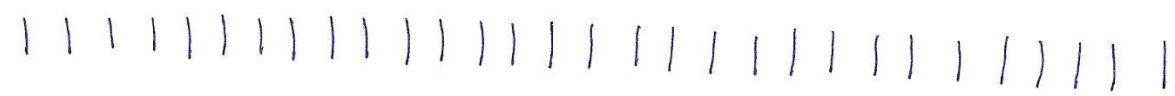



Test David
2/2

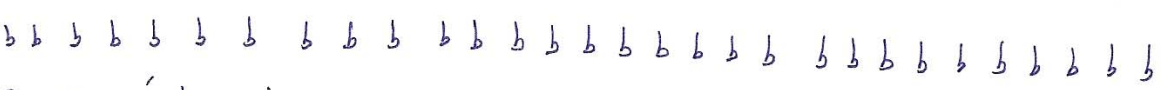
1 - semicorchea



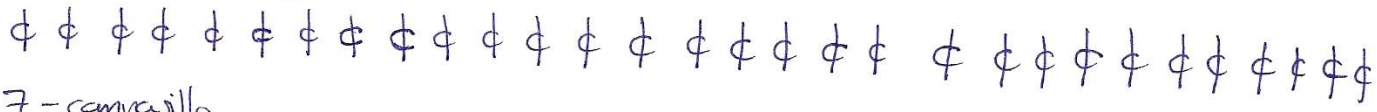
2 - barraDiv



5 - bemol

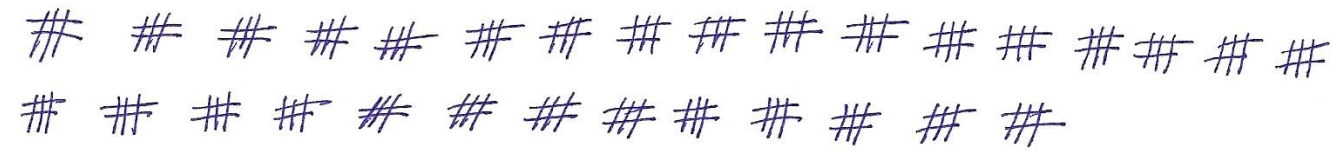


6 - compás binario



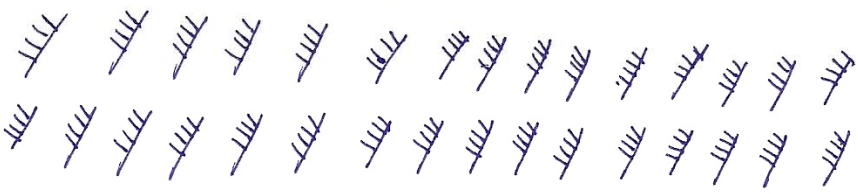
7 - compasillo

8 - sostenido y medio

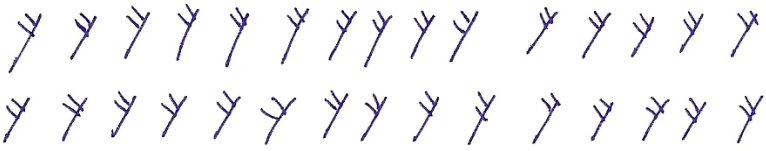


9 - sostenido

10 - silencio Semifusa

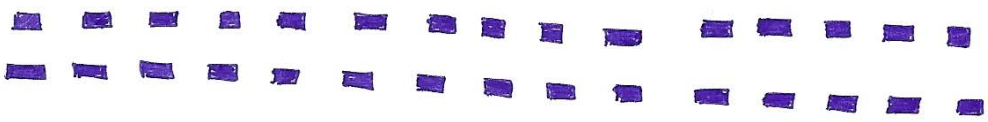


11 - silencio semicorchea

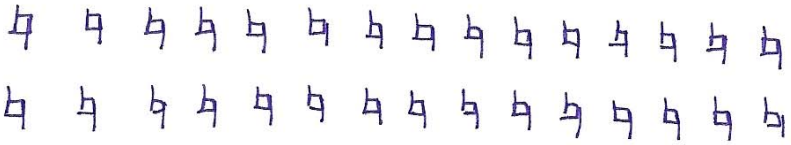


David 2

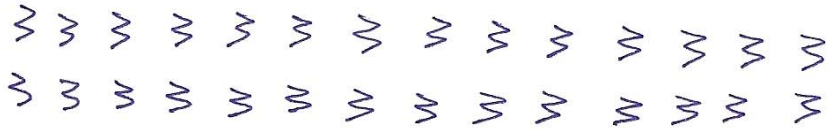
12 - silencio Redonda / Blanca



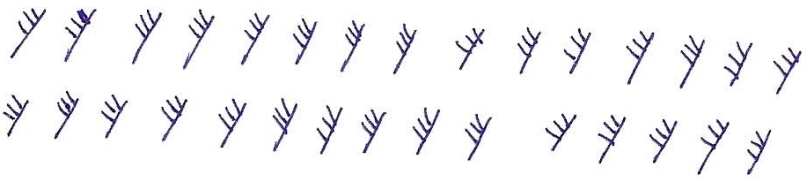
13 - bemuadro



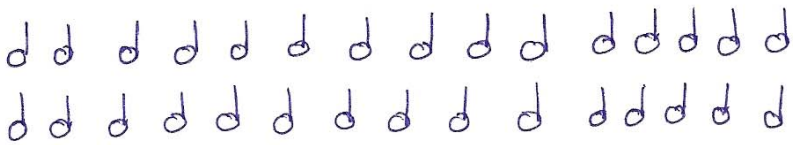
14 - silencio Negra



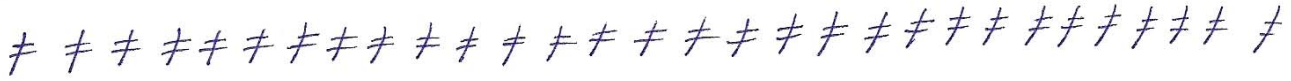
15 - silencio Fusa



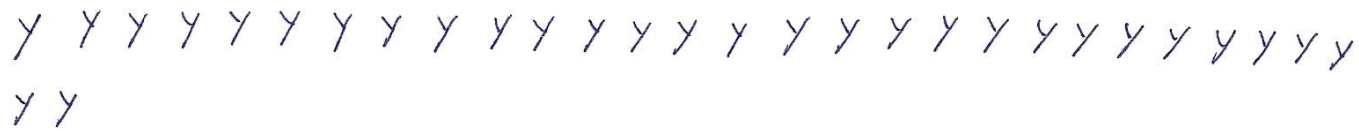
16 - blanca



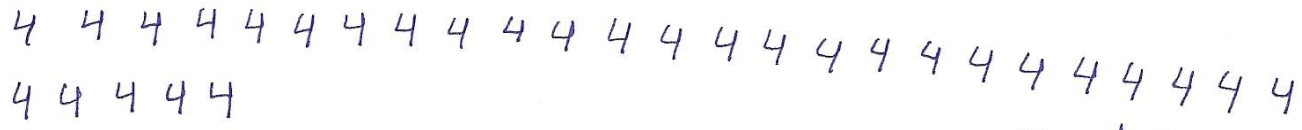
17 - semisostenido



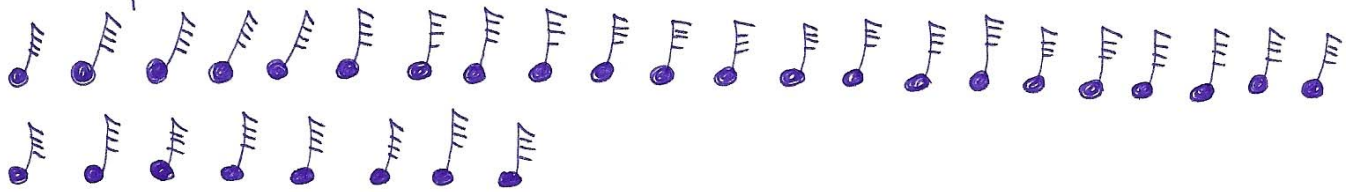
18 - silencio Corchea



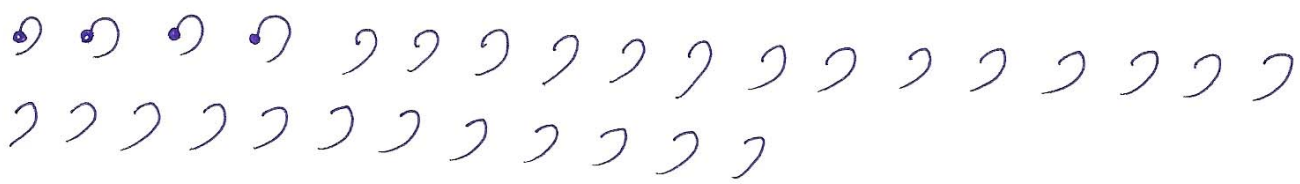
19 - cuatro



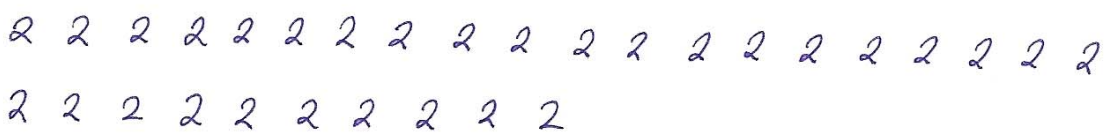
20- semifusa



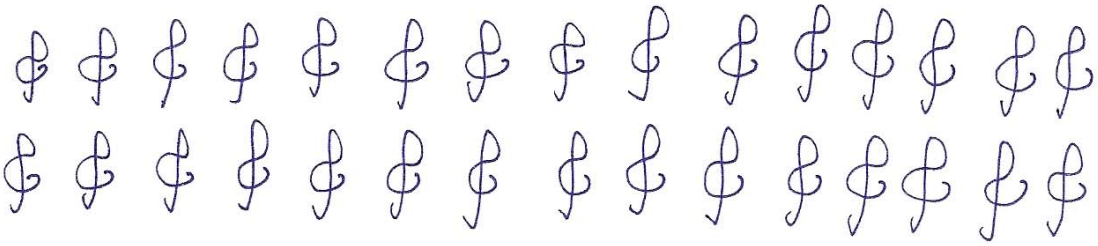
24 - clave Fa



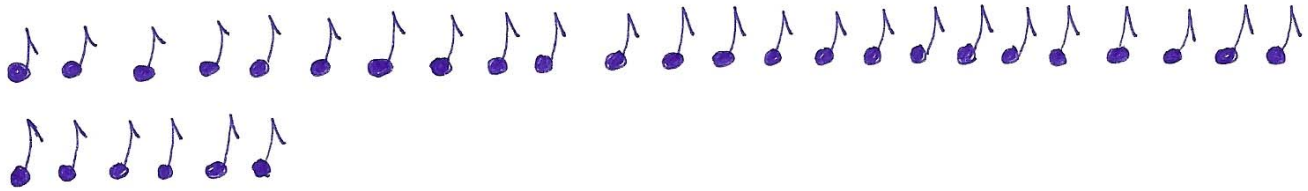
25- clave



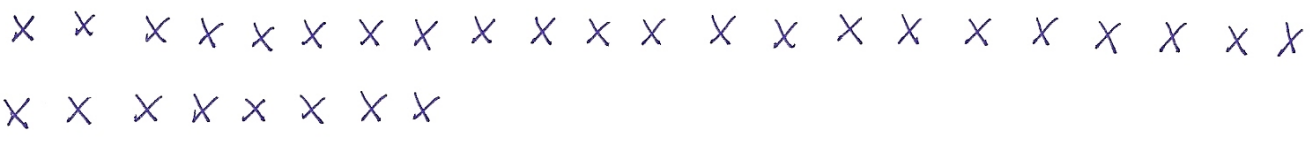
26 - clave Sol



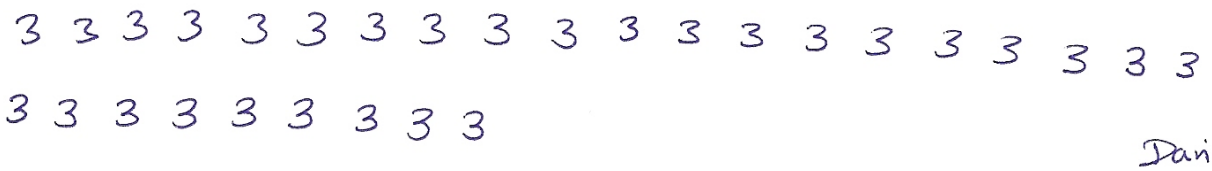
27 - corchea



32 - doble sostenido

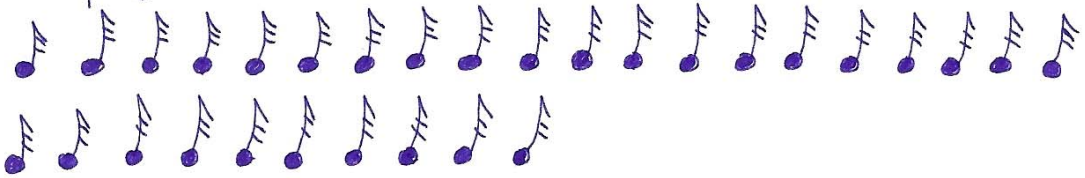


33 - tres

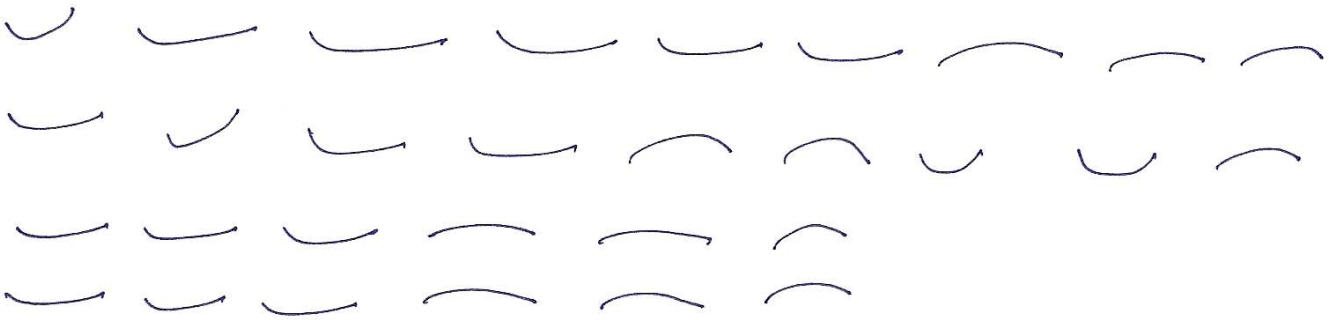


David 2 3/4

35 - fusa



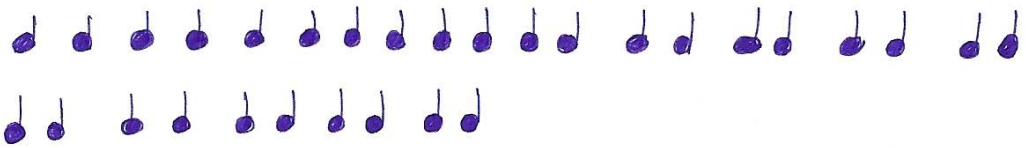
38 - ligadura



41 - seis

6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6

42 - negra



43 - ocho

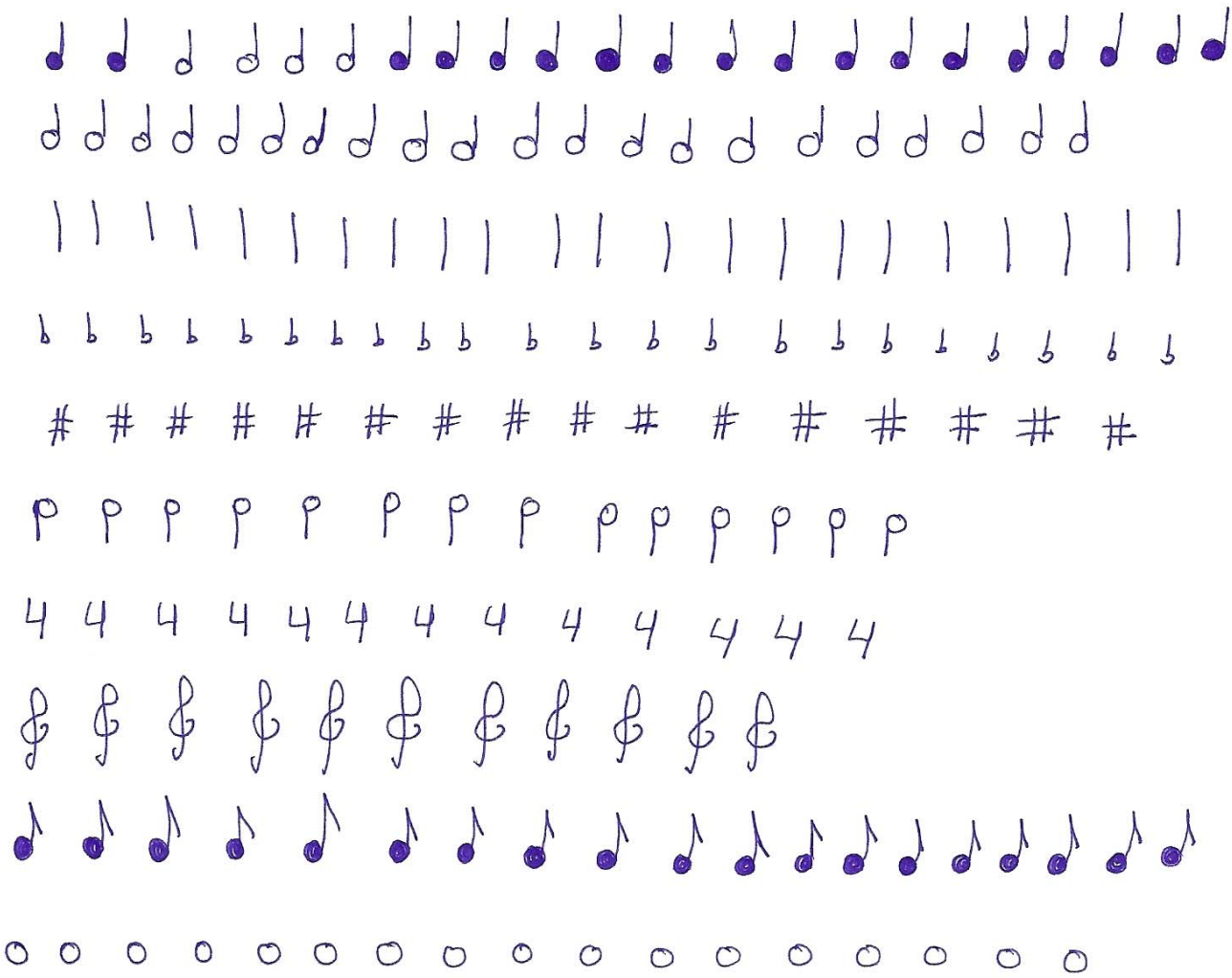
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8

45 - puntillo

46 - redonda

Band 2.

4/4



David 1/1