UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

Corso di Laurea magistrale in
Informatica

OPTICAL MUSIC RECOGNITION PER DOCUMENTI IEEE 1599 BASATO SU FILTRI IMMAGINE E OPERAZIONI MORFOLOGICHE

Relatore: Prof. Luca Andrea Ludovico
Correlatore: Prof. Adriano Baratè

Tesi di Laurea di:
Jacopo Matteo Carravieri
Matr. Nr. 902589

ANNO ACCADEMICO 2019-2020

Questo lavoro è dedicato a tutti gli studenti

"Io studio,
ma studiate pure voi,
che se studio solo io non serve a un c. . . o"

– Gli scarabocchi di Maicol & Mirco

"No tale is so good
that it can't be spoiled
in the telling"

– Proverbio

Indice

Indice **ii**

1 Introduzione **1**
1.1 Abstract . . . 1
1.2 Motivazione e contesto . . . 1
1.3 Organizzazione del lavoro . . . 2
1.4 Optical Music Recognition . . . 3
1.4.1 Architettura . . . 6
1.4.2 Preprocessamento . . . 6
1.4.3 Stima delle grandezze . . . 10
1.4.4 Rimozione del rigo . . . 12
1.4.5 Segmentazione . . . 12

2 Il formato IEEE 1599 **13**
2.1 Panoramica . . . 13
2.2 Layer . . . 14
2.3 Sincronizzazione . . . 15

3 Implementazione **19**
3.1 Tecnologie utilizzate . . . 19
3.1.1 Sistema operativo . . . 19
3.1.2 Versioning dei sorgenti . . . 20
3.1.3 Ambiente di sviluppo . . . 20
3.1.4 Gestione delle dipendenze . . . 21
3.1.5 Versione di Java . . . 21
3.1.6 Framework per la Computer Vision . . . 22
3.2 Processo di sviluppo . . . 22
3.3 Scelte generali . . . 23
3.4 User stories . . . 23
3.4.1 FCal . . . 24
3.4.2 GraphicMapper . . . 24

3.5 Diagramma del classi . . . 26
3.5.1 FCal . . . 26
3.5.2 GraphicMapper . . . 27
3.6 Codice sorgente . . . 27
3.6.1 FCal . . . 27
3.6.2 GraphicMapper . . . 28
3.7 Filtri e parametri . . . 30
3.7.1 Descrizione . . . 30

4 Test **37**
4.1 Metodologia . . . 37
4.2 Metriche considerate . . . 38
4.3 Dataset degli spartiti . . . 38
4.4 Risultati . . . 39
4.5 Considerazioni . . . 41

5 Conclusioni **43**
5.1 Risultati raggiunti . . . 43
5.2 Sviluppi futuri . . . 44

Bibliografia **46**

Capitolo 1

Introduzione

1.1 Abstract

Presso il Laboratorio di Informatica Musicale dell'Università degli Studi di Milano è stato sviluppato il formato IEEE 1599, con l'intento di fornire uno strumento per raggruppare e rendere fruibili elementi diversi associabili ad un brano musicale. Nel caso degli spartiti si è voluto rendere possibile evidenziare note, accordi e pause per sincronizzarle con l'andamento della traccia audio. Tale compito inizialmente era portato a termine tramite il tracciamento manuale di riquadri simbolo per simbolo, mentre ora questo processo è semi automatico: grazie all'impiego di filtri immagine e operazioni morfologiche è sufficiente un click su un pixel qualsiasi del simbolo perché attorno ad esso venga disegnata una bounding box.

1.2 Motivazione e contesto

Questo lavoro di tesi si inserisce nel contesto dello standard IEEE 1599, sviluppato e curato dal Laboratorio di Informatica Musicale dell'Università degli Studi di Milano, ed ha come obbiettivo di aumentarne la sua diffusione facilitando la produzione di nuovi contenuti per l'archivio musicale[1].
Lo standard IEEE 1599 sarà discusso nel capitolo 2, tuttavia come informazione di contesto è importante notare come gli spartiti musicali, ed in particolare alcuni dei simboli ivi contenuti, rappresentino un suo costituente fondamentale [1].
Individuare questi simboli era un'operazione manuale, che richiedeva di disegnare un riquadro delle giuste dimensioni attorno ad ognuno di essi, come visibile in figura 1.

[1] `https://ieee1599.lim.di.unimi.it/music_collection.php`

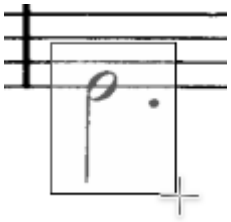


Figura 1: Esempio di selezione manuale

Nel corso del tempo si è appurato come questa fosse un'operazione lunga e ripetitiva, un limite al numero di nuovi contenuti che era possibile produrre e per la quale si è dunque voluto trovare una soluzione alternativa. Questo processo è ora semi-automatico: in figura 2 è visibile come sia sufficiente un click del mouse su una parte qualsiasi del simbolo affinché venga disegnato intorno ad esso un riquadro delle giuste dimensioni. È poi possibile modificarne posizione e dimensione. In sviluppi futuri si vuole fare in modo che nemmeno questa operazione sia più necessaria, come discusso successivamente nel paragrafo 5.2.

Figura 2: Selezione semi-automatica di una nota; a destra la regione cliccabile

La natura del problema e le soluzioni individuate ricadono sotto la definizione di *Optical Musical Recognition*, tuttavia in questo caso la lista dei simboli musicali è già nota, occorre unicamente individuarli all'interno dell'immagine. Si tratta dunque di un problema di **riconoscimento informato**.

1.3 Organizzazione del lavoro

Questo lavoro si articola nei seguenti capitoli:

- **Introduzione**: la parte rimanente del capitolo introduttivo si addentrerà nel concetto di *optical music recognition*, ovvero il riconoscimento automatico dei simboli musicali a partire da un'immagine digitale. Saranno discusse le difficoltà che questo comporta, le soluzioni individuate dalla comunità nonché le problematiche ancora aperte.

- **IEEE 1599**: si discuterà in dettaglio del formato XML sviluppato dal Laboratorio per la rappresentazione a tutto tondo dei contenuti musicali, il secondo pilastro fondamentale di questo lavoro di tesi.
- **Implementazione**: questo capitolo – senz'altro il più corposo – tratterà nella sua interezza il processo di sviluppo che ha portato alla realizzazione del software di riconoscimento, poi divenuto parte della soluzione ufficiale per la creazione di nuovi contenuti.
- **Test**: saranno qui descritti natura e risultati dei test condotti sul software per valutarne le capacità di riconoscimento. Verranno discussi i limiti individuati nonché le potenziali strade da seguire per il loro superamento.
- **Conclusioni**: ultimo, breve capitolo di conclusioni sul lavoro nel suo complesso e sui contributi che ha apportato. Verranno anche forniti spunti per futuri ampliamenti.

1.4 Optical Music Recognition

Il termine *optical music recognition* – d'ora in poi **OMR** – si riferisce sia alla problematica che all'insieme delle tecniche ideate per estrarre i simboli dalle immagini digitali di spartiti musicali. Anche se quantitativamente più abbondanti, il termine simbolo non si riferisce soltanto a note e pause, ma anche a tutta una serie di altri elementi ugualmente importanti per la corretta interpretazione del brano:

- chiave
- durata della misura
- tempo metronomico
- alterazione della nota
- prolungamento della durata
- indicazioni di dinamica
- segni di articolazione
- legature
- annotazioni testuali

L'OMR rappresenta senz'altro un problema particolarmente complesso [2], a causa anche della natura **bidimensionale** della scrittura musicale: sull'asse orizzontale vi è la progressione del brano nel tempo, mentre sull'asse verticale è codificata l'altezza e soprattutto possono esservi tutta una serie di simboli, posti spesso anche molto vicini alla nota vera e propria. La figura 3 illustra un caso – peraltro non particolarmente complesso – dove sono presenti quattro differenti simboli al medesimo tempo: accento, inizio di legatura, nota e informazione sulla dinamica. A questa considerazione si aggiunge il fatto che le note e spesso anche i simboli giacciono direttamente sul pentagramma, fondendosi con esso, fatto che rende la loro separazione automatica ulteriormente problematica.

Figura 3: Quattro simboli organizzati sull'asse verticale

In figura 4 sono invece presenti ben dieci simboli: tre note vuote a formare un accordo, un'alterazione – nella forma di un diesis –, tre prolungamenti di durata – simboli puntiformi – ed un secondo accordo composto da tre note piene:

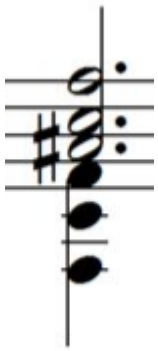



Figura 4: Numerosi simboli musicali organizzati sull'asse verticale

Spesso l'OMR viene paragonato all'*optical character recognition* che, seppur simile in termini concettuali e in una parte nelle tecniche impiegate, si differenzia fortemente proprio sotto questo aspetto: la natura dei testi è strettamente monodimensionale [2], con un singolo carattere da riconoscere in un determinato luogo dell'asse temporale. Esistono delle eccezioni a questa regola, tuttavia sono rare e non vanno perciò a definire la natura generale del problema.

Oltre alla questione sulla bidimensionalità appena discussa, occorre notare come le immagini degli spartiti possano essere vittima – come spesso accade quando si devono processare immagini scansionate – di uno o più dei seguenti problemi:

- bassa risoluzione e/o definizione dell'immagine
- fenomeni di sbiadimento dell'inchiostro
- variazioni di luminosità causate dalle lampade dello scanner
- rumore introdotto dagli equipaggiamenti di cattura
- allineamento non in asse dell'immagine
- contrasto inadeguato

Tali problematiche possono essere mitigate almeno in parte con un adeguato *preprocessing* [2], impiegando algoritmi comunemente utilizzati anche in altri contesti. Questo aspetto sarà discusso nel sottoparagrafo 1.4.2.
Da un punto di vista semantico è invece la complessità ed ambiguità della scrittura musicale a rendere il problema di difficile soluzione: contrariamente all'OCR nell'OMR esistono spesso modi diversi di rappresentare lo stesso concetto. Non c'è giusto o sbagliato in questo caso: la forma scelta dipende unicamente dalle preferenze dell'autore. Vi sono dei progetti nati per introdurre nuove forme di notazione musicale più semplici e prive di ambiguità, come The Music Notation Project [3]. In figura 5 è possibile notare come quest'ultima sia ad esempio priva di alterazioni, codificando l'altezza della nota unicamente sfruttando la posizione verticale nel rigo.

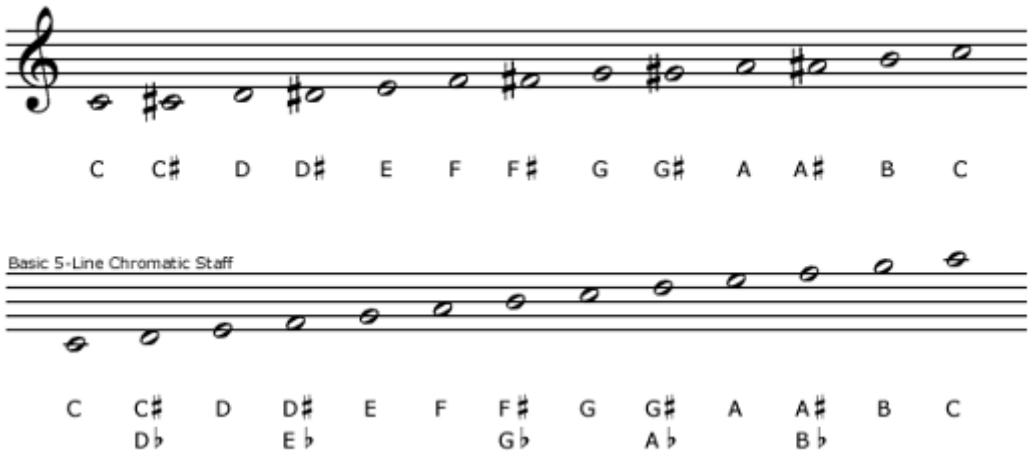


Figura 5: Dalla notazione tradizione ad una senza alterazioni

Come ultima nota prima di addentrarsi nella descrizione delle tecniche di OMR vale la pena constatare come, in maniera simile all'OCR, il riconoscimento di spartiti musicali scritti a mano sia notevolmente più difficile rispetto a quelli stampati [2][4][5]. Questo fatto, che è intuitivo, trova conferma nei test svolti su diversi sistemi di OMR: in presenza di scrittura manuale quasi tutti falliscono oppure forniscono risultati di bassa qualità.

1.4.1 Architettura

Il problema del riconoscimento è, come illustrato in figura 6, comunemente suddiviso in quattro fasi [2]:

- preprocessamento dell'immagine: miglioramento dal punto di vista della qualità e conversione in un formato adeguato alle fasi che seguono;
- riconoscimento dei simboli: estrapolazione dell'informazione musicale dallo spartito finalizzata alla creazione di una lista bidimensionale di simboli primitivi;
- ricostruzione dell'informazione musicale: conversione della lista di simboli primitivi in una descrizione logica e completa del brano;
- costruzione di un modello di notazione musicale: rappresentazione binaria dell'informazione ottenuta al fine di poterla memorizzare.

La terza fase non è presente nel contesto di questo lavoro, poiché la descrizione logica del brano è inserita manualmente, mentre la quarta fase è strettamente legata alla natura della specifica applicazione e soprattutto non presenta particolari sfide tecnologiche. Nei seguenti capitoli verranno perciò trattati in dettaglio i passaggi che compongono preprocessamento e riconoscimento, secondo quello che è lo stato dell'arte.

1.4.2 Preprocessamento

In questa fase esistono diverse tecniche che possono essere combinate, il cui utilizzo dipende in modo significativo dalla qualità dell'immagine di partenza:

- miglioramento del contrasto
- normalizzazione dell'istogramma
- correzione della luminosità
- rimozione del rumore

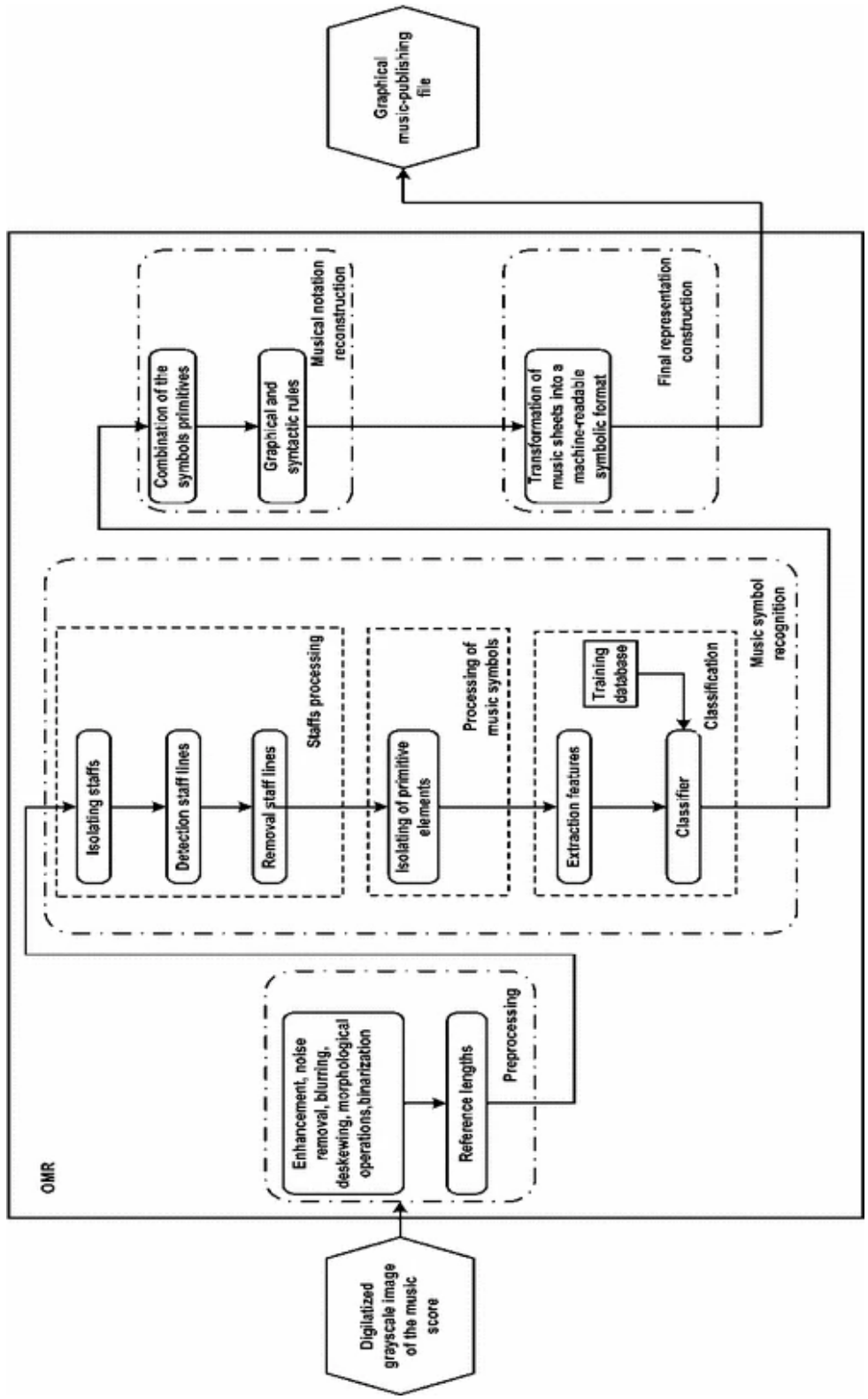


Figura 6: Architettura tipica di un sistema di OMR

- correzione dell'allineamento

La rimozione del rumore, il filtro forse più interessante tra quelli citati, si basa sul principio di sostituire ogni pixel con la media – semplice o pesata – di un certo numero di pixel simili, o per meglio dire identici al netto proprio di questo elemento di disturbo. Esiste una relazione statistica che lega il numero n dei termini della media alla quantità di rumore residua ed è data dalla seguente formula:

$$noise_n = \frac{noise}{\sqrt{n}}$$

Individuando nove pixel simili all'interno dell'immagine è dunque possibile dividere – teoricamente – la quantità di disturbo per tre [6].
La somiglianza è stabilita esaminando non solo il valore del singolo pixel, ma anche di un certo numero di pixel vicini: dimensioni tipiche per questa finestra di ricerca e confronto sono 3×3, 5×5 o 7×7 e le due finestre sono confrontante calcolando l'errore quadratico medio, che deve essere minimo. Il principio che sta alla base di questo metodo è che un pixel è simile solo se contribuisce effettivamente a creare un disegno simile: il singolo valore non ha infatti un particolare significato.
Tecniche meno recenti cercavano valori simili in una regione limitata dell'immagine, ogni pixel oltre una certa distanza dalla posizione corrente non veniva dunque preso in considerazione. Gli autori della tecnica denominata *Non-Local Means Denoising*, che è impiegata all'interno del software, fanno però notare come non ci sia alcun motivo per cui un pixel simile debba anche trovarsi vicino [6]. Si consideri ad esempio la figura 7, dove diversi elementi architetturali sono pressoché identici ma distanti tra loro.

Figura 7: Regioni dell'immagine simili ma distanti tra loro

La fase finale del preprocessamento consiste in un'operazione ben specifica: la produzione di un'immagine binaria. Questo passaggio serve a convertire l'informazione, che fin qui è in scala di grigi, a solo due valori ammissibili: bianco per le regioni di interesse e nero per lo sfondo.
Vi sarebbero algoritmi progettati per lavorare specificatamente sugli spartiti musicali, tuttavia il loro utilizzo è ancora raro [2]. Questa operazione viene invece portata a termine impiegando metodi generici, come ad esempio **Otsu** [2], che non prevede parametri di funzionamento. Nella sua forma più semplice questo algoritmo cerca una soglia che minimizzi la varianza all'interno delle due classi che compongono l'immagine: background e foreground. Oppure che, in maniera maniera del tutto analoga, massimizzi la varianza tra le due classi, rappresentata in figura 8 dalla linea curva in rosso.

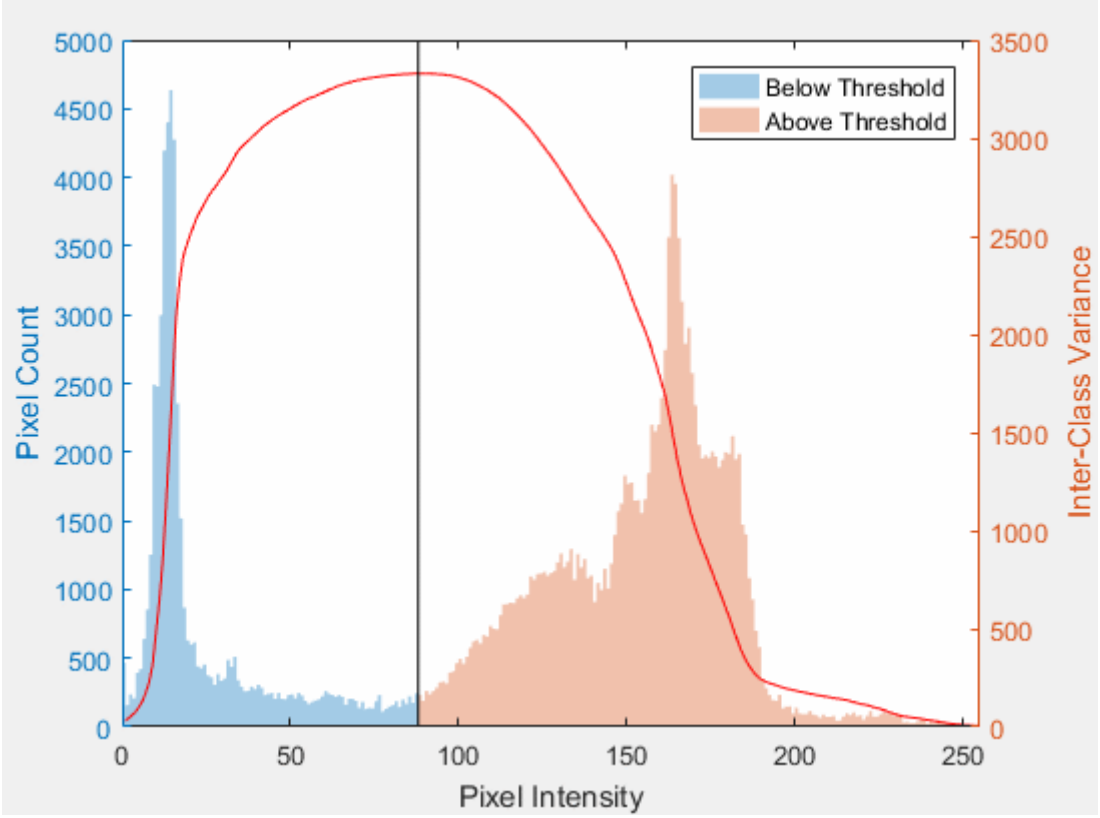


Figura 8: Rappresentazione visuale della ricerca di una soglia tramite Otsu

1.4.3 Stima delle grandezze

Prima di eliminare definitivamente il rigo musicale, procedimento che sarà oggetto del capitolo seguente, vi sono due importanti grandezze da ricavare: dimensione dello spazio bianco tra due linee e spessore di una linea. Questi sono due valori importanti in quanto forniranno un'indicazione sulla dimensione della notazione musicale in generale [7], la quale può variare anche considerevolmente da spartito a spartito ma rimane sempre e comunque proporzionata al rigo. Le due grandezze prendono il nome di *staffspace height* e *staffline height* e sono rappresentate in figura 9.

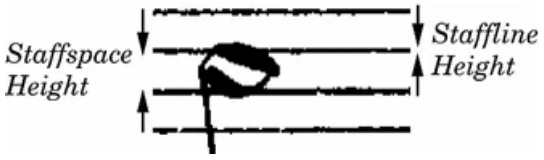


Figura 9: Staffspace height e staffline height

Conservare queste due misure permetterà di utilizzarle in seguito quali parametri per gli algoritmi, evitando in questo modo di ricorrere a numeri fissi – o *magic numbers* – che potrebbero inficiare la qualità del riconoscimento. L'utilizzo di parametri prefissati renderebbe infatti l'intero sistema meno flessibile [2] e dunque meno capace di adattarsi ai diversi formati di stampa.
Il metodo più utilizzato per ottenere queste grandezze si basa sull'utilizzo del noto algoritmo di compressione Run Length Encoding [2], dove le ripetizioni di un determinato valore sono sostituite da una tupla composta dal valore stesso e il numero delle ripetizioni. *6 6 6 A A A A* si traduce dunque in *(6 3) (A 4)*. Essendo però l'input composto solo dall'alternanza dei valori binari 1 e 0 è possibile, dopo aver stabilito per convenzione che il primo valore è sempre 1, omettere la prima parte della tupla e indicare unicamente la lunghezza della ripetizione. Qualora il primo valore fosse invece 0 è sufficiente indicare una prima ripetizione di lunghezza zero. Applicando questo metodo ad ogni colonna ed aggregando i risultati in due istogrammi è possibile identificare lo staffline height come la lunghezza che si presenta più volte per le ripetizioni di nero, viceversa lo staffspace height come la lunghezza più frequente per le ripetizioni di bianco.
Vi sono tuttavia immagini affette da importanti quantità di disturbo, dovuto sia ad artefatti nell'originale che ai processi di acquisizione e binarizzazione. Per ovviare a queste situazioni e rendere l'algoritmo più robusto viene proposto un miglioramento [2]: considerare le ripetizioni a coppie, sia bianco-nero che nero-bianco, mappandone il valore sulla somma delle singole ripetizioni. La somma che appare più di frequente è la stima del valore *staffspace height + staffline height*. Questo processo è illustrato in figura 10.

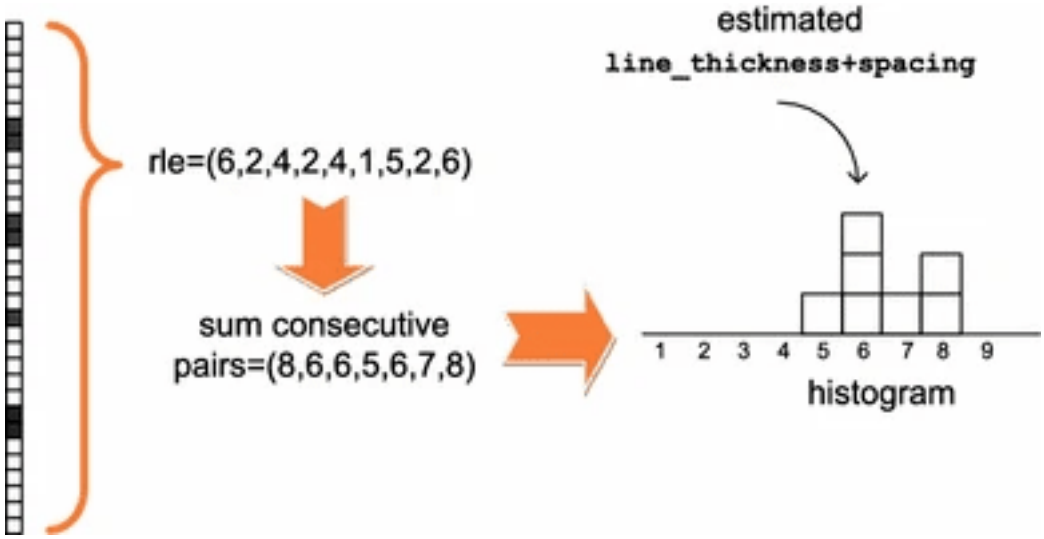


Figura 10: Miglioramento proposto da Cardoso e Rebelo

1.4.4 Rimozione del rigo

La rimozione del rigo rappresenta una fase fondamentale del processo di OMR, in quanto permette di creare spazio vuoto tra i simboli che rimarrebbero altrimenti collegati da questo elemento grafico [2]. Si tratta di una condizione fondamentale in vista della fase successiva, che prevede la segmentazione ovvero l'isolamento dei simboli in regioni dell'immagine indipendenti. Il compito di identificare e rimuovere il rigo è tuttavia potenzialmente complicato da una serie di problematiche: succede infatti che esso non sia orizzontale, che le linee non siano dritte, parallele tra loro o che siano presenti della discontinuità. Potrebbero verificarsi uno o tutti questi problemi insieme, soprattutto – ma non solo – nel caso di spartiti realizzati a mano [8].
Uno degli algoritmi più recenti proposti per portare a termine questo compito sfrutta la proprietà globale più importante del rigo musicale: essere l'unico oggetto nero di grandi dimensioni sul foglio, che si estende dal margine sinistro al margine destro [8]. Si tratta del primo algoritmo a fare uso di una proprietà globale, tutti i lavori precedenti basano il loro funzionamento solo su proprietà locali [2]. Modellando lo spartito come un grafo pesato, dove i nodi rappresentano i pixel ed assegnando i pesi in modo da preferire – gestendo correttamente eventuali discontinuità – il proseguimento del percorso attraverso pixel neri, è possibile individuare le linee del rigo cercando gli *shortest paths* che attraversano il foglio: prima applicando la ricerca partendo da entrambi i lati e infine combinando i due risultati [8].

1.4.5 Segmentazione

La segmentazione dei simboli avviene seguendo un approccio gerarchico: innanzitutto l'immagine viene suddivisa sulla base del rigo, in seguito le sezioni così ottenute sono analizzate in cerca dei simboli presenti al loro interno [2]. In questa fase i diversi elementi primitivi che compongono una nota – cerchio, e quando presenti linea, coda e punto – sono considerati come entità separate, essi saranno uniti a formare un unico simbolo solo nel corso delle fasi successive [2].
Vale la pena notare come alcuni lavori propongano di eseguire segmentazione e classificazione in un unico passaggio, tramite l'utilizzo di modelli di Markov nascosti [2]. Fin'ora questo approccio è stato però applicato solo a spartiti molto semplici, dove vi è un solo simbolo in un ogni istante temporale e non sono presenti legature tra le note [2].

Capitolo 2

Il formato IEEE 1599

2.1 Panoramica

IEEE 1599 è un formato XML nato dall'esigenza di descrivere in maniera coerente i diversi aspetti che compongono un brano musicale: il più importante – nonché quello virtualmente sempre presente – è senz'altro la performance audio[1], tuttavia si possono individuare numerosi altri elementi, come ad esempio:

- immagini digitali degli spartiti
- lista dei simboli musicali
- video musicali ed esibizioni dal vivo
- fotografie e immagini artistiche
- copertina e booklet dell'album
- locandine relative ad eventi
- performance MIDI

Descrivere in maniera completa questi aspetti non è un compito semplice, poiché sono per loro natura fortemente *eterogenei*. Vi è tuttavia un particolare interesse nel rendere accessibili queste informazioni, a dimostrazione di questo fatto si può notare come il numero dei progetti di digitalizzazione nati in ambito musicale sia in continua crescita [1].

[1] Anche più di una nel caso di diverse interpretazioni.

2.2 Layer

IEEE 1599 è strutturato sul concetto di *layer*, di cui ne esistono sei:

- **general**: metadati di varia natura relativi al brano
- **logic**: lista degli eventi
- **structural**: oggetti musicali e relazioni tra loro
- **notational**: rappresentazione grafica degli spartiti
- **performance**: esecuzione tramite linguaggi di performance
- **audio**: tracce audio del brano

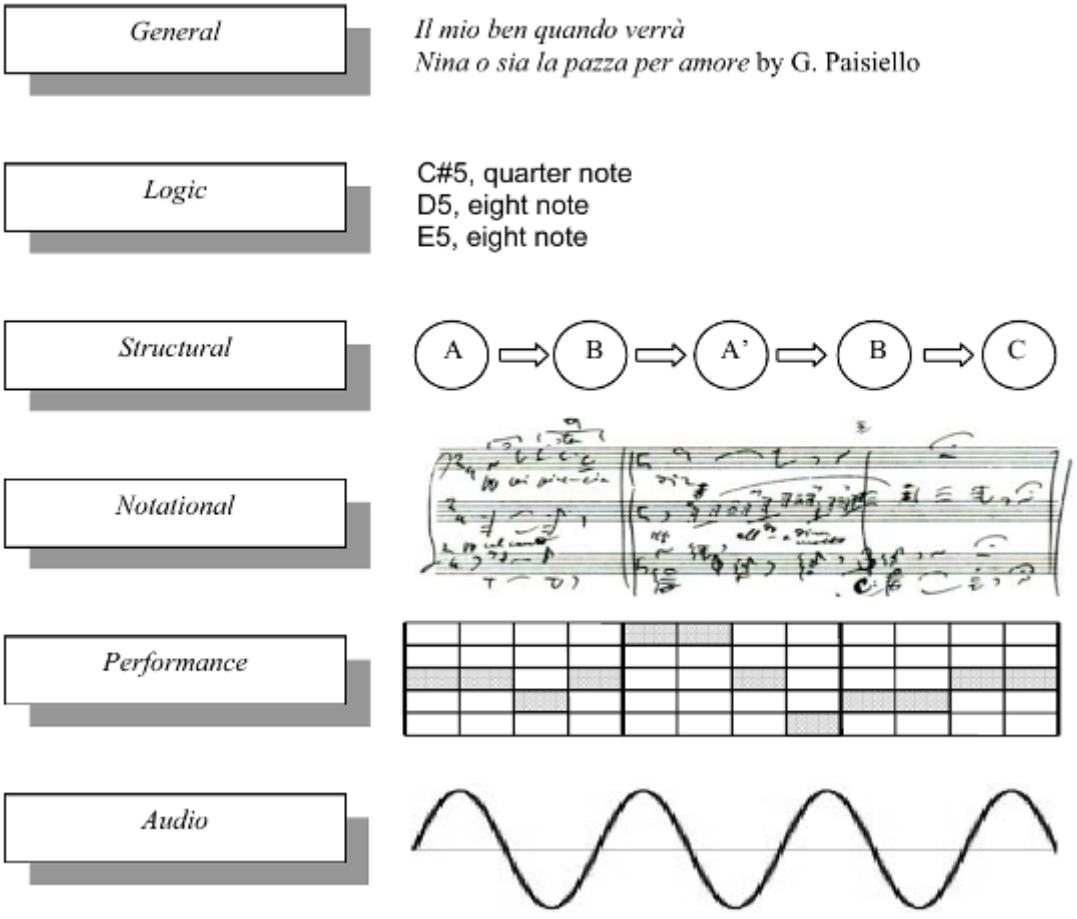


Figura 11: I sei layer che compongono IEEE 1599

Ogni layer permette di definire al suo interno più *istanze*, la cui tipologia dipende naturalmente dallo specifico layer. In *audio* le istanze saranno spesso le registrazioni del brano, in *notational* le immagini degli spartiti, e così via. Questa natura *multi-layer* e *multi-instance* contribuisce a rendere il formato particolarmente flessibile [1].
Layer centrale per questo lavoro è quello notazionale, dove vi è il riferimento alle immagini degli spartiti e le coordinate dei rettangoli che evidenziano i vari simboli musicali. Il layer è organizzato come gerarchia dei seguenti elementi XML, enumerati dal più esterno al più interno:

- **notational**: radice della gerarchia.
- **graphic_instance_group**: contenitore logico per le diverse pagine che compongono lo spartito. L'unico attributo presente è *description*, una descrizione testuale che sarà visibile nei software che supportano IEEE 1599.
- **graphic_instance**: riferimento al file immagine della pagina. Gli attributi sono il percorso relativo *file_name*, la doppietta *file_type* e *file_format* che descrive il tipo di codifica, *position_in_group* dove è riportato il numero della pagina ed infine *measurement_unit*, quasi sempre pixel.
- **graphic_event**: simbolo di interesse. Vi sono l'attributo *event_ref* che fa riferimento all'identificatore univoco dell'evento musicale e le quattro coordinate per il disegno dell'area rettangolare: *upper_left_x*, *upper_left_y*, *lower_right_x* e *lower_right_y*.

Anche questo layer supporta un numero qualsiasi di istanze: è possibile specificare tanti *graphic_instance_group* quanti sono gli spartiti che si vogliono includere. Potrebbe trattarsi di versioni diverse del medesimo spartito, così come di più spartiti relativi ai singoli strumenti musicali: anche qui il formato lascia all'utilizzatore massima libertà.
Si può notare come le immagini che compongono uno spartito risiedano nel filesystem insieme al file XML stesso; questo si applica anche agli oggetti multimediali negli altri layer. Pietra miliare del formato è infatti il voler descrivere unicamente le informazioni del brano nel suo complesso, demandando la memorizzazione dei singoli oggetti multimediali al gran numero di formati esistenti [9].

2.3 Sincronizzazione

Molte istanze hanno una natura temporale e prima di essere fruite devono essere opportunamente sincronizzate. Si pensi ad esempio a uno spartito e alla performance audio che esso descrive: le note sono suonate in un momento ben preciso

dell'esecuzione. In questo caso tramite sincronizzazione è possibile collegare una regione rettangolare dello spartito ad un preciso momento temporale della traccia audio; il risultato di questa operazione è visibile in figura 12.

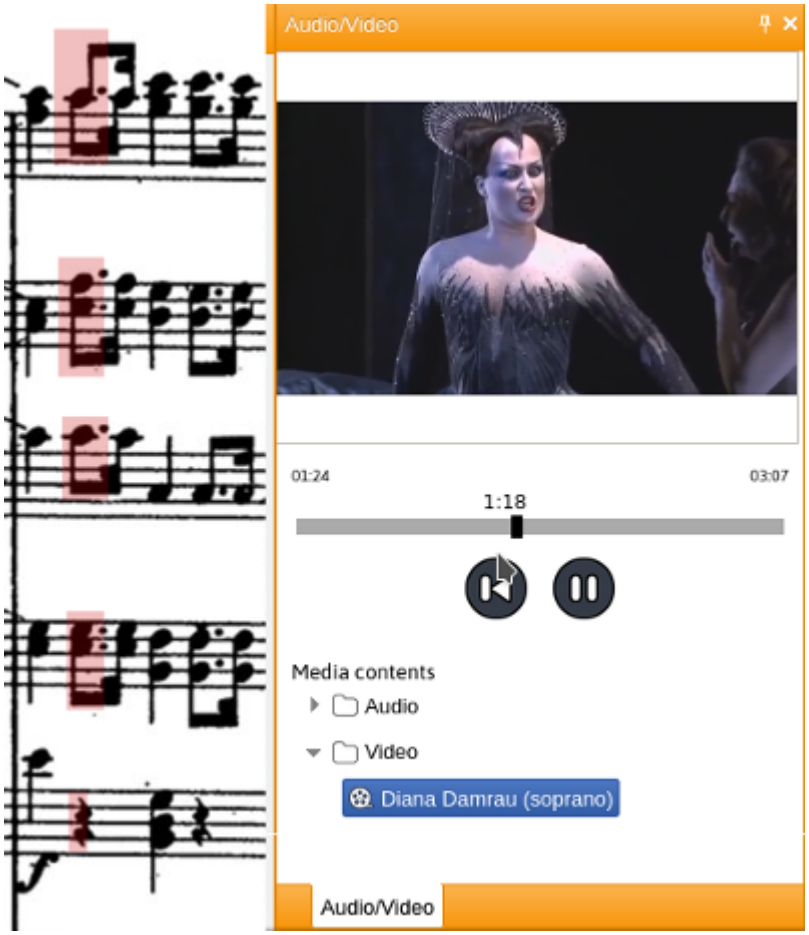


Figura 12: Sincronizzazione tra spartito musicale e performance

Questo è un altro pilastro fondamentale del formato IEEE 1599: supporto esteso alla sincronizzazione dei contenuti *time-based* [9] tramite la definizione di *eventi* e *occorrenze*. Concetto centrale è lo *spine*, una lista ordinata di eventi definita nel layer logico. Gli eventi possono poi essere richiamati in qualsiasi altro layer per mezzo di un identificatore univoco. La definizione di uno spine è illustrata nell'esempio seguente:

```
<ieee1599>
    <logic>
        <spine>
```

```
            <event id="e0" timing="0" hpos="0"/>
            <event id="e1" timing="2" hpos="2"/>
            <event id="e2" timing="2" hpos="2"/>
        </spine>
    </logic>
</ieee1599>
```

È importante notare come un evento possa comparire in layer diversi, più volte nello stesso layer così come più volte nella medesima istanza. Questa flessibilità permette di ottenere tre diversi tipi di sincronizzazione, le prime due di natura *intra-layer* mentre la terza *inter-layer*:

- **Interna**: le occorrenze di eventi diversi formano la descrizione della traccia in un solo layer e una sola istanza. È il caso delle note di un brano, dove è presente un solo spartito oppure gli spartiti non hanno note in comune.
- **Locale**: un evento ha più occorrenze all'interno del medesimo layer, ma in istanze diverse. È il caso della medesima nota riportata su versioni differenti dello spartito, ad esempio una scritta a mano e una stampata.
- **Globale**: un evento occorre in layer diversi. È il caso dell'esempio riportato in figura 12, dove le occorrenze mettono in collegamento il layer notazionale – lo spartito – e il layer performance – la registrazione a teatro –.

Un esempio di sincronizzazione locale è illustrato in figura 13.

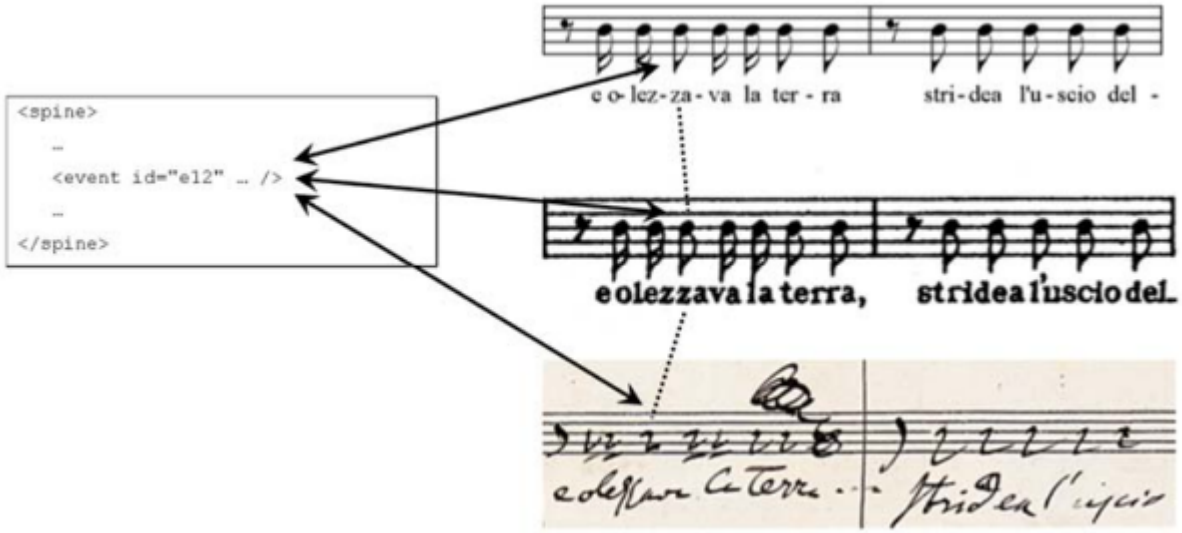


Figura 13: Tre spartiti fanno riferimento alla stessa nota

Combinando questi aspetti è possibile creare una struttura interconnessa di descrizioni per gli eventi musicali, che prende la forma di una rete semantica locale [10].

Capitolo 3

Implementazione

3.1 Tecnologie utilizzate

Una fase fondamentale di ogni progetto è la scelta degli strumenti di sviluppo. Questa considerazione si applica naturalmente ad ogni ambito: un falegname dovrà avere cura di scegliere strumenti adeguati e di buona qualità, pena la pessima riuscita del lavoro.
In questo capitolo saranno elencati i software scelti per lo sviluppo dell'applicazione e la redazione dell'elaborato. Gli unici vincoli veri e propri sono due: lo sviluppo del progetto in Java, perché il software ufficiale IEEE 1599 è scritto in questo linguaggio, e l'utilizzo di LaTeX, poiché ogni documento è stato redatto con questo strumento.

3.1.1 Sistema operativo

Il sistema operativo scelto è Kubuntu 18.04 LTS. L'utilizzo di un sistema Linux per lo sviluppo di applicazioni porta con sé una serie di vantaggi:

- il gestore dei pacchetti permette di installare i software necessari al processo di sviluppo in maniera pressoché istantanea;
- il linking delle librerie native è fortemente facilitato poiché installazione e aggiornamento sono a cura dal sistema operativo;
- Bash è un ambiente di scripting estremamente potente e ricco di tool di ogni genere, difficilmente sarà necessario implementare qualcosa manualmente;
- documentazione ufficiale, tutorial e domande e risposte a quesiti tecnici sono reperibili in gran numero con una semplice ricerca.

La scelta di utilizzare KDE rispetto a Gnome non ha invece motivazioni tecniche ma è dovuta semplicemente ad una questione di preferenza personale.

3.1.2 Versioning dei sorgenti

Il processo di sviluppo si è articolato in più sedi, per questo si è reso necessario utilizzare un sistema di versioning nel cloud. La scelta è caduta su GitHub, risorsa estremamente popolare e supportata egregiamente dai principali ambienti di sviluppo. Il fatto che sia costruito su Git ha inoltre il vantaggio che operazioni particolari possano essere svolte da terminale: in un caso si è reso necessario utilizzare il comando *bisect* per individuare e correggere un bug.

3.1.3 Ambiente di sviluppo

Come discusso più avanti nel paragrafo 3.3 il processo di sviluppo si è articolato su due fasi ben distinte. Per la prima è stato scelto l'ambiente di sviluppo IntelliJ IDEA. Il motivo principale sono le conoscenze pregresse, tuttavia vale la pena notare come esso sia anche l'IDE Java più utilizzato dagli sviluppatori. Questo dato è riportato nel grafico visibile in figura 14.

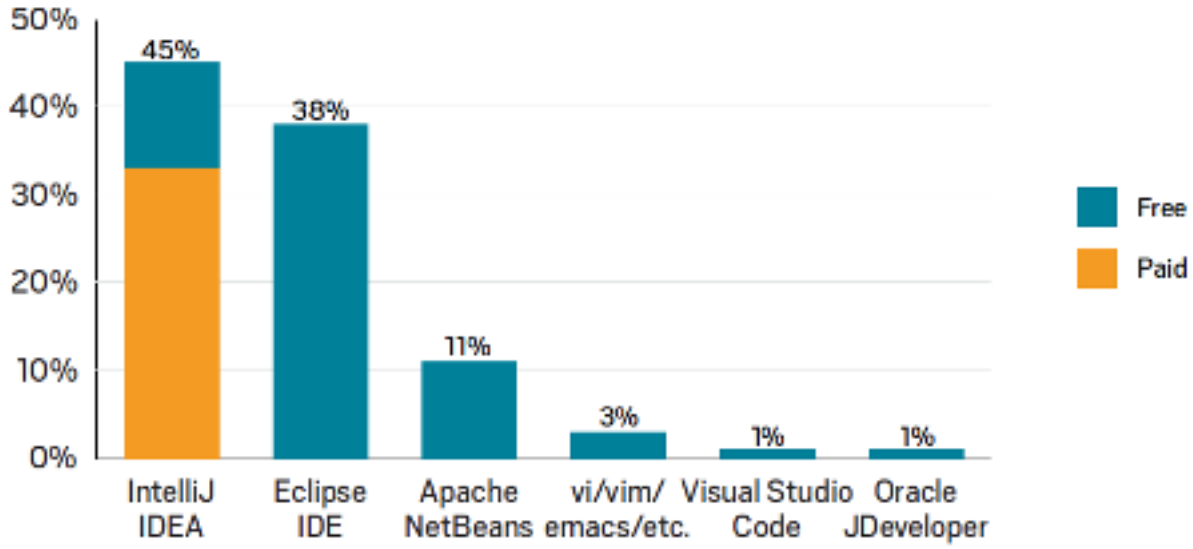


Figura 14: Percentuali di utilizzo dei vari IDE per Java [11]

La seconda fase è stata invece portata a termine in Apache Netbeans, l'ambiente dove viene già sviluppato e mantenuto da parte del Laboratorio il software ufficiale IEEE 1599 per la produzione dei contenuti.

3.1.4 Gestione delle dipendenze

Il software per il riconoscimento si appoggia su alcune librerie esterne. Il compito di reperirle e mantenerle aggiornate è stato demandato a Maven, dotato di un repository centrale ricco di pacchetti Java pronti all'uso. Non essendo necessarie particolari feature è stato preferito a Gradle, in quanto giudicato sensibilmente più semplice da utilizzare.

3.1.5 Versione di Java

Trattandosi di un nuovo progetto, far uso di caratteristiche recenti del linguaggio è sembrato un approccio assolutamente sensato. La versione minima di Java è stata fissata alla release 9, resa disponibile nel corso del 2017.
Di particolare utilità è stato giudicato il riconoscimento automatico del tipo per le variabili locali, che permette di rendere il codice decisamente più conciso in un gran numero di situazioni. L'esempio seguente è estrapolato direttamente dai sorgenti del software ufficiale[1]:

```
for (Map.Entry<String, ArrayList<MappedRectangle>> entry :
 mainPanel.mappedRectangles.entrySet())
 if (entry.getValue() != null)
  for (MappedRectangle mappedR : entry.getValue())
   ...
}
```

diventa sensibilmente più leggibile con:

```
for (var entry : mainPanel.mappedRectangles.entrySet())
 if (entry.getValue() != null)
  for (var mappedR : entry.getValue())
   ...
}
```

Una caratteristica decisamente più interessante e corposa sono le Stream API, che portano nel linguaggio il concetto di programmazione funzionale. Una trattazione della programmazione funzionale è al di fuori degli scopi di questo elaborato, ci si limiterà dunque a notare come essa permetta di risolvere una classe di problemi in modo decisamente più efficace rispetto ad un'analoga soluzione imperativa. Un esempio di utilizzo reale è riportato nel sottoparagrafo 3.6.1.

[1] Linea 1086 di IEEEGraphicMapper.java.

3.1.6 Framework per la Computer Vision

Per via dei vincoli di progetto la scelta non poteva che ricadere su OpenCV: si tratta di una libreria C++ che tuttavia grazie al proliferare di Android include anche un supporto ufficiale a Java.
Si è ipotizzato anche di reimplementare gli algoritmi utilizzati in linguaggio Java, tuttavia per ragioni di performance – è risaputo come la computazione ad alte prestazioni non sia tra i punti forti del linguaggio, senza contare che OpenCV fa largo uso di OpenCL per ottimizzare ulteriormente i tempi di esecuzione – e del tempo decisamente lungo che questo avrebbe richiesto, si è deciso immediatamente di non procedere su questa strada.

3.2 Processo di sviluppo

In una serie di incontri preliminari con i relatori sono stati definiti i seguenti aspetti:

- natura dettagliata del progetto e dei requisiti
- forma che il software avrebbe dovuto prendere
- selezione di paper sull'OMR ritenuti più pertinenti
- strumenti di sviluppo utilizzati dal laboratorio
- algoritmi da utilizzare per il riconoscimento

Dopo questa fase iniziale il lavoro è stato puramente implementativo e si è articolato nei seguenti passi, ripetuti ciclicamente:

1. definizione delle funzionalità da implementare nel corso delle settimane successive;
2. lavoro di sviluppo individuale basato su quanto discusso nel punto 1 ed eventualmente 3;
3. convalida del lavoro svolto e raccolta delle richieste di modifica e miglioramento riguardo interfaccia e funzionalità.

Per il punto 2 è stata fissata una durata indicativa di 2-3 settimane, in accordo con quanto stabilito dal framework di processo *Scrum* per la durata di uno sprint.

3.3 Scelte generali

Nel paragrafo 1.4 è stata fornita una panoramica dettagliata su quello che è lo stato dell'arte di un sistema di OMR, dalla struttura generale agli algoritmi più recenti che animano i singoli componenti. Appare evidente di come si tratti di un sistema complesso, la cui implementazione avrebbe richiesto un lungo lavoro di debugging e ottimizzazione. Prima di imbarcarsi in un'impresa di questa portata si è voluto tentare un approccio alternativo: valutare potenzialità e limiti di una soluzione più snella, basata unicamente su una sequenza ben studiata di **filtri immagine e operazioni morfologiche**[2].
Questo modo di procedere ha avuto origine anche da una seconda importante motivazione: nel capitolo 1 si è discusso di come il progetto mirasse a sviluppare un sistema di OMR informato, più semplice dunque rispetto a una soluzione tradizionale in quanto privo della componente di classificazione, particolarmente complessa [2] e che richiede che non via sia alcuna distorsione o erosione dei simboli musicali presenti sullo spartito. Fenomeni di questo tipo potrebbero verificarsi impiegando un numero elevato di filtri immagine, ma non pregiudicano la bontà di un sistema informato: una nota potrebbe benissimo diventare un simbolo indefinito, fintantoché le sue dimensioni rimangono grossomodo invariate e che ogni pixel rimane raggiungibile dagli altri, formando un solo *connected component*.
Solo se questo approccio si fosse rivelato inadeguato si sarebbe lavorato ad un sistema tradizionale, basato sugli algoritmi discussi nel paragrafo 1.4. In caso contrario il lavoro sarebbe stato poi integrato nel software del Laboratorio, come è effettivamente avvenuto. Per effettuare questa valutazione è stato deciso lo sviluppo di un programma dedicato denominato FCal, che permettesse di sperimentare con i filtri immagine messi a disposizione da OpenCV mostrando i cambiamenti in tempo reale. Il progetto si compone dunque di due parti distinte: FCal, e la nuova versione del plugin GraphicMapper del software IEEE 1599, dove avviene la selezione delle note presenti sugli spartiti. Nei capitoli seguenti verranno analizzate entrambe le componenti.

3.4 User stories

Le funzionalità dal punto di vista dell'utente verranno enunciate sulla base del formalismo proposto da Mike Cohn per le user stories di Scrum [12]:

> As a (role) I want (something) so that (benefit).

[2] D'ora in poi solo filtri immagine.

In questo caso il ruolo verrà però omesso, poiché per FCal si tratta sempre della persona incaricata di individuare una sequenza di filtri adeguata, mentre per GraphicMapper esso è sempre il creatore di contenuti che si occupa di selezionare i simboli musicali sugli spartiti.

3.4.1 FCal

- Voglio scegliere da un ampio spettro di filtri immagine e operazioni morfologiche, così da poter individuare la combinazione che produca la mappa binaria migliore.
- Voglio variare i parametri dei filtri tramite scorciatoie da tastiera e visualizzare il risultato in tempo reale, così da poter sperimentare rapidamente nuove combinazioni.
- Voglio studiare l'effetto dei filtri su più spartiti contemporaneamente, così da poter trovare il miglior compromesso possibile nelle diverse condizioni di stampa.
- Voglio visualizzare la mappa binaria sovrapposta all'immagine di partenza, così da poter avere un'idea precisa sulla bontà del riconoscimento.
- Voglio selezionare i simboli tramite un click del mouse, così da poter essere sicuro che la bounding box venga disegnata nella posizione corretta e con le giuste dimensioni.
- Voglio salvare e ricaricare la lista dei filtri, così da poter proseguire il lavoro in un secondo momento nonché poter inizializzare automaticamente il plugin GraphicMapper.
- Voglio modificare manualmente la lista dei filtri in un editor di testo, così da poter operare correzioni minori senza bisogno di aprire FCal.

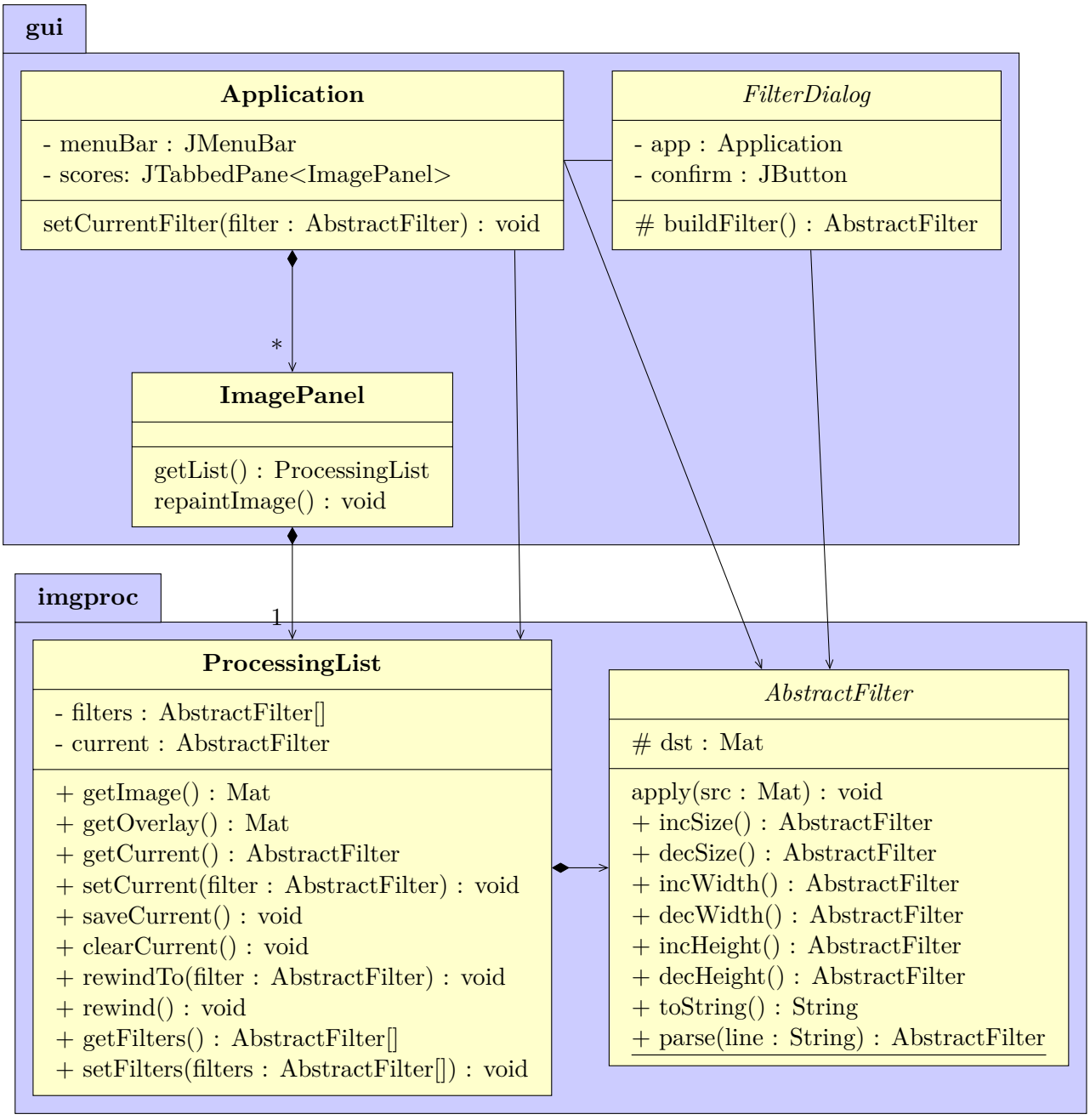
3.4.2 GraphicMapper

- Voglio selezionare note e pause con un click su una parte qualsiasi del simbolo, così da poter creare più rapidamente nuovi contenuti.
- Voglio lavorare nel contesto del plugin esistente, così da poter utilizzare le operazioni già implementate.
- Voglio visualizzare un margine tra la bounding box e il simbolo musicale, cosi da poter ottenere una migliore resa grafica.

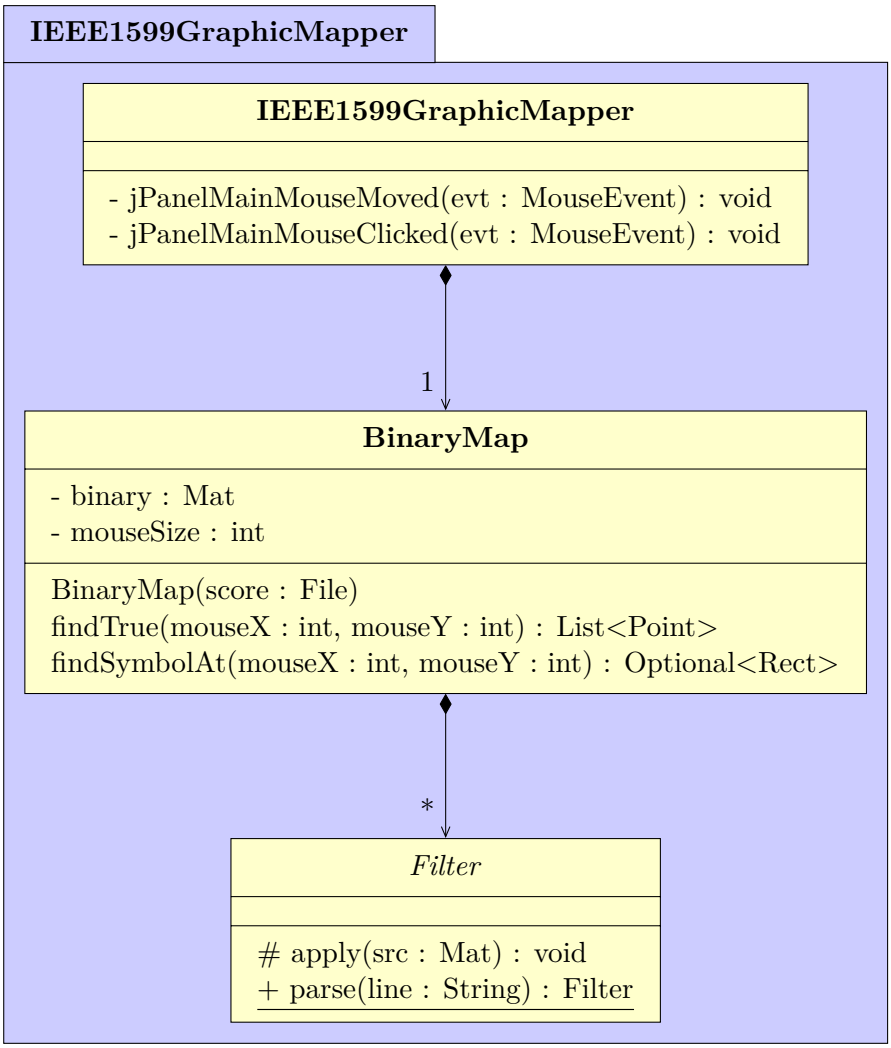
- Voglio visualizzare una barra di progresso durante l'applicazione dei filtri, così da avere un'idea del tempo rimanente.

3.5 Diagramma del classi

3.5.1 FCal



3.5.2 GraphicMapper



3.6 Codice sorgente

3.6.1 FCal

La lista dei filtri è memorizzata in un normale file di testo, uno per riga. La riga è composta dal nome della classe che implementa il filtro, uno spazio e la lista dei parametri in formato JSON:

```
Negative {}
Denoising {"strength":10}
Morphology {"type":"TopHat","shape":"Rect","width":25,"height":1}
```

```
Morphology {"type":"Close","shape":"Rect","width":1,"height":7}
Threshold {"type":"Otsu","threshold":127}
FillHoles {}
Morphology {"type":"Open","shape":"Rect","width":1,"height":5}
```

Salvataggio e caricamento avvengono tramite reflection per quanto concerne il nome della classe, rispettivamente utilizzando la libreria Gson per la lista dei parametri:

```
@Override
public String toString() {
    return getClass().getSimpleName() + " " + new Gson().toJson(this);
}

public static AbstractFilter parse(String line) {
    var split = line.split(" ");
    try {
        var cls = Class.forName(AbstractFilter.class.getPackageName()
        + "." + split[0]);
        return (AbstractFilter) new Gson().fromJson(split[1], cls);
    }
    catch(ReflectiveOperationException e) {
        throw new RuntimeException(e);
    }
}
```

3.6.2 GraphicMapper

In risposta al movimento del mouse viene mostrata una porzione ingrandita della mappa binaria, centrata in corrispondenza della nuova posizione. Nel caso in cui il puntatore si trovi su un pixel bianco, o ve ne sia uno nelle immediate vicinanze, la rappresentazione del puntatore viene disegnata in verde, in caso contrario in rosso:

```
synchronized BufferedImage getScaledPopup(int x, int y, Dimension size) {
    // ritaglia una regione intorno alla posizione del mouse
    var ul = new Point(
        max(x * scaling - size.width / 2, 0),
        max(y * scaling - size.height / 2, 0)
    );
    var br = new Point(
        min(x * scaling + size.width / 2, scaled.width() - 1),
        min(y * scaling + size.height / 2, scaled.height() - 1)
```

```
    );
    var rect = new Rect(ul, br);
    var lens = scaled.submat(rect).clone();
    // aggiungi spazio vuoto se il mouse è vicino ai bordi
    if(lens.width() < size.width) {
        var fill = new Mat(lens.height(), size.width - lens.width(),
        lens.type(), all(0xFF));
        hconcat(x < binary.width() / 2 ? List.of(fill, lens) :
        List.of(lens, fill), lens);
    }
    if(lens.height() < size.height) {
        var fill = new Mat(size.height - lens.height(), lens.width(),
        lens.type(), all(0xFF));
        vconcat(y < binary.height() / 2 ? List.of(fill, lens) :
        List.of(lens, fill), lens);
    }
    // disegna il puntatore del mouse al centro della lente
    cvtColor(lens, lens, Imgproc.COLOR_GRAY2BGR);
    var color = findTrue(y, x).size() > 0 ? GREEN : RED;
    circle(lens, new Point(lens.width() / 2, lens.height() / 2),
    mouseSize, color, 2);
    return (BufferedImage) toBufferedImage(lens);
}
```

In seguito ad un click del mouse viene invocato il metodo *findTrue*, che restituisce la lista di pixel positivi nella immediate vicinanze. Per migliorare l'esperienza utente il puntatore del mouse viene considerato avente dimensione quadrata di alcuni pixel per lato, anziché puntiforme: in questo modo anche un click che cade leggermente al di là del simbolo dà comunque origine all'evento di selezione, richiedendo all'utente meno precisione in fase di selezione.
Dopo aver ordinato la lista in base alla distanza dalla posizione reale del mouse, sul pixel più vicino viene invocato l'algoritmo FloodFill di OpenCV: la bounding box così ottenuta – e opportunamente allargata, secondo quanto discusso nel sottoparagrafo 3.4.2 – rappresenta il simbolo musicale riconosciuto:

```
synchronized Optional<Rect> findSymbolAt(int mouseX, int mouseY) {
    // usa l'occorrenza più vicina al puntatore del mouse
    return findTrue(mouseY, mouseX).stream().sorted((p1, p2) -> {
        var d1 = sqrt(pow(p1.x - mouseX, 2) + pow(p1.y - mouseY, 2));
        var d2 = sqrt(pow(p2.x - mouseX, 2) + pow(p2.y + mouseY, 2));
        return Double.compare(d1, d2);
    // esplora e contorna il simbolo
```

```
    }).findFirst().map(point -> {
        var rect = new Rect();
        // preserva la mappa originale per eliminare e aggiungere
        nuovamente lo stesso simbolo
        Imgproc.floodFill(this.binary.clone(), new Mat(), point,
        Scalar.all(0), rect);
        return enlarge(rect);
    });
}
```

3.7 Filtri e parametri

Individuare una sequenza adeguata di filtri immagine è stata una parte importante e onerosa del processo di sviluppo. Seguendo tuttavia un approccio incrementale, basato sullo stato attuale dell'immagine, è stato possibile – filtro dopo filtro – produrre una mappa binaria dei simboli di qualità soddisfacente:

1. rigo musicale e simboli in bianco, sfondo in nero: **Negative**
2. rimozione del rumore, se presente: **Denoising 10×**
3. binarizzazione dell'immagine: **Otsu**
4. rimozione del rigo musicale: **Morphological TopHat 25×1**
5. ricostruzione delle note tagliate dal rigo: **Morphological Closing 1×7**
6. riempimento delle note vuote: **FillHoles**
7. rimozione dei residui del rigo, separazione orizzontale delle note, eliminazione di legature e altri simboli minori: **Morphological Opening 1×5**

3.7.1 Descrizione

Il risultato dell'applicazione del primo filtro – l'inversione dei colori – su uno degli spartiti del dataset è visibile in figura 18. La trasformazione avviene applicando la seguente operazione ad ogni pixel:

$$pixel = BitwiseNot(pixel)$$

La rimozione del rumore utilizza l'algoritmo *non-local means denoising*, precedentemente discusso nel sottoparagrafo 1.4.2. Non essendo presente alcun artefatto il

risultato rimane in questo caso invariato, come mostrato in figura 19.
La binarizzazione utilizza Otsu, algoritmo precedentemente trattato nel sottoparagrafo 1.4.2, mentre il risultato dell'operazione è visibile in figura 20.
Prima di trattare la trasformazione *TopHat* o l'apertura morfologica, su cui essa si basa, è necessario discutere delle due operazioni morfologiche fondamentali: *erosion* e *dilation*. Queste due operazioni fanno uso di un argomento denominato *structuring element*, una forma geometrica 2D – che può essere di qualsiasi tipo, ma è spesso rettangolare o circolare – che viene fatta scorrere lungo l'immagine e in base alla quale viene deciso il nuovo valore di ogni pixel. Si consideri, per l'erosione, l'illustrazione in figura 15: un pixel viene eroso se una parte qualsiasi dello structuring element, dopo essere stato centrato su di esso, ricade sul background. Analogamente, nel caso della dilatazione un pixel diventa parte del foreground se una parte dello structuring element ricade su di esso, come illustrato in figura 16.

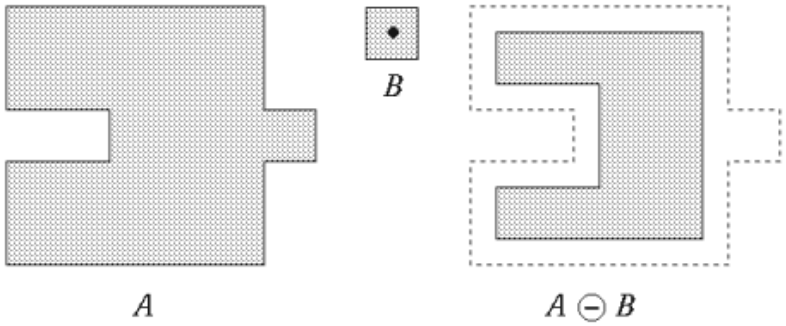


Figura 15: Erosione di A con structuring element B quadrato

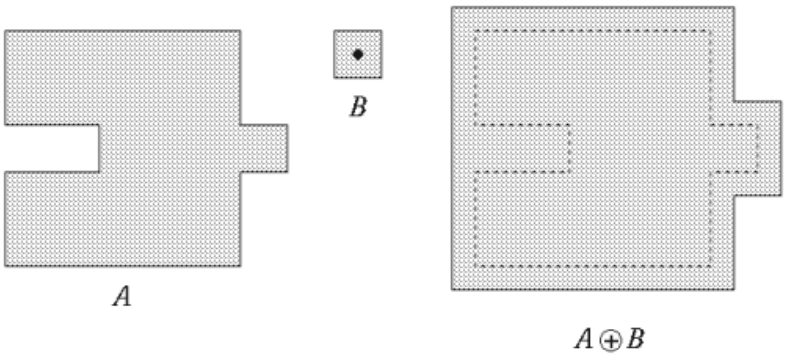


Figura 16: Dilatazione di A con structuring element B quadrato

Le due trasformazioni possono essere combinate per creare operazioni composite, come nel caso dell'*opening*: tramite erosione è possibile rimuovere piccoli

oggetti che non raggiungono le dimensioni dello structuring element, mentre la dilatazione applicata in seguito permette di ricostruire forma e dimensione degli elementi che sono stati invece preservati:

$$O_s(I) = D_s(I) \circ E_s(I)$$

Nel caso della trasformazione TopHat l'apertura morfologica viene sottratta all'immagine di partenza, rendendo possibile rimuovere in un colpo solo tutti gli elementi che abbiano una dimensione pari almeno allo structuring element, come appunto il rigo musicale. In figura 21 è visibile il risultato dell'operazione. La formula della trasformazione è la seguente:

$$T_s(I) = I - O_s(I)$$

La chiusura morfologica consiste in una dilatazione seguita da erosione, ed è utile per riempire dei buchi senza però alterare forma e dimensione degli altri elementi presenti nell'immagine. In figura 22 è visibile il risultato dell'operazione. La sua formula è:

$$C_s(I) = E_s(I) \circ D_s(I)$$

Il riempimento delle note vuote – necessario per rendere cliccabile anche l'interno di questa tipologia di simboli – è un filtro composito creato specificatamente per questo progetto. Si basa ancora una volta sull'algoritmo FloodFill di OpenCV: esso viene applicato su una copia della mappa binaria, partendo da un pixel qualsiasi del background. In questo modo tutto, tranne l'interno delle note vuote, diventa parte del foreground. Applicando ora un *bitwise not* all'intera immagine e combinandola con lo spartito tramite un *bitwise or* è possibile portare a termine il riempimento. In figura 22 è visibile il risultato dell'operazione. Le formule di questo filtro sono:

$$I_0 = Clone(I)$$

$$I_0 = FloodFill(Image = I_0; SeedAt = (0, 0))$$

$$I_0 = BitwiseNot(I_0)$$

$$I = BitwiseOr(I, I_0)$$

Il risultato dell'apertura morfologica – l'ultimo filtro necessario alla produzione della mappa binaria – è visibile in figura 23.

Figura 17: Immagine di partenza

Figura 18: Inversione dei colori

Figura 19: Rimozione del rumore

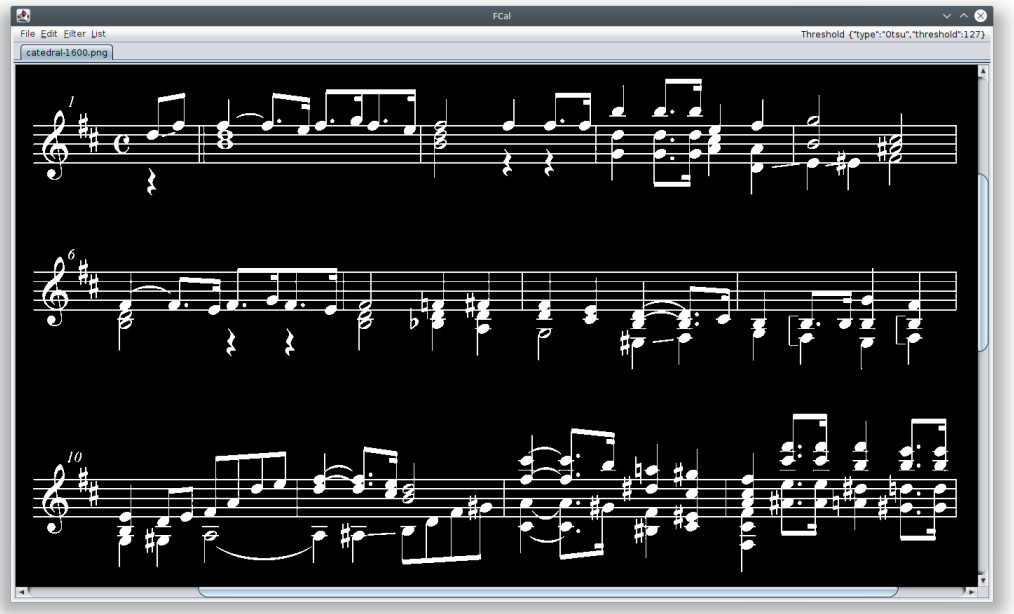


Figura 20: Binarizzazione dell'immagine

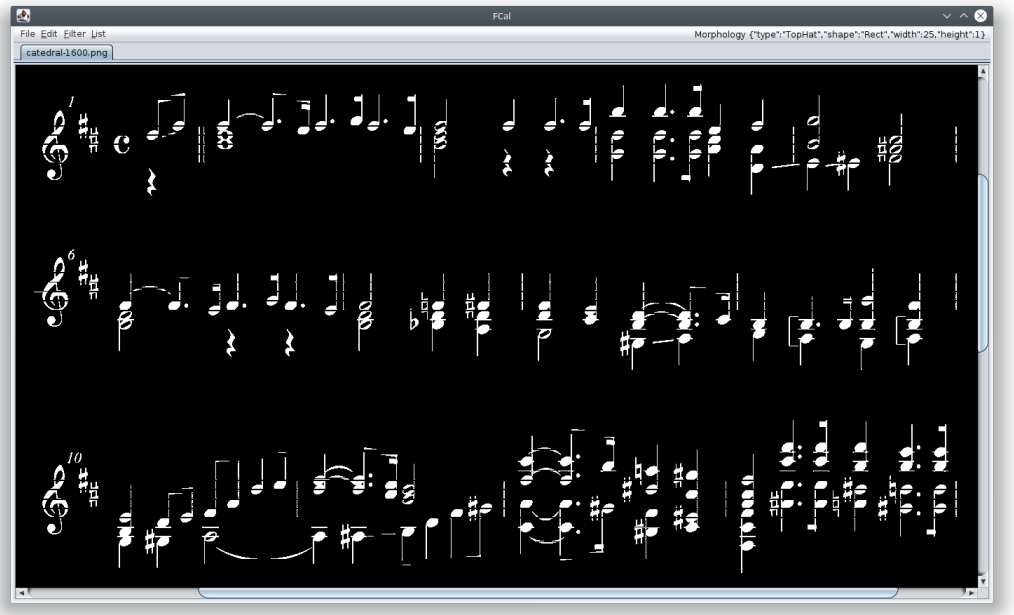


Figura 21: Rimozione del rigo musicale

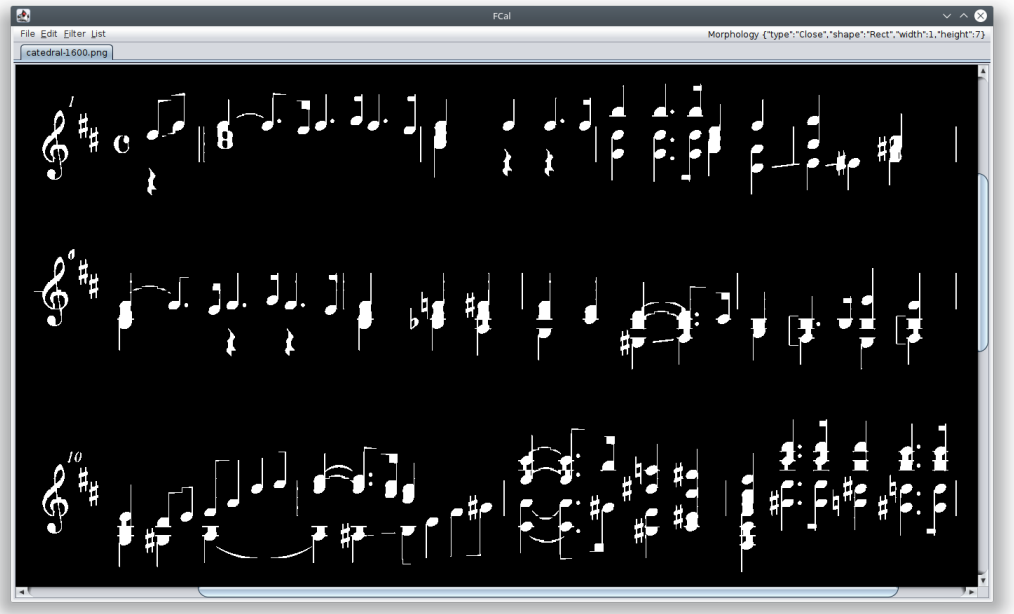


Figura 22: Ricostruzione verticale dei simboli e riempimento delle note vuote

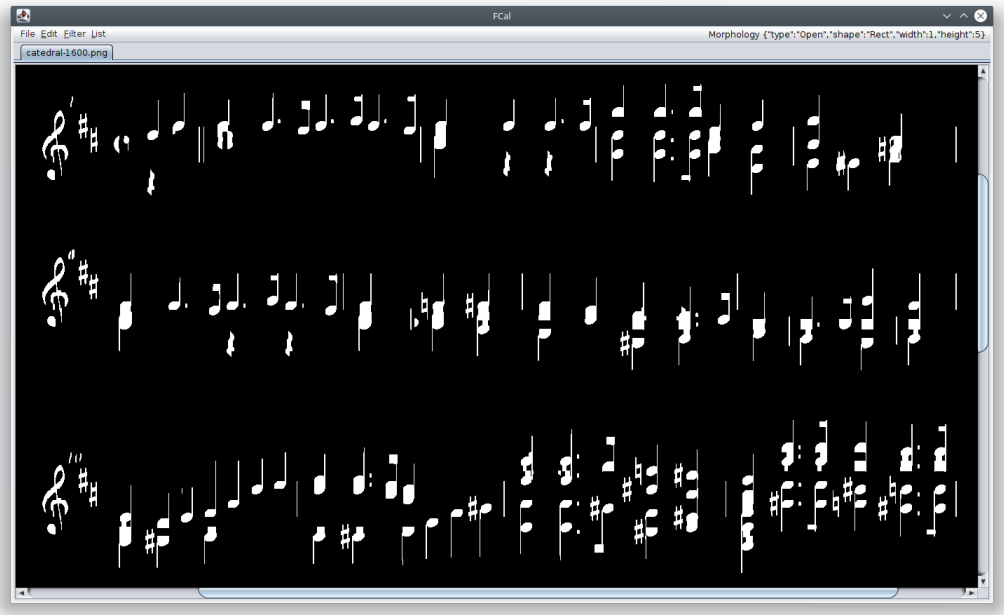


Figura 23: Separazione orizzontale dei simboli

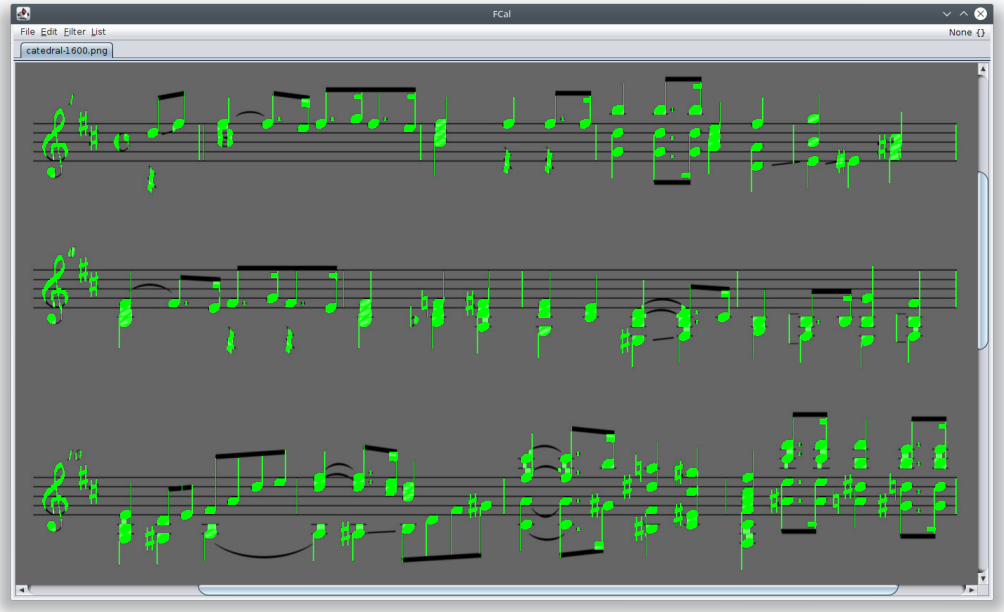


Figura 24: Mappa binaria sovrapposta allo spartito

Capitolo 4

Test

4.1 Metodologia

Al fine di mettere alla prova il riconoscimento informato sono stati selezionati un certo numero di spartiti dall'archivio ufficiale dei contenuti IEEE 1599[1]. Dopo uno primo test di riferimento con lo spartito *Andante religioso*, dove le condizioni di stampa sono ideali, la selezione è stata operata con l'intento di differenziare il più possibile le condizioni di test e attribuire così maggiore spessore ai risultati finali. Per ogni spartito sono state scelte due righe, rappresentative dell'intero contenuto ma al tempo stesso dotate di peculiarità interessanti: non si è ritenuto necessario analizzare l'intero spartito, spesso infatti le righe sono molto simili o addirittura si ripetono.
Nel paragrafo 4.3 sono elencati i motivi che hanno portato alla scelta di uno spartito e rispettivamente di una particolare riga: così facendo al lettore sono possibili considerazioni su questo particolare aspetto del protocollo di test. I risultati sono organizzati in forma tabellare, secondo tre categorie:

- simboli riconosciuti;
- simboli parzialmente riconosciuti, ovvero aggregati in un unico simbolo oppure al contrario per i quali la bounding box ha una dimensione insufficiente, andando a tagliarne una parte;
- simboli non riconosciuti, ovvero totalmente assenti dalla mappa binaria.

[1] `https://ieee1599.lim.di.unimi.it/music_collection.php`

4.2 Metriche considerate

La metrica più importante da considerare è senz'altro il rapporto tra i simboli riconosciuti e la totalità dei simboli presenti su uno spartito: è questo numero infatti che ci informa su quanti click dell'utente siano effettivamente andati a buon fine. Assume tuttavia una certa importanza anche analizzare i mancati riconoscimenti: sarebbe infatti auspicabile, in questo caso, che vi fosse almeno un riconoscimento parziale, poiché questa situazione è rimediabile con la semplice pressione di un tasto. Si consideri il caso di più simboli aggregati in uno solo: con una scorciatoia da tastiera è possibile dividere in due la bounding box e rimediare immediatamente all'errore. Un numero elevato di riconoscimenti totalmente mancati indicherebbe invece un sistema particolarmente rigido, caratteristica questa che un utilizzatore potrebbe senz'altro trovare problematica.
Non sono stati analizzati i tempi di esecuzione, in quanto essi sono piuttosto brevi – 2-3 secondi per l'applicazione di tutti i filtri – e soprattutto non variano in maniera rilevante in funzione dei parametri dei filtri. Discorso analogo si applica alla risoluzione degli spartiti: risultati migliori, ad esempio, su una risoluzione più alta del medesimo spartito indicherebbero solamente che i parametri dei filtri così come sono stati calibrati si adattano meglio a quella particolare risoluzione. Sarebbe tuttavia sufficiente modificarli per invertire questo esito ed ottenere un riconoscimento migliore sullo sparito a risoluzione più bassa. In altre parole, una valutazione di questo tipo non sarebbe indicativa delle performance assolute del sistema.

4.3 Dataset degli spartiti

La lista seguente elenca le linee che sono state scelte per entrare a far parte del dataset di test, seguendo la forma:

> Nome dello spartito, risoluzione orizzontale, numero della linea: lista delle peculiarità

- **Andante religioso, 1600, 2:** condizioni di stampa ideali, test di riferimento;
- **Andante religioso, 1600, 3:** presenza di molti simboli ravvicinati tra loro;
- **Andante religioso, 1200, 3:** presenza di molte annotazioni testuali in prossimità dei simboli;
- **A Chloris, 1600, 2**: presenza di rumore nell'immagine, note unite in ottavi e sedicesimi;

- **A Chloris, 1600, 3**: accordi che si estendono verticalmente oltre il rigo;
- **Der Hölle, 1600, 7**: definizione insufficiente per Audiveris, rigo non orizzontale, presenza di pause tra le note;
- **Der Hölle, 1600, 16**: variazione pronunciata nell'altezza delle note, simbolo quasi cancellato;
- **Marcia trionfale, 1500, 1**: definizione insufficiente per Audiveris, notazione inusuale con staffa separata dalla testa della nota;
- **Marcia trionfale, 1500, 11**: note unite in ottavi e terzine di sedicesimi;
- **Les sons, 1600, 1**: definizione insufficiente per Audiveris, presenza di legature tra le note;
- **Les sons, 1600, 3**: presenza di alterazioni in numero elevato;
- **Prélude, 900, 1**: risoluzione inusuale, simboli non standard per la rappresentazione delle note.

4.4 Risultati

Nome spartito	Simboli totali	Riconosciuti	Parzialmente	Mancati	% Riconosciuti	% Parzialmente	% Mancati
Andante religioso	68	68	0	0	100%	0%	0%
Andante 1200px	39	35	4	0	89,75%	11,43%	0%
A Chloris	60	32	28	0	53,33%	46,67%	0%
Der Holle	103	95	8	0	92,23%	7,77%	0%
Marcia trionfale	47	46	0	1	97,87%	0%	2,13%
Les sons	46	37	9	0	80,43%	19,57%	0%
Prélude	23	15	8	0	65,22%	34,78%	0%
	386	328	57	1	84,97%	14,77%	0,26%

$$\frac{Riconosciuti}{Totali} = 84,974\%$$
$$\frac{Non\ riconosciuti}{Totali} = 0,259\%$$

4.5 Considerazioni

Innanzitutto vale la pena considerare il test di riferimento sullo spartito *Andante religioso*: su di esso sono stati riconosciuti tutti i simboli, un risultato sperato ma assolutamente non scontato. Negli altri casi, con un riconoscimento medio che raggiunge quasi l'85%, si può senz'altro affermare che il sistema vanta delle buone prestazioni, soprattutto considerando che tre degli spartiti sottoposti a test sono stati rifiutati dal tool di terze parti Audiveris, con la motivazione che la definizione era insufficiente. Allo stesso modo solo una quantità estremamente bassa di simboli viene totalmente mancata: un riconoscimento parziale impone sì all'utente almeno un passaggio aggiuntivo – la pressione di un tasto per dividere o rispettivamente ingrandire la bounding box –, ma rappresenta comunque una situazione facilmente rimediabile.
Vi sono alcuni spartiti dove il riconoscimento si attesta sotto la media: A Chloris con circa il 53% e Prélude con un 65%. Nel primo caso questo è da attribuire ad una scelta di stampa non ottimale: diverse note sono infatti estremamente ravvicinate, separate solo dal rigo. La figura 25 illustra questa problematica. Nel secondo caso il risultato sotto la media è quasi certamente da attribuire alla bassa risoluzione e alla notazione musicale personalizzata, per la quale i filtri non erano stati pensati ma si sono comunque dimostrati in grado di riconoscere una parte dei simboli. In figura 26 è visibile uno spaccato dello spartito.

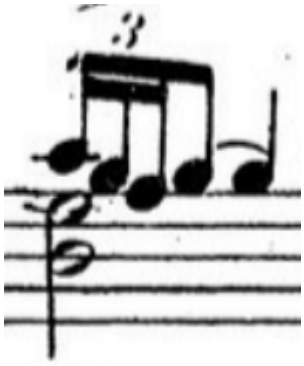



Figura 25: Accordo, in basso a sinistra, estremamente ravvicinato alle altre note

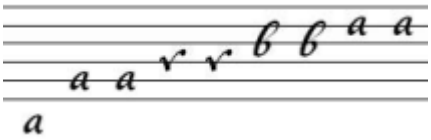


Figura 26: Notazione musicale inusuale

Sarebbe interessante provare ad innalzare ulteriormente le performance del sistema agganciando i parametri dei filtri – ed in particolare della chiusura morfologica, filtro dal quale le prestazioni del sistema dipendono in maniera particolare – alla grandezza denominata staffline height discussa nel sottoparagrafo 1.4.4. Attualmente infatti questo filtro, che serve proprio a ricostruire le note che sono rimaste tagliate in seguito alla rimozione del rigo, fa uso di un cosiddetto *magic number* che assume il valore di 7. Questa scelta è una conseguenza del fatto che OpenCV non mette a disposizione alcun algoritmo di estimazione: volendo introdurne uno lo si dovrebbe implementare in un file C++ dedicato e linkarlo manualmente alla Java Virtual Machine, complicando considerevolmente l'architettura del sistema. Non è infatti plausibile, al pari dei filtri immagine, pensare di implementare un algoritmo così oneroso direttamente in linguaggio Java.

Capitolo 5

Conclusioni

5.1 Risultati raggiunti

Il sistema è in grado di riconoscere una percentuale elevata dei simboli musicali. Questo fatto si tradurrà in un notevole risparmio di tempo durante la creazione di nuovi contenuti IEEE 1599, dando una spinta alla diffusione di questo nuovo formato dalle notevoli potenzialità.
È interessante notare come, confermando quanto auspicato nelle fasi preliminari, sia stato possibile raggiungere questo risultato impiegando una soluzione software dalla complessità contenuta, senza dover implementare un sistema di OMR completo. Come discusso nel paragrafo 3.3, ciò avrebbe richiesto un lavoro notevolmente più lungo, sia da un punto di vista implementativo che di successiva messa a punto, senza la garanzia che i risultati finali sarebbero stati soddisfacenti: nel paragrafo 4.3 si è documentato come Audiveris, che implementa uno di questi sistemi di OMR, si sia rivelato incapace di processare alcuni spartiti, che invece risultano normalmente utilizzabili in GraphicMapper.
Si è comunque rivelato molto istruttivo dover imparare a consultare la letteratura, operazione necessaria sia per valutare se le soluzioni classiche fossero adeguate al particolare problema di riconoscimento informato, sia per redarre il paragrafo 1.4 del presente elaborato. Per la prima volta ci si è potuti addentrare nel mondo della ricerca, imparando a scegliere i giusti paper ed estrapolandone le informazioni rilevanti, competenza che senz'altro risulterà utile ogni qual volta ci si troverà a lavorare su un problema di una certa complessità e per il quale una soluzione non si riveli immediata. La considerazione più importante è che la fase preliminare di ogni buon progetto sia un inquadramento completo del problema da risolvere e delle soluzioni già studiate; dalle quali è naturalmente lecito discostarsi ma unicamente in seguito ad una riflessione approfondita che tenga conto di benefici e potenziali problemi.

5.2 Sviluppi futuri

Come anticipato nel paragrafo 4.5, il primo miglioramento è rappresentato dall'utilizzo di valori dinamici come parametri dei filtri, anziché dei *magic number* come avviene adesso. L'algoritmo di estimazione delle grandezze è stato discusso nel sottoparagrafo 1.4.3: ai fini di ottenere una maggiore robustezza gli autori hanno sacrificato la possibilità di individuare separatamente i valori *staffline height* e *staffspace height*, pertanto solo la loro somma può essere determinata. Questo non dovrebbe comunque rappresentare un problema, in quanto sarebbe sufficiente utilizzare delle frazioni di essa. Una volta individuati dei coefficienti adeguati è lecito pensare come questi si adatterebbero ad un gran numero di spartiti, poiché la proporzione tra lo spessore delle linee rispetto allo spazio bianco tra di esse appare in molti casi visivamente costante. Qualora questo non fosse il caso, o semplicemente si volesse rendere il sistema totalmente libero dall'utilizzo di valori fissi, si potrebbe invece svolgere un lavoro di ricerca sull'algoritmo, affinché esso sia nuovamente in grado di restituire i singoli valori pur preservando le importanti caratteristiche di robustezza.
Un altro potenziale sviluppo, decisamente più corposo ma di grande impatto, e anticipato nel paragrafo 1.2, mirerebbe a rendere il sistema del tutto automatico, eliminando la necessità di cliccare sui simboli di interesse. Un algoritmo di segmentazione è già disponibile all'interno della libreria OpenCV, la sfida in questo caso è invece riuscire in seguito a scartare tutti i numerosi elementi che non sono rilevanti, preservando unicamente note e pause. La soluzione più ovvia sarebbe l'utilizzo di un classificatore binario, tuttavia questo avrebbe lo svantaggio di dover essere addestrato con un dataset quanto più corposo possibile, la cui costruzione richiederebbe un notevole lavoro manuale. Appare più sensato tentare invece di individuare una soluzione algoritmica, anche attraverso una ricerca nella letteratura.

Bibliografia

[1] Luca A Ludovico. Ieee 1599: A multi-layer approach to music description. *Journal of Multimedia*, 4(1), 2009.

[2] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre RS Marcal, Carlos Guedes, and Jaime S Cardoso. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1(3):173–190, 2012.

[3] The music notation project: exploring alternative music notation systems.

[4] Alicia Fornés, Josep Lladós, and Gemma Sánchez. Primitive segmentation in old handwritten music scores. In *International Workshop on Graphics Recognition*, pages 279–290. Springer, 2005.

[5] Arnau Baro, Pau Riba, and Alicia Fornes. Towards the recognition of compound music notes in handwritten music scores. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 465–470. IEEE, 2016.

[6] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. Non-local means denoising. *Image Processing On Line*, 1:208–212, 2011.

[7] David Bainbridge and Tim Bell. The challenge of optical music recognition. *Computers and the Humanities*, 35(2):95–121, 2001.

[8] Carlos Guedes, Guilherme Artur Capela, Ana Maria Rebelo, Joaquim Pinto da Costa, and Jaime Cardoso. Staff detection with stable paths. 2009.

[9] Luca A Ludovico. Key concepts of the ieee 1599 standard. In *Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects, IEEE CS, Lugano, Switzerland*, pages 15–26, 2008.

[10] Luca A Ludovico and Adriano Baratè. Local and global semantic networks for the representation of music information. *Je-LKS*, 12(4), 2016.

[11] Simon Maple and Andrew Binstock. The largest survey ever of java developers, 2018.

[12] Mike Cohn. *User stories applied: For agile software development.* Addison-Wesley Professional, 2004.

Progetto sviluppato presso il Laboratorio di Informatica Musicale
https://www.lim.di.unimi.it