POLITECNICO DI TORINO

Corso di Laurea Magistrale

in Ingegneria Informatica

Tesi di Laurea Magistrale

Progetto e realizzazione di un programma per il riconoscimento della semiografia musicale attraverso l'uso di tecniche della Computer Vision

Relatore
Bartolomeo Montrucchio

Candidato
Giacomo Gustavino

Anno Accademico 2019/2020

Indice

Indice figure

1 Introduzione

Negli ultimi anni la tecnologia ha profondamente rivoluzionato la quotidianità di ognuno di noi, rendendosi imprescindibile per lo svolgimento di innumerevoli azioni, dal lavoro alla comunicazione.

Persino in istituzioni sacre come i Conservatori di musica, fra i cui corridoi si respira ancora oggi il fascino di epoche passate, non si è saputo rinunciare alle infinite opportunità che le innovazioni tecnologiche avevano da offrire.

Presto i Maestri di musica hanno aperto le porte delle proprie aule ad impianti di registrazione audio e a strumenti di musica elettronica;

Al giorno d'oggi si è sentita la necessità di inserire, nell'offerta formativa della maggior parte dei Conservatori, corsi che educhino gli studenti all'uso dei principali software musicali.

Esiste una vastissima gamma di software musicali:

- software di notazione musicale
- software per il montaggio audio
- software per la sintetizzazione di suono
- software per l'esecuzione di file multimediali
- software musicali per scopi didattici

Di questi, i software di notazione musicale sono certamente quelli che si sono dimostrati più utili nell'ambito della musica classica;

1.1 Cos'è un programma di notazione musicale?

Un programma di notazione musicale è un software che consente la scrittura di partiture musicali.

Possiamo dire che un software di questo tipo sta ad una partitura come un software di videoscrittura sta al testo.

Attraverso l'uso di questi software l'utente ha l'opportunità di comporre, stampare, ascoltare una partitura, oltre che esportarla nel formato desiderato (xml, midi, pdf, fin, sib, ecc). [1]

1.2 Obiettivo della tesi

Spesso maestri e studenti di musica sono costretti a dover trascrivere, utilizzando uno dei suddetti programmi, partiture prese da libri o da immagini digitali per potervi apporre modifiche, o per poter procedere al loro ascolto.

Ho pensato che sarebbe stato di estrema utilità un programma che venisse incontro a questa necessità, provvedendo alla semplificazione dell'acquisizione dei dati musicali da immagine digitale.

Piuttosto celebri sono i programmi OCR (optical character regognition), il cui scopo è quello di riconoscere i caratteri di un testo e di codificarli in un formato che sia leggibile da un calcolatore. [2]

Il mio intento è quello di creare un programma analogo che permetta il riconoscimento della semiografia musicale di una partitura e che la codifichi in formato compatibile con un qualsiasi programma di notazione musicale, così da facilitare il processo di acquisizione delle informazioni contenute in una partitura.

Il riconoscimento ottico dei simboli musicali nelle partiture stampate, in realtà, è iniziato dalla fine degli anni '60, quando gli istituti di ricerca ebbero la possibilità di usufruire dei primi scanner. Lo sviluppo di sistemi per il riconoscimento ottico della musica, coinvolge molti campi di ricerca, fra cui Computer Vision, Data Analysis e MIR[1] (Music Information

[1] Music Information Retrieval è un campo di ricerca che si occupa del recupero di informazioni dalla musica [18]

Retrieval). Al giorno d'oggi esistono dozzine di applicazioni che sfruttano tali sistemi; purtroppo i software che garantiscono i risultati più soddisfacenti sono proprietari e quindi a pagamento.

L'obiettivo di questa tesi è quello di progettare e realizzare un programma Open Source che a partire da un'immagine digitale, sfruttando le tecniche della Computer Vision, restituisca un file, leggibile da un programma di notazione musicale.

2 Background

2.1 Elementi di una partitura

Una partitura (il cui significato letterale è ‘insieme di parti’) è definita come “l'organizzazione grafica di più pentagrammi musicali contemporanei, ad uso del compositore o del direttore d'orchestra, al fine di controllare e gestire con un colpo d'occhio l'intera simultaneità delle parti che concorrono all'opera musicale”. [3]

Si rende necessario perciò l’utilizzo di una partitura nell’ambito della musica d’insieme, corale, da camera e in alcuni casi, anche nella musica elettronica.

Ogni brano musicale è il risultato dell'insieme di più misure; le misure, o battute, sono le porzioni di brano delimitate da linee verticali dette linee di misura; l’unità di tempo (ad esempio 4/4) è il valore che indica la durata di una battuta e ne denota metrica ed accentuazioni.

Di solito, se il brano musicale è per un gruppo di strumenti vi sarà una parte per ciascun strumento e tali parti saranno allineate verticalmente l’una con l’altra rispetto alle singole battute. Quindi, un direttore d’orchestra è in grado di leggere una partitura tenendo sotto controllo le battute di ciascuna parte, sulla stessa colonna.

Di seguito andremo a prendere visione di alcuni degli elementi che più frequentemente fanno parte di una partitura.

L’elemento base della semiografia musicale è la figura. Le figure principali sono le note e le pause. Le note sono associate al suono, mentre le pause corrispondo a momenti di silenzio. Andiamo a descrivere alcune delle caratteristiche principali appartenenti alle figure musicali.

2.1.1 Note

In questo paragrafo andiamo ad ispezionare la nota nelle sue componenti e nelle sue proprietà.

Testa

La parte di una nota rappresentata con un'ellisse è detta testa; la testa di una nota può essere piena o vuota; come vedremo in seguito tale caratteristica è legata al valore della nota, ovvero alla sua durata.

Gambo

Il gambo è un segmento verticale collegato alla testa della nota. Generalmente il gambo viene posizionato verso l'alto quando la testa della nota è in una posizione superiore al terzo rigo del pentagramma, verso il basso altrimenti. Il gambo viene a posizionarsi sulla sinistra della testa della nota quando è rivolto verso il basso, mentre è posizionato a destra della testa se rivolto verso l'alto. Anche il gambo, come la testa, è una componente legata alla durata della nota; tutte le note di durata uguale o inferiore di una minima sono dotate di gambo.

Travi

Le travi sono tratti rettilinei che collegano i gambi di due o più note di durata di un ottavo, ovvero le crome. La trave è doppia quando collega i gambi di semicrome, e via via sarà tripla per le biscrome e quadrupla per le semibiscrome.

Altezza di una nota

Le componenti viste fin ora si riferiscono al valore di durata che una nota assume; una nota che non sia posizionata su un pentagramma non definisce alcuna informazione legata al suono. L'altezza di una nota è determinabile andando ad analizzare come la testa va a posizionarsi all'interno dei righi e degli spazi di un pentagramma; tuttavia, anche l'informazione della

posizione sul pentagramma non è sufficiente; andiamo ad enunciare i simboli della notazione musicale che determinano l'altezza di una nota.

1) la chiave è un simbolo presente all'estrema sinistra di ogni pentagramma e definisce a quale altezza, ciascuno spazio e ciascun rigo, corrispondono; nella chiave di basso ad esempio il secondo rigo corrisponde al fa della terza ottava;
2) la testa di una nota può essere posizionato su uno spazio o su un rigo del pentagramma; procedendo dal basso verso l'alto incontriamo il primo rigo, poi il primo spazio, subito il secondo rigo e così via; i suoni corrispondenti a tali posizioni saranno quelli della scala diatonica[2] della tonalità legata all'armatura d'impianto;
3) un'alterazione posta prima di una nota modifica l'altezza dell'intonazione della nota a cui si riferisce. Le alterazioni possono essere ascendenti, diesis, e discendenti, bemolle; rispettivamente corrispondono all'aumento o alla diminuzione dell'intonazione di una nota di un semitono. L'insieme delle alterazioni che vanno a collocarsi immediatamente dopo la chiave è denominato armatura di chiave. Le alterazioni facenti parte dell'armatura di chiave si riferiscono a tutte le note del brano che hanno altezza corrispondente;

Durata

Come già accennato la durata di una nota è descritta da alcune caratteristiche degli elementi che la compongono:

• Il tipo della testa della nota (vuota o piena);

• La presenza o meno del gambo;

[2] Una scala diatonica è una scala di sette suoni, che sono legati da relazioni intervallari ben precise, cinque toni e due semitoni. [22]

• La presenza o meno di *flag* (nei gruppi di note, queste vengono convertite in travi);

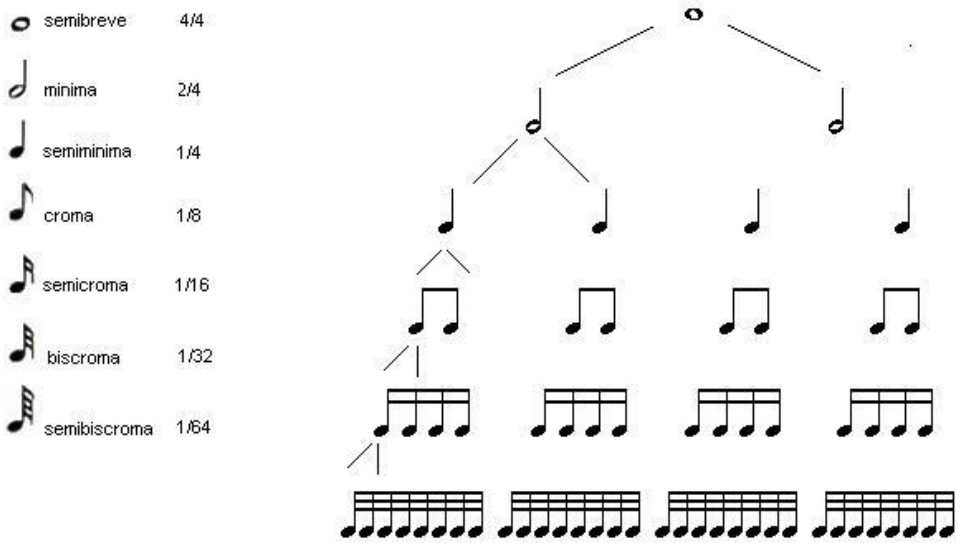


Figura 1 Durata delle note

A seconda della tipologia e della disposizione di queste componenti, siamo in grado di attribuire il corretto valore ad una nota, ovvero la durata che la nota assume nel contesto musicale in cui si viene a trovare.

Andiamo ad elencare le principali tipologie di nota in base alla durata:

- la semibreve esprime una nota di durata pari all'intero, ovvero 4/4; essa viene raffigurata come un'ellisse vuota priva di gambo;
- la minima esprime una nota di durata pari a 2/4; essa viene raffigurata come un ellisse vuota dotata di gambo;
- la semiminima è una nota di durata un quarto; essa viene raffigurata con un'ellisse piena dotata di gambo;
- se al gambo di una semiminima giustapponiamo da una a quattro code, otteniamo note con una durata rispettivamente di 1/8, 1/16, 1/32 1/64 dette rispettivamente croma, semicroma, biscroma e semibiscroma.

Se a qualsiasi delle note appena enunciate affianchiamo un punto sulla destra, esso comporterà l'aumento del valore di durata della nota di una quantità pari alla metà del suo valore originario.

La durata di una nota è legata anche relativamente al valore della pulsazione; questo valore è detto indicazione di tempo, viene generalmente a collocarsi all'inizio del brano, ed indica la velocità con cui dovrà essere riprodotto il brano; ad esempio la dicitura '♪=100 bpm' vorrà dire che la croma avrà pulsazioni pari a 100 battiti al minuto.

2.1.2 Aggregazione di note

Le note possono essere componenti di cinque diversi tipi di raggruppamenti:

- gli accordi sono rappresentati da due o più teste di note disposte verticalmente collegate tutte al medesimo gambo; secondo la teoria musicale queste note devono essere suonate contemporaneamente;
- due o più note, grazie all'uso delle travi, possono essere raggruppate sotto una figurazione ritmica che può essere omogenea, se formata da note della stessa durata, o mista se coinvolge note di diverso valore;
- i gruppi irregolari sono note che, come nel caso precedente, hanno i gambi collegati da una trave singola, doppia, tripla ecc; a differenza del caso precedente però hanno durata irregolare; ad esempio la quintina di crome è una figurazione ritmica in cui ciascuna nota ha durata un quinto di una semiminima; sopra la trave è apposto un numero che ne determina il tipo (la quintina è caratterizzata dal numero '5'); il gruppo di note posizionato in corrispondenza del '4' in figura 2 rappresenta una terzina;

- in forme musicali a più voci come la fuga[3], è possibile trovare due linee melodiche nello stesso pentagramma; in questi casi una voce manterrà sempre i gambi verso l'alto e l'altra verso il basso.

2.1.3 Principali componenti di una partitura

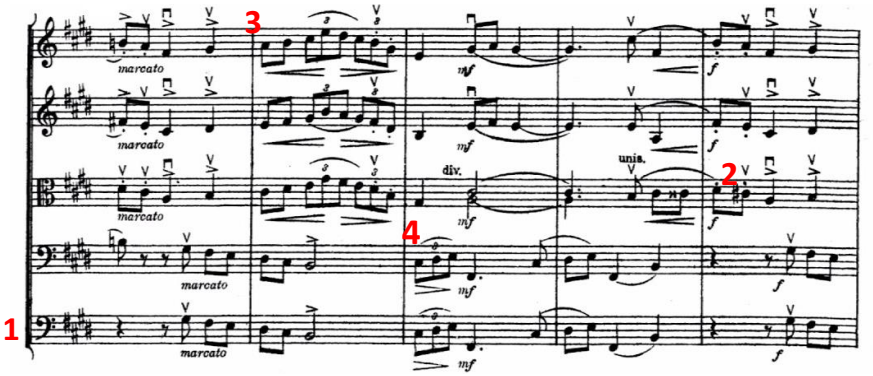


Figura 2 Partitura di un'orchestra d'archi

In figura 2 è mostrata una porzione di una partitura per orchestra d'archi; all'interno di essa possiamo individuare alcuni simboli comuni a tutte le partiture. Andando ad osservare figura 2 possiamo osservare che la porzione di partitura rappresentata è composta da cinque pentagrammi, a loro volta composti da cinque righi equispaziati e paralleli. La linea verticale che collega l'estrema sinistra dei pentagrammi sta ad indicare i pentagrammi che fanno parte della stessa accollatura; ogni pentagramma si riferisce ad una parte; in questo caso avremo cinque parti distinte che devono essere eseguite insieme. Procedendo da sinistra verso destra incontriamo, per ciascuna parte, la propria chiave. Essa sta ad indicare a quale altezza sonora, un determinato luogo del pentagramma, corrisponde; ad esempio, in corrispondenza del numero '1' troviamo la chiave di basso, anche detta chiave di Fa: essa è denominata chiave di Fa in quanto i due punti che le sono di fianco, indicano il rigo corrispondente al Fa della terza ottava. A destra di ogni chiave troviamo un gruppo di simboli; tale gruppo

[3] La fuga è una tecnica compositiva contrappuntistica a due o più voci, costruita su un soggetto (un tema musicale) che viene introdotto all'inizio e poi ripetuto più volte ad altezze diverse esplorando le sue possibilità espressive. [19]

di simboli è detto alterazione di chiave ed indica che le note corrispondenti alla posizione di ciascuna alterazione presente nel gruppo, devono essere alterate nell'intonazione in una particolare maniera per tutta la durata del brano; questa alterazione sarà crescente se il simbolo è un diesis, e decrescente se l'alterazione è un bemolle. L'alterazione che si trova in corrispondenza del numero '2' è detta temporanea; in questo caso l'alterazione è un diesis, ciò significa che la nota che la segue verrà alterata di un semitono in senso crescente; le alterazioni temporanee hanno validità limitata ad una battuta. Le battute, o misure, sono le porzioni di brano delimitate dalle linee di misura; una linea di misura è rappresentata da un segmento verticale che attraversa il pentagramma; possiamo osservare ciò in corrispondenza del numero '3'; le linee di misura per parti che fanno parte della stessa accollatura sono incolonnate. Esistono poi moltissime altre indicazioni riguardanti ad esempio l'agogica, l'articolazione e le tecniche esecutive, per le quali non verrà affrontato un approfondimento specifico.

2.2 Programmi di notazione musicale

Come già detto in precedenza un programma di notazione musicale è un software utilizzato per creare, modificare, ascoltare e stampare partiture.

La rapida crescita dei computer desktop negli anni '80 ha portato allo sviluppo di dozzine di programmi di notazione musicale.

Questi programmi furono accolti come una manna dai giovani compositori, insegnanti di musica e studenti di composizione poiché fornivano un modo molto più agevole per creare partiture e parti per musica orchestrale ed altre opere.

Negli anni novanta furono creati Finale e Sibelius. Essi permisero la produzione di partiture di altissima qualità; inoltre, la vastissima gamma di sofisticatissime funzionalità con cui venivano incontro alle esigenze dei

propri utenti, sancì una netta supremazia nei confronti delle applicazioni concorrenti.

Degno di menzione è sicuramente MuseScore, creato negli anni 2000; è un programma di notazione musicale Open Source che negli ultimi anni è riuscito a guadagnarsi una posizione di rilievo, accostandosi come celebrità ai due programmi precedentemente citati.

2.3 Acquisizione dei dati musicali

Sebbene fin da metà novecento era ben noto il grande potenziale che i computer avrebbero potuto avere nel campo della musica, le difficoltà legate all'acquisizione dei dati hanno rallentato lo sviluppo delle applicazioni inerenti alla musica. Negli ultimi decenni sono stati tentati numerosi approcci per venire incontro a questo problema.

2.3.1 Alfanumerico

Uno dei primi approcci fu quello di codificare la notazione musicale attraverso un codice alfanumerico; il più usato fra questi è chiamato DARMS (Digital Alternate Representation of Musical Score). Nonostante il sistema fosse capace di codificare praticamente ogni tipo di simbolo musicale, questo progetto fu presto abbandonato. La procedura di acquisizione dati con questa metodologia è molto lenta nonché soggetta ad errori; oltretutto per utilizzare questo metodo è richiesto un lungo studio del sistema di codifica.

2.3.2 Grafico

Il sistema grafico permette all'utente di selezionare il simbolo predefinito desiderato da un menu visualizzato sullo schermo di un computer e posizionarlo appropriatamente nel pentagramma grazie all'uso del cursore.

Questo metodo si dimostra efficace sin tanto che l'informazione musicale da acquisire è relativamente ridotta; in ogni caso questo è probabilmente il

metodo più pratico per acquisire dati musicali, ed è uno dei metodi utilizzabili nei programmi di notazione musicale di cui abbiamo parlato.

2.3.3 Tastiera MIDI

L'uso di una tastiera MIDI collegata con il computer, offre un efficiente metodo di acquisizione: l'utente suona il brano desiderato alla tastiera per trasferire l'informazione al computer. Uno svantaggio di questo metodo è la perdita di alcune informazioni, come la separazione di diverse voci coesistenti nel medesimo pentagramma; il protocollo MIDi si preoccupa di memorizzare solo ciò che è utile ai fini della riproduzione di un brano, ma non ciò che riguarda la visualizzazione di un brano musicale.

2.3.4 Fonte sonora

Per quanto riguarda l'acquisizione da fonte sonora possiamo dire che, se per l'acquisizione di una monodia si sono ottenuti risultati piuttosto soddisfacenti, lo stesso non può essere detto per la musica polifonica; infatti ad oggi gli algoritmi per la determinazione di quest'ultima non offrono ancora una sufficiente precisione. Come nell'acquisizione MIDI anche in questo caso, andremo incontro ai medesimi problemi inerenti alla perdita dell'informazione visuale. [4]

2.4 Cos'è un sistema OMR

I sistemi OMR hanno come obiettivo quello di semplificare il compito dell'acquisizione dei dati musicali nella circostanza in cui si disponga di tali informazioni in forma di immagine digitale.

Gli scenari a cui questi sistemi apportano significativi benefici sono molteplici:

- un clarinettista che necessita della trasposizione di un brano;
- un qualsiasi solista che necessita di accompagnamento;

- un copista che necessita delle parti separate di ogni singolo strumento di una partitura orchestrale;

L'utilità dell'estrazione di informazioni dai documenti cartacei inoltre, coincide con la sempre più presente necessità di gestire le informazioni in modo automatico. Al momento attuale infatti l'informazione, nel senso più generale del termine, prodotta e conservata sotto forma cartacea è nettamente superiore a quella archiviata in forma digitale.

Nonostante il problema di un riconoscitore ottico per caratteri musicali sia allo studio da molti anni, resta ancora un campo di ricerca nel quale rimangono irrisolti ancora molti problemi.

I sistemi OMR non sono infallibili, pertanto, se è sicuramente vero che questi programmi semplificano moltissimo il processo di acquisizione dei dati, è vero anche che i risultati da essi prodotti potrebbero presentare delle imprecisioni, il che rende necessaria in ogni caso la revisione da parte dell'utente.

Lo scenario più probabile è quello in cui un sistema OMR venga utilizzato per elaborare la maggior parte dei simboli del brano da codificare, seguito da una fase di editing, operata dall'utente, utilizzando un programma di editor musicale standard, per correggere errori e omissioni commessi dal sistema nel corso della conversione.
Sono stati proposti molti approcci a proposito dei sistemi OMR; la maggior parte di essi condivide un'architettura, in cui ogni fase del processo esegue una determinata operazione, come la rilevazione e la rimozione delle linee del pentagramma prima di passare alla fase successiva che per esempio avrà come scopo l'individuazione di un particolare simbolo. Un problema comune con questo approccio è che gli errori e gli artefatti che sono stati commessi in una fase vengono propagati attraverso il sistema e possono influire pesantemente sulle prestazioni. Ad esempio, se la fase di rilevamento della linea del rigo non riesce a identificare correttamente

l'esistenza del rigo musicale, i passaggi successivi probabilmente ignoreranno quella regione dell'immagine, portando a informazioni mancanti nell'output.

Il riconoscimento ottico della musica è spesso sottovalutato a causa della natura apparentemente facile del problema: se dotati di una scansione perfetta della partitura, il riconoscimento visivo può essere effettuato con una sequenza di algoritmi abbastanza semplici, come il *template matching*[4]. Tuttavia, il processo diventa significativamente più difficile per scansioni scadenti o per musica scritta a mano, che molti sistemi non riescono a riconoscere del tutto.

Perciò, nonostante tutti i simboli di un'immagine potrebbero essere rilevati perfettamente da un esecutore, le frequenti ambiguità e talvolta la violazione di regole, porterebbero portare il sistema OMR all'errore.

2.4.1 OMR vs OCR

Il riconoscimento ottico della musica (OMR) è stato spesso paragonato al riconoscimento ottico dei caratteri (OCR).

La differenza più grande è che la notazione musicale è un sistema di scrittura funzionale. Ciò significa che mentre i sistemi OCR si riducono al riconoscimento di primitive ben definite, nel riconoscimento della simbologia musicale il significato dei simboli varia in funzione di dove essi sono posizionati e della maniera nella quale essi sono in relazione con gli altri simboli.

Non è sufficiente l'individuazione dei simboli in se per determinarne la semantica; molto spesso per trovare la giusta interpretazione di un simbolo musicale, è necessario verificare in che modo esso è in relazione con gli altri; molto spesso simboli della stessa categoria possono dare vita a

[4] Il *template matching* è una tecnica di elaborazione dell'immagine il cui obiettivo è quello di rintracciare i luoghi in cui un oggetto 'modello' è presente [20]

significati totalmente diversi a seconda della maniera in cui sono combinati fra di loro. C'è da aggiungere che la notazione musicale coinvolge relazioni spaziali bidimensionali, mentre il testo può essere letto come un flusso di informazioni unidimensionale, una volta stabilita la linea di base. [4]

Un'ultima considerazione riguarda il fatto che non esiste un dizionario delle strutture musicali analogo ad un vocabolario per le parole, questo perché nella musica, le regole di combinazione tra i simboli non sono riconducibili a quelle usate per il testo scritto; perciò mentre per un testo è in un certo senso possibile verificare la coerenza di una parola, lo stesso non può essere fatto nella musica.

2.4.2 Modello di un algoritmo OMR

Un sistema OMR è un problema complesso, reso gestibile attraverso la scomposizione in sotto-problemi.

Molti degli algoritmi già esistenti seguono un modello diviso in sette passaggi fondamentali:

- digitalizzazione dello spartito musicale
- pre-elaborazione grafica
- riconoscimento e/o rimozione del pentagramma
- localizzazione dei simboli musicali
- classificazione dei simboli musicali
- scelta del valore semantico da attribuire ai simboli musicali
- generazione della rappresentazione simbolica nel formato scelto;

Le fasi di analisi grafica sono centrali nei sistemi OMR; per questo motivo, saranno queste le fasi che richiederanno maggiore studio ed attenzione.

Fondamentali per la soluzione del problema del riconoscimento musicale sono anche le procedure di applicazione della conoscenza sintattica e

semantica del processo, non sempre facili da delineare in quanto in letteratura musicale si riscontrano frequentemente situazioni equivoche ed eccezioni alle regole.

2.4.3 Output del processo

Spesso le applicazioni che sfruttano sistemi OMR includono la creazione di una versione riproducibile della partitura musicale. Un modo comune per creare tale versione è generare un file MIDI, che può essere sintetizzato in un file audio. I file MIDI, tuttavia, non sono in grado di memorizzare tutti i dettagli riguardo alla disposizione delle note e degli altri simboli nella partitura, ma contengono solo le informazioni necessarie alla riproduzione audio.

Se il nostro obiettivo è quello di recuperare lo spartito nella sua integrità, con lo scopo di essere letto o modificato dall'utente è necessario che il file prodotto dal processo sia dotato di una codifica strutturata, che includa informazioni precise sul layout. Fra i principali formati adatti per memorizzare queste informazioni citiamo MEI e MusicXML. [4]

2.4.4 Storia degli OMR e algoritmi in letteratura

Fu Dennis Pruslin, a metà degli anni '60, a pubblicare il primo sistema OMR. Tale sistema riusciva a riconoscere un sottoinsieme dei principali simboli musicali e a rilevare accordi complessi. Era assente tuttavia la possibilità di riconoscere simboli quali chiavi, indicazioni di tempo e pause.

2.4.4.1 David Prerau

Nel 1975 il ricercatore David Prerau tentò di ampliare l'algoritmo di Pruslin; tuttavia, egli si accorse che la tecnica utilizzata da Pruslin, al fine di rimuovere i pentagrammi, danneggiava la maggior parte dei neumi musicali. Fu così che Prerau ideò un approccio sostitutivo per la rimozione dei pentagrammi e andò alla ricerca di tutti gli stili possibili attraverso i quali potevano essere rappresentati i simboli musicali, raccogliendoli in un database. La ricerca nel database veniva sfruttata con l'obiettivo di ridurre

in maniera rapida il numero di possibili neumi musicali che un “segno sconosciuto” poteva rappresentare.

Questo processo di identificazione dei simboli musicali può essere così schematizzato:

- Analisi dei pentagrammi al fine di rilevare le porzioni dei simboli musicali che si intersecano o si sovrappongono ai righi; l’obiettivo di questa operazione è quello di rimuovere i pentagrammi tenendo conto delle porzioni di simboli in comune con le linee di quest’ultimo;
- Assemblaggio dei frammenti per costituire nuovamente i simboli interi. Le regole di tale processo appaiono talora eccessivamente semplici in quanto sfruttano la sola sovrapposizione orizzontale delle parti estratte, non rivelandosi pertanto sempre efficaci;
- Quantificazione delle dimensioni di ciascun simbolo sulla base delle dimensioni del rettangolo circoscritto, ritenute da Prerau caratteristiche più che sufficienti per la classificazione dei simboli;
- Riconoscimento dei simboli mediante il confronto delle misure rilevate con quelle di riferimento contenute in una tabella. Tali valori di riferimento sono stati individuati misurando il numero più alto possibile di tipologie di simboli musicali. Normalmente si rilevano dai tre ai cinque riscontri per ciascun simbolo e solo a seguito di test euristici si è in grado di identificare correttamente il simbolo musicale. Tali test sono basati sulla rilevazione della posizione e di caratteristiche proprie di ciascun simbolo, così come su alcune informazioni sintattiche.

L’algoritmo di Prerau ha rivelato la sua capacità di identificare anche simboli più complessi quali alterazioni, semiminime, chiavi, crome

(composte anche in gruppi con travi multiple), ma non semicrome e accordi.

Va considerato che questi primi approcci erano considerevolmente penalizzati dalla limitatezza tecnologica dell'epoca e uno dei passaggi fondamentali di tali progetti era la semplice acquisizione di dati di immagine.

Fortunatamente il progresso tecnologico e il crescente interesse verso questa direzione ha stimolato il raggiungimento di una migliore comprensione dei problemi riguardanti la realizzazione dei sistemi OMR.

2.4.4.2 Wabot-2

In Giappone, agli inizi degli anni '80, venne sviluppato uno straordinario robot, Wabot-2, il quale era in grado di leggere la musica e suonare una tastiera. Per la prima volta, dunque, si è cercato di ovviare al problema OMR attraverso un sistema On-line il quale leggeva in tempo reale, analizzava i dati e forniva un output. Per rispettare i vincoli real-time, tale sistema utilizza un riconoscimento locale e diretto, bypassando ogni sorta di attività di pre-elaborazione (come la rimozione del pentagramma).

Il progetto si basa su un riconoscimento suddiviso in due fasi a cui corrispondono due diversi livelli gerarchici: per il rilevamento di alcuni simboli di maggior importanza come linee di pentagramma o di battuta, viene effettuato un algoritmo di *template matching* implementato via hardware, il quale garantisce rapidità di esecuzione e affidabilità dei risultati; per il rilevamento di simboli che rappresentano alcuni dettagli di minor peso come alterazioni o articolazioni, viene utilizzato un algoritmo di *template matching* implementato via software.

2.4.4.3 Martin e Bellisant

Nel 1991 venne proposto un nuovo sistema che sfruttava una rete neurale artificiale (Artificial Neural Network); le reti neurali sono generate grazie a

modelli matematici che simulano il sistema nervoso dell'uomo, per la catalogazione dei simboli.

Queste reti neurali artificiali sono pensate come reti costituite da molte unità di elementi semplici di calcolo, che rappresentano i neuroni, ciascuna delle quali può essere dotata di una memoria locale. Le diverse unità sono legate tra loro per mezzo di vie di comunicazione, che rappresentano le sinapsi, lungo le quali scorre un segnale di tipo numerico ma non simbolico. Le unità agiscono esclusivamente sui dati locali e sugli input ricevuti attraverso le connessioni. Molte di queste reti neurali artificiali sono dotate di sistemi di "apprendimento" i quali iterativamente adattano il valore associato alle connessioni, ovvero, a partire da esempi, imparano per prove ed errori mostrando di essere in grado di generalizzare a prescindere dai dati iniziali. È proprio questo il principale vantaggio derivante dall'uso di questi tipi di rete: non è indispensabile conoscere con esattezza le relazioni tra i dati che intendiamo esaminare in quanto le reti neurali risultano funzionali anche con dati disturbati o connessi da relazioni non lineari.

Le reti neurali vengono ampiamente sfruttate nel riconoscimento automatico dei caratteri (OCR) poiché risultano essere estremamente funzionali nel riconoscimento dei modelli.

2.4.4.4 McGee e Merkley: MusicReader

Nel 1994 William F.McGee e Paul Merkley realizzarono MusicReader, uno strumento deputato al riconoscimento della simbologia musicale il quale offriva l'opportunità di interazione da parte dell'utente, durante il processo di riconoscimento.

Questo sistema ha però evidenziato sin da subito i propri limiti tra cui l'inadeguatezza della memoria del calcolatore per cui era stato sviluppato (una macchina con processore 386 a 16MHz e sistema operativo MS-DOS); questo problema ebbe ricadute sul processo di classificazione in quanto era

necessario stabilire volta per volta cosa fosse necessario memorizzare. MusicReader è in grado di esportare i dati in formato DARMS o MIDI; in maniera simile ai precedenti processi, MusicReader prevede la rimozione dei pentagrammi prima del riconoscimento delle varie componenti della partitura.

Il rendimento di MusicReader può definirsi ottimo per il riconoscimento di monodie o brani composti da parti a singola voce; per brani pianistici o partiture più complesse le prestazioni calano sensibilmente.

2.4.4.5 Ichiro Fujinaga

Ichiro Fujinaga realizzò nel 1988, l'Adaptive Optical Music Recognition, un sistema adattativo. Tale sistema è caratterizzato da una estrema precisione nel riconoscimento di partiture scritte a mano.

L'Adaptive Optical Music Recognition si fonda sull'acquisizione e memorizzazione di esempi con cui confrontare i simboli sconosciuti per rilevarne la somiglianza o meno. Tale sistema è infatti costituito da un database di simboli e tre processi tra loro legati: un rilevatore di simboli musicali che sfrutta un algortmo k-nearest neighbour; dispone poi di un'interfaccia per la correzione manuale di eventuali errori; infine il sistema è dotato di un meccanismo di apprendimento attraverso il quale modifica continuamente le informazioni del database.

2.4.4.6 Cantor

Presso l'Università di Waikato e Canterbury, in Nuova Zelanda, David Bainbridge e Timm Bell svilupparono nel 1996 un sistema chiamato Cantor, fra le cui caratteristiche troviamo l'estensibilità, ovvero, eventuali estensioni non richiedevano la sostituzione del codice sorgente. Questa caratteristica è resa possibile da una architettura modulare che suddivide il sistema in sotto-processi quali: rilevazione di pentagrammi e isolamento dei simboli musicali, identificazione degli elementi primitivi per mezzo di un

linguaggio di nome “Primela”, individuazione delle relazioni che intercorrono tra i simboli riconosciuti.

Per quanto concerne l’identificazione dei simboli primitivi, essa avviene attraverso un approccio grammaticale, ovvero il Definite Clause Grammars (DCG), il quale valuta quantitativamente la distanza tra i simboli e consente di lavorare su partiture che presentano pentagrammi di varie dimensioni.

2.4.4.7 MidiScan

Newell e Homeda crearono nel 1998 MidiScan; per la prima volta fu msso in commercio un software deputato al riconoscimento delle partiture e alla loro ricostruzione.

MidiScan elabora come input una immagine in formato TIFF; successivamente esegue una ricerca dei pentagrammi rilevandone le estremità; se il riconoscimento risulta non esatto, è possibile eseguirlo nuovamente in modo manuale. Per quanto riguarda invece l’identificazione delle *feature* della partitura non sono previste funzionalità interattive; il risultato del processo è codificato tramite la descrizione MNOD che permetterà poi di ricomporre la partitura.

Il sistema MidiScan tralascia però simboli quali le legature, gli accenti, il tempo etc. L’output del processo è un file di tipo Midi.

Per poter utilizzare MidiScan, inoltre, è necessario che i pentagrammi siano correttamente in posizione diritta e che essi presentino una chiave in apertura. Il programma si dimostra piuttosto inaffidabile per quanto riguarda l’individuazione delle indicazioni di tempo e delle figure irregolari, come le terzine.

2.4.4.8 Fahmy e Bolstein

Secondo Famhy e Blostein l'identificazione dei simboli elementari è un processo che richiede il confronto di ciascuno di questi ultimi con una serie di possibili interpretazioni. Per fare in modo che il riconoscimento sia il più preciso possibile vengono estratti alcuni parametri dall'immagine, i quali verranno usati per stabilire dei vincoli e per effettuare valutazioni di grandezza o di relazione a proposito dei simboli.

Per determinare l'entità di un oggetto assegniamo ad esso una lista di etichette che ne definiscono la probabile categoria; procedendo attraverso valutazioni di tipo posizionale e andando a considerare il soddisfacimento di determinati vincoli, si va, via via, ad escludere le categorie dalla lista.

Il procedimento di Fahmy e Bolstein può essere definito con il nome di *Graph-rewriting,* in quanto definisce un metodo per il quale un grafo subisce continue trasformazioni [5].

2.4.4.9 Software presenti sul mercato

Riporto ora una rassegna dei programmi di tipo commerciale più conosciuti:

• Sharpeye2 (http://www.visiv.co.uk)

• Smartscore (https://www.musitek.com)

• Photoscore (https://www.neuratron.com)

• Playscore 2 (https://www.playscore.co/)

• MuseScore (https://www.musescore.com)

3 Analisi dell'immagine

3.1 Cos'è la Computer Vision

La Computer Vision è un campo di ricerca che si occupa di come i computer possano creare un modello del mondo che ci circonda a partire da immagini o video digitali. [6] Questa disciplina ha come scopo quello di rendere i computer in grado di riconoscere ed interpretare correttamente il contenuto di un'immagine o di un video digitale. In altre parole si potrebbe dire che la Computer Vision cerca di ricreare lo stesso processo cognitivo che avviene nell'uomo quando acquisisce le informazioni visive dal mondo che lo circonda, attraverso gli occhi. L'analisi dell'immagine digitale ha come scopo quello di individuare quali oggetti sono presenti e dove si trovano. Possiamo definire due branchie principali della Computer Vision: *Object Classification* e *Object Detection*.

L'*Object Classification* è una branca della Computer Vision che riguarda l'abilità di un computer di riconoscere e classificare oggetti in un'immagine; in altre parole l'*Object Classification* si occupa di determinare se particolari oggetti sono o non sono presenti in un'immagine. Per *Object Detection*, invece, si intende l'individuazione, a seguito di una scansione dell'immagine, di un oggetto che presenta determinate caratteristiche e l'acquisizione di alcuni parametri che lo riguardano, fra cui le coordinate sopra le quali tale oggetto giace.

3.2 Image processing

Un'immagine può essere definita come una funzione bidimensionale $f(x,y)$ dove x e y sono le coordinate di ciascun pixel dell'immagine, e l'ampiezza di f per ogni coppia di coordinate (x,y) è chiamata intensità di grigio. L'*Image Processing* può essere definito come l'insieme di attività, eseguite tramite un calcolatore, al fine di processare un'immagine per ottenere un determinato risultato.

In base al tipo di output generato, possiamo suddividere le operazioni inerenti all'*Image Processing* in:

- operazioni su immagini che restituiscono immagini;
- operazioni su immagini che restituiscono grandezze che si riferiscono a caratteristiche dell'immagine;

Un'altra classificazione che può essere fatta a proposito degli operatori applicabili all'immagine è fra operatori punto ed operatori spaziali.

Operatori punto

Gli operatori punto eseguono una trasformazione del valore di un pixel non curandosi del valore dei pixel vicini; chiamando *u* un pixel dell'immagine di partenza e *v* il pixel posto sulle stesse coordinate dell'immagine di output, possiamo scrivere: $v = f(u)$; chiaramente l'immagine che otterremo come output di tale processo dipenderà dal tipo di funzione $f(u)$ utilizzata. Una delle applicazioni più frequenti di questo tipo di operatore, è la *sogliatura;* tale procedura consiste nella valutazione di ciascun pixel preso singolarmente; se il valore di tale pixel supera una certa soglia, il valore del pixel dell'immagine di destinazione avrà come valore fissato, ad esempio zero; se il valore del pixel nell'immagine originale non eccede la soglia, il pixel corrispondente dell'immagine di destinazione avrà lo stesso valore del sorgente.

Operatori spaziali

Gli operatori spaziali utilizzano, per produrre il valore di un pixel dell'immagine di destinazione, oltre al valore del pixel corrispondente, anche il valore assunto dai pixel posizionati in un determinato vicinato; il vicinato è rappresentato da tutti i pixel che si trovano ad una certa distanza dal pixel considerato; a seconda delle esigenze, l'utente può scegliere una certa dimensione ed una certa forma della finestra che definisce la regione in cui l'operatore va a lavorare. Fra gli operatori spaziali maggiormente

conosciuti menzioniamo il filtro media, che calcola la media aritmetica dei pixel all'interno della "finestra" e impone tale valore, e il filtro mediano, il quale invece calcola la mediana statistica. I due filtri appena enunciati hanno come obiettivo quello della riduzione del rumore. [7]

3.3 Segmentazione

Segmentare un'immagine significa partizionarla in aree significative. Questa operazione si dimostra assai utile nell'ambito dell'elaborazione digitale dell'immagine poiché consente di avere una rappresentazione più compatta dell'informazione visiva; per questo motivo la segmentazione può rendere più semplice alcuni processi della Computer Vision come il *feature extraction*.

In particolare, la segmentazione è il processo grazie al quale siamo in grado di far emergere quei gruppi di pixel che hanno caratteristiche in comune.

Per effettuare la segmentazione di un'immagine è possibile sfruttare una delle tre principali categorie di algoritmi:

- algoritmi basati sull'istogramma;
- algoritmi basati sulla *crescita/divisione delle regioni*;
- algoritmi basati sul rilassamento; [8]

3.3.1 Features extraction

L'estrazione delle caratteristiche (*Features extraction*), viene a collocarsi quasi sempre in una fase successiva a quella di segmentazione. La segmentazione di solito offre come risultato dati in forma di pixel grezzi; questi possono costituire o il confine di una regione (cioè l'insieme di pixel che separa una regione dell'immagine da un'altra) o tutti i punti presenti nella regione stessa. L'estrazione delle caratteristiche consiste nel rilevamento e nella descrizione delle *feature*.

La descrizione delle caratteristiche assegna attributi quantitativi alle caratteristiche rilevate. Ad esempio, potremmo rilevare angoli in una regione e descrivere quegli angoli in base al loro orientamento e posizione; entrambi questi descrittori sono attributi quantitativi. I metodi di elaborazione delle caratteristiche sono suddivisi in tre categorie principali, a seconda che siano applicabili a confini, regioni o intere immagini. Alcune funzionalità sono applicabili a più di una categoria. I descrittori delle caratteristiche dovrebbero essere il più insensibili possibile alle variazioni di parametri come scala, traslazione, rotazione, illuminazione e la prospettiva.

Vengono perciò impiegate tecniche in grado di rilevare determinate 'forme' con affidabilità e robustezza qualunque siano i parametri che ne condizionano l'aspetto. Uno degli invarianti base è la luminosità. Si cerca di individuare una certa caratteristica a prescindere che essa sia chiara o scura. Alcuni fra gli invarianti più importanti oltre la luminosità, sono la posizione, l'orientamento e la dimensione.

Le tecniche presentate in questo capitolo hanno come scopo quello di individuare una forma in un'immagine, discriminandola dagli elementi di sfondo.

Questo può essere fatto considerando le informazioni di intensità dei pixel o confrontando i pixel con un dato *template*.

Estrarre delle caratteristiche significa estrarre un vettore o un insieme di vettori che descrivono un'immagine.

3.4 Image enhancement

Il miglioramento dell'immagine è il processo di trasformazione di un'immagine che produce un output più adeguato rispetto all'originale per una certa applicazione. Le tecniche di miglioramento sono orientate al problema che si deve affrontare. Pertanto, un metodo che risulterebbe utile

a migliorare le immagini a raggi X, potrebbe non essere l'approccio più adatto per migliorare le immagini a infrarossi. Non esiste una "teoria" generale per il miglioramento dell'immagine. Quando un'immagine viene elaborata per l'interpretazione visiva, lo spettatore è il giudice finale di quanto funziona un particolare metodo. Quando si ha a che fare con la percezione della macchina, il miglioramento è più facile da quantificare. Ad esempio, in un sistema di riconoscimento dei caratteri automatizzato, il metodo di miglioramento dell'immagine più appropriato è quello che si traduce nel miglior tasso di riconoscimento, tralasciando altre considerazioni come i requisiti computazionali. Uno dei metodi principali per il miglioramento dell'immagine è l'applicazione di filtri spaziali. Il filtraggio spaziale modifica un'immagine sostituendo il valore di ogni pixel con una funzione dei valori del pixel e dei suoi vicini. Un filtro spaziale lineare esegue un'operazione di somma dei prodotti tra un'immagine *f* e un *kernel* ω, che rappresenta il filtro. Il *kernel* è un array la cui dimensione definisce 'l'intorno' dell'operazione e i cui coefficienti determinano la natura del filtro. Altri termini usati per fare riferimento a un *kernel* di filtri spaziali sono maschera, *template* e finestra. In generale, il filtraggio spaziale lineare di un'immagine di dimensione M × N con un *kernel* di dimensione m × n è dato dall'espressione:

$$g(x,y) = \sum_{s=0}^{M} \sum_{t=0}^{N} (\omega(s,t) * f(x+s, y+t)]$$

3.5 Filtraggio spaziale

Consideriamo tre approcci di base per la costruzione di filtri spaziali. Un approccio si basa sulla formulazione di filtri basati su proprietà matematiche. Ad esempio, un filtro che calcola la media dei pixel in un determinato intorno, sfoca un'immagine; un filtro di media produce un effetto analogo ad un filtro di integrazione; al contrario, un filtro che calcola la derivata locale di un'immagine ne evidenzia i bordi. Un secondo

approccio si basa sul campionamento di una funzione spaziale 2-D la cui forma ha una proprietà desiderata. Ad esempio, mostreremo in seguito che i campioni di una funzione gaussiana possono essere utilizzati per costruire un filtro a media ponderata (passa-basso). Un terzo approccio consiste nel progettare un filtro spaziale con una risposta in frequenza specificata. Questo approccio rientra nell'area della progettazione di filtri digitali.

3.6 Filtri di Smoothing

I filtri spaziali di *smoothing* sono utilizzati con lo scopo di mitigare le transizioni brusche di intensità. Dato che il rumore bianco è tipicamente costituito da transizioni brusche di intensità, un'applicazione ovvia del filtro di *smoothing* è la riduzione di tale rumore. Un altro scopo per il quale viene applicato un filtro di *smoothing* è l'eliminazione dei falsi contorni venutisi a creare in seguito all'utilizzo di un numero insufficiente di livelli di intensità in un'immagine. Il filtro Gaussiano di *smoothing* è considerato un ottimo filtro per lo smussamento dei contorni di un'immagine. I coefficienti del *kernel* che andrà convoluto con l'immagine sono ricavati dalla funzione Gaussiana bidimensionale:

$$g(x,y) = e^{-\frac{x^2+y^2}{2\sigma}}$$

in cui σ rappresenta la deviazione standard.

Gli effetti di questo filtraggio con *kernel* di diverse dimensioni sono mostrati nella seguente figura. [9]

Figura 3 Effetti dell'applicazione del filtro di smoothing Gaussiano [9]

3.7 Order-statistic filters

I filtri di questa categoria sono non lineari; la risposta si basa sull'ordinamento (ranking) dei pixel contenuti nella regione compresa dal filtro (*neighborhood*). Lo *smoothing* in questo caso può essere ottenuto sostituendo il valore del pixel centrale con il valore determinato dal risultato del ranking. Il filtro più noto in questa categoria è il filtro mediano, che, come suggerisce il nome, sostituisce il valore del pixel centrale con la mediana dei valori di intensità di quel pixel nella regione considerata dal filtro. I filtri mediani forniscono eccellenti capacità di riduzione del rumore per alcuni tipi di rumore casuale, con una sfocatura notevolmente inferiore rispetto ai filtri di *smoothing* lineari di dimensioni simili. I filtri mediani sono particolarmente efficaci in presenza di rumore impulsivo, a volte chiamato rumore sale e pepe, in quanto si presenta come punti bianchi e neri sovrapposti a un'immagine. [10]

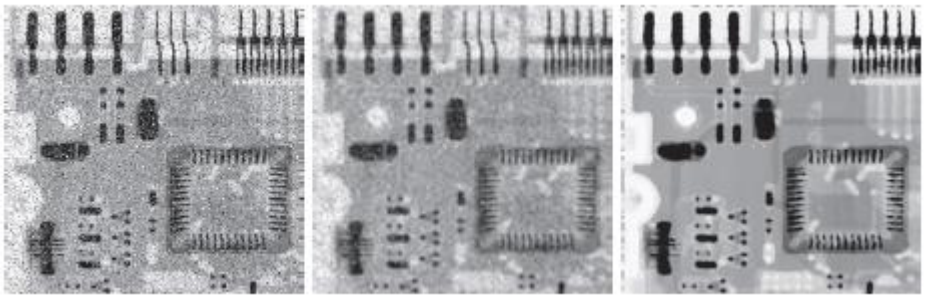


Figura 4 a)Immagine a raggi di un circuito stampato b) riduzione del rumore con filtro Gaussiano c) riduzione del rumore con filtro mediano [10]

3.8 Correlazione

La correlazione consiste nello spostare il centro di un *kernel* su un'immagine e calcolare la somma dei prodotti in ciascuna posizione.

$$g(x) = \sum_{s=-a}^{a} (\omega(s) * f(x+s)]]$$

La correlazione dà una misura del '*match*' tra l'immagine *f* ed il *template* ω, in quanto i luoghi in cui *f* e ω mostrano un 'match' migliore, restituiranno

valori maggiori nella funzione di correlazione. In questo caso, quindi la ricerca di un oggetto, descritto nel *template* ω, corrisponderà ad una ricerca dei massimi locali all'interno della funzione di correlazione.

3.9 Convoluzione

Il meccanismo della convoluzione spaziale, è il medesimo della correlazione, tranne per il fatto che il *kernel* è ruotato di 180 gradi.

$$g(x) = \sum_{s=-a}^{a} (\omega(s) * f(x-s)]\!]$$

La convoluzione è di fatto la base della teoria dei sistemi in cui l'output di un sistema è ottenuto dalla convoluzione di uno stimolo con la funzione di trasferimento *f* di tale sistema. Definendo *f(x-s)* come funzione di memoria, possiamo dire che il processo di convoluzione somma gli effetti di uno stimolo moltiplicato per la funzione di memoria. Il risultato ottenuto è la risposta cumulativa ad uno stimolo.

Una matrice di convoluzione ω, è una matrice, generalmente di piccole dimensioni rispetto all'immagine, ed in molti casi rappresenta un filtro da applicare all'immagine. Risulta dunque molto utile per l'applicazione di filtri di *smoothing*, *sharpening* o per il riconoscimento dei contorni e altro ancora.

3.10 Sharpening

L'operazione di *sharpening* è un processo che evidenzia le transizioni di intensità.

Il modo in cui l'intensità dell'immagine cambia all'interno di un'immagine è un'informazione importante e viene utilizzata per molte applicazioni, come vedremo. La variazione di intensità può essere descritta con le derivate *Ix* e *Iy* dell'immagine in scala di grigi (per le immagini a colori, le derivate sono generalmente prese per ciascun canale di colore).

Il calcolo delle derivate dell'immagine può essere eseguito utilizzando approssimazioni discrete. Le derivate discrete vengono implementate attraverso la convoluzione:

$$I_x = I * D_x \quad \text{e} \quad I_y = I * D_y.$$

in cui Dx e Dy sono rappresentati generalmente o dal filtro Prewitt o da quello di Sobel, che equivalgono rispettivamente a:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad D_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad D_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Il gradiente dell'immagine è il vettore che ha come componenti le due derivate:

$$\nabla I = [I_x, \ I_y]^T$$

Il gradiente ha due importanti proprietà fondamentali:

la magnitudo, che descrive quanto forte è la variazione di intensità dell'immagine;

$$|\nabla I| = \sqrt{I_x^2 + I_y^2}$$

e l'angolo del gradiente, che indica la direzione della variazione di intensità maggiore in ogni punto (pixel) dell'immagine

$$\alpha = \arctan2(I_y, I_x)$$

Di seguito un esempio di cosa visivamente è il risultato di queste operazioni:

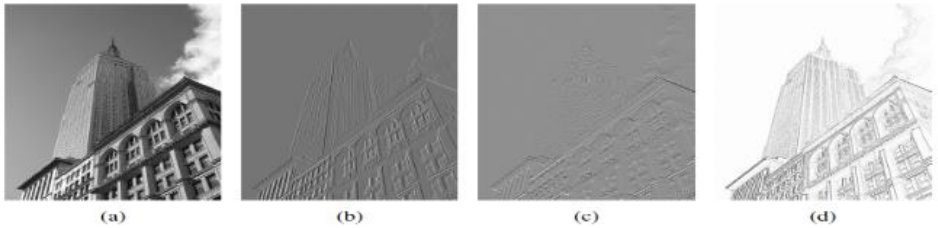


Figura 5 Un esempio di derivata di un'immagine utilizzando il filtro derivativo di Sobel: (a) immagine originale, (b) derivata x, (c) derivata y, modulo(d)

I filtri di *sharpening* hanno come obiettivo quello di rendere l'immagine più nitida. Spesso i filtri di *sharpening* utilizzano operatori derivativi del secondo ordine, poiché più sensibili alle variazioni di intensità. Il più semplice operatore di questo tipo è l'operatore laplaciano.

L'approccio consiste nel definire una formulazione discreta della derivata di secondo ordine. Per compiere quest'operazione utilizzeremo un *kernel* i cui coefficienti sono ottenuti dalla funzione Laplaciana:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Le derivate seconde rispetto ad x e y in una immagine digitale sono calcolate nel seguente modo:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

da cui il Laplaciano:

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

Il filtro Laplaciano assumerà perciò la seguente forma:

0	1	0
1	−4	1
0	1	0

Figura 6 Filtro Laplaciano invariante ad inclinazioni di 90 gradi

Poiché le derivate di qualsiasi ordine sono operazioni lineari, il laplaciano è un operatore lineare. Il Laplaciano evidenzia le transizioni di intensità nette in un'immagine e de-enfatizza le regioni la cui intensità varia lentamente. Ciò tenderà a produrre immagini che hanno linee di bordo grigiastre e altre discontinuità, tutte sovrapposte su uno sfondo scuro e senza dettagli. Le caratteristiche dello sfondo possono essere "recuperate" pur conservando l'effetto di nitidezza del laplaciano aggiungendo l'immagine laplaciana all'originale.

$$g(x,y) = f(x,y) + c\left[\nabla^2 f(x,y)\right]$$

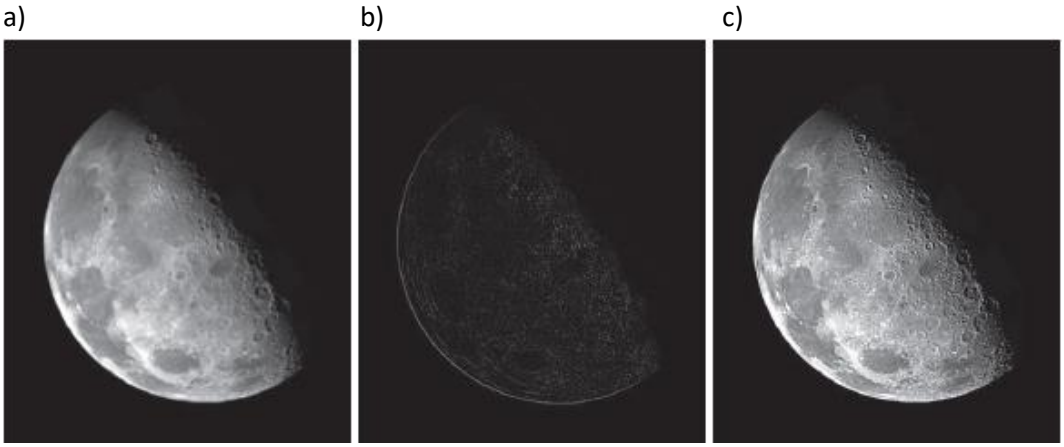


Figura 7 a) immagine originale b) immagine filrata da kernel Laplaciano c) Immagine dopo il processo di sharpening [10]

3.11 Unsharping mask

La sottrazione di una versione non nitida (smussata) di un'immagine all'immagine originale è un processo utilizzato dagli anni '30 dall'industria

della stampa e dell'editoria per rendere più nitide le immagini. Questo processo, chiamato mascheramento di contrasto, consiste nei seguenti passaggi:

- Sfocare l'immagine originale.
- Sottrarre l'immagine sfocata dall'originale (la differenza risultante è chiamata maschera).
- Sommare la maschera all'originale.

Indicando con $\overline{f}$ l'immagine sfocata, la maschera di unsharping è ottenuta dalla seguente formula:

$$g_{\text{mask}}(x,y) = f(x,y) - \overline{f}(x,y)$$

Infine sommiamo all'immagine originale la maschera pesata da un coefficiente k:

$$g(x,y) = f(x,y) + k\,g_{\text{mask}}(x,y)$$

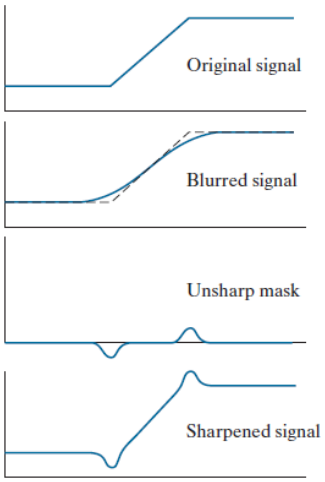


Figura 8 Passaggi di una procedimento di unsharping mask [10]

Figura 9 a) immagine origina b) immagine sfocata c) unsharping mask d) risultato del processo [10]

3.12 Sogliatura

Uno dei metodi più immediati per effettuare la segmentazione di un'immagine è rappresentato dall'operazione di sogliatura (thresholding). Il risultato ottenuto per mezzo di questa operazione, applicata ad un'immagine in scala di grigio, è un'immagine binaria, ovvero un'immagine i cui pixel sono o totalmente bianchi o totalmente neri. Il primo passo in una operazione di sogliatura è la determinazione di un certo valore di soglia; secondo questo procedimento, generalmente, viene assegnato il valore 1 se il loro valore supera quello della soglia, in caso contrario il loro valore verrà impostato a 0. La funzione che descrive l'operazione di sogliatura può essere espressa come segue:

$$I_{binaria}(x) = \begin{cases} 0 & \text{se} \quad I_{originale}(x) < soglia \\ 1 & \text{se} \quad I_{originale}(x) \geq soglia \end{cases}$$

dove *x* rappresenta la coordinata *(x,y)* del generico pixel.

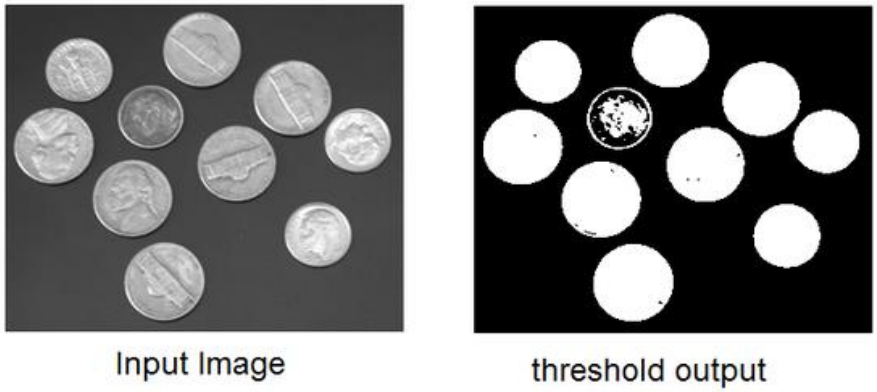


Figura 10 Input ed output di un processo di thresholding

3.13 Operatori di insieme applicati alle immagini

Gli operatori di insiemi principali sono l'unione e l'intersezione. Per le immagini in scala di grigio, l'unione è sostituita dall'operatore di massimo e l'intersezione è sostituita dall'operatore di minimo. L'operatore di

massimo e minimo tra due immagini *f* e *g*, che hanno lo stesso dominio di definizione, sono definiti come segue per ogni punto x:

$$(f \vee g\ )(x) = \max[f(x),\ g(x)\ ],$$
$$(f \wedge g\ )(x) = \min[f(x),\ g(x)\ ].$$

Fra gli operatori di insieme di base più usati vi è il *complemento*. Attraverso l'applicazione di questo operatore otteniamo, per ogni pixel dell'immagine sorgente, un pixel dell'immagine destinazione il cui valore è pari al valore massimo che un pixel può assumere meno il valore del pixel sorgente corrispondente.

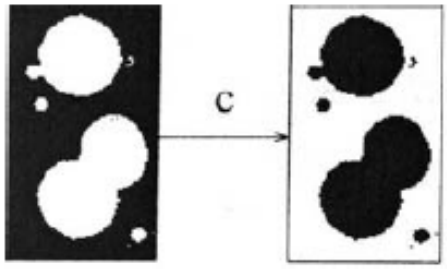


Figura 11 Complemento di un'immagine binaria

La differenza fra due insiemi A e B viene ottenuta facendo l'intersezione tra l'insieme A e il complemento dell'insieme B. Tale operazione è applicabile solo ad immagini binarie.

3.14 Operatori morfologici

La parola morfologia denota comunemente un ramo della biologia che si occupa della forma e della struttura di animali e piante. Usiamo la stessa parola per indicare una serie di operatori dell'*image processing*, utilizzati come mezzo per individuare componenti dell'immagine; tali componenti sono utili in quanto in grado di rappresentare particolarità degli oggetti nell'immagine come scheletri e bordi. La morfologia matematica è una procedura non lineare frequentemente usata per evidenziare o isolare oggetti all'interno di un'immagine, che posseggono una determinata forma.

Il valore di ogni pixel dell'immagine di destinazione è il risultato del confronto tra il pixel corrispondente dell'immagine sorgente con i suoi 'vicini'. In precedenza abbiamo osservato operatori come la convoluzione, in cui il risultato era ottenuto dalla somma pesata dei valori dei pixel vicini a quello considerato.

Con le immagini binarie possiamo combinare i valori solo attraverso l'algebra booleana.

Possiamo introdurre una convoluzione binaria sostituendo la moltiplicazione con l'operatore AND e la somma con l'operatore OR.

$$g'_{mn} = \bigvee_{m'=-R}^{R} \bigvee_{n'=-R}^{R} m_{m',n'} \wedge g_{m+m',n+n'}.$$

Consideriamo che l'immagine *G* sia convoluta con la maschera simmetrica M di dimensione $2R+1 \times 2R+1$ i cui coefficienti sono tutti uguali a 1.

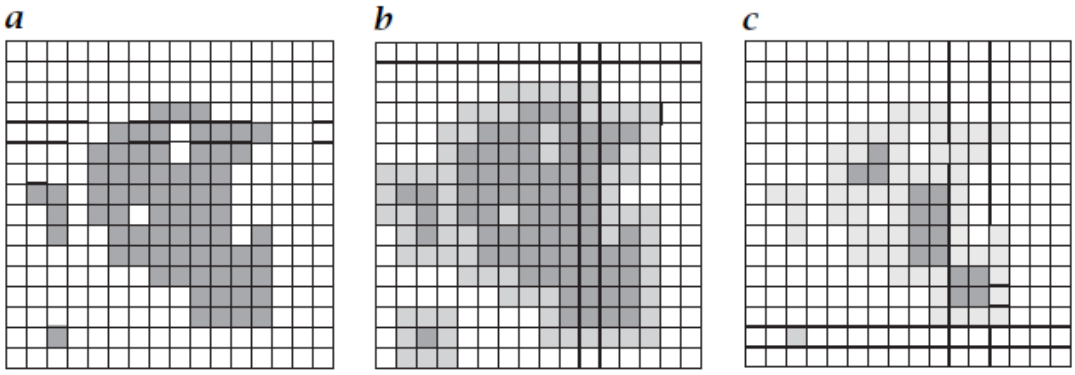


Figura 12 b) Dilatazione e c) erosione dell'immagine binaria a) con maschera 3x3 [11]

Quello che otteniamo è una dilatazione; i contorni divengo più smussati e i piccoli buchi sono riempiti.

L'operatore ottenuto viene perciò detto operatore di dilatazione.

L'operatore duale della dilatazione è l'erosione e può anch'esso essere ricavato dalla convoluzione binaria tramite l'operatore logico *AND*.

$$g'_{mn} = \bigwedge_{m'=-R}^{R} \bigwedge_{n'=-R}^{R} m_{m',n'} \wedge g_{m+m',n+n'}$$

Quello appena proposto è un metodo non propriamente convenzionale di presentare gli operatori morfologici. Normalmente, queste operazioni sono definite come operazioni su insieme di pixel, in quanto prendono in considerazione il sottoinsieme dei pixel non nulli che appartengo all'immagine G; M rappresenta l'insieme dei pixel non nulli della maschera. [11]

Erosione e Dilatazione vengono generalmente definite dalle seguenti formule:

$$\boldsymbol{G} \ominus \boldsymbol{M} = \{p : \boldsymbol{M}_p \subseteq \boldsymbol{G}\}$$

$$\boldsymbol{G} \oplus \boldsymbol{M} = \{p : \boldsymbol{M}_p \cap \boldsymbol{G} \neq \emptyset\}.$$

Dati gli insiemi *G* e *M*, la dilatazione di *G* utilizzando la maschera M si ottiene spostando *M* su ogni punto *x* dell'immagine *G*. La dilatazione è quindi è l'insieme di tutti gli spostamenti *x* di *M* tali che *M* e *A* si sovrappongono per almeno un elemento diverso da zero.

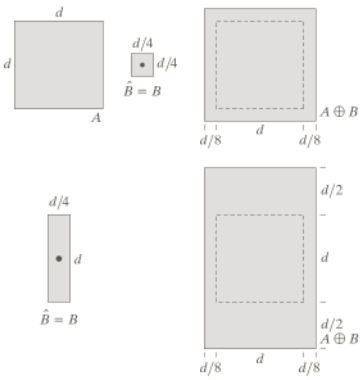


Figura 13 esempio dilatazione [10]

Mentre l'erosione di *A* da parte di *B* è l'insieme di tutti i punti *x* tali che *B*è contenuto in *A*.

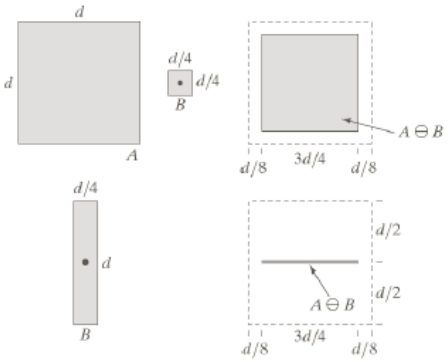


Figura 14 Esempio di erosione [10]

Combinazione di operatori morfologici

Utilizzando gli operatori elementari di dilatazione ed erosione possiamo sviluppare ulteriori utilissime operazioni per lavorare sulla forma degli oggetti.

L'operazione di erosione è utile per rimuovere piccoli oggetti. Tuttavia, ha lo svantaggio che tutti gli oggetti rimasti sono assottigliati. Possiamo evitare questo effetto dilatando l'immagine dopo l'erosione utilizzando il medesimo elemento di struttura (o *kernel*). Questa combinazione di operazioni è detta operazione di *opening*.

$$G \circ M = (G \ominus M) \oplus M.$$

L'*opening* filtra tutti gli oggetti che in nessun punto contengono l'elemento di struttura, ma evita l'assottigliamento generale della dimensione degli oggetti.

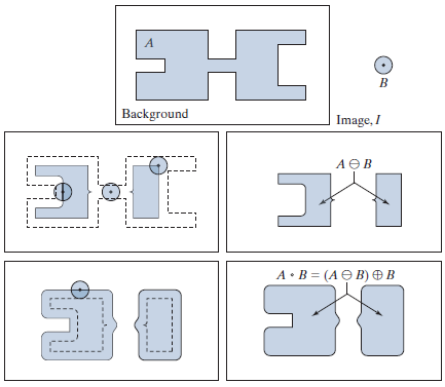


Figura 15 Processo di opening [10]

L'opening è anche un'operazione ideale per rimuovere linee di spessore minore del diametro dell'elemento di struttura. L'opening ha come effetto anche quello di rendere i contorni più smussati. Abbiamo potuto osserva come l'operatore di dilatazione ingrandisce gli oggetti e riempie piccoli 'buchi'. Il generale ingrandimento degli oggetti può essere evitato attraverso una successiva applicazione dell'operatore di erosione.

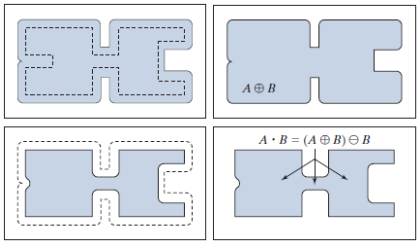


Figura 16 Processo di closing [10]

Questa combinazione di operazioni è detta operazione di *closing*. [10]

$$G \bullet M = (G \oplus M) \ominus M.$$

Spesso è richiesto di applicare un operatore che eroda l'oggetto ma non lo rompa in due pezzi separati. Con un operatore del genere, la topologia dell'oggetto viene preservata e le strutture sottili, come ad esempio le linee, vengono al più ridotte allo spessore di un pixel. Tale operatore è detto operatore di *tinning*; un operatore di *tinning* deve rispettare le seguenti condizioni:

- un oggetto non deve esse diviso in due parti;
- un punto di confine non deve essere rimosso cosicché l'oggetto non divenga più piccolo;
- nessun oggetto deve essere rimosso.

Per ottenere il risultato desiderato, per un'oggetto con 8 connessioni di prossimità bisogna compiere un'erosione utilizzando come elemento di struttura un *kernel* con 4 connessioni di prossimità.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Gli operatori morfologici possono essere usati per estrarre i bordi di un oggetto binario. Un'operazione di erosione con una maschera che contiene tutti i possibili 'vicini' del pixel considerato, rimuove tutti i punti giacenti sul bordo. Il bordo è ottenuto attraverso la differenza (operatore /) tra l'oggetto e l'oggetto eroso:

$$\begin{aligned} \partial G &= G/(G \ominus M_b) \\ &= G \cap \overline{(G \ominus M_b)} \\ &= G \cap (\tilde{G} \oplus M_b). \end{aligned}$$

In maniera simile il bordo dello sfondo è ottenuto sottraendo l'oggetto alla sua versione dilatata:

$$\partial G_B = (G \oplus M_b)/G.$$

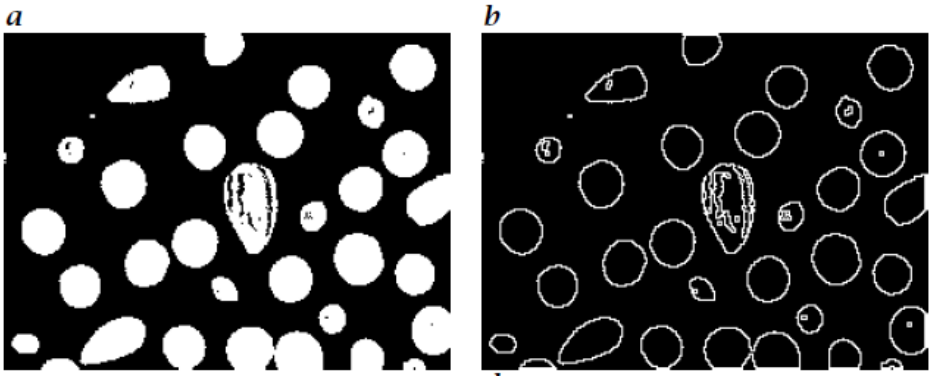


Figura 17 a) immagine originale b) bordi degli oggetti contenuti nell'immagine [11]

3.15 Edge detection

Il riconoscimento dei contorni (in inglese *edge detection*) è una branca della Computer Vision, ed in particolare del *features extraction*, che ha come obiettivo principale quello di individuare le zone di un'immagine in cui l'intensità dei pixel varia rapidamente. [12] Una brusca variazione di

intensità nell'immagine spesso corrisponde ad un cambiamento o ad un evento che si verifica nella scena del mondo reale che l'immagine rappresenta (discontinuità della profondità, discontinuità dell'orientamento delle superfici, discontinuità di materiali, e variazioni dell'illuminazione). Quello che ci aspettiamo come risultato da un processo di *edge detection*, è un'immagine composta solo dalle linee di contorno degli oggetti della scena. L'immagine di output conterrà una quantità di informazioni inferiore rispetto all'immagine di partenza, in quanto i dati preservati saranno solamente quelli indispensabili alla descrizione dei contorni rilevati e della loro struttura. Possiamo raggruppare i principali algoritmi di *edge detection* in due categorie:

- algoritmi che sfruttano la derivata spaziale di primo ordine dell'immagine, in cui il problema della ricerca dei contorni si traduce nella ricerca di massimi locali all'interno del gradiente;
- algoritmi che sfruttano la derivata spaziale di secondo ordine, in cui si andrà alla ricerca dei punti in cui la derivata seconda attraversa lo zero (*zero-crossing*).

Al termine di un'operazione di *edge-detection*, viene spesso utilizzata una sogliatura uniforme al fine di selezionare solo i punti con i valori più significativi.

Molti approcci per l'interpretazione delle immagini sono basati sulla ricerca dei contorni, poiché il loro rilevamento è in genere insensibile ai cambiamenti dei livelli di illuminazione.

Come già detto, però, un *edge detector* rileva i repentini cambiamenti di intensità, e per questo motivo dimostra una certa sensibilità al rumore casuale; per andare incontro a questo problema sarebbe prudente incorporare un filtro di *smoothing* all'interno del processo di *edge detection*.

3.15.1 Roberts cross operator

L'operatore a croce di Roberts ideato nel 1965 è uno dei primi operatori per l'*edge detection*. Questo operatore implementa una versione base di *edge detector* del primo ordine attraverso l'uso di due *template*, che sottraggono i valori dei pixel in diagonale, opponendosi così alle direzioni degli assi. I due *template* sono:

+1	0
0	–1

(a) M^-

0	+1
–1	0

(b) M^+

Il punto di contorno è ottenuto come il massimo fra i due valori ottenuti convolvendo i due *template* con l'immagine P [9]:

$$\mathbf{E}_{xy} = \max\ \{|M^+ * \mathbf{P}_{xy}|, |M^- * \mathbf{P}_{xy}|\} \quad \forall x, y \in 1, N-1$$

3.15.2 Operatore di Prewitt

Nell'operatore di *edge detection* di Prewitt i *template*, Mx ed My sono di dimensione 3x3, e quantificano le variazioni di intensità rispettivamente lungo x e lungo y:

1	0	–1
1	0	–1
1	0	–1

1	1	1
0	0	0
–1	–1	–1

Il formato vettoriale di questo *edge detection*, ci permette di sapere il tasso di cambiamento lungo i due assi; possiamo calcolare la magnitudo del contorno M e la sua direzione Θ:

$$M = \sqrt{Mx(x,y)^2 + My(x,y)^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{My(x,y)}{Mx(x,y)}\right)$$

3.15.3 Sobel

L'operatore di Sobel è un operatore differenziale discreto, calcola un'approssimazione del gradiente della funzione di intensità dell'immagine.

Il peso del pixel centrale viene raddoppiato per entrambe i *template* di Prewitt; otteniamo così l'operatore di Sobel.

1	0	–1
2	0	–2
1	0	–1

(a) *Mx*

1	2	1
0	0	0
–1	–2	–1

(b) *My*

Figura 18 Operatori di Sobel

Dopo aver applicato l'operatore di Sobel, ogni punto nell'immagine risultante corrisponderà ad un particolare vettore gradiente. L'operatore di Sobel si basa sulla conversione dell'immagine con un filtro piccolo, separabile e di valore intero applicato nelle direzioni verticale e orizzontale; il che si traduce in meno calcoli quindi in un basso costo computazionale. Il risultato dell'approssimazione del gradiente è grezzo, in particolare per le variazioni ad alta frequenza nell'immagine.

La maggior parte dei metodi di rilevamento dei bordi si basa sul presupposto che i bordi si trovino nell'immagine dove c'è discontinuità. Sulla base di questa ipotesi, è possibile trovare i bordi prendendo la derivata del valore dell'intensità e trovando quei punti in cui le derivate dell'intensità hanno il valore maggiore.

L'operatore di Sobel calcola il gradiente dell'intensità dell'immagine in ogni punto e fornisce anche la direzione del possibile aumento da chiaro a scuro e la velocità di cambiamento di direzione. I risultati mostrano quanto bruscamente o gradualmente l'intensità del punto cambia e quanto è probabile che determinati pixel rappresentino un bordo. [13]

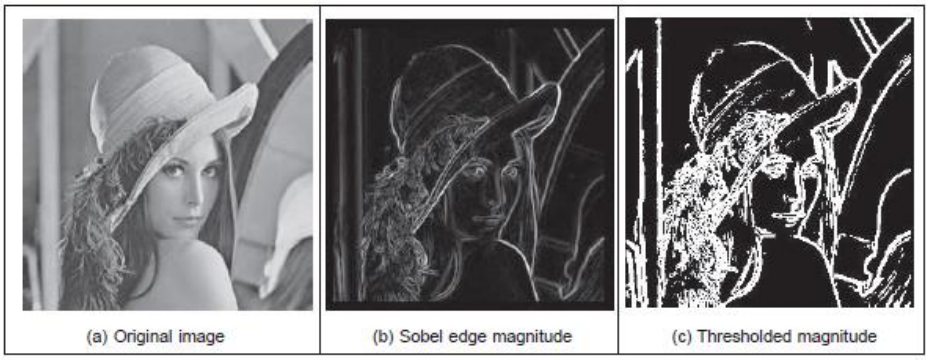


Figura 19 Applicazione dell'operatore di Sobel [10]

3.15.4 Canny edge detector

L'algoritmo di Canny per il riconoscimento dei contorni ideato nel 1986 da John F. Canny, ad oggi è forse la tecnica di *edge detection* più popolare; verte su tre obiettivi fondamentali:

- Basso tasso di errore; tutti i bordi devono essere trovati e non dovrebbero esserci risposte spurie.
- I punti del bordo devono essere localizzati con precisione. I bordi localizzati devono essere il più vicino possibile ai veri bordi. Cioè, la distanza tra un punto contrassegnato come bordo dal rilevatore e il centro del bordo reale deve essere minima.
- Risposta singola: il rilevatore deve restituire un solo punto per ogni punto di bordo reale. Cioè, il numero di massimi locali attorno al bordo reale deve essere minimo.

Dato che i risultati ottenuti da rilevatori di bordi sono spesso affetti da errori causati dal rumore presente nell'immagine analizzata, è essenziale per prima cosa applicare un filtro che escluda al meglio il rumore tutelandoci, per quanto possibile, da errori nei rilevamenti. Per fare ciò, l'immagine viene convoluta con un filtro Gaussiano. Canny fu il primo a dimostrare che l'applicazione di un filtro Gaussiano di *smoothing*, fosse particolarmente adatto per questo tipo di algoritmi. Ricordiamo che l'operatore Gaussiano è dato da:

$$g(x,y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Il gradiente di tale funzione è dato da:

$$\nabla g(x,y) = \frac{\partial g(x,y)}{\partial x} U_x + \frac{\partial g(x,y)}{\partial y} U_y$$

$$= -\frac{x}{\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} U_x - \frac{y}{\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} U_y$$

Da questa equazione possiamo calcolare i coefficienti di un *template* che combina la derivata del primo ordine con lo smussamento Gaussiano. Il gradiente di un'immagine sfocata non restituisce dei contorni netti. Per tracciare un contorno nei punti corretti, possiamo convolvere l'immagine con un operatore che restituisce la derivata prima nella direzione normale al contorno. Il massimo di questa funzione rappresenta il picco dei dati del contorno. Vogliamo ottenere perciò un operatore G_n , che rappresenta la derivata prima della funzione g nella direzione normale $n_\perp$:

$$G_n = \frac{\partial g}{\partial \mathbf{n}_\perp}$$

in cui $n_\perp$, può essere ottenuto dalla derivata prima della funzione Gaussiana g, convoluta con l'immagine P, con appropriato scalamento ottenuto dividendo per il modulo:

$$\mathbf{n}_\perp = \frac{\nabla(\mathbf{P} * g)}{|\nabla(\mathbf{P} * g)|}$$

Il luogo dei punti di contorno è ottenuto calcolando il massimo della funzione ottenuta dalla convoluzione dell'immagine P con l'operatore G_n:

$$\frac{\partial(G_n * \mathbf{P})}{\partial \mathbf{n}_\perp}$$

e quindi ponendo a zero la derivata della funzione qui sopra:

$$\frac{\partial^2 (G * \mathbf{P})}{\partial \mathbf{n}_\perp{}^2} = 0$$

Tramite questi procedimenti otteniamo il soddisfacimento del primo criterio voluto da Canny, ovvero il rilevamento dei contorni nel giusto luogo. Il prossimo passo è l'esclusione dei punti di non-massimo. Questa operazione consiste nel trattenere alcuni dei punti ottenuti dal procedimento di *edge detection*; questa operazione assottiglia i contorni dell'immagine appena calcolata e ci garantisce l'ottenimento di :

- punti posizionati nel giusto luogo
- assenza di risposte multiple
- minima risposta al rumore

Tuttavia non è possibile ottenere un'esatta implementazione del rilevatore di Canny, in quanto è richiesto il calcolo della direzione normale al contorno, che è difficile da stimare. [9]

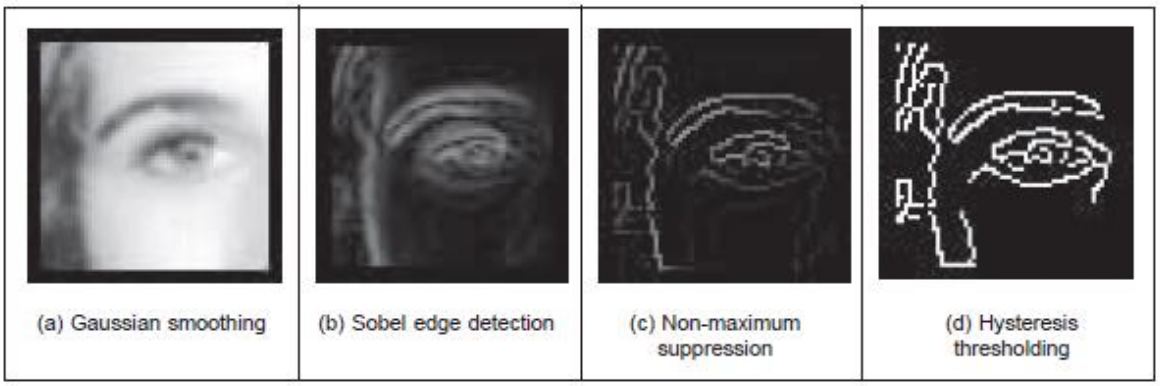


Figura 20 Applicazione del edge detector di Canny [9]

L'algoritmo di rilevamento dei bordi del processo di Canny è comunemente approssimato seguendo 4 passaggi fondamentali:

1) il rumore dell'immagine viene ridotto tramite l'applicazione di un filtro gaussiano;

2) viene calcolato il gradiente d'intensità dell'immagine, usando l'operatore di Sobel;
3) vengono esclusi punti di non-massimo così da liberarsi di eventuali falsi positivi;
4) viene applicata la doppia sogliatura e l'analisi della connettività per rilevare e collegare i bordi.

Filtro Gaussiano

Dato che i risultati ottenuti dai rilevatori di bordi sono spesso affetti da errori causati dal rumore presente nell'immagine analizzata, è essenziale per prima cosa applicare un filtro che escluda al meglio il rumore tutelandoci quanto possibile da falsi rilevamenti. Per fare ciò l'immagine viene convoluta con un filtro Gaussiano. Questo passaggio smussa l'immagine per ridurre gli effetti del rumore sul rilevatore di bordi.
L'equazione di un filtro Gaussiano *(2k+1)×(2k+1)* è:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Per un filtro Gaussiano di dimensione 5x5, per σ=1, abbiamo:

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

La dimensione del filtro Gaussiano condiziona le performance del rilevatore. Più è piccola la dimensione del filtro, più il rilevatore è sensibile al rumore.

Calcolo del gradiente d'intensità dell'immagine

L'algoritmo di Canny si basa sulla ricerca dei punti in cui l'intensità dell'immagine in scala di grigio cambia in modo repentino. Questi punti

sono individuati determinando il gradiente dell'immagine. Il gradiente per ogni pixel viene calcolato applicando l'operatore di Sobel. Per ogni pixel, viene calcolata la derivata parziale su x e su y, mediante la convoluzione dell'immagine con i *kernel* di Sobel visti nel paragrafo precedente:

La magnitudo del gradiente G è $|G| = |Gx| + |Gy|$, mentre direzione del bordo viene così calcolata: $\theta = arctan\ (Gy / Gx)$.

Dato che utilizziamo una maschera 3x3, ci sono solo quattro direzioni possibili con cui possiamo descrivere l'orientamento: 0 gradi (in direzione orizzontale), 45 gradi (lungo la diagonale positiva), 90 gradi (in direzione verticale) o 135 gradi (lungo la diagonale negativa). Quindi l'orientamento del bordo deve essere approssimato in una di queste quattro direzioni.

Soppressione dei non-massimi

La soppressione dei non-massimi è una tecnica di assottigliamento dei bordi, utilizzata per trovare i luoghi in cui la variazione è più netta. Per ogni pixel dell'immagine del gradiente, l'intensità del bordo è confrontata con i pixel nella stessa direzione positiva e negativa del gradiente; se l'intensità del pixel corrente è maggiore rispetto a quella dei pixel con la stessa direzione allora il valore viene preservato, altrimenti soppresso.

In altre parole solo i punti corrispondenti a massimi locali vengono presi in considerazione dai successivi passaggi dell'algoritmo.

Tracciare i bordi attraverso l'isteresi

Nonostante le accortezze usate, vengono ancora rilevati bordi spuri a causa di disturbi e variazioni di colore. Per tenere conto di queste risposte spurie, è essenziale filtrare i pixel del bordo attraverso l'uso di due valori di soglia: una bassa ed una alta, che vengono confrontate con il gradiente in ciascun punto:

- se il valore del gradiente in un punto è superiore alla soglia alta, allora il punto è considerato facente parte di un bordo;

- se il valore è minore della soglia bassa, allora il pixel è scartato;
- se il valore è compreso fra le due soglie, allora il pixel viene considerato appartenente ad un bordo solo se contiguo ad un punto precedentemente accettato.

Questo procedimento di doppia sogliatura richiama al comportamento del fenomeno di isteresi, e viene applicato in quanto è difficile stabilire, a priori un unico valore di soglia che possa da solo stabilire se una certa intensità giustifica la presenza di un bordo oppure no. Il risultato ottenuto al termine di questi passaggi è un'immagine binaria in cui ogni pixel è classificato come facente o non facente parte di un bordo. Abbiamo così una mappa contenente un insieme di linee di contorno, mediante elaborazioni successive, può essere descritta da una poligonale. [14]

3.16 Laplaciano

Gli *edge-detector* del primo ordine si basano sulla premessa che la derivata evidenzia le variazioni di intensità; nel grafico della derivata prima troviamo un picco laddove il tasso di variazione è massimo; nella derivata seconda avremo un picco laddove la derivata prima ha crescita massima e zero quando la derivata prima è costante; perciò un massimo locale nella derivata prima corrisponde ad un *zero-crossing* nel grafico della derivata seconda. L'operatore Laplaciano implementa la derivata del secondo ordine. Questo diverso meccanismo di ricerca del contorno è più preciso e robusto, rispetto a quello del procedimento che sfrutta l'operatore di Sobel; inoltre è meno sensibile al rumore e spesso non è necessario agire tramite sogliatura in fase di post-processing [15]. La derivata del secondo ordine può essere approssimata da una differenza fra due derivate del primo ordine adiacenti:

$$f''(x) \cong f'(x) - f'(x+1)$$

$$f''(x) \cong -f(x) + 2f(x+1) - f(x+2)$$

Da queste considerazioni formuliamo il *template* orizzontale del secondo ordine:

–1	2	–1

Combinando l'operatore Laplaciano riferito ad *x* e quello riferito a *y*, otteniamo:

0	–1	0
–1	4	–1
0	–1	0

L'operatore Laplaciano è raramente usato nella sua forma base, in quanto risente negativamente del rumore casuale; spesso l'operatore di *smoothing* Gaussiano viene incorporato ad esso;

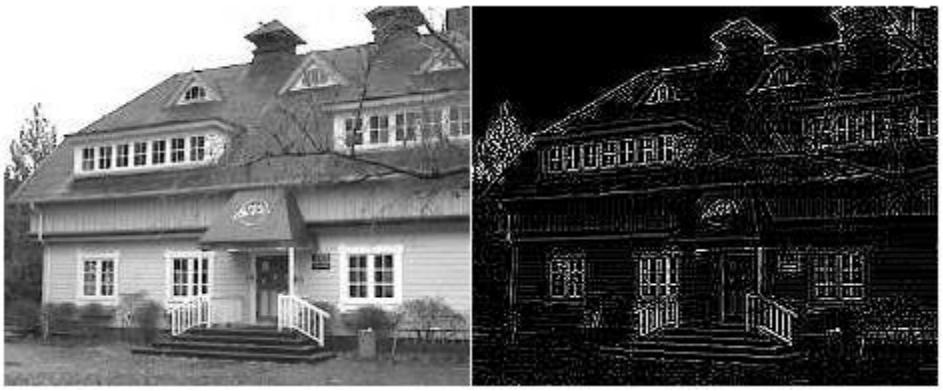


Figura 21 Applicazione dell'operatore Laplaciano

Sia l'operatore Laplaciano che il gradiente ottenuto con l'operatore di Sobel evidenziano le discontinuità di intensità all'interno di un'immagine; è stato dimostrato che il Laplaciano è superiore nel rilevare dettagli fini, ma fornisce immagini più rumorose, mentre il gradiente è migliore nel rilevamento dei bordi. [16]

3.17 Rilevatore di Harris

Intuitivamente, pensiamo a un angolo come un rapido cambio di direzione in una curva. Gli angoli sono caratteristiche molto efficaci perché sono ben

distinguibili e ragionevolmente invarianti rispetto alla prospettiva. Grazie a queste caratteristiche, gli angoli vengono utilizzati regolarmente per abbinare le caratteristiche dell'immagine in applicazioni come il tracciamento per la navigazione autonoma, algoritmi di Computer Vision stereo e *query* di database di immagini.

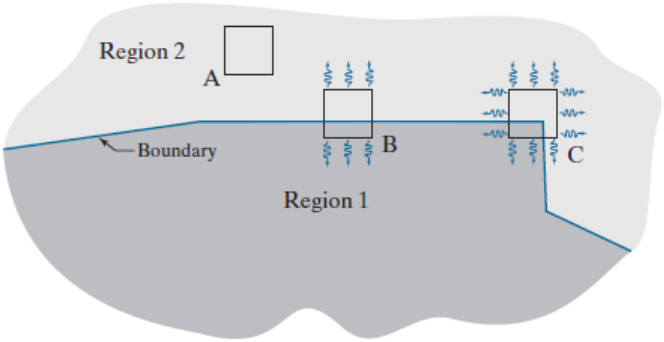


Figura 22 Zone di interesse nell'algoritmo di Harris [10]

Per spiegare l'idea di base del rilevatore di Harris prendiamo in considerazione figura 22.

Gli angoli vengono rilevati convolvendo una piccola finestra con un'immagine, come abbiamo visto ad esempio per il filtraggio spaziale. La finestra del rilevatore è progettata per calcolare i cambiamenti di intensità. Siamo interessati a tre scenari: (1) Aree di zero (o piccola) variazione di intensità in tutte le direzioni, una condizione che si verifica quando la finestra si trova in una regione costante (o quasi costante), come nella posizione A in figura 22; (2) aree di cambiamento in una direzione ma nessun (o piccolo) cambiamento nella direzione ortogonale, cosa che accade quando la finestra si estende su un confine tra due regioni, come nella posizione B; e (3) aree di cambiamenti significativi in tutte le direzioni, una condizione che si verifica quando la finestra contiene un angolo (o punti isolati), come nella posizione C.

Sia *f* un'immagine e sia *f (s, t)* una finestra dell'immagine definita dai valori *(s, t)*. Una finestra della stessa dimensione, ma spostata di *(x, y)*, è data da *f(s+ x, t + y).* Quindi, la somma ponderata delle differenze al quadrato tra le due finestre è data da

$$C(x,y) = \sum_s \sum_t (\omega(s,t)[f(x+s, y+t) + f(s,t)]^2$$

dove *ω (s, t)* è una funzione che determina i pesi con cui i valori interni alla finestra devono essere considerati. La funzione C presenterà valori maggiori in zone in cui c'è una variazione di intensità maggiore.

La finestra spostata può essere approssimata dai termini lineari della serie di Taylor:

$$f(s+x, t+y) = f(s,t) + xf_x(s,t) + yf_y(s,t)$$

Possiamo scrivere:

$$C(x,y) = \sum_s \sum_t \left(\omega(s,t)\left[xf_x(s,t) + yf_y(s,t)\right]\right)^2$$

L'equazione può essere scritta in forma matriciale come:

$$C(x,y) = \begin{vmatrix} x \\ y \end{vmatrix} M \begin{vmatrix} x & y \end{vmatrix}$$

in cui :

$$M = \sum_s \sum_t (\omega(s,t)A \qquad A = \begin{vmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{vmatrix}$$

La funzione di ponderazione ω(s, t) utilizzata nel rivelatore di Harris è isotropa ed ha generalmente una delle due forme seguenti:

- forma a gradino, con valore 1 all'interno della finestra e 0 altrove, indicata quando il rumore è molto poco o quando c'è necessita di alte velocità di risposta;
- ω può corrispondere ad una funzione esponenziale che ne descrive la forma (ad esempio una campana di Gauss).

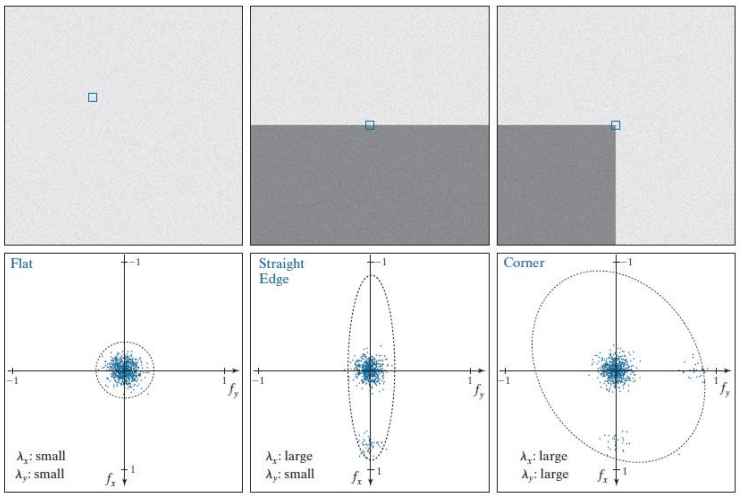


Figura 23 Comportamento autovalori nei tre diversi casi [10]

Come illustrato in Figura 23, un angolo è caratterizzato da valori elevati nella regione C, in entrambe le direzioni spaziali. Tuttavia, quando la finestra si estende su un unico bordo, ci sarà una risposta in una direzione.

Gli autovettori di una matrice simmetrica reale (come M sopra) sono non negativi e puntano nella direzione della massima distribuzione dei dati e gli autovalori corrispondenti sono proporzionali alla quantità di dati distribuiti nella direzione degli autovettori. In effetti, gli autovettori sono gli assi maggiori di un'ellisse che si adatta ai dati e la grandezza degli autovalori indica le distanze dal centro dell'ellisse ai punti in cui interseca gli assi maggiori. La Figura 23 mostra come possiamo usare queste proprietà per distinguere i tre casi a cui siamo interessati. Tuttavia, invece di utilizzare gli autovalori (che sono costosi da calcolare), il rivelatore di Harris utilizza una misura basata sul fatto che la traccia di una matrice quadrata è uguale alla

somma dei suoi autovalori e il suo determinante è uguale al prodotto dei suoi autovalori. La misura è definita come:

$$R = \lambda_x \lambda_y - k(\lambda_x + \lambda_y)^2$$
$$= \det(\mathbf{M}) - k\,\mathrm{trace}^2(\mathbf{M})$$

La costante k è determinata empiricamente e il suo intervallo è generlmente compreso fra 0.04 e 0.06

La misura R ha grandi valori positivi quando entrambi gli autovalori sono grandi, indicando la presenza di un angolo; ha grandi valori negativi quando un autovalore è grande e l'altro piccolo, indicando un bordo; e il suo valore assoluto è piccolo quando entrambi gli autovalori sono piccoli, indicando che la finestra dell'immagine in esame ha valori di intensità pressoché costanti. Generalmente si fa uso di un valore di soglia in maniera da prendere solo i valori più significativi. [10]

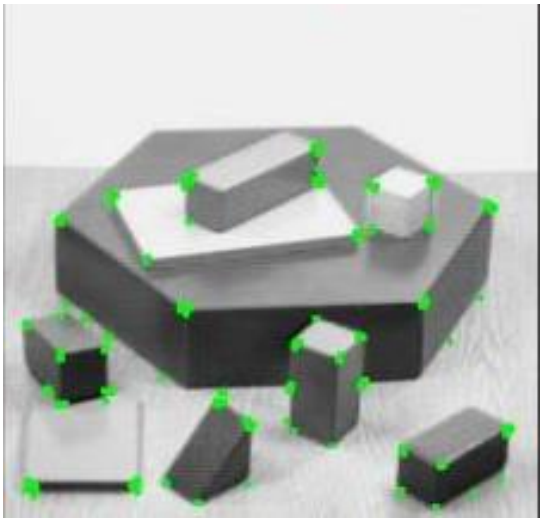


Figura 24 Individuazione degli angoli attraverso l'algoritmo di Harris

3.18 Template-matching

Il *template matching* è concettualmente un processo semplice. Ciò che vogliamo fare è confrontare un *template* con un immagine; il template è rappresentato da un'immagine di dimensione generalmente ridotta, che contiene la forma che stiamo cercando di trovare. Quindi posizioniamo il

template sull'immagine e contiamo pixel per pixel quanti punti coincidono con l'immagine. La procedura è ripetuta per l'intera immagine ed il punto in cui il posizionamento del *template* offre un conteggio maggiore è considerato essere il punto, all'interno dell'immagine, in cui la forma ricercata giace. Questa operazione può essere considerata simile alla convoluzione con un *template*; la differenza è che in questo caso non avremo una somma pesata dei dati dell'immagine, ma ciò che sommeremo sarà il numero di match. [9]

In precedenza abbiamo parlato brevemente della funzione di correlazione:

$$(\omega \circledast f)(x,y) = \sum_{s=-a}^{a} (\omega(s,t) * f(x+s, y+t)]\!]$$

Questa equazione viene valutata per tutti i valori delle variabili di spostamento *x* e *y,* quindi tutti gli elementi di ω visitano ogni pixel di *f*. La correlazione ha i suoi valori più alti nella regione dove *f* e ω sono uguali o quasi uguali. In altre parole troviamo le posizioni in cui ω corrisponde a una regione di *f,* ricercando i luoghi di massimo della funzione di correlazione. Ma questa equazione ha lo svantaggio che il risultato è sensibile ai cambiamenti nell'ampiezza di entrambe le funzioni. Al fine di normalizzare la correlazione ai cambiamenti di ampiezza in una o entrambe le funzioni, eseguiamo il *matching* utilizzando invece il coefficiente di correlazione:

$$\gamma(x,y) = \frac{\sum_s \sum_t [w(s,t) - \bar{w}][f(x+s, y+t) - \bar{f}_{xy}]}{\left\{ \sum_s \sum_t [w(s,t) - \bar{w}]^2 \sum_s \sum_t [f(x+s, y+t) - \bar{f}_{xy}]^2 \right\}^{\frac{1}{2}}}$$

dove $\overline{\omega}$ è il valore medio del *kernel* (calcolato una sola volta), e $\overline{f}$ è il valore medio di *f* nella regione coincidente con ω. Nella correlazione fra immagini, ω spesso rappresenta un *template* (cioè un'immagine del prototipo) e la correlazione è indicata come processo di *template matching*.

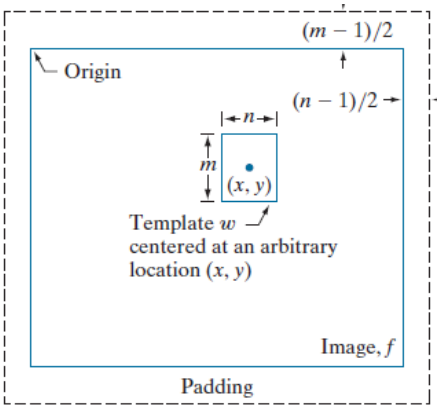


Figura 25 Meccanismo di template matching [10]

Si può dimostrare che il coefficiente $\gamma(x,y)$ assume valori compresi fra −1 e 1, ed è quindi normalizzato ai cambiamenti nelle ampiezze di ω e *f*. Il valore massimo si verifica quando la ω normalizzata e la corrispondente regione normalizzata in *f* sono identiche.

Alla fine della procedura, cerchiamo il massimo in $\gamma(x,y)$ per trovare dove si è verificata la migliore corrispondenza. È possibile avere più posizioni in $\gamma(x,y)$ con lo stesso valore massimo, il che indica la presenza di più corrispondenze tra ω e *f*. [10]

La trasformata di Hough, che vedremo più avanti, ad esempio definisce un efficace implementazione di *template-matching* per *template* binari. Questa tecnica è in grado di estrarre forme semplici come linee, ellissi, rettangoli.

3.19 Trasforamata di Hough

La trasformata di Hough è una tecnica grazie alla quale è possibile localizzare forme all'interno di un'immagine. In particolare è utilizzata per estrarre linee, cerchi ed elissi. Essa fu ideata da Hough nel 1962. Fu estesa ed implementata da Richard Duda e Peter Hart nel 1972, che ne riconobbero il grande potenziale come algoritmo per l'*image processing*; il principale vantaggio consiste nell'opportunità di ottenere uno stesso risultato rispetto ad un algoritmo di *template matching*, ma in un tempo minore. Questo è ottenuto grazie ad una riformulazione del processo di

template matching, basato su un approccio detto *evidence gathering* (raccolta delle prove), in cui le prove corrispondono ai conteggi raccolti in un accumulatore. La trasformata di Hough definisce un *mapping* fra i punti dell'immagine e lo spazio dell'accumulatore (*Hough space*). Il *mapping* è ottenuto in maniera computazionalmente vantaggiosa basandosi su una funzione che descrive la forma da ricercare. Questo *mapping* richiede molte meno risorse computazionali rispetto al *template matching*.

In un piano cartesiano i punti appartenenti ad una linea, sono messi in relazione da un coefficiente angolare m e da un coefficiente noto c:

$$y = mx + c$$

La stessa equazione può essere scritta in forma omogenea:

$$Ay + Bx + 1 = 0$$

dove $A=-1/c$ e $B=m/c$. Quindi una retta è definita dalla coppia (A,B).

Possiamo anche dire che una coppia (x,y) definisce una retta nello spazio con parametri (A,B). Quindi possiamo considerare l'equazione come una fascio di rette passante per un punto fissato, oppure come l'equazione di una retta con parametri (A,B) fissati. Quindi le due coppie possono essere usate per definire punti o linee allo stesso tempo. La trasformata di Hough raccoglie evidenze della coppia (A,B) considerando che tutti i punti (x,y) definiscono la stessa linea nello spazio (A,B). Dunque se l'insieme dei punti appartenenti ad una retta definiscono la retta (A,B) allora:

$$Ay_i + Bx_i + 1 = 0$$

In termini di parametrizzazione cartesiana abbiamo:

$$c = -x_i m + y_i$$

Quindi , per determinare la retta possiamo trovare il valore dei parametri (m,c). Nella figura 26 possiamo notare come i punti (x_i, y_i) e (x_j, y_j)

rappresentano due punti di una retta nello spazio cartesiano, mentre rappresentano due rette nello spazio (A,B). Ogni volta che due punti dello spazio cartesiano appartengono alla stessa linea, essi definiscono due linee nello spazio di Hough. In maniera efficiente la trasformata di Hough conta le potenziali soluzioni in un accumulatore che registra le prove. Più punti appartengono alla stessa linea più grande sarà il valore dell'accumulatore per quei particolari parametri.

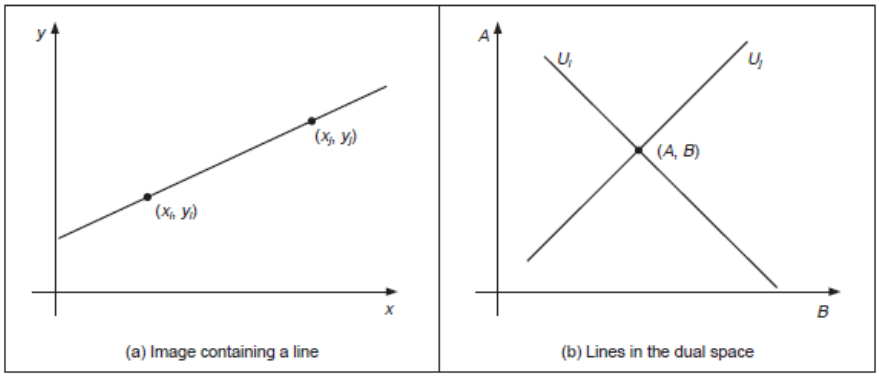


Figura 26 Esempio Trasformata di Hough [9]

Il problema di rilevamento delle linee viene trasformato in questo modo in un problema di ricerca di massimo all'interno dell'accumulatore. Questa strategia è robusta e in grado di gestire rumore ed occlusioni. [9]

Il procedimento prescinde dal tipo di parametrizzazione utilizzata; potrebbe essere ad esempio utilizzata una parametrizzazione in coordinate polari.

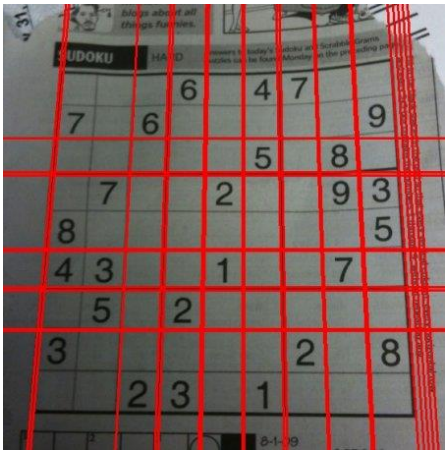


Figura 27 Linee rilevate attraverso trasformata di Hough

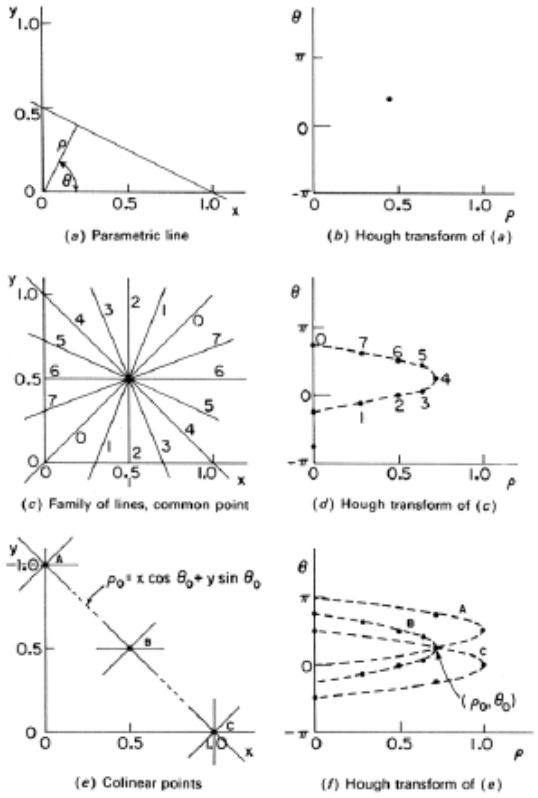


Figura 28 Trasformata di Hough [17]

3.20 Python

Python è il linguaggio scelto per implementare il codice di questo progetto di tesi.

Python è un linguaggio di programmazione Open Source molto popolare.

Soprattutto ultimamente si è dimostrato particolarmente adatto per quanto riguarda l'implementazione di software inerenti al campo della statistica e del *data-science*.

Nasce nel 1991 ideato da Guido van Rossum; è un linguaggio di programmazione *general purpose*, *high-level*, la cui enfasi è stata posta fin dall'inizio sulla leggibilità e sulla compattezza del codice scritto.

Python dispone di una grandissima quantità di librerie; in particolare nel campo della Computer Vision impossibile non citare *opencv*, la quale

implementa moltissime delle tecniche più utilizzate per l'analisi e la manipolazione delle immagini digitali.

Un'altra libreria rivelatasi assai utile è Music21. Essa oltre ad offrire una serie di strumenti utili soprattutto nel campo dell'analisi, offre la possibilità di implementare la creazione di un brano e la sua esportazione in vari formati come MIDI e XML.

4 Algoritmo ed implementazione

L'idea dello sviluppo di questo progetto nasce dall'esigenza, mia e di molti miei colleghi studenti presso il conservatorio, di velocizzare il processo di inserimento di brani, posseduti in forma cartacea o in un qualsiasi formato per la codifica delle immagini digitali, in programmi di editor musicale. Ad esempio nel corso Tecniche di orchestrazione, un tipico esercizio consiste nel dover orchestrare, rispettando determinate indicazioni, un brano per piano forte; quindi il maestro invia lo spartito[5] per posta elettronica; l'acquisizione automatica di tale spartito, anche se non perfetta, consentirebbe il risparmio di tempo prezioso da parte dello studente; la correzione dei saltuari errori commessi dal programma richiederebbe una quantità di tempo minima; lo studente potrebbe impiegare così la maggior parte del tempo affrontando la parte più significativa del compito, essendo il processo meccanico di trascrizione, per gran parte, svolto.

Lo scenario per cui ho pensato questo programma perciò non necessità di tempi di risposta brevissimi, né tanto meno di tempi di risposta *rael-time*. L'attenzione è stata rivola principalmente al riconoscimento, il più possibile fedele, di partiture o spartiti monofonici e polifonici a singola voce.

Questo progetto non indaga a fondo sulle tecniche degli algoritmi OCR per il riconoscimento dei caratteri, perciò non ripone fiducia negli algoritmi OCR utilizzati; questo significa che, a livello di testo, sono tenute in considerazione solo informazioni marginali, come il titolo e l'autore del brano, le quali, a seguito di rilevamenti errati, non avrebbero comunque alcuna conseguenza sul processo di codifica delle informazione musicale; per quanto riguarda invece informazioni fondamentali, come ad esempio la frazione numerica, che indica la durata di una battuta, ho ritenuto necessario

[5] Lo spartito è una riduzione per canto e pianoforte di un brano originariamente destinato a canto ed orchestra [21]

l’inserimento manuale, da effettuarsi prima che l’intero processo abbia inizio.

4.1 Segmentazione

La prima operazione da fare per ottenere una partizione dell’immagine nei suoi elementi base è l’individuazione e la rimozione dei pentagrammi; il motivo per cui vogliamo rilevare e rimuovere le linee del pentagramma risiede nella necessità di isolare le *feature* musicali affinché esse possano essere rilevate nella maniera più affidabile possibile; tuttavia il procedimento di rimozione dei pentagrammi potrebbe andare a degradare altri simboli; esistono infatti autori che hanno suggerito algoritmi che non affrontano questo passaggio. La rimozione dei pentagrammi facilita dei procedimenti nelle fasi di rilevamento successive, ma introduce rumore. Ad esempio, i simboli sono spesso spezzati nel corso di questo processo, oppure frammenti di linee che non vengono rimosse completamente possono essere interpretati come parte di simboli o nuovi simboli. D’altro canto la conservazione di quante più informazioni possibili per i passaggi successivi, rischia di aumentare il costo computazionale e di rendere troppo difficoltoso il processo di rilevamento dei dati. La scelta presa per l’implementazione di questo algoritmo, coinvolge due diversi metodi per la rimozione dei pentagrammi; a seconda della *feature* che si intende estrarre, viene utilizzato il metodo di rimozione del pentagramma che la degrada meno.

Il rilevamento del pentagramma può essere complicato per una serie di motivi. Spesso immagini ottenute mediante scansione o foto, presentano linee di pentagramma distorte o curve; nella versione attuale dell’algoritmo tale problematica non è affrontata; per il corretto funzionamento del programma sarà necessario ricevere come input partiture con linee dritte e parallele fra di loro.

4.2 Struttura del processo

La struttura generale del processo può essere schematizza in tre fasi principali:

- **fase di segmentazione**: si procede con all'acquisizione dell'immagine; vengono effettuate operazioni di pre-processing come la conversione dell'immagine in scala di grigi e l'applicazione di un filtro che ne riduca il rumore; successivamente si procede col rilevamento dei pentagrammi, e col calcolo del parametro che sarà l'unità di misura di tutte le nostre misurazioni in termine di dimensione, ovvero la dimensione di uno spazio fra due righi;
- **fase di ricerca dei simboli:** attraverso l'uso di alcune delle tecniche dell'*image processing* di cui abbiamo parlato nel capitolo precedente, si procede alla ricerca delle *feature* a cui siamo interessati, in alcuni casi considerando l'immagine nelle sua interezza, in altri, localizzando la ricerca in particolari regioni di interesse; le primitive estratte vengono sottoposte a processi di confronto che ne definiscono categoria e relazioni; gli elementi estratti vengono memorizzati successivamente in una struttura dati che sarà l'input della fase successiva;
- **fase di costruzione della partitura:** grazie alle informazioni raccolte al passo precedente, abbiamo gli elementi, che legati a procedimenti logici che ricostruiscono le regole della grammatica musicale, ci permettono di assemblare, mattone per mattone, la nostra partitura; questa fase produce un file xml che può essere visualizzato, ascoltato e modificato tramite un programma di notazione musicale.

4.2.1 Metodo di rilevamento del pentagramma

Il metodo di rilevamento dei pentagrammi utilizzato in questo programma prevede tre fasi:

- fase di segmentazione: attraverso l'uso di operatori morfologici viene eliminato tutto ciò che non sia pentagramma;
- fase di rilevamento: tramite l'applicazione dell'algoritmo per il riconoscimento dei contorni di Canny, individuiamo dimensione e centro di massa di ciascuna delle forme mantenute dall'immagine originale;
- fase di applicazione delle regole musicali: grazie all'implementazione delle regole musicali siamo in grado di determinare quali rilevamenti siano coerenti con l'oggetto della nostra ricerca; avremo quindi la possibilità di scelta riguardo il mantenimento o l'esclusione dei righi individuati;

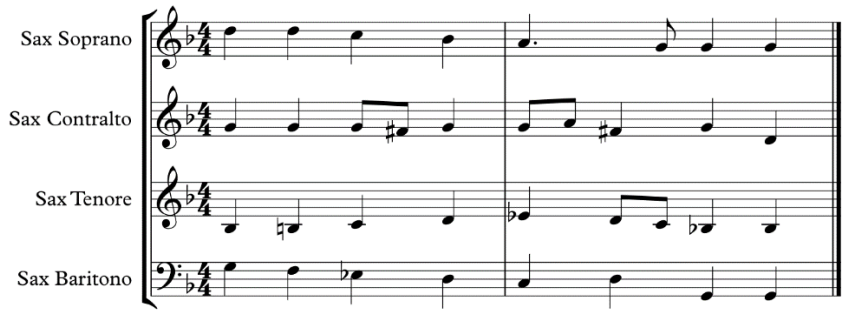


Figura 29 Immagine di una partitura

Fase di segmentazione

Come già spiegato nel capitolo precedente, le operazioni morfologiche delle immagini binarie si basano sulla relazione geometrica o sulla connettività dei pixel che si ritiene appartengano alla stessa classe; le operazioni elementari per le trasformazioni morfologiche sono dilatazione ed erosione; esse rispettivamente dilatano ed erodono l'immagine, secondo le dimensioni di un *kernel*; in questo primo passaggio del processo vogliamo

che solo i pentagrammi rimangano visibili e quindi che solo essi siano facilmente rintracciabili.

L'idea alla base di questo passaggio è stata quella di eliminare tutti quegli elementi dell'immagine con larghezza inferiore a quella di un pentagramma; non essendo a questo punto del processo la misura della larghezza del pentagramma disponibile, si è scelto di prendere come riferimento la dimensione della pagina, giacché verosimilmente il pentagramma si estenderà per quasi tutta la sua ampiezza; per la rimozione degli oggetti indesiderati viene eseguita l'operazione di erosione utilizzando un *kernel* che ha altezza unitaria (di un pixel) e larghezza pari a metà della dimensione orizzontale della pagina. Questo è quello che otteniamo:

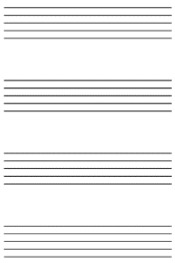


Figura 30 Immagine che ha subito un processo di erosione

Come visto nel capitolo precedente per ottenere le dimensioni originali degli oggetti rimasti eseguiamo l'operazione di dilatazione con lo stesso *kernel*, ottenendo:

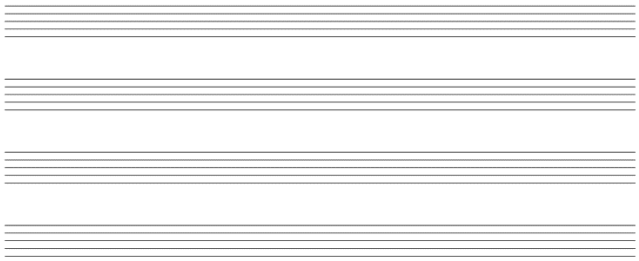


Figura 31 Immagine che ha subito processo di opening

L’operazione che abbiamo messo in atto ha il nome di *opening*; abbiamo ottenuto in questo modo l’immagine segmentata, la quale sarà oggetto del processo di rilevamento dei contorni nella fase successiva. Come già detto alla base di questo passaggio abbiamo scelto di utilizzare l’algoritmo per il rilevamento dei contorni di Canny.

Fase di rilevamento pentagrammi

Questa fase ha lo scopo di ottenere le coordinate e il centro di massa degli oggetti che abbiamo estratto nella fase di segmentazione. Si procedere con l’esecuzione dell’algoritmo di Canny, grazie al quale otteniamo come risultato l’immagine di seguito:

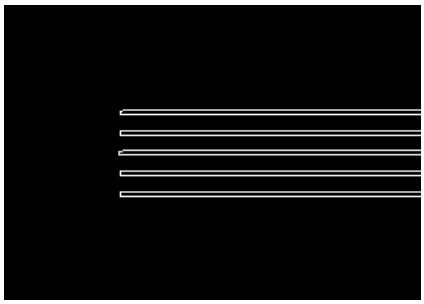


Figura 32 Ingrandimento di un'immagine prodotta dall'algoritmo di edge detection di Canny

Per trovare dimensioni e coordinate ci serviamo della funzione *findContours* di *opencv*, la quale ci permette di elaborare i contorni ottenuti con l’algoritmo di Canny. Questa funzione restituisce un array di oggetti contorno ai quali possiamo applicare diverse funzioni di analisi. In particolare sfruttando la funzione nativa di *opencv boundingBox*, siamo in grado di recuperare i valori che descrivono le coordinate e le dimensioni del rettangolo (con base ed altezza parallele agli assi) che circoscrive il contorno considerato. I dati rintracciati vengono memorizzati in un array e forniti alla fase successiva.

Inizialmente per questa particolare fase, di rilevamento dei pentagrammi, si era preso in considerazione l’uso della trasformata di Hough per il

rintracciamento delle linee; tuttavia questa operazione non ci fornisce le coordinate e le misure esatte del segmento individuato, bensì l'equazione della retta su cui questo segmento giace; per questo motivo si è ritenuto che la trasformata di Hough non fosse la tecnica migliore da prendere in considerazione per il problema da affrontare.

Fase di applicazione delle regole musicali

Ora che possediamo coordinate e posizione di tutti i righi presenti nella pagina andiamo ad analizzare i righi nel loro insieme, oltre che estrarre parametri di cui ci serviremo in futuro. A partire dalle conoscenze della grammatica musicale possiamo implementare la logica che lega questi elementi. Come ben sappiamo un pentagramma è composto da cinque righi equidistanti fra di loro; ordinando i righi rilevati secondo le ascisse andremo alla ricerca di tuple composte da cinque righi equidistanti. Memorizziamo dunque i risultati di questa analisi in una struttura dati. La distanza fra un rigo e l'altro, ovvero lo *spazio*, è la dimensione a cui faremo riferimento per tutte le misurazioni successive. Essa coincide approssimativamente con il diametro della testa di una nota; questa considerazione è valida per qualsiasi partitura di qualsiasi dimensione; inoltre non solo le note, ma anche gran parte degli altri elementi potranno avere la propria dimensione approssimata da un fattore moltiplicativo associato a questo 'calibro'.

4.2.2 Metodi di rimozione del pentagramma

La rimozione dei pentagrammi è un processo comune a moltissimi dei sistemi OMR in letteratura; essa si dimostra un passaggio in alcuni casi essenziale, in quanto le linee di pentagramma intersecano la maggioranza degli elementi che si intende identificare, rendendo difficoltoso il loro tracciamento; tuttavia, i processi di rimozione del pentagramma introducono rumore che spesso corrompe l'integrità delle figure originali. In questo progetto si è scelto di utilizzare due diverse tecniche per rimuovere il pentagramma; a seconda della *feature* che si vuole estrarre,

utilizziamo il procedimento che la degrada di meno. La prima delle due tecniche fa uso solo di operazioni morfologiche, l'altra invece, prevede la costruzione di una maschera, che sarà, insieme all'immagine originale, termine di operazioni logiche fra immagini.

4.2.2.1 Rimozione del pentagramma tramite opening (metodo 1)

Questo primo metodo consiste in una semplice operazione di *opening*. *L'opening* spesso si dimostra assai utile in molte applicazioni per l'eliminazione di un particolare tipo di rumore che nell'ambito delle telecomunicazioni è denominato *noisy white*; possiamo vedere un esempio in figura 33.

Figura 33 Rimozione del rumore tramite opening

Alla base del primo procedimento di rimozione del pentagramma, c'è l'idea di considerare le linee di pentagramma come fossero un rumore da eliminare; eseguendo l'operazione di opening con un *kernel* di larghezza unitaria e altezza pari allo spessore del rigo (ottenuto in precedenza misurando l'altezza del rettangolo che lo conteneva) otteniamo:

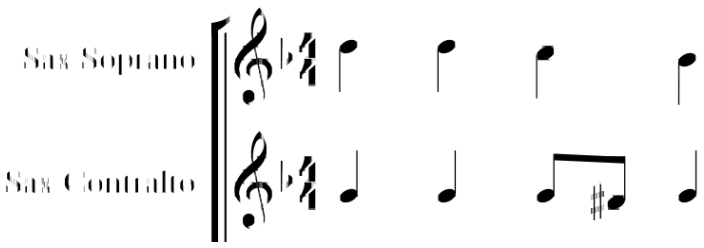
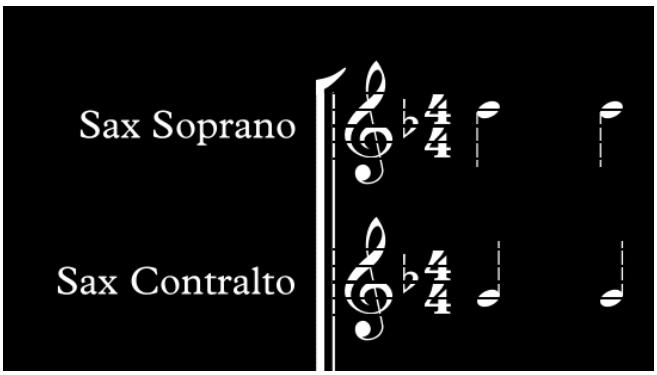


Figura 34 Rimozione dei pentagrammi con il primo metodo

Dalla figura 34 possiamo notare come, se per le note, la trasformazione sembra non aver sortito alcun effetto negativo; notiamo invece quanto le alterazione risultino degradate da questo procedimento.

4.2.2.2 Rimozione del pentagramma tramite maschera (metodo 2)

Questa seconda procedura coinvolge l'uso di operatori logici fra immagini. Come primo passo provvediamo alla creazione di una maschera che contenga la porzione di immagine che intendiamo eliminare, ovvero i pentagrammi; in fin dei conti possiamo utilizzare la medesima immagine ottenuta nel paragrafo 4.2.1, mostrata in figura 31, utilizzata per il rilevamento del pentagramma. Il procedimento di sottrazione della maschera all'immagine originale avviene attraverso l'utilizzo di operatori logici. Dapprima è necessario fare il complemento dell'immagine originale ottenuto tramite l'esecuzione dell'operatore logico NOT; per eliminare i pentagrammi dall'immagine originale sarà sufficiente compiere l'operazione logica AND fra il negativo dell'immagine completa e la maschera; il risultato ottenuto è l'immagine senza righi in negativo:



Applicando ora, nuovamente l'operatore NOT otteniamo finalmente il risultato desiderato; per colmare le fratture venutisi a creare nei simboli, laddove una volta passavano i righi del pentagramma, possiamo eseguire un'operazione di *closing* con un *kernel* che abbia altezza di un terzo di *spazio* e larghezza unitaria, ottenendo:

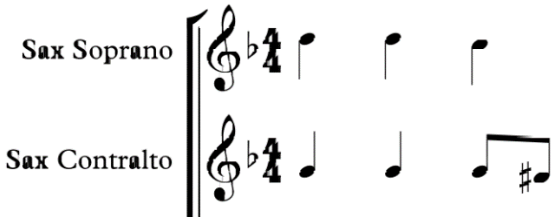


Figura 35 Rimozione pentagrammi con il metodo 2

Possiamo notare come con questo secondo metodo le alterazioni siano molto meno degradate rispetto al risultato ottenuto con la prima procedura; con questo secondo criterio però, l'operazione di *closing* eseguita alla fine porta ad effetti indesiderati: oltre a colmare piccoli buchi il *closing* potrebbe andare a fondere elementi vicini, come ad esempio potrebbero essere le note di un accordo, come possiamo osservare in figura 36. Questa seconda tecnica perciò non è indicata, ad esempio, se il fine ultimo è il rilevamento delle note.

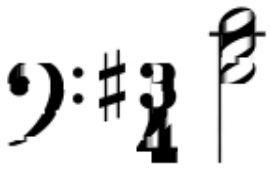


Figura 36 Effetti indesiderati del closing

4.3 Segmentazione e riconoscimento dei simboli

L'estrazione dei simboli musicali è l'operazione che segue il rilevamento e la rimozione delle linee di pentagramma. Il processo di segmentazione consiste nel localizzare ed isolare gli oggetti musicali per identificarli. La complessità di questa operazione riguarda l'identificazione di simboli

spezzati e sovrapposti che possono presentarsi in diverse dimensioni. L'approccio più comune per la segmentazione dei simboli è una scomposizione gerarchica dell'immagine. Una partitura viene prima analizzata e suddivisa per pentagrammi, quindi vengono estratti i simboli grafici elementari: teste di nota, pause, punti, gambi, *flag*, ecc. Sebbene in alcuni approcci le teste delle note vengano nuovamente unite con gambi e *flag* per la fase di classificazione, nella fase di segmentazione questi simboli sono considerati oggetti separati. Alcune volte, il passaggio di segmentazione delle primitive viene eseguito insieme all'attività di classificazione delle stesse.

In questo progetto vengono messi in atto procedimenti differenti a seconda dell'oggetto che si intende ricercare: in particolare le note vengono trattare in maniera differente rispetto ad altri simboli come le alterazioni e le chiavi; questa necessità sorge nel momento in cui si prende atto che, mentre questi ultimi assumono sempre la medesima forma, sia pure scalata o traslata, le note hanno una forma che è il risultato della composizione di una serie di forme geometriche, assemblate appositamente al fine di conferirle il valore e l'altezza desiderati. Per le note, il procedimento di riconoscimento prevede l'uso di una sequenza di operatori morfologici che ne isolino espressamente le particolari forme geometriche di cui esse si compongono (ellissi piene e vuote, gambi, travi, punti etc). La semplice conformazione delle primitive offre l'opportunità di utilizzare metodi di riconoscimento basati su semplici proprietà geometriche: le dimensioni del *boundingBox*, l'area, le coordinate, etc. Una volta individuate queste primitive avviene una fase di confronto che ne denota le relazioni.

Per quanto riguarda gli altri simboli, la ricerca avviene in due fasi: la prima prevede l'esecuzione di operazioni morfologiche e lo studio delle dimensioni degli oggetti ancora sconosciuti, candidati ad essere i simboli ricercati; in questo modo vengono individuate le zone in cui gli oggetti di

interesse potrebbero essere posizionati; la fase successiva consiste nell'effettuare l'operazione di *template maching* per diverse scale del campione da confrontare.

In questo stesso modulo del progetto sono analizzate le linee di battuta; grazie a questa procedura, determiniamo, non solo da quante battute è composto il brano, ma anche da quanti pentagrammi è composto una accollatura[6]; come è stato detto in precedenza infatti le linee di battute interne ai pentagrammi che si riferiscono a strumenti che devono suonare insieme, sono incolonnate. La buona riuscita di questa operazione è fondamentale, in quanto ne va della corretta organizzazione delle informazioni acquisite e della successiva codifica.

4.3.1 Rilevamento teste delle note

Il rilevamento delle note è sicuramente uno dei passaggi fondamentali di questo algoritmo; come possiamo ben immaginare, sono le note coloro che vanno a costituire concretamente il corpo di un brano musicale; poi naturalmente ci sarà tutta una serie di simboli che andranno a conferire a queste ultime un significato particolare o ne descriveranno alcune caratteristiche di esecuzione.

I passaggi che ci conducono ad acquisire i valori di altezza e durata delle note sono molteplici, e sono di seguito elencati:

- ricerca e analisi delle teste piene delle note;
- ricerca e analisi delle teste vuote delle note;
- identificazione dell'altezza della nota;
- ricerca e analisi dei gambi;

[6] Si dice *sistema* o *accollatura* la rappresentazione dell'insieme dei pentagrammi corrispondenti a parti che devono suonare insieme. La *linea di simultaneità* è un segmento che congiunge l'inizio di tutti i righi che fanno parte della stessa accolltura. [23]

- ricerca e analisi delle travi singole (successione di crome) e doppie (successione si semicrome);
- ricerca e analisi degli angoli formati fra gambi e travi nelle successioni di crome e semiminima;
- ricerca delle *flag* delle crome e delle semicrome;

Nella versione attuale del progetto non ci si è spinti oltre rilevamento del valore di durata della semicroma; il programma non sarà perciò in grado di rilevare note con una durata minore o uguale della biscroma.

4.3.1.1 Ricerca e analisi delle teste piene delle note

La notazione musicale prevede due tipi di testa di nota:

- piena: posseduta da tutte quelle note che hanno valore uguale o minore della semiminima;
- vuota: caratteristica propria della minima e della semibreve.

I due casi richiederanno dei trattamenti diversi per il loro rilevamento.

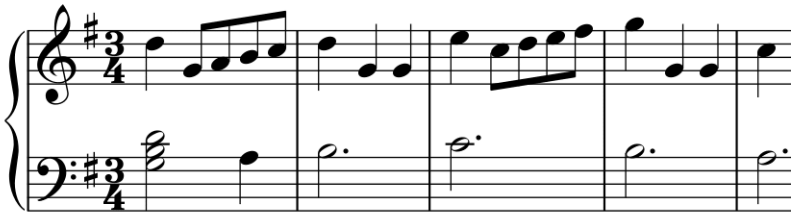


Figura 37 Frammento di partitura preso in esame per il rilevamento delle note

Segmentazione teste piene

Andiamo ad esporre ora il primo caso che è il più semplice ed immediato. Come già noto un primo passaggio sarà rappresentato dalla rimozione dei pentagrammi. Nel paragrafo 4.2.2 abbiamo mostrato come per l'individuazione delle note il metodo di rimozione del pentagramma maggiormente indicato è il primo; esso prevede una trasformazione tramite

l'operazione morfologica di *opening* con *kernel* di altezza pari allo spessore del rigo e larghezza unitaria. Come già mostrato in figura 34 ciò che otteniamo è un'immagine che ritrae, oltre alle chiavi e alle alterazioni visibilmente degradate, le teste delle note piene con il proprio gambo. Per isolare le teste è necessaria la rimozione dei gambi; un gambo è un segmento verticale che ha spessore comparabile a quello di una linea di pentagramma; la maniera attraverso la quale si è pensato di procedere è forse anche la più intuitiva, ovvero, l'immagine senza pentagrammi verrà nuovamente sottoposta a processo di opening, con il medesimo *kernel* ma trasposto, ossia di altezza unitaria e larghezza pari allo spessore di un rigo; l'immagine che si ottiene è la seguente:

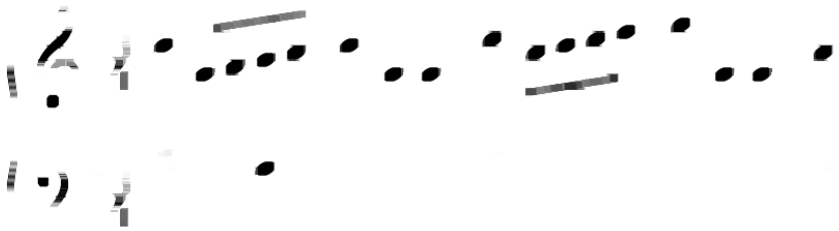


Figura 38 Segmentazione delle teste delle note

Possiamo osservare come le teste delle note piene siano ben visibili mentre il resto della partitura risulti, se non invisibile, comunque estremamente degradato; per eliminare le fioche ombre dei simboli che vogliamo escludere possiamo applicare un'operazione di sogliatura.

Rilevamento teste piene

Il passo successivo sarà dedicato al rilevamento e all'analisi dei contorni delle forme rimaste in figura 38. Per fare ciò, andremo a compiere un procedimento analogo a quello per il rilevamento dei righi; ovvero applicheremo l'algoritmo di Canny e successivamente, per ogni contorno trovato, andremo ad analizzare le dimensioni del rettangolo che lo

circoscrive; trattandosi di figure geometriche elementari, ci è consentito di basare le nostre misurazioni a partire dalle dimensioni del rettangolo circoscritto; difatti pensando alla testa di una nota come ad un cerchio, il lato del quadrato che lo circoscrive sarà la sua circonferenza. Basandoci sul fatto che la dimensione di uno *spazio*, calcolata nel paragrafo 4.2.1, è all'incirca pari al diametro di una testa, possiamo andare a compiere una prima selezione dei contorni rilevati; di tutti i contorni rilevati andremo a salvare solamente quelli che hanno la base e l'altezza del rettangolo che li circoscrive di dimensione simile a quella di uno spazio; i contorni che non rispettano tali criteri verranno scartati; andremo invece a memorizzare il centro di massa dei rettangoli che rispettano i criteri di selezione.
Nell'immagine seguente mostriamo in verde l'interno contorni rilevati (riempiamo i contorni solo per una visione più chiara), in rosso i rettangoli circoscritti a contorni ammessi ed in blu quelli che circoscrivono contorni scartati:

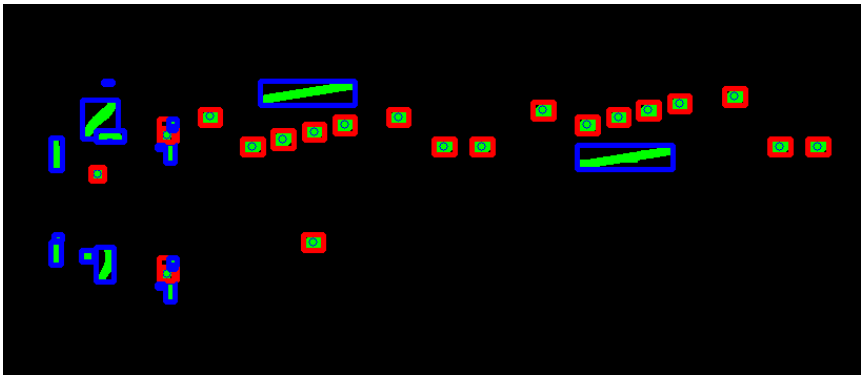


Figura 39 Risultato del rilevamento delle teste delle note

Possiamo notare la presenza di falsi positivi nella zona di inizio accollatura, in corrispondenza delle chiavi; le operazioni di opening hanno frammentato la chiave di violino, e i suoi frammenti hanno dato vita a rilevamenti fallaci; presto avremo strumenti necessari per poter escludere questi falsi positivi; verranno esclusi, ad esempio tutti i rilevamenti di note piene che non siano affiancate da un gambo (condizione necessaria per qualsiasi nota piena), oppure i rilevamenti il cui centro di massa giace all'interno di zone proibite.

4.3.1.2 Ricerca e analisi delle teste vuote delle note

Il procedimento di identificazione delle minime e delle semibrevi non è tanto intuitivo quanto il precedente; esso infatti richiede un numero assai maggiore di passaggi. Le difficoltà che si vengono ad aggiungere sono molteplici. Come prima cosa il processo di rimozione del pentagramma attraverso l'*opening*, potrebbe andare a fratturare l'ellisse che descrive la testa di nota vuota; perciò non è possibile applicare il metodo di rimozione del pentagramma tramite opening, o almeno non è possibile fare questa operazione come primo passaggio.

L'idea alla base di questa procedura, è quella di ricondurre questo caso a quello affrontato in precedenza, ovvero al caso di rilevamento delle teste piene; quello che andremo a fare è trasformare le teste di nota vuote in teste di nota piene. Il nostro obiettivo ora sarà colmare i piccoli contorni interni a alle minime e alle semibrevi. Per rilevare i contorni interni a minime e semiminime, ci serviremo della funzione *findContoour* di *opencv*; questa volta però non forniremo come input un'immagine elaborata da algoritmo di Canny, il quale fornisce i contorni di tutte le forme nell'immagine, bensì forniremo come input un'immagine, seppur segmentata, composta da forme integre; in queste condizioni la funzione *findContour* restituisce come risultato tutti i contorni chiusi presenti di per se nell'immagine; raccoglierò solo quelli che hanno dimensione comparabile a quella di una nota, ovvero di altezza e larghezza pari circa a quella di uno *spazio.*

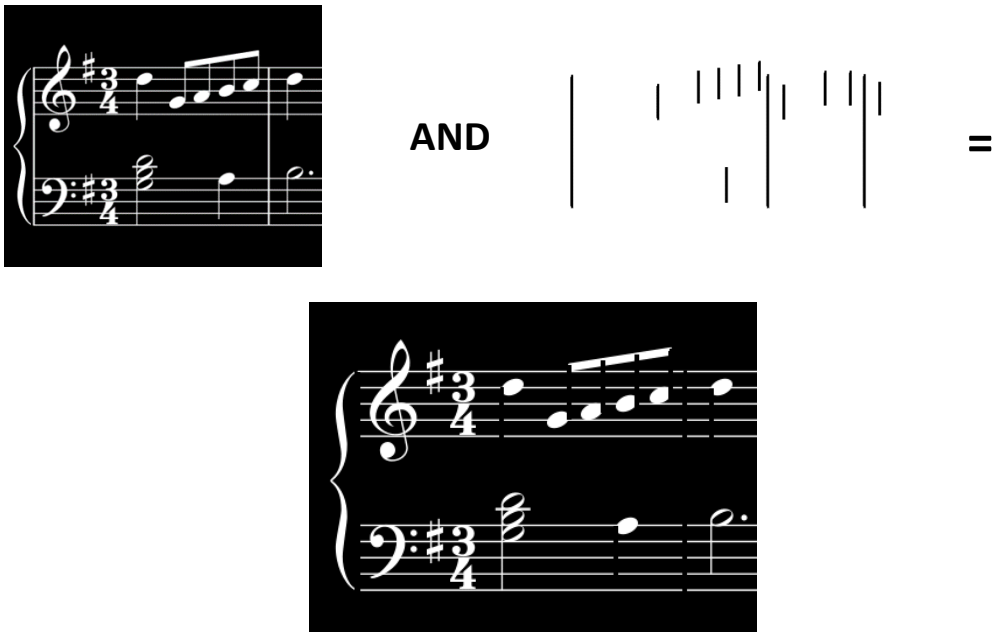
Andiamo ad elencare i passaggi necessari all'individuazione delle teste di nota vuote:

- rimozione dall'immagine completa di linee verticali e teste di nota piene;
- creazione di una maschera contenente il riempimento delle minime e delle semibrevi;

- somma della maschera con l'immagine completa;
- rilevamento delle note con la stessa tecnica utilizza per l'individuazione delle note con testa piena.

Rimozione di teste piene e linee verticali

Per tutelarci dal rilevamento di contorni che non siano quelli interni a minime e semibrevi, sarebbe opportuno eliminare dall'immagine completa ciò che fino ad ora abbiamo rilevato. Tuttavia, la nostra prima preoccupazione in questa fase, è quella di non fratturare le ellissi delle teste di nota vuote; per questo motivo posticiperemo la rimozione del pentagramma al momento in cui tali ellissi saranno riempiti; inoltre avremo cura di preservare le teste di nota piena anche durante l'eliminazione delle linee verticali; infatti elimineremo solo linee verticali molto lunghe, misurandole attraverso boundingBox, oppure le linee verticali vicine alle teste piene, di cui già conosciamo le coordinate. La rimozione delle linee verticali avviene nella medesima maniera usata per le linee di pentagramma metodo 2 (paragrafo 4.2.2.2), l'unica cosa che cambia è il contenuto della maschera in cui avremo linee verticali invece che orizzontali:



Tramite l'uso di una maschera contenente le teste di note piene trovate in precedenza, le eliminiamo, ottenendo:

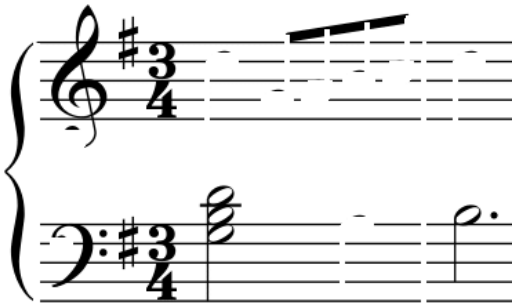


Figura 40 Immagine segmentata al fine di individuare le ellissi delle note vuote

Creazione maschera

A questo punto l'immagine è pronta per essere data in input a *findContour*, senza passare prima per l'algoritmo per il rilevamento dei contorni di Canny, in modo tale da trovare i contorni chiusi di per se esistenti in figura 40.

Attraverso l'uso del *boundingBox* selezioneremo solo i contorni le cui dimensioni sono comparabili a quelle di una nota; andremo a creare una maschera che presenta solo questi 'ripieni'.

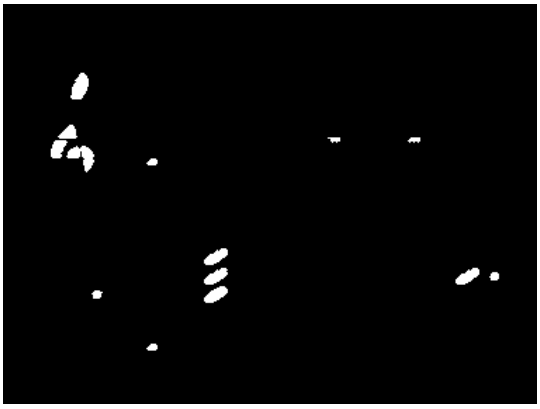


Figura 41 Maschera per il riempimento di minime e semibrevi

Tale maschera andrà sommata poi al negativo dell'immagine completa; il complemento di questa somma restituisce:

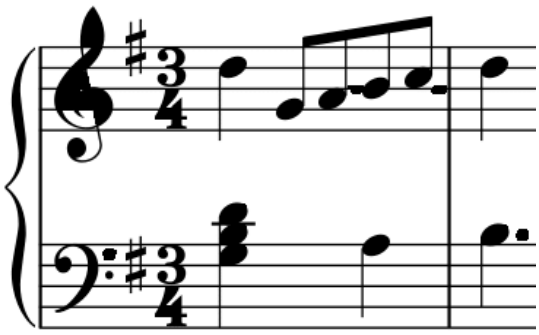


Figura 42 Immagine ottenuta riempiendo i contorni di minime e semibrevi

A questo punto possediamo una partitura di sole note con testa piena; possiamo finalmente utilizzare il medesimo procedimento eseguito nel paragrafo 4.3.1.1.

Otterremo dapprima la seguente rappresentazione:

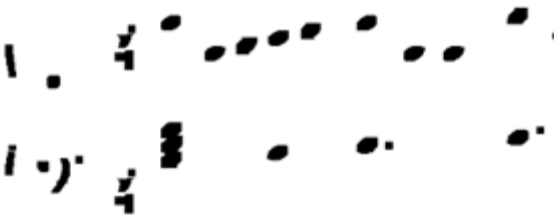


Figura 43 Immagine segmentata per rilevamento di minime

Andando poi a procedere con il rilevamento dei contorni di immagine otterremo come output un vettore contenente le coordinate delle teste piene più le coordinate delle teste vuote; già dal paragrafo precedente però conosciamo le coordinate delle note con testa piena; tutte le note appartenenti all'insieme delle note totali, che non appartengono al sottoinsieme delle note con testa piena, appartengono al sottoinsieme delle note con testa vuota.

Come è possibile notare in figura 44, l'accordo all'inizio del secondo pentagramma non viene rilevato in quanto le note sono andate a fondersi,

creando una forma fuori misura; nel prossimo paragrafo vedremo in che modo riusciremo ad ovviare a questo problema sfruttando le proprietà del gradiente dell'immagine.

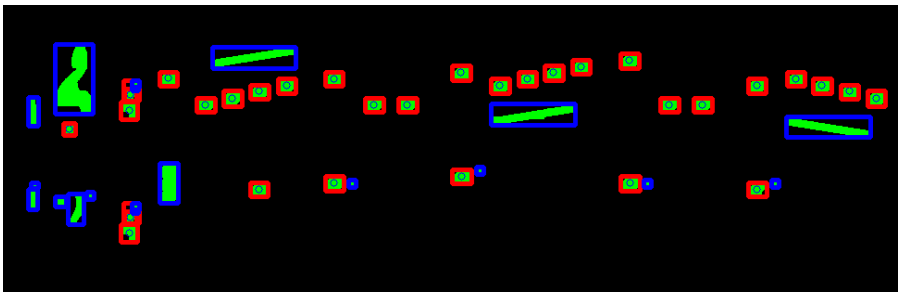


Figura 44 In verde l'interno di tutti i contorni, in rosso i BoundingBox dei contorni validi, in blu i BoundingBox dei contorni scartati

4.3.1.3 Utilizzo del gradiente per separare note vicine

Come abbiamo potuto notare in precedenza, nel caso di accordi il rilevatore di contorni rileva un'unica grande forma; quello che desideriamo è riuscire a distinguere le tre note in maniera distinta.

Ciò che si è pensato di fare è ottenere una maschera a partire dal gradiente dell'immagine ottenuta riempiendo le teste di nota vuote; tale maschera verrà sottratta all'immagine mostrata in figura 43.

Applicando il gradiente all'immagine in figura 42, otteniamo:

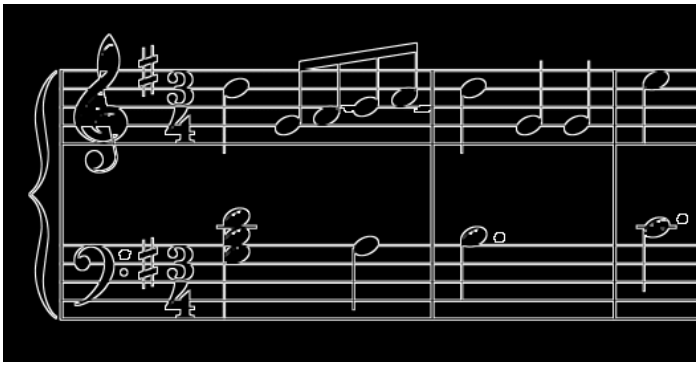
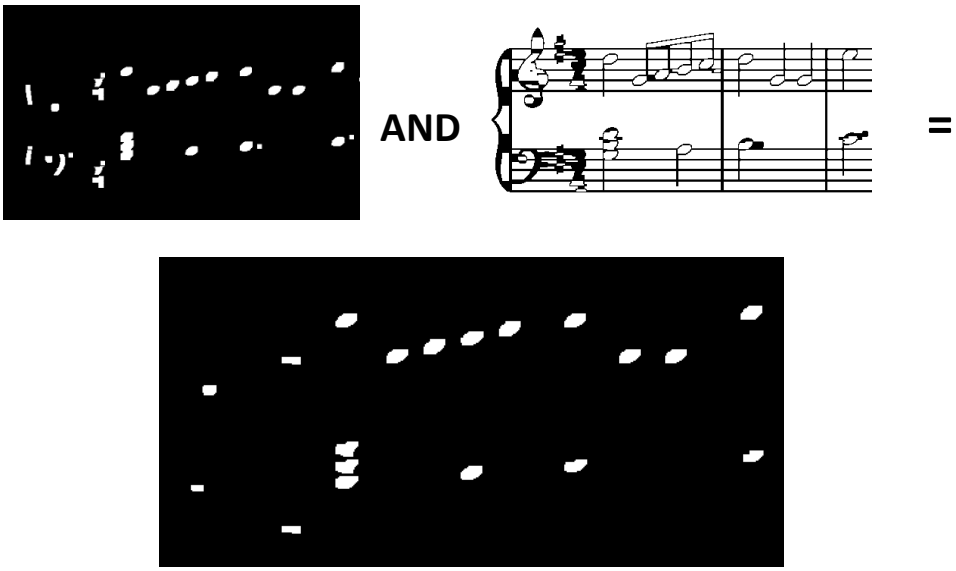


Figura 45 Gradiente dell'immagine

Quello che vogliamo ottenere è che i contorni di ciascuna nota dell'accordo risultino chiusi; andremo ad eseguire un'operazione di *closing* sull'immagine di figura 45, con *kernel* di altezza unitaria e larghezza pari ad

un terzo della dimensione di uno *spazio;* la maschera ottenuta andrà sottratta all'immagine usata per il precedente rilevamento, ovvero l'immagine le minime sono state riempite e le note dell'accordo risultano unite:



A questo punto, proseguiamo eseguendo con il solito processo che sfrutta l'algoritmo di Canny e la valutazione delle misure dei *boundingBox*; quello che otteniamo è mostrato di seguito:

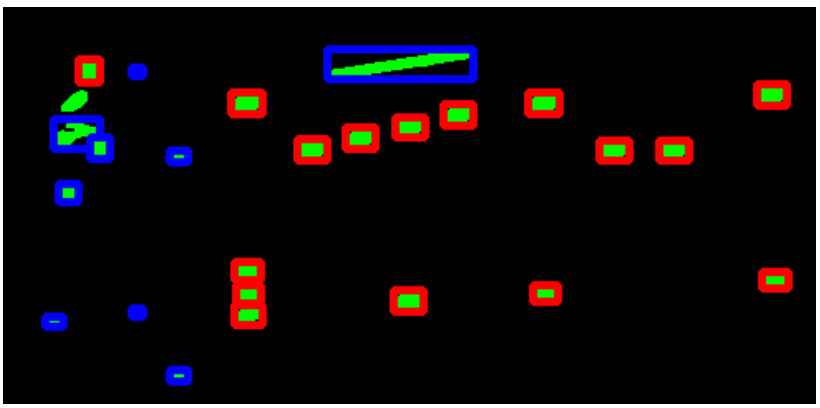


Figura 46 Individuazione di teste di note piene e vuote

Al termine di questo procedimento, conosciamo sia le coordinate delle teste di nota piene che le teste di nota totali, di conseguenza siamo in grado di determinare le coordinate delle teste di nota vuote. In un passaggio

successivo si dovrà capire se le testo vuote sono minime o semibrevi, e se le teste piene sono semiminime, crome o semicrome.

4.3.2 Identificazione dell'altezza della nota

Le operazioni compiute fino ad ora hanno sempre considerato l'immagine nella sua interezza; le fasi successive, che ci condurranno all'identificazione dell'altezza e del valore della nota, verranno eseguite considereranno un pentagramma per volta; Andremo perciò a considerare di volta in volta un pentagramma con tutti i simboli che abbiamo rintracciato, presenti nel suo intorno.

Il passaggio di identificazione delle note, in realtà trova la sua collocazione in un momento successivo al rilevamento della chiave; chiave, che come sappiamo, ha la funzione di fissare l'altezza relativa alle note; andiamo tuttavia a vedere il criterio utilizzato per l'implementazione di questo procedimento, considerando che per il pentagramma corrente è stata individuata una chiave di violino.

Arrivati a questo punto dell'algoritmo, disponiamo delle coordinate di tutte le teste di note, vuote e piene; queste informazioni unite alla conoscenza delle coordinate e dimensioni dei pentagrammi, ci permettono di calcolare l'altezza di ciascuna nota.

Come prima cosa, creiamo un array di stringhe che contiene tutte le note che possiamo rintracciare in un pentagramma; per descrivere l'altezza di una nota utilizziamo la notazione anglosassone; in notazione anglosassone le note hanno il nome delle lettere dell'alfabeto: il la si scrive A, il si, si scrive B, e via di seguito. Ciascuna nota è accompagnata da un numero che ne indica l'ottava; ad esempio il do centrale si scrive C4, essendo, per gli anglosassoni, la quarta ottava, quella centrale.

In posizione zero di questo array di note avremo la nota più grave, ovvero F3 (il fa della terza ottava) e poi via via tutte le altre.

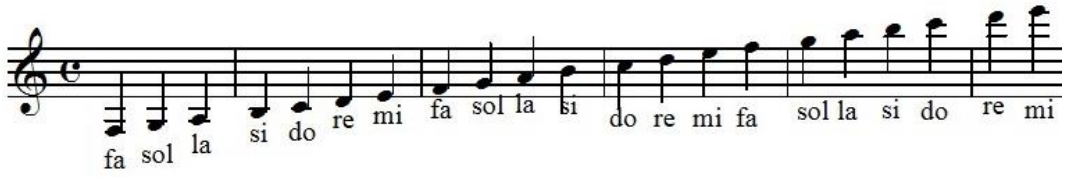


Figura 47 Note di un pentagramma in chiave di violino

Per trovare l'altezza di una nota necessitiamo di sapere il valore della sua ordinata ed il valore delle ordinate dei righi che compongono il pentagramma a cui la nota si riferisce; in realtà, conoscendo il valore della dimensione di uno spazio, che chiameremo *spazio*, necessitiamo di conoscere l'ordinata solo del suo ultimo rigo, quello più in basso, che chiameremo *y0*; il fa basso, che corrisponde alla prima nota a destra in figura 47, avrà valore di ascissa pari a *y_min = y0-3*spazio*. Il fa basso, preso in considerazione ora, è la nota più bassa che intendiamo identificare; sarà il nostro valore zero. Dato che la posizione di note consecutive si alterna fra righi e spazi, andremo a stabilire la distanza in termine di ordinata fra una nota e la sua successiva; denominiamo tale distanza col nome di *step= spazio/2*.

Nel momento in cui dobbiamo identificare l'altezza di una nota, ne andiamo a normalizzare il valore dell'ordinata; abbiamo detto che *y_min* è il nostro zero; chiamando l'ordinata della nota *y_nota*, il suo valore normalizzato sarà *y_norm = y_nota – y_min*.

Possiamo sintetizzare il processo di identificazione della nota attraverso il seguente frammento di codice:

```
array_note=[F3 G3 A3 C4 D4 E4 F4 G4 A4 B4 C5 D5 E5 F5 G5 A5 B5 C6 D6 E6]

trovato=False

i=0

while trovato==False and i < len(array_note):
```

```
{
 if (y_norm > (i-1/2)* step  and  y_norm ≥ (i+1/2)* step):
  nome_nota=array_note[i]
  trovato=True
 i++
 }
```

In poche parole andiamo a verificare uno *step* alla volta, lungo tutto l'intervallo, partendo da *y_min* a salire, dove la nota si colloca.

Dopo avere analizzato ogni nota di ogni rigo, disporremo di una struttura dati che per ogni pentagramma ha associato un array di elementi nota, che contengono le coordinate e il nome della nota.

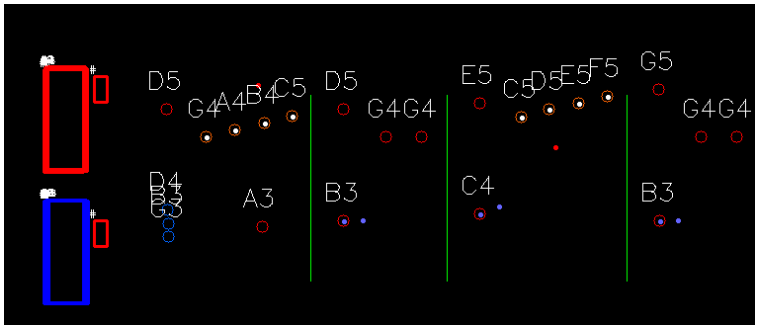


Figura 48 Un'immagine di controllo intermedia dell'algoritmo sviluppato, che mostra il nome delle note

4.3.3 Rilevamento gambi

Il rilevamento e l'analisi dei gambi è necessaria per distinguere una minima da una semibreve; infatti, mentre quest'ultima è raffigurala come un'ellisse, la minima è composta da un'ellisse affiancata da un gambo.

Figura 49 Semibreve e minima

Qualsiasi testa di nota piena è affiancata da gambo; perciò non è grazie alla presenza di questo che capiamo, in quest'altro caso, il valore della nota.

Possiamo però sfruttare l'informazione delle coordinate dei gambi per individuare eventuali rilevamenti fallaci; se una qualsiasi testa di nota piena non sarà affiancata da un gambo sapremo che, o tale nota è stata erroneamente rilevata, oppure si è verificato il mancato rilevamento di un gambo.

Il rilevamento dei gambi può essere effettuato in maniera analoga al sistema di rilevamento del pentagramma, descritto nel paragrafo 4.2.2.1; segmentiamo l'immagine tramite *l'opening* con un *kernel* di larghezza unitaria e altezza equivalente al doppio della dimensione di uno *spazio,* così da essere sicuri di eliminare tutte le teste nonché tutti i simboli di altezza inferiore a due volte la dimensione di uno *spazio*; otteniamo così l'immagine contenente solo le linee verticali; procediamo col rilevamento dei contorni e alla stima delle misure e della posizione tramite l'esecuzione dell'algoritmo di Canny e l'acquisizione dei parametri del *boundingBox*.

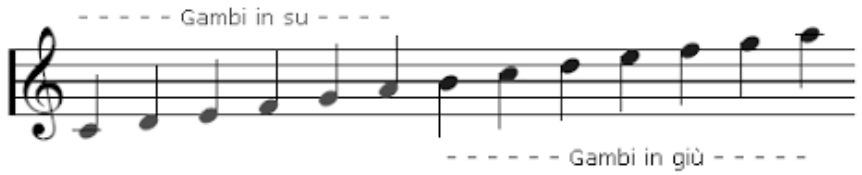


Figura 50 Gambi

Se una testa vuota di nota ha un gambo che si trova in sua prossimità, posizionato in alto a destra o in basso a sinistra rispetto ad essa, allora essa avrà valore di minima; se nessun gambo rispetta tali condizioni la nota è una semibreve; inoltre se viene rilevato che una testa vuota di nota ha gambo posizionato in alto a sinistra rispetto ad essa, tale nota verrà scartata, in quando si presume di essere di fronte ad un bemolle (♭).

4.3.4 Rilevamento di gruppi di crome e semicrome

Abbiamo determinato a quale tipo di nota appartenessero le teste di nota vuote rilevate; ora rivolgiamo la nostra attenzione alla determinazione del

tipo di nota a cui appartengono le teste di nota piene rilevate; in questo programma esse possono essere di tre tipi: semiminime, crome, semicrome.

Le componenti di una semiminima sono state già rilevate a questo punto del programma, ovvero, teste e gambi. Quindi se nel corso di questa fase non risultano particolari rilevamenti, relativi ad una testa di nota piena, essa sarà considerata essere una semiminima. Il nostro compito sarà quello di determinare quale delle le teste di nota piene sia croma o semicroma.

Iniziamo da crome e semicrome che si presentano in gruppo; esse avranno i propri gambi collegati da una 'trave', singola se si tratta di crome, doppia se si tratta di semicrome.

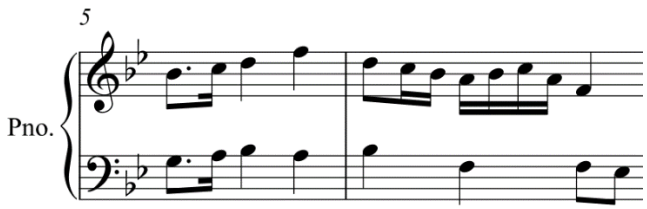


Figura 51 Cellule ritmiche composte da crome e semicrome

Per prima cosa procediamo all'individuazione delle travi.

Il primo passaggio di segmentazione prevede la rimozione del pentagramma, che può essere effettuata attraverso il metodo 2 (quello che prevede l'uso di una maschera) spiegato nel paragrafo 4.2.2.2; non usiamo il metodo 1 per non correre il rischio che operando l'*opening*, scompaiano anche le travi, insieme ai righi di pentagramma; anche se in genere le travi hanno spessore maggiore di diversi fattori rispetto alle linee di pentagramma.

Per procedere al loro rilevamento le isoliamo rimuovendo tutte le linee verticali nell'immagine, con un procedimento analogo al metodo 1 di rimozione dei pentagrammi, ovvero attraverso l'uso dell'operazione di *opening*; in questo modo non creeremo fratture interne alle travi. Quello che otteniamo è mostrato in figura:

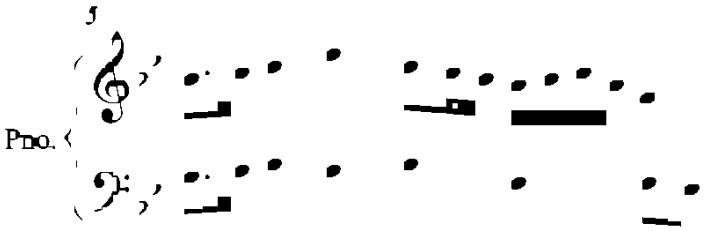


Figura 52 Immagine segmentata per rilevamento Travi

Come al solito si procede con la determinazione dei contorni attraverso l'algoritmo di Canny.

Le travi posso avere una inclinazione che varia a seconda delle note a cui si riferiscono; questa volta non è sufficiente approssimare le misure attraverso il *boundingBox*, in quanto le travi, come detto, possono essere inclinate; per una maggiore precisione sfrutteremo l'utilizzo di un'altra funzione offerta da *opencv* chiamata *minAreaRect*. Grazie a tale funzione siamo in grado di estrarre le dimensioni del rettangolo di minor area che contiene un contorno; il rettangolo che otteniamo, a seconda del contorno che raccoglie, avrà una particolare inclinazione.

Grazie a tali dati possiamo distinguere i due casi:

- se la barra rilevata ha spessore di circa metà *spazio* allora, probabilmente avremo trovato un gruppo di crome;
- se la barra ha dimensione di circa uno spazio, avremo trovato probabilmente un gruppo di semicrome.

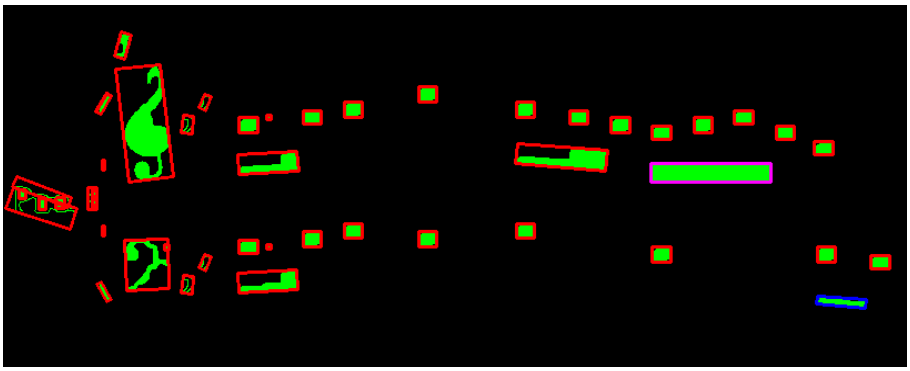


Figura 53 In rosso tutti i contorni rilevati; in viola le travi doppie di semicroma; in blu le travi di crome, in verde il ripieno dei contorni

Si è scritto ‘probabilmente’ poiché bisognerà verificare l’effettiva corrispondenza fra travi e note; questo può essere effettuato tramite un confronto di coordinate.

C’è un’altra importante considerazione da fare; come si può osservare in figura 53, esistono figurazioni miste, in cui sono presenti insiemi di crome e semicrome; per distinguere questi casi ci serviamo del valore dell’area di un particolare contorno rilevato che stiamo considerando, ottenibile grazie alla funzione *contourArea*, e del valore dell’area del rettangolo ruotato che circoscrive il contorno in questione.

Se l’area del contorno e l’area del rettangolo ruotato hanno un valore simile, allora consideriamo valida la trave rilevata; essa quindi è una barra che si riferisce ad un gruppo di crome o semicrome; nel caso le travi che esprimono figurazioni di crome o semicrome omogenee, esse riempiono il rettangolo che le circonda quasi completamente; questo non accade per le configurazioni miste, le quali hanno una zona in cui c’è la doppia barra ed una zona dove la barra è singola; in questo caso la barra in esame verrà scartata ed il valore delle note presenti nella figurazione mista verrà determinato attraverso il procedimento di cui parleremo nel prossimo paragrafo.

4.3.5 Rilevamento di figurazioni ritmiche miste

Per il rilevamento del tipo di nota, per teste di note piene componenti di una figurazione ritmica mista, faremo affidamento sul rilevatore di angoli di Harris. Partiremo anche questa volta da un’immagine in cui i pentagrammi sono stati rimossi con metodo 2, per la motivazione già spiegata nel paragrafo precedente.

Successivamente andiamo ad eliminare le linee verticale in una maniera analoga al metodo 2 per la rimozione del pentagramma, ovvero attraverso l’uso di una maschera, spiegato nel paragrafo 4.2.2.2; quello che otteniamo è mostrato in figura.

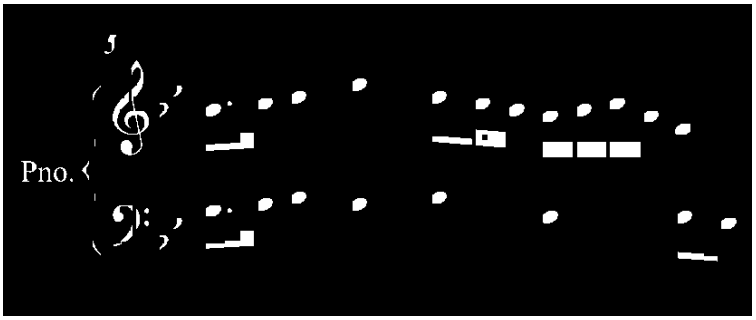


Figura 54 Immagine segmentat per il rilevamento degli angoli

Per questo particolare caso si dimostreranno utili le fratture venutesi a creare nelle travi in corrispondenza delle teste di nota piene; queste interruzioni infatti hanno dato vita alla presenza di angoli; questi angoli verranno rilevati dall'algoritmo di Harris come mostrato in figura:

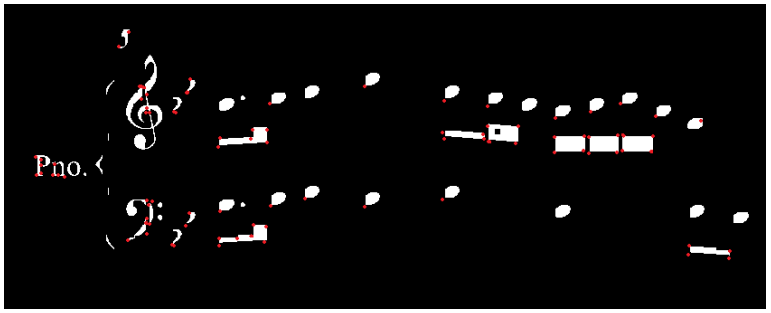


Figura 55 Risultato dell'algoritmo di Harris

Per tutte le note che non sono state ancora identificate, né come crome né come semicrome, verifichiamo la presenza o meno di angoli in loro corrispondenza; laddove è stata la rilevata la presenza di un angolo, diremo che la nota corrispondente è una croma; se si rileva la presenza di due angoli con stessa ascissa ad una distanza, l'uno dall'altro, di circa metà spazio, per quanto riguarda le ordinate, la nota è una semicroma.

Rimangono da individuare quelle crome e semicrome che trovandosi isolate assumono la nota configurazione con la codina arricciata o *flag* (singola per le crome e doppia per le semicrome). Avendo le code delle crome e semicrome una forma che è difficile da approssimare con una forma geometrica elementare, si è preferito demandare questo tipo di rilevamento

a processi che prevedono l'uso del *template matching*; l'argomento verrà perciò ripreso nei paragrafi successivi.

4.3.6 Rilevamento punti di valore

Siamo arrivati ad un punto in cui per ogni testa trovata conosciamo il tipo di nota a cui appartiene. L'ultimo passaggio per poter definire il valore di durata di ogni singola nota è il rilevamento di eventuali punti di valore. Un punto di valore indica l'aumento del valore della durata di una nota, di metà del suo valore; se una nota ha durata *d*, la stessa nota con un punto di valore avrà durata *d+d/2*.

Il procedimento che ci porta al rilevamento dei punti è molto semplice, in quanto riprende procedure già spiegate in precedenza.
Andiamo per prima cosa a rimuovere il pentagramma con il metodo 2, andando quindi a sottrarre la maschera dei pentagrammi all'immagine completa; non si può usare il metodo 1 in quanto l'*opening* potrebbe rimuovere anche i punti insieme al pentagramma, poiché il diametro di un punto ha una misura paragonabile allo spessore di un rigo di pentagramma.
Successivamente possiamo passare immediatamente alla rilevazione dei contorni e all'analisi delle dimensioni dei *boundingBox*. Non compiamo ulteriori segmentazioni in quanto non vogliamo creare frammenti che potrebbero essere confuse per punti; andando a prendere i contorni per i quali i *boundingBox* assumono dimensioni di altezza e larghezza di circa lo spessore di un rigo, otteniamo i seguenti rilevamenti:

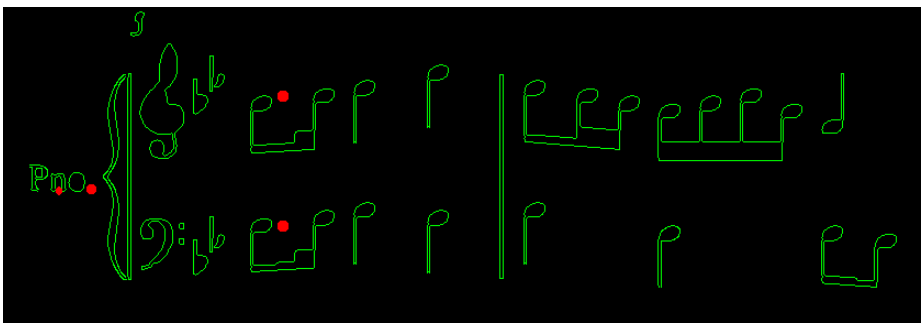


Figura 56 Rilevamento punti di valore

Andando a confrontare le coordinate di questi punti con quelle delle note trovate, possiamo semplicemente identificare tutte le note alla cui destra è posizionato un punto. Ovviamente i punti che non sono prossimi ad alcuna nota vengono scartati.

Dopo quest'ultimo passaggio, disponiamo di tutti gli elementi per calcolare il valore di durata di ciascuna nota; questi valori vanno a collocarsi all'interno di una struttura dati che è composto da un array di elementi nota per ciascun rigo; ogni elemento nota contiene un campo per le coordinate, uno per il nome (che ne determina l'altezza), uno per la durata.

4.3.7 Rilevamento alterazioni temporanee

Le alterazioni di cui andremo alla ricerca sono il diesis, il bemolle e il bequadro; il ruolo di queste alterazioni in musica è quello di alterare l'altezza della nota a cui si riferiscono:

- il diesis aumenta l'altezza di una nota di un semitono;
- il bemolle diminuisce l'altezza di una nota di un semitono;
- il bequadro si usa per indicare che una nota va suonata con la sua altezza naturale, senza alterazioni; questo è necessario se la tonalità del brano prevede alterazioni permanenti, o se una nota che poco prima è stata alterata, deve tornare nel suo stato normale.

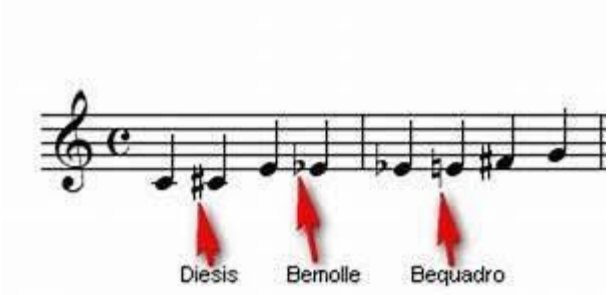


Figura 57 Alterazioni temporanee

Come possiamo osservare da figura 57, le alterazioni non hanno una forma che si presta facilmente alla scomposizione in primitive più semplici, come invece è accaduto per le note.

Il criterio attraverso il quale si è scelto di procedere, avrà in comune con i procedimenti visti fino ad ora solo le tecniche di segmentazione; l'algoritmo principale che ci condurrà al rilevamento di questi caratteri è il *template matching*. Come esplicitato nel paragrafo 3.18, l'algoritmo di *template matching* sfrutta l'operazione di correlazione per rintracciare i punti di maggiore somiglianza, fra il *template*, ovvero l'oggetto da rintracciare, e la porzione di immagine considerata; la funzione di correlazione possiede valori maggiori laddove il *template* e la porzione dell'immagine considerata, mostrano maggiori corrispondenze; il problema si traduce perciò in una ricerca di massimi locali all'interno della funzione di correlazione. Dato che non tutti gli stili grafici di musica utilizzano esattamente la stessa dimensione dei caratteri, questa operazione verrà effettuata per diverse scale del *template*.

La prima cosa che intendiamo fare è ridurre i luoghi di ricerca; vogliamo andare ad individuare i luoghi in cui un'alterazione ha più probabilità di giacere; le alterazioni hanno tutte un'altezza di circa tre volte di uno *spazio* e un'altezza all'incirca doppia rispetto ad uno *spazio*; andiamo ad eseguire alcune trasformazioni sull'immagine in maniera che sia più semplice individuare le forme di tali dimensioni. Si procede in primis con la rimozione del pentagramma attraverso il metodo 2, cosicché le alterazioni non vengano degradate eccessivamente.

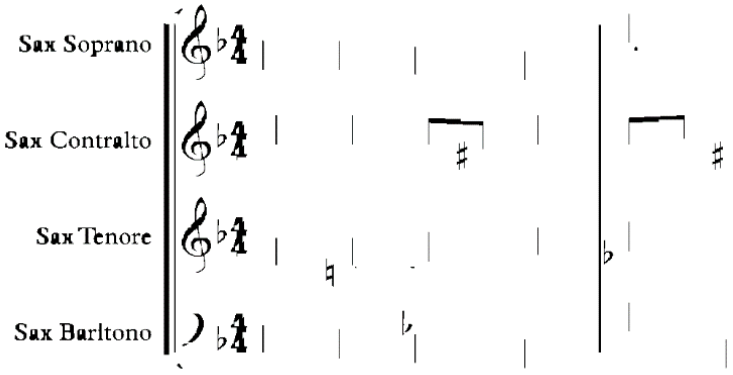


Figura 58 Immagine senza righi e senza note

Sfruttando poi le immagini che ci sono servite per l'individuazione delle teste delle note, come quella in figura 38, creiamo una maschera da sottrarre, ottenendo l'immagine in Figura 58.

Si vogliono eliminare anche le linee verticali, senza però andare ad eliminare i piccoli segmenti che compongono le alterazioni; attraverso un procedimento che sfrutta le tecniche che abbiamo già ampiamente discusso in precedenza, siamo in grado, a partire da una maschera delle linee verticali, di individuare i segmenti più piccoli, e di creare una maschera che non li contenga. Sottraendo tale maschera otteniamo:

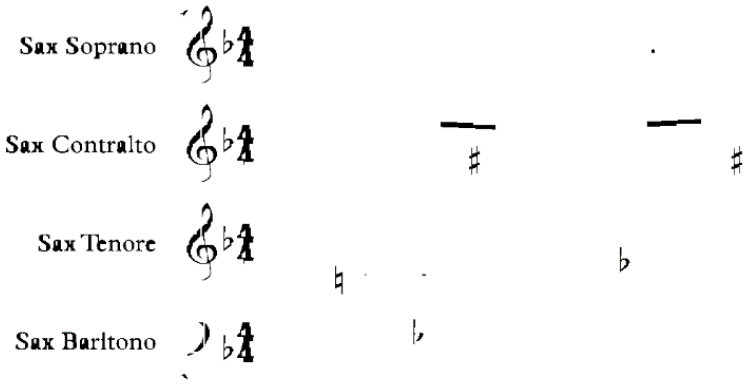


Figura 59 Immagine segmentata per il rilevamento delle alterazioni

A questo punto tramite rilevamento dei contorni e analisi delle misure di *boundingBox*, individuiamo i luoghi in cui probabilmente giaceranno le alterazioni temporanee.

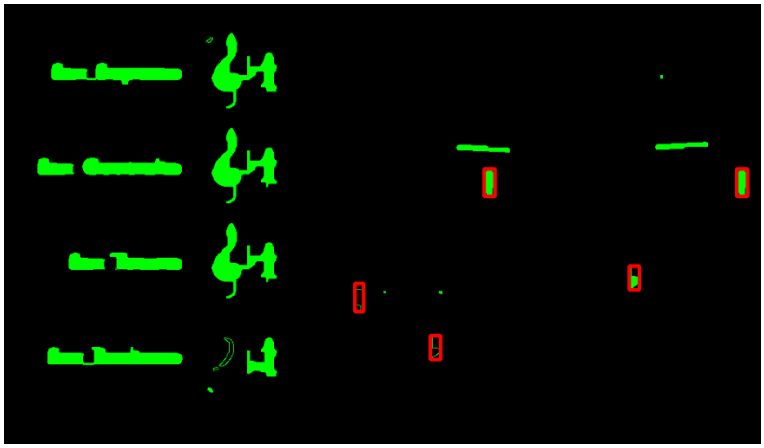


Figura 60 Rilevamento luoghi in cui potrebbero essere delle alterazione

Nelle zone rilevate verrà eseguito il *multi-scale template matching* fra l'immagine del brano priva di pentagrammi (rimossi con metodo 2) e l'alterazione da rintracciare; in figura 60 il risultato dei rilevamenti.

Al termine di questa operazione andiamo a rintracciare, tramite confronto di coordinate, le note alla cui sinistra giace una alterazione; ne verrà modificato appropriatamente il nome che ne determina l'altezza.

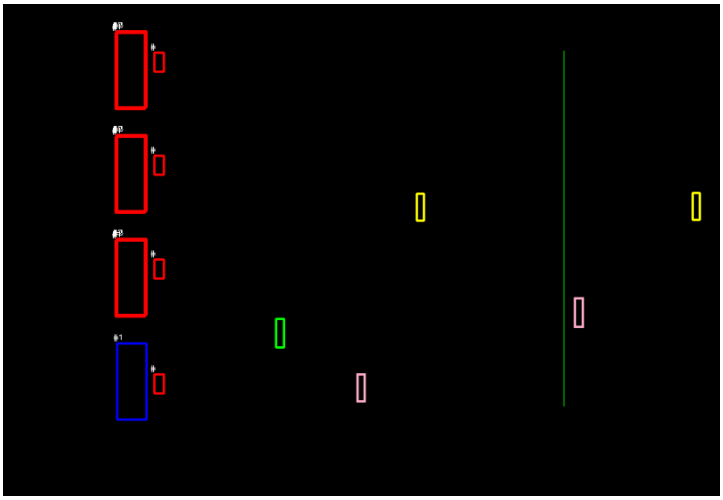


Figura 61 Rilevamento alterazioni: in verde i bequadro, in giallo i diesis, in rosa i bemolle

Le alterazioni che fanno parte dell'armatura armonica, non vengono rilevate nel corso di questo procedimento. Il rilevamento delle alterazioni dell'armatura, come anche il rilevamento delle chiavi in realtà è un operazione che cronologicamente si colloca prima del rilevamento delle teste di nota; il rilevamento dell'armatura e quello delle chiavi ci aiutano infatti a decidere quale sia la zona in cui ammettiamo che una nota giaccia; non è infatti possibile trovare in alcuna partitura una nota posizionata prima dell'armatura di chiave.

4.3.8 Rilevamento chiave ed armatura

I risultati della procedura che andremo a spiegare, si dimostrano assai utili per restringere il campo di ricerca, in quanto ci aiutano a determinare la zona di validità dei simboli ricercati; per questo motivo sarebbe bene collocare tale operazione subito dopo l'identificazione del pentagramma,

prima di intraprendere il processo di identificazione delle teste di note. Un altro motivo, per il quale questo processo sia da collocare prima del rilevamento delle teste di nota, consiste nel fatto che è solo in base al tipo di chiave presente ad inizio pentagramma, che la particolare posizione di una nota su di esso assume un certo valore di altezza sonora.

Si è scelto di collocare la spiegazione di questo procedimento solo ora, poiché si è voluto affrontare in primis il rilevamento di simboli scomponibili in figure geometriche elementari, che non necessitassero del *template matching* per il loro rilevamento.

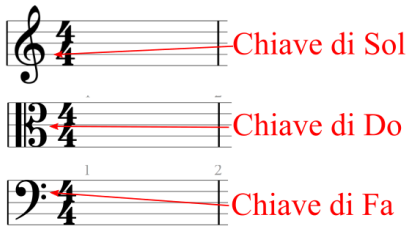


Figura 62 Chiavi musicali

Il momento in cui realmente si colloca questa procedura abbiamo detto essere dopo il rilevamento del pentagramma; conosciamo perciò coordinate e dimensioni dei pentagrammi, nonché la dimensione di uno *spazio.* Chiamando l'ascissa di inizio rigo *x0,* possiamo dire che la ricerca delle chiavi potrà essere effettua in una zona dell'immagine che ha come ascissa un valore compreso fra *x0* e *x0 + 5*spazio.* Si è deciso inoltre di effettuare la ricerca del *template* sull'immagine intera priva di trasformazioni; infatti la chiave di violino verrà a presentarsi, in qualsiasi partitura, sempre nella stessa posizione fra i righi di un pentagramma; in questo caso perciò i righi non risultano essere di alcun disturbo, anzi possono essere considerati una caratteristica dell'oggetto da rilevare. Al termine di questa operazioni disporremo di un array di chiavi, di dimensione pari al numero dei pentagrammi della partitura analizzata.

Considerazioni analoghe possono essere fatte per l'armatura di chiave, ovvero il gruppo di alterazioni permanenti collocate subito dopo la chiave. La presenza di queste alterazioni è da ricercarsi in una zona che va dall'ascissa in cui è stata trovata la chiave per quel rigo, in poi.

Il processo di rilevamento delle alterazioni dell'armatura, può essere eseguito in maniera simile al rilevamento delle alterazioni temporanee, spiegato nel capitolo precedente.

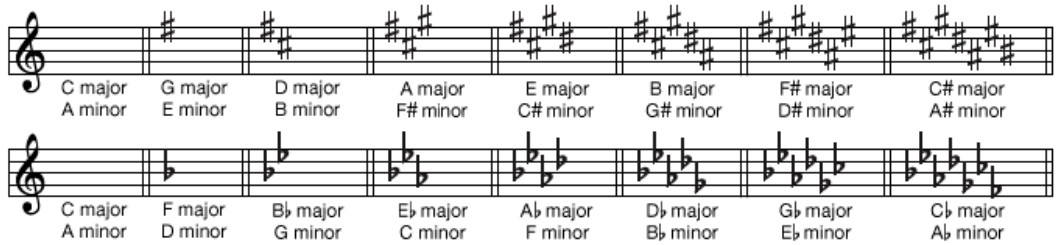


Figura 63 Armature di chiave

Come possiamo osservare da figura 63, esiste un numero limitato di configurazioni di armature di chiave; ciascuna alterazione deve essere dello stesso tipo delle altre e la posizione di ogni nuova alterazione trovata, deve essere a circa uno *spazio* dalla precedente.

Alcuni strumenti, detti traspositori, producono suoni reali che differiscono da quelli scritti sul pentagramma; i pentagrammi che si riferiscono agli strumenti traspositori presenteranno un'armatura di chiave diversa dalle altre; per ogni pentagramma della partitura, sarà necessario memorizzare il tipo e il numero di alterazioni dell'armatura di chiave ad esso riferita.

4.3.9 Rilevamento pause e di code di croma e semicroma

Le pause come le alterazioni hanno forme che difficilmente sono scomponibili in elementi geometrici semplici, fatta eccezione per le pause di minima e semibreve, come mostrato in figura 64.

Per le pause di semibreve e minima si è scelto di eseguire una procedura simile a quella per il rilevamento dei punti di valore, ovvero un algoritmo basato sull'analisi delle dimensioni del *boundingBox* che ne circoscrive i

contorni; per il rilevamento delle restanti pause, si è proceduto invece mediante un algoritmo che sfrutta la tecnica del *template matching,* il cui svolgimento va a ripercorrere pressappoco i medesimi passaggi compiuti per il rilevamento delle chiavi e delle alterazioni.

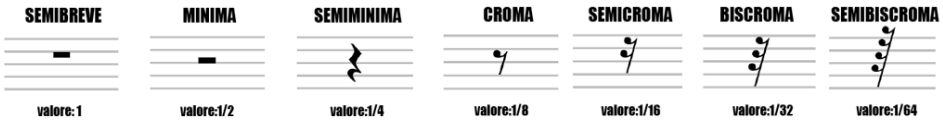


Figura 64 Pause

Lo stesso discorso vale anche per le *flag* delle crome e delle semicrome.

Figura 65 Croma e semicroma

Le ben note codine, flag in inglese, che contraddistinguono le crome e le semicrome non appartenenti a gruppi ritmici collegati da travi, sono di difficile descrizione geometrica; per questo motivo, anche in questo caso si è scelta la strada del rilevamento tramite *template matching*. Le coordinate dei *flag* rilevati, sono confrontate con quelle delle teste di nota piene, la cui tipologia non sia stata già identificata dai metodi precedenti; qualora si venissero a presentare le corrette condizioni di posizione, alla nota verrebbe assegnato l'adeguato valore di durata di un ottavo o di un sedicesimo.

4.3.10 Rilevamento e analisi delle linee di misura

Questa procedura è di fondamentale importanza per il riordinamento delle informazioni estratte fino ad ora; le informazioni sono in questo momento organizzate per pentagrammi, mentre la libreria *music21* richiede una composizione della partitura per battute.

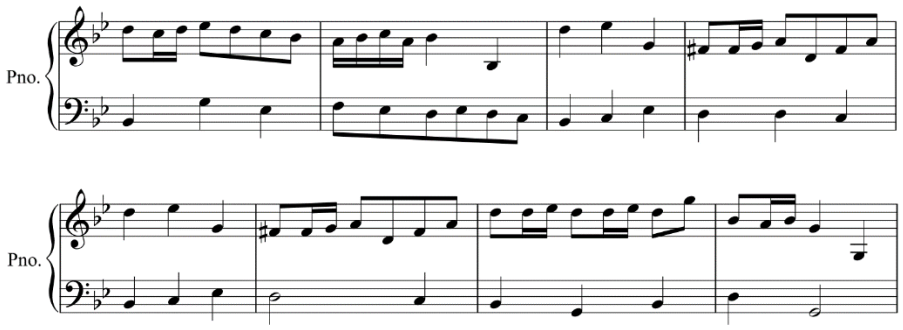


Figura 66 Accollatura composta da 2 parti

Il primo passaggio per il rilevamento delle battute è la segmentazione: operando sull'immagine completa un *opening* con un *kernel* verticale di larghezza unitaria ed altezza pari a quattro volte la misura di uno spazio, otteniamo un'immagine contenente tutti gli elementi verticali maggiori di *4*spazio*.

Di quest'immagine andiamo poi a considerare i contorni che hanno *boundingBox* con larghezza minore di uno *spazio* ed altezza maggiore o uguale di quattro volte la dimensione di uno *spazio.*

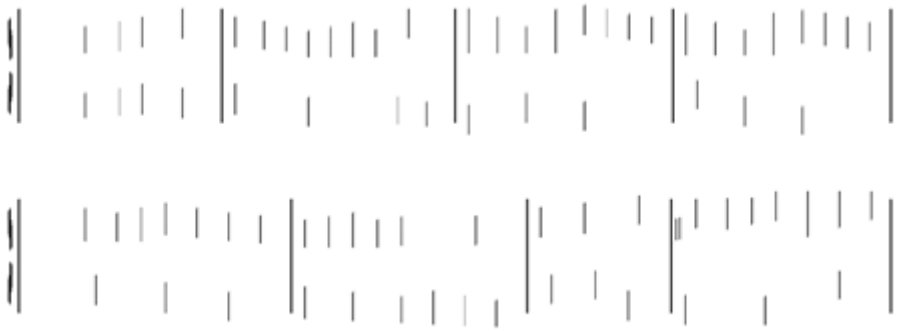


Figura 67 Immagine contenente linee verticali

Possediamo così dimensioni e centro di massa di tutte linee verticali candidate ad essere linee di battuta.

Prima di procedere col rilevamento delle linee di battuta, andiamo ad analizzare le linee verticali che precedono ciascuna chiave (linee di simultaneità); congiungono i pentagrammi appartenenti alla stessa accollatura, e trovandosi in una zona proibita per le note, non possono

essere confuse con gambo di nota. Confrontando le coordinate e le dimensioni delle linee di simultaneità con quelle dei pentagrammi siamo in grado di determinare a quale accollatura ciascun pentagramma appartiene.

Torniamo ora al rilevamento delle line di battuta; tutte le linee che distano per ascissa di una misura inferiore ad uno spazio, da una qualsiasi nota, vengono scartate in quanto si presuppone di avere a che fare con un gambo di una nota e non con una linea di battuta; le linee rimanenti, si presuppone siano linee di battuta.

Dato che alcune note potrebbero non essere state rilevate, non possiamo fare totale affidamento sulla relazione fra linee verticali e note. Per ogni pentagramma vengono confrontate le linee verticali ad esso appartenenti, con quelle appartenenti al pentagramma precedente; se le linee si trovano in corrispondenza, ovvero sulla stessa ascissa, vuol dire che abbiamo trovato una linea di battuta e che i pentagrammi considerati, appartengono alla stessa accollatura

Da questo procedimento apprendiamo con discreta certezza, non solo coordinate e dimensione delle linee di battuta, ma anche da quanti pentagrammi è composta un'accollatura.

4.3.11 Struttura delle informazioni acquisite

Tutti i dati raccolti fino ad ora devono essere utilizzati al fine di ricreare una partitura, in formato xml, il più possibile simile a quella di partenza. In, precedenza è già stato accennato che l'operazione di ricostruzione, mattone per mattone, della partitura viene eseguita sfruttando la libreria *music21*; andiamo a vedere come si compone un brano costruito implementando un codice che sfrutta le funzioni offerte da *music21*; partendo dagli elementi più piccoli, a crescere:

- note e pause;
- *measure*: rappresenta una singola battuta, per una singola parte;

- *part*: rappresenta una parte, ovvero l'insieme di battute riferite un singolo strumento (o nel caso del piano forte ad una singola mano)
- *stream*: rappresenta sostanzialmente l'intera partitura, come insieme delle parti.

Si intende perciò strutturare le informazioni raccolte a seconda delle battute e a seconda della parte a cui si riferiscono.

I procedimenti passati ci hanno garantito la conoscenza delle coordinate di ogni singolo elemento, incluse quelle delle linee di battuta; siamo a conoscenza anche del numero di pentagrammi per accollatura, quindi sappiamo da quante parti distinte è composta la partitura in esame.

Per ottenere una struttura dati che ci consenta di inserire i dati raccolti in maniera appropriata, andiamo ad implementare i seguenti passaggi:

- ordinare rigo per rigo gli elementi trovati secondo le ascisse;
- per ogni rigo, suddividere i suoi elementi, a seconda della battuta a cui appartengono;
- conoscendo il numero di pentagrammi per accollatura siamo in grado di dedurre quali pentagrammi si riferiscono ad una particolare parte.
- la struttura finale che conterrà tutte le informazioni acquisite, è rappresentato da una lista di parti, in cui ogni parte è una lista di misure.

La struttura dati così composta, costituisce il vettore di input per la fase di costruzione della partitura.

4.4 Costruzione della partitura

Per la creazione del file xml vero e proprio abbiamo fatto uso della libreria *music21.* In questa fase oltre al semplice inserimento dei dati raccolti, verranno applicate alcune regole della grammatica musicale che non è stato

possibile implementare in precedenza, data la necessità di una visione globale degli elementi.

Come già accennato in precedenza, la struttura di un brano per questa particolare libreria consiste in una struttura ad albero, il cui nodo radice è denominato *stream;* esso è composto dagli oggetti *part,* i quali rappresentano la parte di uno degli elementi facenti parte dell'organico; ogni oggetto *part* è composto da un insieme di misure; si presuppone che tutte le parti abbiano lo stesso numero di misure; ogni misura contiene note e pause.

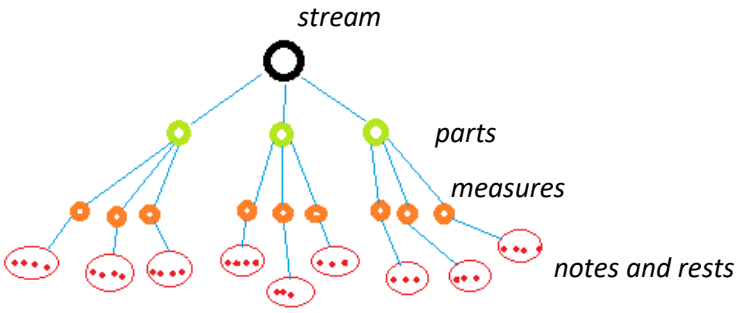


Figura 68 Struttura di un brano in music21

Grazie al vettore di informazioni appositamente ordinato nel passaggio precedente, sarà facile implementare una serie di cicli che ci consentano il corretto popolamento della partitura. Nel corso della procedura di inserimento verranno implementate alcune regole della teoria musicale.

Innanzitutto per la prima battuta di ciascuna delle parti, è previsto l'inserimento di:

- chiave;
- armatura di chiave;
- indicazione di tempo;

Per quanto riguarda la chiave non ci sono considerazioni da fare, dato che già in fase di assegnazione dell'altezza di una nota è stata tenuta in considerazione la chiave vigente nel rigo considerato.

L'armatura di chiave indica la presenza di alterazioni permanenti; per ogni nota che andremo ad inserire è necessario verificare se per quella particolare nota, è prevista un'alterazione; se quella nota è una di quelle corrispondenti all'armatura di chiave, allora si provvede alla modifica dell'altezza della nota.

Tutto questo avverrà a meno che la nota in questione non abbia giustapposto l'indicazione di bequadro; in quel caso qualsiasi segno di alterazione precedente non avrà validità per la nota corrente.

Altre considerazioni possono essere fatte riguardo alle alterazioni; in musica quando è presente una alterazione temporanea riferita ad una nota, questa è da considerarsi vigente per tutto il corso della battuta, per tutte le note della stessa altezza; in figura 69 i due *do* all'interno della battuta cerchiata in rosso, sono entrambe da considerarsi diesis; dato che *music21* non tiene conto di questa regola si è resa necessaria l'implementazione di un controllo.

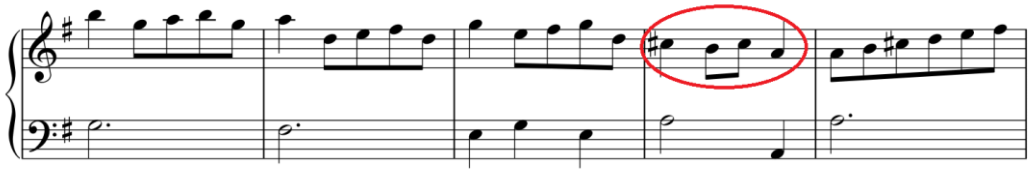


Figura 69 Intervallo di validità delle alterazioni temporanee

L'indicazione di tempo, espressa generalmente con una frazione, determina la quantità di valori che una singola battuta può contenere, oltre che l'accentuazione ritmica e metrica. Si è ritenuto implementare un controllo a tal proposito; nel corso dell'inserimento di note e pause all'interno di una battuta, viene tenuta traccia della somma dei loro valori in termini di durata; tenendo conto dell'indicazione di tempo, viene effettuato un controllo

affinché l’aggiunta di un nuovo elemento, non comporti il superamento del limite di durata per una battuta; si è scelto di implementare tale monitoraggio per evitare il propagamento di eventuali errori di rilevamento; senza tale ispezione potrebbe capitare che a seguito di un rilevamento sbagliato, le varie parti dei singoli strumenti, che compongono un brano, vadano a sfasarsi temporalmente.

A conclusione del processo di inserimento dei dati raccolti, avremo ricostruito l’intera partitura; l’ultimo passaggio sarà delegato alla funzione *write* della libreria *music21,* la quale offre l’opportunità di esportare tali informazioni nel formato desiderato, nel nostro caso in formato xml.

5 Analisi

Nel corso di questo capitolo andremo a valutare le prestazioni dell'algoritmo sviluppato. Al fine di misurare l'efficienza del sistema è stata effettuata una ricerca delle metriche in letteratura; in particolare si sono scelti i criteri di valutazione utilizzati in [5]. Possiamo suddividere le valutazioni che andremo a compiere in quattro tipologie:

- misura del tempo di esecuzione dell'algoritmo;
- tasso di corretto rilevamento dei singoli simboli considerandoli a se stanti;
- tasso di corretta valutazione delle relazioni che legano i simboli;
- infine valuteremo il guadagno in termini di tempo ottenuto grazie all'uso del programma rispetto ad una acquisizione dei dati completamente manuale.

Per eseguire la fase di valutazione sono stati scelti due brani presi dal Quaderno di Anna Magdalena di Bach e la prima pagina di opera 499 e 464 di Mozart.

5.1 Capacità di riconoscimento dei singoli elementi

Una parziale valutazione delle prestazioni può essere effettuata andando ad analizzare la capacità di riconoscimento dei singoli elementi della partitura, considerando ciascuno di essi individualmente.

Si procederà con il conteggio, per ogni categoria di simbolo, delle occorrenze rilevate e di quelle attese. Andiamo ad elencare le diverse categorie: *Testa di nota vuota, Testa di nota piena, Gambi, Pausa di semibreve, Pausa di minima, Pausa di semiminima, Pausa di croma, Pausa di semicroma, Trave singola Trave doppia, Angoli di figurazioni ritmiche miste, Diesis, Bemolle, Bequadro, Chiave di violino, Chiave di basso,*

Chiave di contralto, Uncino singolo (durata 1/8), Uncino doppio (durata 1/16).

L'insieme dei simboli che andremo a ricercare non è completo; qualora la partitura analizzata presentasse elementi che non rientrano in questo insieme, il processo dovrà essere in grado di escluderli; se questo non avvenisse l'elemento erroneamente rilevato verrà considerato come errore.

Per facilitare la formulazione delle valutazioni, faremo uso di alcuni indicatori, i quali si riferiscono a ciascuna delle categorie di simboli che sono stati ricercati:

- con E_i indichiamo il numero effettivo di occorrenze, di una particolare categoria di simboli *i*, presenti in un brano;
- con *N* indichiamo il numero delle categorie dei simboli ricercati;
- con E_{tot} indichiamo la somma delle occorrenze conteggiate sul brano originale $E_{tot} = \sum_{i=1}^{N} E_i$;
- con T_i indichiamo il numero di simboli di categoria *i*, correttamente rilevati;
- con F_i indichiamo il numero dei simboli di categoria *i* erroneamente rilevati;
- con M_i indichiamo il numero di occorrenze del simbolo di categoria *i* che avrebbero dovuto esserci, ma non sono state rilevate;

Per effettuare la valutazione si procede con il conteggio delle occorrenze sul brano originale e poi sul brano ricostruito; tenendo conto delle grandezze sopra enunciate siamo in grado di calcolare un parametro di valutazione che chiameremo *tasso di successo percentuale*, che calcoleremo in questo modo:

$$T = \frac{\sum_{i=1}^{N} T_i}{\mathrm{E_{tot}}} * 100$$

Questo sistema di valutazione, può considerarsi solo in parte soddisfacente; esso presuppone implicitamente che il rilevamento di un simbolo abbia lo stesso peso a prescindere dalla categoria a cui appartiene; sappiamo che la realtà non è questa; ad esempio, il mancato rilevamento di un diesis, non comporterebbe alcun danno a livello di struttura del brano; oltretutto, la successiva correzione, a seguito di questo mancato rilevamento, richiederebbe solo pochi secondi; lo stesso non lo si può dire nel caso a non essere rilevato fosse un pentagramma; l'errore in questo secondo caso, andrebbe a condizionare il rilevamento di molti altri simboli; i vantaggi che trarremmo dall'uso del programma a quel punto sarebbero molto minori; sorge la necessità perciò di dare un peso che quantifichi l'importanza che il rilevamento di una particolare categoria possiede.

Definendo con p_i un peso per ogni categoria con valore compreso fra 0 e 10 abbiamo:

$$TP = \frac{\sum_{i=1}^{N} \mathrm{p_i} T_i}{\sum_{i=1}^{N'} \mathrm{p_i} E_i} * 100$$

Definiamo come tasso di confusione:

$$TC_R = \frac{\sum_{i=1}^{N} F_i + \mathrm{M_i}}{E_{\mathrm{tot}}} * 100$$

5.2 Valutazione della partitura ricostruita

Nel corso dell'analisi precedente sono stati raggruppati nello stesso insieme tutti gli elementi che sono stati oggetto di rilevamento, indipendentemente dal fatto che essi avessero rapporti di relazione con altri simboli o che avessero un significato coerente a se stante; riguardo al primo caso possiamo dire che il corretto rilevamento di un componente non sempre corrisponde ad una corretta valutazione del simbolo musicale nella sua interezza. Ad esempio, seppure rileviamo correttamente la testa e il gambo di una croma, il mancato rilevamento della sua *flag* ne renderà errato il

valore della durata; esso sarà di un quarto invece che un ottavo; perciò si dovrebbe considerare errato il rilevamento della nota nella sua interezza. Un altro caso potrebbe riguardare il rilevamento delle alterazioni; infatti il rilevamento corretto di un'alterazione risulterà vano, se non siamo in grado di identificare la nota a cui si riferisce. Per tali motivi, risulta necessario, nell'ambito della valutazione delle prestazioni dell'algoritmo, considerare la correttezza dei legami che vengono instaurati fra i simboli nel corso della ricostruzione.

Si vuole procedere perciò con una valutazione delle prestazioni, che riguardi la corretta interpretazione dei simboli e delle relazioni che li legano; questa valutazione estende in qualche modo la precedente, in quanto se una componente di un simbolo è stata rilevata male, il simbolo verrà interpretato in modo scorretto; inoltre potrebbe verificarsi che componenti individuate correttamente siamo messe in relazione in maniera scorretta; queste considerazioni ci annunciano che assisteremo ad un calo del valore di prestazione, rispetto al criterio utilizzato precedentemente.

Questa volta invece di conteggiare la correttezza o meno del rilevamento di un simbolo di una particolare categoria, faremo riferimento alla corretta *stima* di un simbolo o di una relazione che lega più simboli.

Andiamo a vedere in che modo è opportuno verificare la veridicità delle diverse tipologie di *stime* fatte a proposito di simboli e di relazioni, determinate nel corso del processo:

- **Nota:** per verificare la che la stima di una nota sia corretta dobbiamo considerare l'altezza e la durata che le sono state assegnate; se anche uno solo dei due valori risulta non essere corretto, la stima della nota è da reputarsi errata.
- **Pause:** per valutare che la stima di una pausa sia corretta è necessario verificarne il valore della durata;

- **Note con alterazione temporanea:** la stima è corretta se nota e alterazione sono appropriatamente associate;
- **Chiavi:** la stima è corretta se si verifica l'appropriata associazione fra chiave e pentagramma a cui si riferisce;
- **Armatura di chiave:** la stima è corretta se le alterazioni rilevate sono del giusto numero e della giusta tipologia;
- **Battute:** la stima è corretta se il numero di battute nel brano è giusto, se le battute per gli strumenti che devono suonare insieme sono incolonnate e se la durata del contenuto della battuta coincide che la durata della battuta stessa;
- **Pentagrammi:** la stima è corretta se si verifica la presenza del giusto numero di pentagrammi in totale e per accollatura;

Andiamo ad esplicitare i valori coinvolti in questa seconda valutazione:

- con E_i' indichiamo il numero di occorrenze della *stima i-esima*;
- con *N'* indichiamo il numero delle diverse tipologie di *stima* del significato dei simboli e delle relazioni fra simboli;
- con E'_{tot} indichiamo la somma delle occorrenze di tutte le diverse *stime* conteggiate sul brano originale $E'_{tot} = \sum_{i=1}^{N'} E_i'$;
- con n_t^i indichiamo il numero di occorrenze della stima *i* del significato dei simboli o delle relazioni fra simboli, corrette;
- con n_f^i indichiamo il numero di occorrenze della stima *i* del significato dei simboli o delle relazioni fra simboli, errate;
- con n_m^i indichiamo il numero di occorrenze della stima *i* del significato dei simboli o delle relazioni fra simboli, mancate;
- con n_a^i indichiamo il numero di occorrenze della stima *i* del significato dei simboli o delle relazioni fra simboli, che non erano

presenti nell'originale, ma sono erroneamente state rilevate nel corso del processo;

Come nel caso precedente, procediamo con la definizione dell'indice di successo percentuale:

$$T' = \frac{\sum_{i=1}^{N'} T_i'}{\mathrm{E}_{\mathrm{tot}}'} * 100$$

Anche questa volta si reputa necessario dare diverso peso a seconda della tipologia di stima considerata; indicando con p_i un coefficiente compreso tra zero e uno, otteniamo il tasso pesato:

$$TP' = \frac{\sum_{i=1}^{N'} \mathrm{p_i} T_i'}{\sum_{i=1}^{N'} \mathrm{p_i} E_i'} * 100$$

Definiamo come indice di errore di ricostruzione:

$$E_{Ric} = \frac{\sum_{i=1}^{M'} \mathrm{n}_f^i + \mathrm{n}_a^i + \mathrm{n}_m^i}{\sum_{i=1}^{N'} \mathrm{p_i} E_i'} * 100$$

5.3 Valutazione guadagno

Come già detto in precedenza lo scenario per cui è stato ideato questo programma, prevede una fase di revisione ed eventuale correzione della partitura riconosciuta; il programma non potrà mai considerarsi al cento per cento affidabile.

Il lavoro di verifica della correttezza, ed eventuale successiva correzione da effettuare attraverso programmi di editing musicale, può richiedere più o meno tempo, a seconda di quanto gli errori commessi dall'algoritmo siano gravi.

In questo paragrafo si intende valutare il vantaggio in termini di tempo, ottenuto dall'acquisizione dei dati in maniera automatica, considerando

anche il tempo necessario alla verifica e alle eventuali correzioni, rispetto all'acquisizione dei dati musicali interamente attraverso inserimento manuale.

Definiremo il guadagno come:

$$G = \frac{\mathrm{C_0}}{\mathrm{C_{OMR}}}$$

In cui $\mathrm{C_0}$ indica il tempo di acquisizione in maniera manuale, e $\mathrm{C_{OMR}}$ definisce il tempo di acquisizione automatica inteso come tempo di verifica e correzione.

Valuteremo l'algoritmo tanto più vantaggioso in termini di tempo, quanto più il valore del guadagno G sarà maggiore.

Per la valutazione di $\mathrm{C_{OMR}}$ dobbiamo tener conto dei tempi di verifica, per ogni simbolo; la verifica andrà effettuata anche se il brano è stato riconosciuto correttamente al cento per cento.

Andiamo a definire alcuni indicatori che ci aiuteranno a definire la formulazione del guadagno:

- Con C_i indichiamo il costo medio di editing, ovvero il tempo necessario per l'inserimento manuale di un simbolo;
- Con C_v indichiamo il tempo medio di verifica della correttezza o meno di un simbolo;
- con C_f indichiamo il costo medio per correggere un particolare errore (questo parametro include il tempo medio di verifica C_v)
- con C_m indichiamo il tempo necessario per aggiungere un simbolo o una relazione non rilevata (questo parametro include il tempo medio di verifica C_v);

- con C_a indichiamo il tempo necessario per rimuovere un simbolo o una relazione non presente sul brano originale (questo parametro include il tempo medio di verifica C_v);

Tenendo conto di tali grandezze possiamo dire che:

$$C_0 = \mathrm{N\ C_i}$$

$$C_{OMR} = \mathrm{C_f + C_a + C_m + C_t}$$

Considerando le formule precedenti otteniamo:

$$\frac{N\ C_i}{C_f n_f + C_a n_a + C_m n_m + C_t n_t} \geq \gamma$$

Tanto più γ sarà grande, tanto più l'utilizzo del programma risulterà vantaggioso.

5.4 Risultati

In questo paragrafo andiamo ad illustrare le valutazioni a proposito delle prestazioni del programma sviluppato; per farlo utilizzeremo i parametri di misura discussi nel paragrafo precedente. Con gli stessi criteri di valutazione si è voluto effettuare un test di comparazione su due fra i più usati software OMR esistenti:

- il primo è disponibile gratuitamente, ed è il software OMR di cui fa uso MuseScore; oltre alla sua funzione principale di editor di notazione musicale, MuseScore offre l'opportunità di acquisizione dati musicali da immagine;
- il secondo è PhotoScore, uno dei software OMR proprietari più celebri in commercio;

Come input sono state scelte quattro immagini:

- le prime due rappresentano brani tratti da 'Il mio primo Bach'; in particolare sono il minuetto in sol maggiore e la polacca in sol

minore; sono brani pianistici generalmente studiati da pianisti principianti;

- le altre due immagini prese come input rappresentano la prima pagina dell'opera 499 e dell'opera 464 di Mozart; questi due brani sono stati scritti per quartetto d'archi; la complessità in termini di eterogeneità e densità dei simboli è notevolmente maggiore rispetto ai primi due brani.

Come già accennato in precedenza lo scenario per cui questa applicazione è stata pensata, non richiede tempi di risposta particolarmente brevi; nel corso dello sviluppo dell'algoritmo si è deciso di non rivolgere particolare attenzione all'ottimizzazione del costo computazionale; il tempo che il processo impiega nell'analisi di una pagina dipende per lo più dal numero di pentagrammi presenti in essa. Di seguito sono mostrati i risultati ottenuti dall'utilizzo dei diversi software.

Tempo di esecuzione del processo

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	41 sec	28 sec	5 sec
Polacca di Bach	58 sec	33 sec	5 sec
Mozart op. 499	2 min e 3 sec	45 sec	10 sec
Mozart op. 464	1 min e 58 sec	42 sec	9 sec

PhotoScore mostra tempi di risposta eccellenti; l'algoritmo utilizzato da MuseScore richiede l'upload del pdf contenente l'immagine del brano da analizzare; questo potrebbe essere uno dei motivi per cui i tempi di risposta

siano notevolmente peggiori rispetto a Photoscore; i tempi di risposta dell'algoritmo da me sviluppato sono i più lunghi; d'altronde la brevità nei tempi di risposta non è il requisito al quale, nel corso dello sviluppo, è stata assegnata la maggiore priorità; i tempi di risposta ottenuti sono tuttavia comparabili ai risultati riportati dai due algoritmi con i quali si è effettuato il confronto.

Successivamente si è proceduto con la misurazione del tasso di successo percentuale; questo parametro viene calcolato sulla base delle corrette individuazioni dei simboli base, in rapporto con il numero totale dei simboli presenti effettivamente sulla partitura originale.

Tasso di successo percentuale

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	100%	99%	100%
Polacca di Bach	99%	98%	100%
Mozart op. 499	91%	87%	98%
Mozart op. 464	92%	90%	99%

Tasso di successo percentuale pesato

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	100%	99%	100%

Polacca di Bach	99%	99%	100%
Mozart op. 499	92%	82%	99%
Mozart op. 464	93%	90%	99%

Tasso di confusione

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	0%	1%	0%
Polacca di Bach	1%	1%	0%
Mozart op. 499	6%	9%	2%
Mozart op. 464	3%	5%	1%

Minuetto e polacca di Bach

Tutti e tre gli algoritmi dimostrano prestazioni eccellenti per i primi due brani; i due brani per pianoforte di Bach hanno una presenza di simboli molto meno densa che rende la rilevazione in sostanza priva di equivoci; infatti l'algoritmo da me sviluppato fallisce solo nell'individuazione di un bequadro nell'analisi della polacca, ed effettua una acquisizione di dati perfetta per quanto riguarda il minuetto; il sistema OMR utilizzato da MuseScore presenta risultati altrettanto buoni; tuttavia è stata riscontrata una certa difficoltà nell'individuazione dei punti di valore; si verifica inoltre, per quest'ultimo, l'aggiunta di un punto di staccato non presente

nella partitura originale. PhotoScore restituisce risultati privi di errore per questi primi due brani.

Quartetti di Mozart

Per quanto riguarda l'analisi dei quartetti di Mozart, il rilevamento dei dati non è stato altrettanto impeccabile; l'algoritmo da me sviluppato in alcuni casi non è stato in grado di individuare la presenza di alterazioni e pause nelle zone di maggiore densità dei simboli; il motivo è da rintracciare nel processo di *template matching*; il problema potrebbe risiedere in un processo di segmentazione che degrada troppo i simboli da rilevare; oppure si potrebbe pensare che il piccolo database da cui preleviamo i *template* da rintracciare, non sia adeguatamente fornito di tutta una gamma di diversi stili con cui uno stesso simbolo può essere rappresentato; il processo da me sviluppato, tuttavia si dimostra piuttosto solido per quanto riguarda il rilevamento delle note, dei pentagrammi e delle battute. Per quanto riguarda i risultati ottenuti dall'algoritmo OMR utilizzato da MuseScore, prendendo come input i quartetti per archi di Mozart, possiamo dire che le imprecisioni prodotte dal sistema sono molteplici; fra gli errori più pesanti vi è la mancata rilevazione di alcune note dalla testa vuota (minime e semibrevi), oltre che la rilevazione errata di alcune figurazioni ritmiche miste; una di queste figurazioni composte è stata confusa persino con una chiave basso; si è potuto notare poi come nel sistema OMR di MuseScore non è implementato un controllo per quanto riguarda la durata delle battute; in seguito a rilevamenti fallaci alcune battute hanno una durata superiore a quella consentita; d'altro canto quest'algoritmo si dimostra assai efficace nell'individuazione delle alterazioni e delle pause. PhotoScore si dimostra assai robusto per l'individuazione di tutte le categorie di simbolo; anche quest'ultimo però, si è rivelato non impeccabile nel rilevamento dei quartetti, in quanto alcune semicrome sono state rilevate come crome.

Andiamo ora ad illustrare gli indici percentuali di valutazione della partitura ricostruita:

Tasso di ricostruzione percentuale

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	100%	99%	100%
Polacca di Bach	99%	99%	100%
Mozart op. 499	81%	76%	95%
Mozart op. 464	82%	79%	97%

Tasso di ricostruzione percentuale pesato

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	100%	99%	100%
Polacca di Bach	99%	99%	100%
Mozart op. 499	79%	75%	93%
Mozart op. 464	81%	77%	95%

Errore di ricostruzione percentuale

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	0%	1%	0%
Polacca di Bach	1%	1%	0%
Mozart op. 499	20%	25%	4%
Mozart op. 464	19%	24%	3%

Come ci aspettavamo, nel considerare la stima del significato semantico dei simboli e delle relazioni, si è verificato un calo delle percentuali di successo, rispetto alle percentuali di semplice individuazione dei simboli; i risultati tuttavia sono in linea con quelli ottenuti dal tasso di riconoscimento percentuale; questo denota che tutti e tre i sistemi valutati si sono dimostrati affidabili nello stimare le relazioni ed il significato da attribuire ad ogni simbolo rilevato.

Nonostante le numerose imprecisioni emerse nel corso del processo di riconoscimento, la percentuale di successo assume dei valori molto alti; come avevamo preventivato un sistema OMR è uno strumento che può facilitare il processo di acquisizione dei dati, ma non può sostituirlo completamente; l'operazione di riconoscimento automatico dovrà essere sempre seguita da una fase di verifica ed eventuale correzione ad opera dell'utente.

Andiamo ora a quantificare il vantaggio di tempo che otterremmo dall'acquisizione dati che sfrutta un sistema OMR rispetto all'acquisizione operata in maniera manuale. Andando ad approssimare il tempo di inserimento, rimozione, correzione ed aggiunta di un simbolo manualmente con $C_i = C_f = C_m = C_a = 5\ sec$, ed il tempo di verifica fatta dall'utente con $C_v = 1\ sec$, calcoliamo il guadagno come rapporto fra tempo necessario all'inserimento manuale e tempo necessario per correzione di un'acquisizione ottenuta tramite sistema OMR.

Guadagno

Input	Programma di tesi	MuseScore OMR	PhotoScore
Minuetto di Bach	5	4,8	5
Polacca di Bach	4,8	4,8	5
Mozart op. 499	3,1	2,4	4,7
Mozart op. 464	3,3	2,9	4.8

Dalla tabella qui sopra possiamo notare come il brano perfettamente acquisito abbia guadagno 5; ricordiamo che il guadagno è calcolato come il rapporto fra tempo di inserimento manuale su tempo di revisione dell'output del sistema OMR. Indicando con N il numero di simboli da inserire otteniamo, per un brano perfettamente riconosciuto, un guadagno pari a:

$$\frac{N * 5s}{N * 1s} = 5$$

Per tanto nel caso migliore, grazie al programma, invece di compiere N operazioni di inserimento, compiremmo N operazioni di verifica, le quali richiedono un quinto del tempo rispetto alle precedenti.

6 Conclusioni

L'obiettivo che si è tentato di raggiungere nel corso di questo progetto di tesi è stato quello di creare un software in grado di diminuire i tempi di inserimento dei dati musicali, posseduti sotto forma di foto o pdf, all'interno di un programma di notazione musicale. Per raggiungere tale scopo è stato progettato un algoritmo in grado di rilevare le *feature* musicali presenti all'interno di un'immagine grazie all'uso di tecniche della Computer Vision.

I risultati ottenuti sono stati soddisfacenti in quanto l'algoritmo si è dimostrato piuttosto solido per quanto riguarda il rilevamento dei simboli più importanti come il rilevamento dei pentagrammi, delle battute e delle note; tuttavia ha mostrato un grado di affidabilità leggermente inferiore per quanto riguarda il rilevamento di caratteri come le alterazione (diesis, bemolle e bequadro); il calo di affidabilità può essere attribuito al fatto che si è fatto uso di un limitato numero di campioni con il quale è stata effettuata la procedura di *template matching;* infatti simboli come alterazioni possono avere stili diversi a seconda di come la partitura sia stata prodotta; un'altra causa potrebbe risiedere in una segmentazione che degrada troppo l'immagine sulla quale è eseguita la ricerca del *template*; infine per eseguire il *template matching* ci si è affidati ad una funzione fornita da *opencv*, non considerando che potrebbero essere trovate soluzioni più performanti.

Da questo punto di vista possono essere fatti sicuramente dei miglioramenti per tentare di raggiungere le prestazioni di programmi proprietari come PhotoScore; in particolare si intende:

- elaborare un metodo di rimozione del pentagramma che non rimuova porzioni dei simboli che vi giacciono sopra;
- estendere l'insieme dei simboli che il processo è in grado di rilevare;

- implementare un algoritmo di *template matching* più performante;
- creare un database che contenga il maggior numero di stili con cui i campioni da ricercare possano essere rappresentati;
- implementazione di un sistema di apprendimento tramite rete neurale per la categorizzazione dei simboli.

L'algoritmo in ogni caso dimostra la sua efficacia anche nello stato attuale.

Nel capitolo precedente abbiamo potuto notare come la valutazione a proposito delle prestazioni sia assolutamente positiva; infatti i risultati ottenuti mostrano un numero di valutazioni corrette di gran lunga superiore agli errori; ne consegue come il vantaggio ottenuto in termini di tempo impiegato, nell'acquisizione dei dati musicali grazie all'utilizzo di questo software, sia notevole.

Bibliografia

[1] [Online]. Available: https://en.wikipedia.org/wiki/Category:Music_software.

[2] [Online]. Available: https://it.wikipedia.org/wiki/Riconoscimento_ottico_dei_caratteri.

[3] [Online]. Available: https://it.wikipedia.org/wiki/Partitura.

[4] I. FuJinaga, «Optical Music Recognition using Projections».

[5] I. Bruno, «ANALISI DI IMMAGINI DI SPARTITI MUSICALI: METODI E STRUMENTI PER IL RICONOSCIMENTO E L'INDICIZZAZIONE AUTOMATICA».

[6] [Online]. Available: https://it.wikipedia.org/wiki/Visione_artificiale.

[7] [Online]. Available: https://it.wikipedia.org/wiki/Elaborazione_digitale_delle_immagini.

[8] [Online]. Available: https://it.wikipedia.org/wiki/Segmentazione_di_immagini.

[9] Nixon, Feature Extraction in Computer Vision and Image Proessing.

[10] Gonzalez, Digital Image Processing.

[11] B. Jahne, Digital Image Processing.

[12] [Online]. Available: https://it.wikipedia.org/wiki/Riconoscimento_dei_contorni.

[13] T. Adlakha, Analytical Comparison between Sobel and Prewitt Edge Detection Techniques.

[14] [Online]. Available: https://en.wikipedia.org/wiki/Canny_edge_detector.

[15] [Online]. Available: https://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid666794.pdf.

[16] S. G. a. R. Porwal, «COMBINING LAPLACIAN AND SOBEL GRADIENT FOR GREATER SHARPENING».

[17] W. K. Pratt, Introduction to Digital Image Processing.

[18] [Online]. Available: https://en.wikipedia.org/wiki/Music_information_retrieval.

[19] [Online]. Available: https://it.wikipedia.org/wiki/Fuga_(musica).

[20] [Online]. Available: https://it.wikipedia.org/wiki/Template_matching.

[21] [Online]. Available: https://www.treccani.it/vocabolario/spartito/.

[22] [Online]. Available: https://it.wikipedia.org/wiki/Scala_diatonica.

[23] [Online]. Available: https://it.wikipedia.org/wiki/Glossario_musicale_(R-Z).