

# A System for Optical Music Recognition and Audio Synthesis

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Medieninformatik**

eingereicht von

**Matthias Wallner**

Matrikelnummer 0451221

an der  
Fakultät für Informatik der Technischen Universität Wien

Betreuer: Prof. Dr. Horst Eidenberger

Wien, 15.09.2014

\_\_\_\_\_  
(Unterschrift Verfasser)

\_\_\_\_\_  
(Unterschrift Betreuer)

# A System for Optical Music Recognition and Audio Synthesis

DIPLOMA THESIS

for acquiring the academic degree

**Master of Science**

in the study program

**Media Informatics**

submitted by

**Matthias Wallner**

matriculation number 0451221

at the  
Faculty of Informatics, Vienna University of Technology

Supervisor: Prof. Dr. Horst Eidenberger

Vienna, 15.09.2014

\_\_\_\_\_  
(Signature of the Applicant)

\_\_\_\_\_  
(Signature of the Supervisor)

Matthias Wallner  
Grenzstraße 50a  
8570 Voitsberg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit, einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Voitsberg, 29.06.2013

---

(Unterschrift Verfasser)

## Abstract

This paper addresses Optical Music Recognition (OMR), a way to convert music notation into a digital representation, and its acoustic rendition. On the basis of a software prototype designed for mobile devices, the necessary steps will be discussed and compared to related work. The focus is on the fields of image processing and pattern recognition as well as audio synthesis. Innovations and experiences that have emerged in the field of OMR have been selected and used throughout the development process of the prototype. There is a particular emphasis on projection-based methods, which have proven highly successful as early as in the 1980s. Of all the works that were carried out, the research of I. Fujinaga has to be named as the strongest influence of this thesis. Both its easy implementation and its efficiency are well suited for the task of OMR on mobile devices. Furthermore, similarities and differences between the techniques applied and other approaches in the field of OMR will be presented. Symbol recognition is carried out by means of simple algorithms of pattern recognition (template matching) while the results are translated into an audible representation via MIDI synthesis. While there exist several performant OMR systems, specially tuned for the use with personal computers and scanners, OMR on mobile devices is still in its infancy. With the assistance of well-known and proven methods in this very field, this thesis provides a playful contact with optical music recognition.

*Die vorliegende Arbeit befasst sich mit Optical Music Recognition (OMR), einer Methode zur Überführung von Musiknoten in eine digitale Form, sowie deren akustischer Wiedergabe. Basierend auf einem Softwareprototyp, der für den Betrieb auf Mobiltelefonen entwickelt wurde, wird das Verfahren im Detail präsentiert und verwandten Systemen gegenüber gestellt. Im Mittelpunkt stehen im Wesentlichen die Bereiche Bildverarbeitung und Mustererkennung sowie Audiosynthese, wobei Erfahrungen und Forschungsergebnisse auf dem Gebiet der OMR selektiert für die Entwicklung des Prototypen herangezogen wurden. Im Besonderen stützt sich das Framework auf projektions-basierte Methoden, womit bereits in den 1980er Jahren beachtliche Ergebnisse erzielt wurden. Stellvertretend dafür ist die Arbeit von I. Fujinaga zu nennen, die einen wesentlichen Bezugspunkt für diese Diplomarbeit darstellt. Die relative Einfachheit der Umsetzung und der für die Anwendung auf mobilen Endgeräten ideale Tradeoff zwischen Erkennungsrate und rechnerischem Aufwand stellen einen großen Vorteil gegenüber anderen Methoden dar, die ihrerseits in dieser Arbeit anhand der ihnen zugrunde liegenden Techniken gegenüber gestellt werden. Die Erkennung der Musiksymbole wird mittels einfacher Algorithmen der Mustererkennung, konkret Template Matching, bewerkstelligt, wobei das Ergebnis mittels MIDI-Synthese in ein für Menschen hörbares Resultat umgewandelt wird. Während bereits leistungsfähige OMR-Systeme für die Nutzung mit Personal Computer existieren, steckt die Notenerkennung auf mobilen Geräten noch in ihren Kinderschuhen. Unter Zuhilfenahme bekannter und einschlägig erprobter Verfahren bietet die Diplomarbeit einen spielerischen Einstieg in das Gebiet der optischen Notenerkennung*

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Goals and Motivation . . . . .	1
1.2	Thesis Outline . . . . .	2
<b>2</b>	<b>Fundamentals</b>	<b>3</b>
2.1	Music Notation . . . . .	3
2.1.1	History . . . . .	3
2.1.2	Terms and symbols . . . . .	3
2.1.3	Other elements . . . . .	6
2.2	Optical Music Recognition . . . . .	6
2.2.1	Pattern Recognition . . . . .	6
2.2.2	OMR vs. Optical Character Recognition . . . . .	7
2.3	State of the Art . . . . .	10
2.3.1	Staff Line Identification and Removal . . . . .	11
2.3.2	Musical Object Location, Classification and Semantics . . . . .	14
2.3.3	Final Representation . . . . .	15
2.3.4	OMR Software . . . . .	17
<b>3</b>	<b>Adapting the OMR Framework</b>	<b>19</b>
3.1	Image Preprocessing . . . . .	19
3.1.1	Cropping . . . . .	19
3.1.2	Deskewing . . . . .	20
3.1.3	Binarisation . . . . .	22
3.1.4	Reference Lengths . . . . .	24
3.2	Staff Line Identification and Removal . . . . .	25
3.2.1	Filtering . . . . .	25
3.2.2	Stave Extraction . . . . .	26
3.3	Feature Classification . . . . .	27
3.3.1	Overview . . . . .	27
3.3.2	Implementation . . . . .	27
3.3.3	Clefs . . . . .	29

3.3.4	Notes . . . . .	31
3.3.5	Key signatures . . . . .	33
3.3.6	Accidental detection . . . . .	34
3.3.7	Detection of other elements . . . . .	35
3.4	Musical Semantics and Transformation . . . . .	37
3.4.1	Symbol Interpretation . . . . .	37
3.4.2	Audio Synthesis . . . . .	38
<b>4</b>	<b>Implementation</b>	<b>43</b>
4.1	Platform and tools . . . . .	43
4.2	Hardware . . . . .	43
4.3	perfOMR . . . . .	44
4.3.1	User Interface . . . . .	44
4.3.2	Symbol templates . . . . .	44
4.3.3	Helper Classes . . . . .	45
4.3.4	Audio Rendering . . . . .	47
4.3.5	Bugs and Limitations . . . . .	47
4.3.6	UML . . . . .	50
4.4	License . . . . .	51
<b>5</b>	<b>Evaluation</b>	<b>53</b>
5.1	Methodology and Metrics . . . . .	53
5.2	Template Dataset . . . . .	54
5.2.1	Image Deformation . . . . .	54
5.3	Results . . . . .	55
5.3.1	Staff line detection . . . . .	56
5.3.2	Symbol Recognition . . . . .	56
5.3.3	Other symbols . . . . .	58
5.3.4	Pitch Recognition . . . . .	60
<b>6</b>	<b>Conclusion and Future Work</b>	<b>61</b>
	<b>Bibliography</b>	<b>63</b>



# Introduction

## 1.1 Goals and Motivation

For thousands of years, music has been a cultural companion of mankind, a way for humans to express and evoke feelings, be it as active musicians or untrained listeners. Throughout this period and particularly in the field of classical music, which is typically performed by larger groups of musicians, there has been a need to depict one's musical ideas in an understandable, graphical format, which led to the introduction of music notation. Consisting of symbols of well-known meaning, sheets of music, albeit different from how we know them today, became the method of choice to communicate and record musical ideas for the ancient Greeks. Notation as a whole, despite having evolved over centuries, helps its readers to globally share a common understanding, no matter what type of music or the degree of musical education.

In every civilisation, music is considered a pivotal part of culture and commonly this is also reflected in general education. As early as elementary school or sometimes even before, children become acquainted with the syntax and semantics of notes. In order to be able to convey the ideas of music, however, it is necessary to have some knowledge of musical conventions.

Assuming there was a system that would translate a piece of music into an auditive representation, not only would musicians greatly benefit from a way to get an idea of the tune, as quite a few of them may be unable to properly read musical notation. But this would also help in a more casual environment. Just consider parents or grandparents who would like to sing a song for or together with their children, and cannot remember the specific tune. In such a case, an application which supports the step of interpretation would be very convenient indeed.

Having been a musician for more than twenty years myself and thus being familiar with the difficulties of understanding music notation from personal experience, I quickly



warmed to the idea of creating a tool that can bridge the gulf described above.

*It is the goal of this thesis to provide a prototype software which runs on the Android mobile operating system and transforms musical notation - in the form of simple children's tunes - into acoustic renditions.*

The application should incorporate features of both individual notes, such as key or duration, and more complex inter-notational characteristics. While many algorithms are based on the acquisition of images by means of a document scanner, the optical sensor of a smartphone will be used for the same task in the present system. The idea of the prototype is to provide a simple to use and playful means of music translation in a typically casual environment, which can take advantage of the massive improvement smartphone technology has seen in recent years.

## **1.2 Thesis Outline**

The structure of the underlying thesis is as follows:

Chapter 2 sketches the fundamentals of Common Music Notation and the means of automatic recognition of musical features by the use of Optical Music Recognition. Chapter 3 gives an in-depth description of the underlying framework, while Chapter 4 provides information on the actual implementation and deployment of the prototype, referring to limitations of operating system and hardware. The thesis concludes with an evaluation of the framework on a test set of sample notations of simple children's tunes.

# CHAPTER 2

## Fundamentals

This chapter provides a brief introduction into the field of music notation before we delve into the more analytical area of Optical Music Recognition (OMR).

### 2.1 Music Notation

#### 2.1.1 History

Throughout history, many different forms of musical notation have been used by the respective cultures. One of the earliest known examples dates back to the Mesopotamia of 2,000 B.C. [18], showing musical instructions using a diatonic scale, which is seen as the foundation of western music as we know it. It is commonly believed, however, that modern notation stems from the ancient Greeks, who assigned a group of three letters of their alphabet to each tone in order to visualise tones as well as their respective semi- and quarter-tones. Later, in the eleventh century A.D., a graphical means was introduced to represent the degrees of the scale, which until then had been described by Latin letters. These elements, which we refer to as *staves* today, will be extensively examined in the following subsections, as they are a key element to OMR.

Not least the efforts of churchmen who tried to find an absolute system of indicating both pitch and time values of sounds through several medieval centuries, have extensively shaped western music notation, which is applied by musicians of all genres throughout the world. [43]

#### 2.1.2 Terms and symbols

The following section contains relevant information on the most important musical symbols which combine into a modern score.

## Staves

Staves are a regular element on a score sheet, usually occurring in sets of five parallel lines of the same length. Between each of these lines, the space is fixed. While a piece of music is used to be played in a strict top-down and left-to-right order, staves may also be grouped, depending on the number of instruments. In this case, these groups of staves are considered to have the same 'timestamp', which means that they are played simultaneously. As this thesis focuses on simple children's tunes, only notations depicting a single instrument or voice shall be considered subsequently. Please note that stems may be enhanced with short ledger lines, which, if necessary, mimick another stave line above or below the regular staves.

While staves as horizontal lines provide a means of organizing the tonal dimension, the temporal dimension is delimited with vertical bars, connecting the topmost with the lowermost line of the stave. Depending on the particular time signature, there is a fixed number of counts within these bar lines while the interval on the x-axis is subject to variation.

## Notes

Individual tones are indicated by notes which are embedded into the staves as discussed above. Notes convey a meaning both with their vertical position, determining the pitch or tone height, and their shape. Usually, a note consists of a note head and a connected stem. In certain cases, where two or more notes are played at the same time, all stems but one are omitted.

The appearance of the note head gives information on the duration of a particular note, which is shown in Fig. 2.1. However, the time value can be increased by a half of its value by a dot inside a small area to the right of the note head. In addition to that, hooks or beams, which are connected to the other end of the stems, have a similar meaning. While hooks reduce the time value of only the adjacent note head, beams have an enlarged scope across two or more notes (Fig. 2.2).

## Key signature and accidentals

Tonality of a music score as a whole is also defined by key signatures which are located at the beginning of a stave or, in some cases, next to a double bar. Depending on the vertical position on the stave lines and on their shape they affect different notes that will be increased or decreased by a semitone.

The same applies to accidentals that are considered as key signatures which change the pitch of the note that it immediately precedes and of any following notes that are either on the same line or space in the measure, which, in turn, is delimited by a vertical bar. Where applicable, previous assignments within a measure are cancelled by the natural sign.

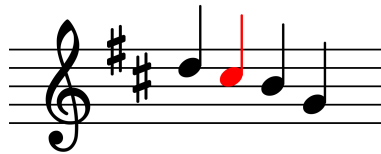


Figure 2.1: Example of a key signature designating the highlighted note to be played one semitone higher.



Figure 2.2: G-clef, F-clef and C-clef.



Figure 2.3: Examples of time signatures.

There are other forms of accidentals such as *e.g. double, courtesy or quarter-tone accidentals* which, due to their limited use, will not be referred to in the following.

### Clefs

Also affecting the tonal value of the notes, the first element occurring on a staff is the clef which comes in three main types (Fig. 2.2). In this paper only one of these, the G- or treble clef will be of interest.

### Time signature

Positioned near the clef sign in most cases, a time signature affects neither the time nor the pitch value of notes. While the numerator specifies the number of beats per bar, the denominator indicates the note value of one beat. A bar is referred to as space between two neighbouring bar lines that are drawn perpendicular to the staff lines. The letter C (see Fig. 2.3) indicates the common time which would otherwise be represented by the 4/4 meter. Common children's tunes, which are subject to this work, have been written



Figure 2.4: Pause symbols of varying length.

in a 3/4 or 4/4 time signature while other musical genres like Jazz or classical music show a higher variability in this particular property.

### 2.1.3 Other elements

In addition to the metrum described above, pauses play a role in defining the temporal layer of a musical piece. Fig. 2.4 shows a selection of common rests of different lengths, corresponding with the length of note values (measure given in beats): *Semibreve* (1), *Minim* (1/2), *Crotchet* (1/4), *Quaver* (1/8), *Semiquaver* (1/16).

## 2.2 Optical Music Recognition

After presenting the key elements of musical notation, we now approach the problem of Optical Music Recognition (OMR), a set of different methods for recognizing musical symbols and their translation into a machine-readable format.

In the following, pattern recognition will be discussed, which is a basic technique in any character recognition framework. Here, I am also going to point out similarities and dissimilarities between the fields of OMR and Optical Character Recognition (OCR) before giving a brief overview of common OMR techniques and applications.

### 2.2.1 Pattern Recognition

Generally speaking, pattern recognition seeks to analyze and categorize signals, assigning labels to given input data. Independent of the respective approach, the process can be divided into a total of four main stages that, in our application, directly follow the capturing stage (see Fig. 2.5):

*Preprocessing* is undertaken in order to restrict input data to a minimum, which often involves a normalization of the input. We will refer to the image preprocessing in the following chapter. When performing pattern recognition on images, *segmentation* then separates components, providing a basis for higher-level recognition processes. Based on the segments, *feature extraction* derives properties of the respective signals (e.g. width or height) which then are differentiated into classes by means of *classification*.

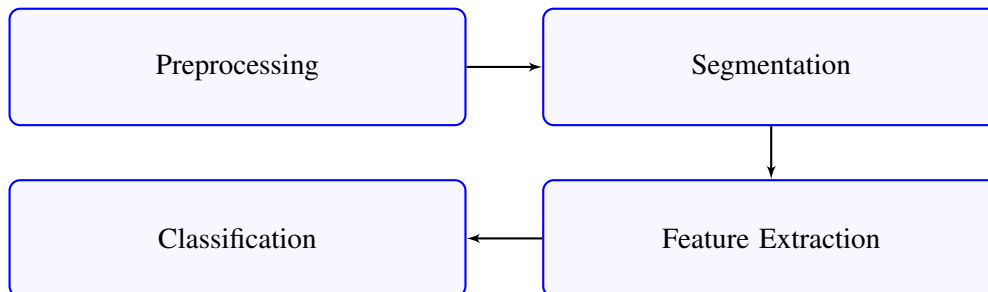


Figure 2.5: Overview over a typical pattern recognition system.

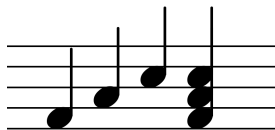


Figure 2.6: Pitch is conveyed through position on y-axis. The figure shows a major triad consisting of f', a' and c''.

### 2.2.2 OMR vs. Optical Character Recognition

Both the field of OMR and Optical Character Recognition belong to the domain of Pattern Recognition. While the inherent problems may look similar at first glance, they are, however, substantially different. Moreover, OMR can be described as a superset of OCR for the following reasons:

According to [5], textual information is usually aligned in a one-dimensional way in which a page is divided by regular spaces between singular text lines. By contrast, musical notation makes extensive use of the vertical dimension by using the y-axis to convey information on pitch. Notes are not restricted to be displayed in a one-linear fashion from left to right. In case of chords, stems may be omitted, resulting in individual notes that are placed on top of each other or with a minor variation along the x-axis. Fig. 2.6 shows three singular, graphically identical crotchets or quarter notes that have been translated to different positions and next to them a chord consisting of the very same notes. This phenomenon, which is known as (selective) translation, leads to a high number of variations, rendering impossible a template matching with finite templates.

While the horizontal distance between two elements in musical notation is mostly uniform, it needs to be stretched or narrowed in order to facilitate readability in certain cases. This possibly results in the same event having more than one graphical representation. At the same time, minor graphical differences in music convey important information while there is a substantial difference in shape between most glyphs in text.

Another property which is common to musical symbols is that many of them consist of multiple components. In western text, most glyphs are made of one region which in some cases is complemented by smaller supporting dots or diacritical marks. What makes the task of OMR even more complicated is that the above cited properties do not only occur isolated but also simultaneously [5]. Furthermore, there is no uniform appearance of musical symbols. They are not only variable in size and shape if one compares handwritten scores to printed scores but also among different printed publications [34]. As mentioned in [42], the superimposition of syntactically unrelated symbols often occurring on music scores with a high density is another problem that OMR has to cope with. Strictly speaking, authors are responsible for making their work as clear as possible but readability is heavily affected by the inherent complexity of the score.

In 2.7, we see a musical piece that, despite all measures of diligence taken, suffers from the issues cited above. The area marked with a red frame shows a diagonal line from the upper left to the lower right corner, denoting a *glissando*, which is an effect where the performer glides from one pitch to another. Obviously, this symbol stretches not only across several staff lines, but it also touches the flat accidental as well as a double beam and a crescendo mark. In the blue bordered area, there is a typical example of accidentals, which, due to obvious reasons of space, are touching each other. While bounding box based approaches are regarded as fast and accurate means for musical object classification, they often fail in cases of superimposed objects [4].

In addition to the musical symbols described earlier, musical scores contain textual information that affects the way music is interpreted, often affecting dynamics (e.g. *mf*: mezzoforte = moderately loud), or tempo (e.g. *rall.*: rallentando = gradual, slow decrease of tempo) of certain areas of the score. Thereby, the position of these tokens is vital in order to identify their scope.

The image displays a musical score for "Hill Song I" by P. A. Grainger, consisting of four systems of piano and vocal staves. The score is annotated with various musical symbols and performance instructions:

- System 1:** The vocal line is marked *plaintively*. The piano accompaniment starts with a dynamic of *f* and includes the instruction *mf louden lots*.
- System 2:** The vocal line has a red rectangular highlight around a specific passage. The piano accompaniment includes the instruction *mf skittishly* and features a blue rectangular highlight around a complex chordal passage.
- System 3:** The piano accompaniment begins with a dynamic of *mf* and includes the instruction *mf skittishly*. A blue rectangular highlight is present around a complex chordal passage.
- System 4:** The piano accompaniment starts with a dynamic of *f* and includes the instruction *mf*. A blue rectangular highlight is present around a complex chordal passage.

The score includes various musical notations such as dynamics (*f*, *mf*, *mp*, *ff*), articulation (*tr*), and performance instructions (*plaintively*, *louden lots*, *skittishly*). The piece is in 2/4 time and features complex harmonic structures, including triplets and sixteenth-note passages.

80468

Figure 2.7: Superimposition of musical symbols using the example of an excerpt of "Hill Song I" by P. A. Grainger.



## 2.3 State of the Art

The difficult task of Optical Music Recognition has been extensively studied in the past few decades and provided a wealth of publications that belong to the fields of image processing, graph theory, pattern recognition or probabilistic coding, to name a few. Efforts have resulted in numerous OMR systems (commercial and non-commercial ones), particularly since the 1980s, when hardware costs fell to a reasonable level. Yet, OMR applications are still considered to be inferior in performance compared to their OCR counterparts [42].

Due to its complexity, the problem of OMR is generally subdivided into a number of subtasks, which are typically processed in a uni-directional fashion. In this pipeline, decisions made in a later stage have no implication on any previous stage. According to Bainbridge [5] the problem of OMR comprises four major steps as depicted in Fig. 2.8.

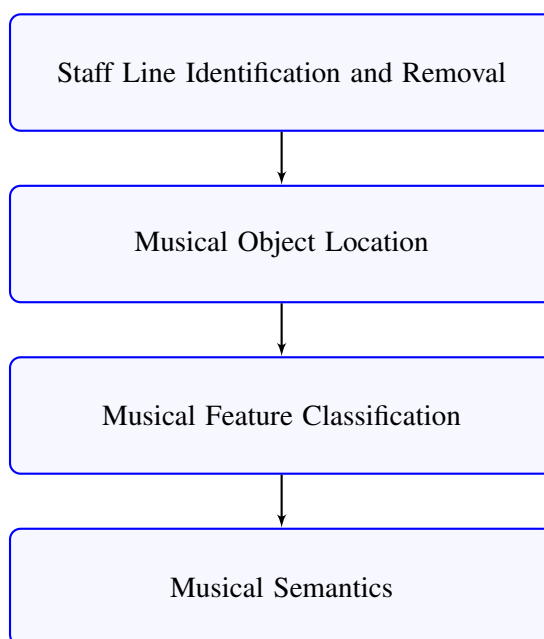


Figure 2.8: Control flow for a typical OMR application.

Related work suggests an alternative terminology which is mainly due to a different approach in decomposition [34]. The framework is not necessarily traversed in a top-down fashion as there are (but few) systems that try to improve recognition results by incorporating knowledge obtained in later steps into earlier stages. Moreover, each of the above stages is, to a greater or lesser extent, subject to variation in terms of used methods – these will be discussed in detail in the following.

### 2.3.1 Staff Line Identification and Removal

Staff line identification is not only considered as a preliminary step for the later removal of staff lines. It also provides an ideal starting point for segmentation-free approaches that do not rely on this potentially problematic task which is prone to fragmentations of symbols [26], [31].



Figure 2.10: Excerpt of “Pictures at an exhibition” (M. Mussorgsky). From Bainbridge and Bell, 2001 [5].

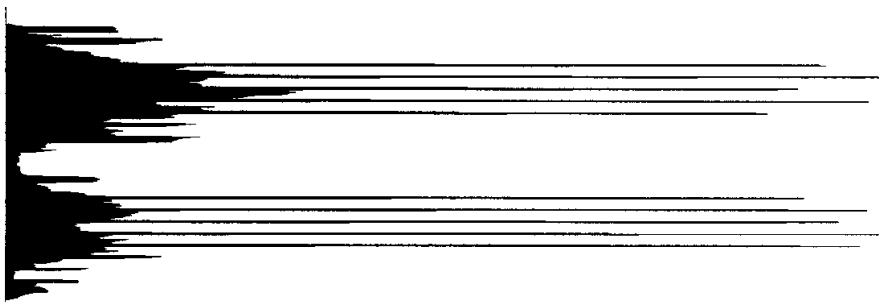


Figure 2.11: Corresponding horizontal projection of the example above. Spikes on the x-axis suggest the occurrence of staff lines for particular y-values. [5].

A variety of OMR work has focused on projections and runlengths in order to approach the task of staffline recognition. Being widely used outside the scope of OMR (e.g. medical applications), projections provide a decent tradeoff between ease of implementation and reliability. Introduced in OMR by Fujinaga [14], the projection method (see Fig. 2.10 and 2.11) has proven successful both for cases when good source material is provided and when other, computationally more expensive, methods are ruled out due to hardware limitations and/or realtime requirements. Working on pixel level, projection profiles are created along each line on the y-axis simply by summing up the number of pixels of the respective line before the most frequent black and white runlengths are used to locate the position of staff lines. In order to remove staff lines, black runlengths larger than a certain height are excluded so that only sufficiently wide connected components are considered later on. The remainder, a set of horizontal segments,

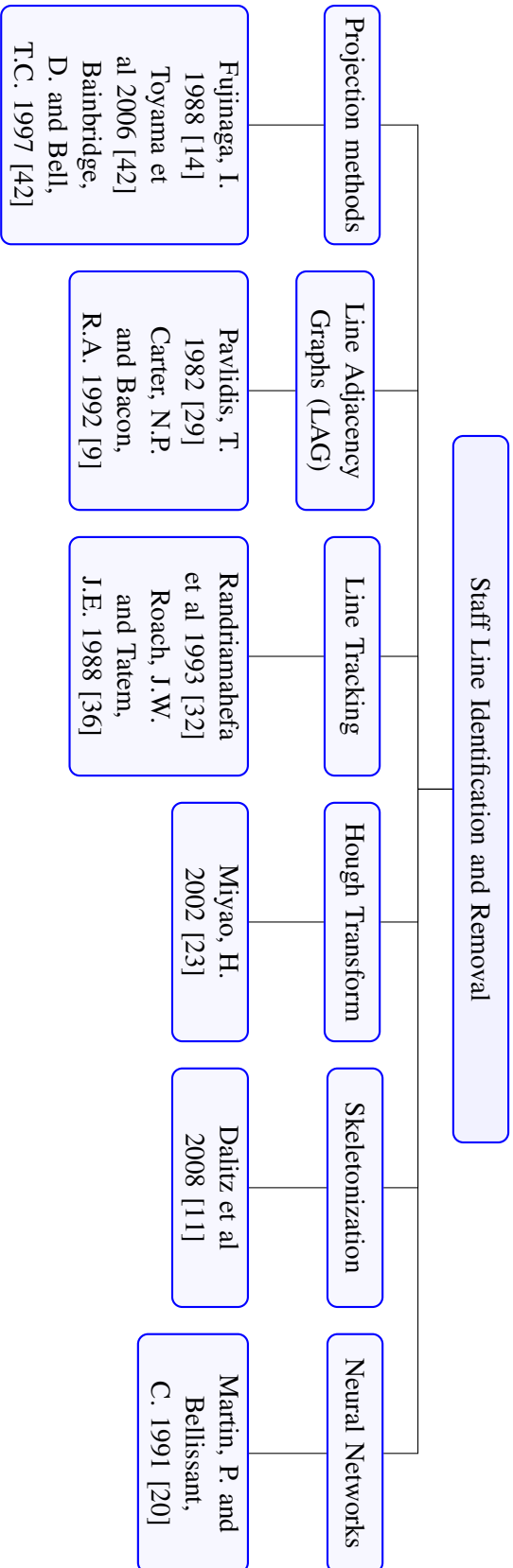


Figure 2.9: Overview of available methods on discriminating between staff lines and other objects in a notation.

is then subtracted from the original image. [42] has presented an alternative approach which relies on the vertical projection profiles by finding staff candidates in a run length histogram of equally spaced vertical scan lines. As the present work makes extensive use of projection methods, deeper insight will be given in Chapter 3 ff..

Line Adjacency Graphs (LAG) [29], a well-known principle in graph theory, are key to another method for removing staff lines. In the fashion of the concept described above it is based on runlengths. Similarly, it does not consider chords in arbitrary directions. Due to feasibility issues, it makes only use of the horizontal and the vertical directions [11] to search for potential sections of lines. In [9], the LAG is obtained by computing vertical runlengths for each column, whereas encodings of adjacent columns are compared. After the transformation, the LAG is filtered by aspect ratio, angle and connectedness [16], yielding a set of staffline sections which are referred to as filaments. The advantage of this method is that it does not contain parts of symbols that intersect staff lines, leaving intact as many elements as possible for the process of Musical Object Location. However, this comes at the expense of a high computational effort since every pixel has to be searched. The system is known to be robust, mastering skew up to ten degrees.

As proposed in [32], Line Tracking provides another hands-on approach to the problem of identifying staff lines. After the staff skeleton has been detected using estimates of both staff line thickness and the spacing between two staff lines of one stave, lines are tracked and vertical black runs around each point of the staff skeleton are removed if they match a certain criterion. As an advantage over other methods, staff lines are found irrespective of degradations of the image (e.g. warp, tilt). Various methods that provide intelligent means to deal with superimposed objects have been presented, only differing in the approach of how pixels are extracted.

In [20], the problem of discriminating between staff line and symbol parts is resolved with a multi-layered neural network using gradient back propagation. An approximation of the image skeleton containing endpoints, junction points and bending points is obtained by thinning. The estimated height of the staff line can be obtained from the histogram of the number of pixels that have been removed during each iteration of the thinning process [26]. Furthermore, the original image can be reconstructed using the skeleton information.

Originally tuned to handwritten scores, the Vector Field method described by [36] seeks to remove all shapes that are known to be perfectly horizontal. This information is gained by computing a function that incorporates both chord length and angle for each foreground pixel thus resulting in a vector field. Staff line pixels are then identified using a labelling scheme, trying to leave intact pixels that possibly belong to music symbols by integrating information of neighbouring pixels into the labelling process.

Feature extraction techniques like Hough transform are widely used in such fields as computer vision or image analysis, and there have been attempts to exploit their ad-

vantages in optical music recognition. Above all, this method is capable of delivering acceptable results in the task of staff line detection, even when there are large inclinations or discontinuities. Results degrade, of course, when staff lines are curved, e.g. due to properties of the camera lens. [23] have proposed an improvement of their earlier method which comprises the extraction of certain candidate points on a staff line and their connection - hereby, the optimisation technique of matching by dynamic programming is taken advantage of. Subsequently, groups of staves are composed and, in a concluding step, edges are extracted. This method delivers satisfactory results even on imperfect scans in terms of the characteristics mentioned above.

### 2.3.2 Musical Object Location, Classification and Semantics

Being logically placed after the step of staff line detection, Musical Object Location deals with the segmentation and classification process of musical primitives. Due to the bidimensional structure of notation, poor printing, degradations caused by previous stages and/or due to the complexity of symbols, musical feature classification is a delicate task that has been addressed by numerous works. The usual strategy is to extract primitives, meaning basic music symbols such as note heads, stems, dots that are assembled and classified in turn. While these two steps are mostly processed together, few research works, including the underlying work, separate between these tasks [33].

Projection profiles play a major role not only in the process of staff line identification but also during the recognition of musical objects [30]. While we already described its ability to detect staff lines using horizontal projections, the same method can be applied to the vertical dimension in order to detect note stems, bar lines in subsequent stages. In [15], the  $k$ -nearest neighbour algorithm ( $k$ -NN) is used for classifying symbols based on the features extracted from the projection profiles.

[9] and [35] have both resorted to LAGs with the latter work introducing a three-step process. During segmentation, each connected component is represented by a compressed LAG whereas textual information is discarded. While lines and curves can be found by simply traversing the graph and merging adjacent lines, accidentals, rests and clefs are detected using character profiles that measure the perpendicular distance of their contour. For note heads, Template Matching is applied as they cannot be isolated by connected component analysis.

Generally, Template Matching, has seen widespread use in the field of OMR in order to detect primitives. In [42], the author introduces a coherence check for primitive candidates which refers to music writing rules. Furthermore, pixel tracking is applied in order to detect shapes such as stems, beams and hooks. [22] adopt a multi-layer neural network that identifies heads and flags from the candidates that have been extracted based on stem positions. Thus, the introduction of a rule-based system is omitted. The system delivers satisfying results on high quality scans.

While other techniques can be applied during several steps of the OMR pipeline, Hidden Markov Models are, for the most part, restricted to the step of object recognition. In his work on early typographic musical prints, Pugin seeks to bypass the sensitive step of removing staff lines. Instead, accurate recognition was achieved by treating staves as a sequence of symbols which in turn was converted into a probabilistic, one-dimensional model. This technique provided good recognition rates on simple scores [31]. Earlier, Kopec and Chou were among the first to integrate all steps of the framework described above into a single optimization process. Using only a subset of printed music notation for pedagogical purpose they successfully applied their document image decoding (DID) approach to music scores [37]. Inherent problems of this approach, which included a fast growth of the underlying Markov models, were addressed by Stückelberg and Doermann who moved from a generative to a descriptive recognition model [38].

An interesting approach is taken by Bainbridge and Bell who have integrated several of the aforementioned techniques into their CANTOR system which can be highly customised through a Primitive Expression Language (PRIMELA). Instead of working on a hard-coded string-based [10] or attributed programmed graph grammar [12], the user is entitled to specify the appearance of musical symbols which allows the framework to adapt to a variety of notations [3].

### 2.3.3 Final Representation

A number of systems that translate musical notation into a machine-readable format have been deployed in recent years. Likewise, there has been an increase in efforts to build up large collections of digital music.

As more and more music sheets are digitized, there has emerged the need for an automated extraction of semantics in order to make them eligible for content-based retrieval and browsing. [13] have introduced an automated procedure for mapping and synchronizing sheet music to audio recordings which has been implemented in the PROBADO music repository. For the mapping procedure, the two domains of notation and audio recordings are translated into the same chroma representation as proposed in [19] where chroma refers to the twelve traditional pitch classes of a scale.

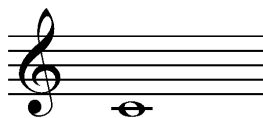


Figure 2.13: A middle C whole note, depicted in standard western notation.

The authors of [2] have presented a system that captures images through a handheld scan device, playing sounds according to the notation data in realtime. Additionally, this tangible device enables the user to further interact by means of velocity, pitch or

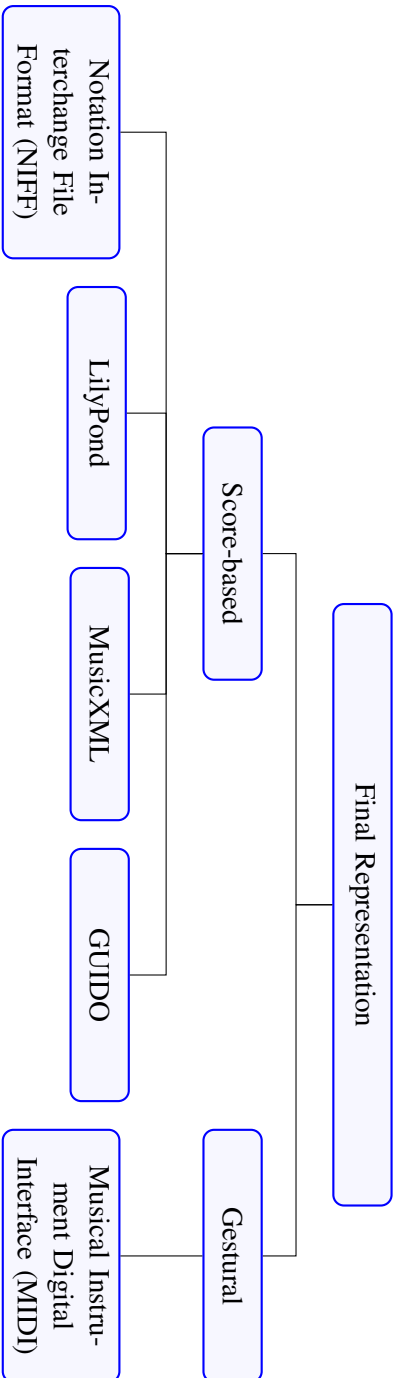


Figure 2.12: Music formats serving as means of storing or processing musical data. This list is not intended to be exhaustive.

instrument changes. In general, there exists a wealth of commercial OMR systems that provide means to export the results of OMR into multi-purpose formats (see Fig. 2.12). For recording previously recognised musical data, this thesis focuses on Musical Instrument Digital Interface (MIDI), which, being a gestural format, is an exception among the numerous representations of music. Among the score-based formats are Notation Interchange File Format (NIFF, obsolete), GUIDO (complete OMR system) and MusicXML which has drawn attention recently with the latter having gained status of a de facto standard. MusicXML allows for musical information exchange across a wide range of applications that are concerned with composition, recognition or playback of music [40]. Another example is GNU LilyPond, which serves both as a file format and a computer program. See Figure 2.13 and Listings 2.1 and 2.2 for a brief example.

---

Listing 2.1: Example of the MusicXML format.

---

```
<note>
  <pitch>
    <step>C</step>
    <octave>4</octave>
  </pitch>
  <duration>4</duration>
  <type>whole</type>
</note>
```

---

Listing 2.2: Example of the LilyPond format.

---

```
\begin{lilypond}
  c'4
\end{lilypond}
```

---

### 2.3.4 OMR Software

Among the well-known proprietary representatives of fully-functional OMR software are Capella Scan, VivaldiScan or ForteScan but moreover, efforts have been made to provide software under the GNU General Public License [1].

In the field of mobile computing, Neuratron's NotateMe [25] is a prominent example for shifting the location of music editing and capturing to the actual workplace of the musician. This system not only allows for recognition of handwritten music through the display touchpad but recently, full-featured OMR capability has been integrated, allowing for functions that have been limited to personal computers in the past. These comprise of editing and exporting of previously captured sheet music while for the latter, MusicXML is used as the format for import and export.





## Adapting the OMR Framework

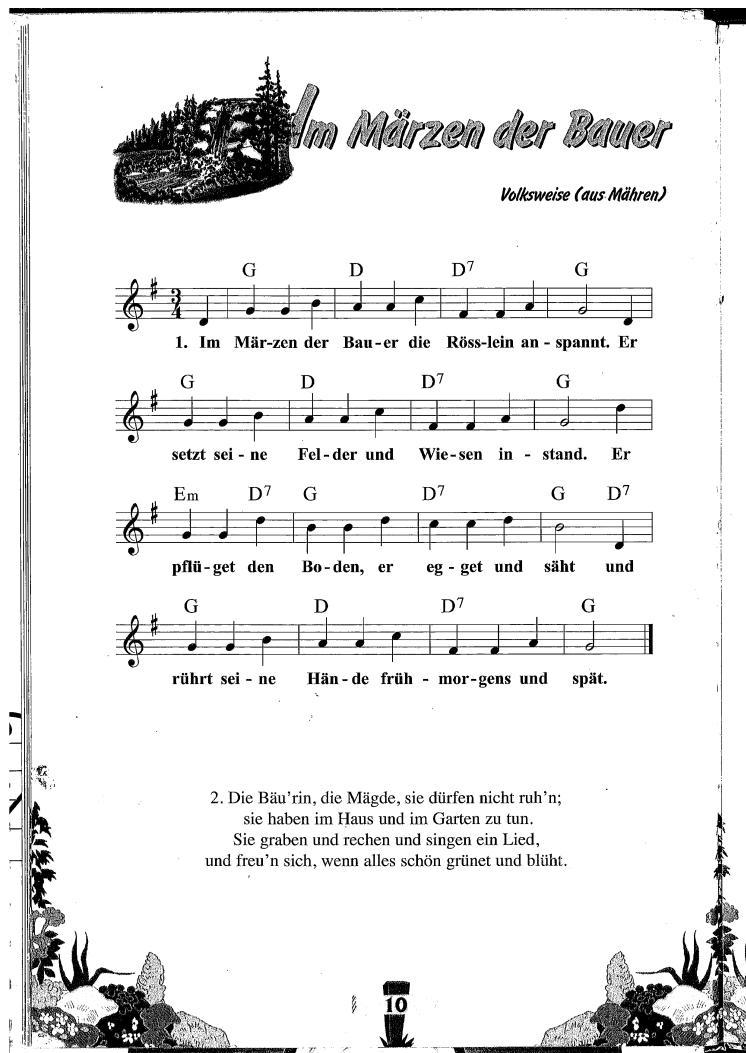
Similar to other implementations, the framework used in the prototype follows the same unidirectional pipeline that has been presented in the previous chapter. In the following, the various stages will be discussed and the adopted techniques will be explained in detail. For illustrational purposes, the classic children’s tune “Im Märzen der Bauer” will be used (see Fig.3.1).

### 3.1 Image Preprocessing

It is important to consider that the system is designed to cope with mobile photographs which are of a lower quality than typical high-resolution scans of notations. Research has shown that minor degradations of image quality have already an enormous effect on the recognition rate. Therefore, exhaustive measures have to be taken to ensure a decent input even in a difficult environment.

#### 3.1.1 Cropping

In the beginning, all portions of the image that are not part of the music sheet are removed. The idea is to detect the largest white element in the photograph, which is assumed to be the notation. As a preparatory step, the image needs to be downscaled in order to remove noise. In addition to that, the morphological operation of erosion is applied to the image which joins disparate elements by replacing the pixel in the anchor point of a kernel  $K$  with the minimal pixel value of image  $I$  in the region covered by  $K$ . Iteratively, a contour detection algorithm based on the algorithm [39] is performed alongside the morphological operation, providing a list of rectangular shapes. Processing stops once a rectangular shape of sufficient size has been detected. For the actual



**Im Märzen der Bauer**  
Volksweise (aus Mähren)

1. Im Mär-zen der Bau-er die Röss-lein an - spannt. Er  
setzt sei - ne Fel-der und Wie-sen in - stand. Er  
pflü-get den Bo-den, er eg - get und säht und  
rührt sei - ne Hän - de früh - mor-gens und spät.

2. Die Bäu'rin, die Mägde, sie dürfen nicht ruh'n;  
sie haben im Haus und im Garten zu tun.  
Sie graben und rechen und singen ein Lied,  
und freu'n sich, wenn alles schön grünnet und blüht.

10

Figure 3.1: Scanned notation prior to preprocessing.

implementation of the process described, I have employed the *OpenCV* library. Results can be observed in Fig.3.2.

### 3.1.2 Deskewing

It is an unfavourable property of photographs or even some image scans, that their main objects are often skewed at an arbitrary angle, which provides a suboptimal starting point for the following steps in the underlying framework. It has been mentioned earlier that this work heavily relies on using information from projections which means it is elementary to properly align the image in order to obtain satisfactory results. In order to

1. Im Mär-zen der Bau-er die Röss-lein an - spannt. Er  
 setzt sei - ne Fel - der und Wie - sen in - stand. Er  
 pflü - get den Bo - den, er eg - get und säht und  
 rührt sei - ne Hän - de früh - mor - gens und spät.

Figure 3.2: The input image after the cropping process.

automatically compensate for the rotation of the mobile phone relative to the notation sheet, it seems logical to determine the angle of rotation by means of projection profile analysis, which takes into account local pixel information [27].

A narrow strip located centrally in the image provides the starting point to the algorithm, on which a *horizontal projection* is applied. The resulting profile, depicting the corresponding image rows with a high number of information (foreground pixels) as peaks, is then used as reference projection to measure the skew between two adjacent strips. Correlation coefficients between a y-projection of the reference strip and a scan-line of the adjacent vertical strip are then computed by shifting the neighbouring strip up and down. The projections are then summed up at the offset providing the highest coefficient, thus forming the new reference projection. Computation continues until we have reached the right bottom corner of the page, before the same procedure is applied to the area to the left of the reference strip.

The sum of offsets, which have been temporarily stored, provides the basis for the actual shearing process in the fashion of Bresenham's Line Drawing Algorithm [8]. Each column of the image is then shifted to the nearest pixel based on a slope value deducting the average deviation per column.

Listing 3.1: Deskewing process of the input image.

---

```

binarise(input)
set margins of centre window
idx = 0

while rightborder < width
    horizontalProjection(window)

    for y = 0 : height;
        coeff = projection1[y]*projection2[y2];
    end

    if (coeff > max_coeff)
        save max_pos and max_coeff;
    end

    for y = 0 : height;
        new_projection[y] = projection1[y] + projection2[y + max_pos];
    end

    offset += max_pos - max_pos[idx-1]
    idx++
end

while leftborder > 0
    ...
end

for x = 0 : width;
    for y = 0 : height;
        idx_new = idx + shift;
        temp_image[idx] = image[idx_new];
    end
    shift += offset_per_pixel
end
shift = 0
for y = 0 : height;
    for x = width-1 : x >= 0;
        idx_new = idx - shift
        image[idx] = temp_image[idx_new]
    end
    shift += offset_per_pixel
end
return image[]

```

---

### 3.1.3 Binarisation

Since musical notations are monochromatic “by nature”, it is reasonable to introduce an early binarization step in order to ease further processing and to eliminate background or noise. There is no consensus, however, on how to handle the conversion of the multivalued image to the according binary representation.

Otsu’s method [28], a variety of global thresholding, is one of the highest-rated

$T=122$
---------

$$T_{global}(g) = \begin{cases} 0 \dots g < t \\ 1 \dots g \geq t \end{cases}$$

Figure 3.3: Global thresholding.

$T=123$	$T=104$
$T=155$	$T=88$

$$T_{local}(x,y) = \begin{cases} 0 \dots g(x,y) < t \\ 1 \dots g(x,y) \geq t \end{cases} \in \text{Region } R_i$$

Figure 3.4: Local thresholding.

$T=144$	$T=116$
$T=115$	
$T=120$	$T=78$

$$T_{adaptive}(x,y) = \begin{cases} 0 \dots g(x,y) < t(N(x,y)) \\ 1 \dots g(x,y) \geq t(N(x,y)) \end{cases}$$

Figure 3.5: Adaptive thresholding.

procedures and has been implemented by many OMR applications. Here, image pixels are divided into two classes, fore- and background, respectively. The parameter  $T$ , by which pixels are discriminated, is computed by maximizing the variance between the two classes (see Fig. 3.3) Its advantage is that there is a low computational effort as only zero- and first order cumulative moments are utilized. Unfortunately, early tests have shown that the approach is not sufficient for the field of mobile photographs due to uneven illumination of the music sheet in certain environments and its inability to deal with strong illumination gradients.

As opposed to the global method, adaptive binarization [27] is capable of delivering satisfactory results even with poor images, as it uses local information from the image

itself. This procedure suggests using the mean and the standard deviation of a pixel's neighbourhood as local information (see Fig. 3.5).

For this prototype, I have opted for a hybrid model called local thresholding (see Fig. 3.4). Here, the image is first divided into chunks whereby the number of eight rows and columns has proven best for the image dataset. Iterating through the pixels of each image partition, the number of foreground and background pixels are counted, based on a threshold value  $T$  which has the initial value of  $0$ . Then, the mean value both of all foreground and background pixels is computed. After each iteration,  $T$  is assigned the average of the two means, resulting in a maximization of variance of the two classes once the algorithm has reached its end. Finally, the threshold  $T$  is applied to every pixel within the current window. This procedure is a fair compromise, overcoming deficits of the global thresholding but remaining computationally efficient.

### 3.1.4 Reference Lengths

Similar to other OMR algorithms, perfOMR heavily relies on a small set of precomputed numbers, which are referred to as *reference lengths*. This term seems appropriate as consecutive steps of the recognition process are performed based on these values. Given the nature of the algorithms chosen for symbol segmentation and identification, wrong reference values will negatively affect the final result considerably or even render impossible further computations.

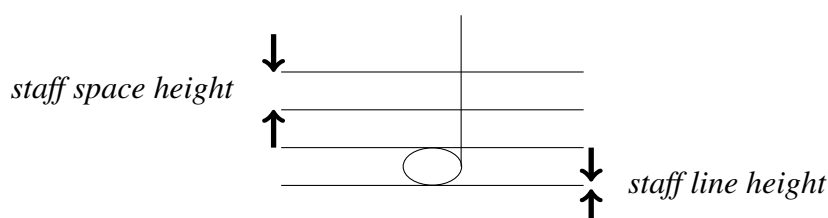


Figure 3.6: Definition of the two reference lengths relative to the model of a staff.

The two parameters which have proven to provide a robust foundation are *staff space height* and *staff line height*. While the first measure describes the distance in pixels between each of the four staff lines per stave, the latter stands for the thickness of staff lines (see Fig. 3.6). The importance of these values becomes self-evident once having delved deeper into the perfOMR system.

*Run-Length Encoding (RLE)*, a simple form of lossless coding, is useful for depicting sequences and has been applied in numerous OMR algorithms over the years. In order to derive the values for the reference lengths mentioned above, it is necessary to encode each column of the notation with RLE, resulting in a run-length matrix. Traversing through the elements, black and white runs are counted and the most-common ones

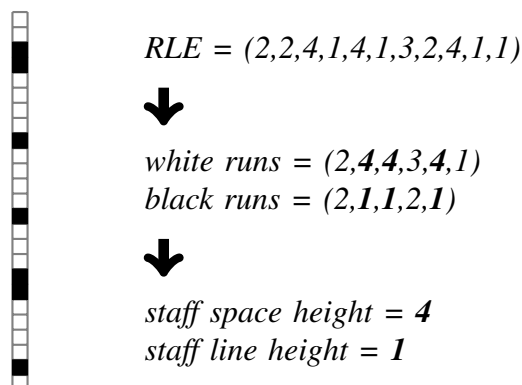


Figure 3.7: Reference lengths are obtained employing Run-Length Encoding.

are stored as *staff space height* and *staff line height*, respectively. For our data set of simple children's tunes, it was sufficient to take one centrally located column as a reference for computing the values. One could imagine, however, to integrate several randomly chosen columns into the encoding process, which would further improve results by providing a greater robustness. The algorithm is illustrated in Fig. 3.7

## 3.2 Staff Line Identification and Removal

Given a valid result (*staff height* > 0 AND *staff space* > *staff height*) from the measuring process, we are now in the position to detect staff lines, a critical task in music recognition, when it comes to isolating musical symbols. The position of staff lines can easily be derived from the maxima of the y-projection. I have mentioned earlier that systems have been designed that bypass staff removal. The underlying work seeks to reduce the amount of data in order to keep the computational cost for following stages low. Therefore, the strategy is to duplicate the image and to remove every item except for the staff lines. Eventually, the computed image  $A'$  can be XORed with the original image  $A$ .

### 3.2.1 Filtering

Since staves and staff lines are supposed to be horizontally aligned after the deskewing process, we want to remove components that exceed the height of staff lines. In this implementation, vertical black runs that are greater than twice the staff height are removed by a pixel tracking algorithm, while the vertical position of these segments being candidates for stems and bar lines is saved for later use.

After applying a horizontal projection we are now in the position to analyse connected components which is done by a labelling algorithm, encompassing two passes. Only columns with a projection value exceeding 0 are computed. Beginning with the



forward pass, traversing through the pixels of the image, for each black pixel, a search for neighbouring positives in a predefined area of preceding pixels is performed. In case there is no adjacent positive pixel, simply a new label  $L$  is assigned. In any other case, all black pixels in the neighbourhood, possibly containing different labels after the first assignment, are assigned the minimal label of this particular area.

After the backward pass which, in general, follows the same principle, connected components are uniquely identified by the according label  $L$ . Concurrently, for each connected component, the margins on both the x- and the y-axis are tracked in order to compute width and height of the respective labels. This allows us to remove all short components from the temporary image array whose vertical or horizontal span is below a predefined threshold  $T$  (see eq. 3.1). Let  $x_{\text{span}}$  be the margin on the x-axis and  $y_{\text{span}}$  be the computed margin on the y-axis. Furthermore,  $s_s$  denotes the staff space.

$$T = \begin{cases} 0 \dots x_{\text{span}} \geq s_s \parallel y_{\text{span}} \leq 2 * s_s \\ 1 \dots x_{\text{span}} < s_s \parallel y_{\text{span}} > 2 * s_s \end{cases} \quad (3.1)$$

In order to improve our preparatory parameters of staff width and staff space, two further runs of our run-length algorithm are applied. These are only separated by another filtering step that is introduced to remove remaining components taller than staff height. Given the exact positions of the five staff lines, which are basically the local maxima of a newly computed vertical projection profile, any fragments in-between these lines can be removed.

The final result of the process described is an image that should contain staff lines exclusively. As referred to earlier, an exclusive disjunction will remove the staff lines from the original image  $A$  by an *XOR* (see Fig.3.9). Pixels of the staff line matrix  $A'$  are subsequently removed from  $A$  if they are black. While the staves have been filtered out of the notation, their position is preserved as this information is crucial for the following tasks.

### 3.2.2 Stave Extraction

Having determined the position and number of staves, the staves which constitute the notation are separated. As for the feature classification, each stave is computed individually. In the introduction we have described a stave as a set of typically five parallel staff lines. Thus, the following extraction procedure has been adopted:

Iterating through all rows of the image, each y-position is checked for a staff flag. For each group of five staff lines, however, it is not sufficient to extract the area within the upper most and the lower most staff. Considering western music notation, notes and other symbols are not restricted to this area but they can appear, to some extent, in regions below or above the relevant stave. Therefore, the exact vertical centre points between two stave systems provide the intersection points between the strips that are to

be extracted. Any staff system counting less than five staff lines will be discarded while the other staves are forwarded to the classification process.

### 3.3 Feature Classification

Symbols are detected and classified based on template matching which works directly on image data. While there exist different styles of notations, let alone the variations in handwritten scores, western music notation follows a very similar scheme. For this work, the following subset of the Unicode Standard 6.2 [41] was chosen to represent the templates (see Fig. 3.8) These elements provide the basis of our symbol recognition step while further properties (most notably pitch) are derived from spatial data of the respective symbols in the image.

It has to be stated that in order to come up for a proportional mismatch between template and the object that is to be recognised (e.g. due to the resolution of the camera used), all templates need to be scaled. Let  $t_h$  be the height of the template provided. Both the values of staff space  $s_s$  and staff height  $s_h$ , which we already explained earlier, are used to compute the scale factor as follows:

$$s_f = \frac{2 * s_s + 2 * s_h}{t_h} \quad (3.2)$$

#### 3.3.1 Overview

Having been featured in many works of Optical Music Recognition, (non-deformable) template matching was chosen as key technique for this work. As opposed to a feature-based approach, the template-based approach operates directly on local image data. Instead of determining the best matching location in the image for each of the given templates, the prototype makes use of the information which has been aggregated so far. As template matching is known to be a rather costly method as opposed to the efficient feature-based approach, it is necessary to provide the smallest possible search window. The *Sum of absolute differences (SAD)* has proven to be a reliable metric for measuring the similarity between the image blocks. Given a search window  $W$  and the template  $T$ , the origin of  $T$  is shifted through the search image. Now, the absolute difference of intensity is computed for a pair of pixels  $w(x,y)$  and  $t(i,j)$  which is defined as:

#### 3.3.2 Implementation

Rather than computing the best matching position of the pattern, a matching coefficient  $C$  is computed based on the SAD and template size  $s$  (see Eq. 3.3). If this error measure










Code	Character	Description
1D11E		treble clef
1D15D		semibreve
1D15E		minim
1D15F		crotchet
1D13C		minim rest
1D13D		crotchet rest
266D		flat
266E		natural
266F		sharp

Figure 3.8: Templates chosen for feature classification

is lower than the predefined threshold, a match has been detected for the appropriate symbol. In order to reduce the computational effort, the matching of a template on the respective position is stopped as soon as the error measure becomes too high and a match is not possible anymore. Remaining pixels are skipped and similarity measurement continues after template T has been shifted to the next position. For each of the different symbols detected, an individual procedure is started after a match has been detected.

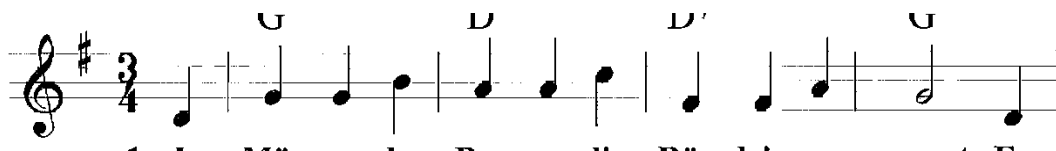


Figure 3.9: First staff of our sample tune after staff line removal.



Figure 3.10: Vertical projection of Fig.3.9 with spikes indicating possible locations of bar lines or note stems.

$$SAD(x, y) = \sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} |w(x + i, y + j) - t(i, j)| \quad (3.3)$$

As stated earlier, the feature classification process is based on isolated strips of the image, comprising a single staff and its symbols that are subject to the recognition step. For each of these image subsets, a run of the method *identifySymbols* is performed. Also, it is characteristic to the system that it relies heavily upon the reference lengths of *Staff space Height* and *Staff line Height* that were computed at the end of the image preprocessing step.

In order to be able to identify individual symbols, it is necessary to locate candidates which is why a projection onto the x-axis is performed. The resulting spikes (see Fig.3.10) give a good indication on the possible location of symbols which helps to rule out regions with a low profile value for the subsequent matching process. Thereby, feasibility of the system is ensured.

### 3.3.3 Clefs

We have already learned that clefs, which indicate the pitch and thus the name of written notes, come in different fashions. Since this work focuses on the recognition of children's tunes, it is freely assumed that a treble or G-clef is present at the beginning of a staff. Furthermore, since staves are perfectly aligned vertically and the type of clef is not subject to change at some point in the musical score, a computation only of the very first staff is sufficient.

Compared to other symbols, the complex appearance of a clef provides a high degree of difficulty for our template matching approach since only minor dissimilarities in shape severely affect the outcome of the matching procedure significantly. Since clefs

commonly have the property of being the first large symbol on a stave, matching with a dedicated template has not been found a viable solution. Instead, the projection profile is used to determine the position of the clef. For that matter, a lookup algorithm is performed within a predefined area, looking for a sequence of positions where the projection value exceeds a threshold.

In case of no clef symbol being detected by the algorithm, the note detection process, which is referred to below, is started at the leftmost position of the extracted stave.

Listing 3.2: Note matching

---

```

for each note type
  find note stems on projection peaks
  apply template matching algorithm to lookup areas (between/on staff lines,on
  supplementary lines)
  scale template according to the template_height : staff_space ratio
  while (error < abort_threshold && not_finished)
    accumulate matching error (Math.abs(value(Image)-value(Template)))
  end
  if error_coefficient < matching_threshold
    note candidate found, save coordinates to notes[][]
    track and remove note stem from image
    hook/beam detection by line tracking:
    if vertical_travel > 1.5*staff_space
      hook detected
    else if (horizontal_travel > 3*staff_space
      beam detected
    end
  end
  end
  find prolongation dots
  derive midi symbol from pitch and duration of note
  save to symbols list
end

```

---

### 3.3.4 Notes

As indicated earlier, notes, in most cases, consist of both note heads and stems which they are connected to. Let us assume that our notation sheet is perfectly aligned in a way that stems stand perpendicular to the x-axis of the ground image. In that case, the x-projection shows a significant peak in the area of the note stem. This very position is taken as a basis for the note lookup algorithm (see Lst. 3.2)

Among the different types of note heads, those of quarter, eight or sixteenth notes are the easiest to match. Due to their property of being filled as opposed to half or full notes which have a hollow shape, their recognition is relatively robust against rotation or deformation of the input image. Furthermore, mismatches between the template chosen and the actual appearance of the symbols used in the notation have no greater impact. In our algorithm, however, we differentiate between three types of note heads. While there are only two forms of appearance, namely full and hollow, in first place, half and full notes differ from each other by the thickness of the border. Thus, the algorithm is run exactly three times per staff.

Throughout this work, it has been the main strategy to minimize the lookup regions as much as possible in order to maintain computational efficiency. At the expense of accuracy, an approach was chosen which decreases the number of note candidates in every iteration.

As an input, a previously extracted strip consisting of a set of staff lines, a fixed threshold, note templates and the result of the x-projection are used among other vari-

ables. First, note candidates are derived out of local peaks in our projection if the predefined threshold is exceeded. A number of  $3*staffspace$  has proven to be a good value to locate note stems. Since a note head is necessarily attached to a stem, we are now in the position to perform a template matching for which an area spanning to the left and to the right of the stem by  $2*staffspace$  has to be matched with the template  $T$ . For classification, the *Sum of absolute differences (SAD)* metric, which we have already referred to, is used. If the sum of errors exceeds a predefined value, the matching is aborted prematurely and the search window is shifted to another position.

Let us now assume that our template matching algorithm was allowed to finish on a given search window. In that case, a correlation coefficient is computed by dividing the SAD by the template size. If this coefficient is low enough, a match has been obtained and the location is saved along with the value (e.g. full, half) and the dimensions of the note head. Concludingly, both the note head and the adjacent stem are removed from the original image. From the dimensions of the search window that has led to the match, the centre of the note head can be derived which is needed as an input for the operations that are described below.

### Stem, beam and hook detection

We have already learned that the properties of a given note are not affected by the note head alone. While adjacent features like accidentals lead to a change in pitch, beams and hooks attribute to the temporal properties. Since hooks or beams extend from the upper or the lower end of a stem (depending on the location of the note head), the stem needs to be traversed starting from the position where it is connected to the note head. In our case, a stem is detected if the number of black pixels tracked vertically equals or extends  $2*staffspace$ .

The endpoint features as a seed point for a tracking algorithm similar to the one that was introduced by [42] for detecting beams and hooks. Hereby, the middle points of the vertical runlength measure is tracked by computing both the “north” and “south” offset and dividing the difference by two. In order to compensate for imperfections of the input image or for gaps caused by the earlier process of stave extraction, runlength measuring ignores areas without foreground pixels  $< 2*staffspace$ . Hence, the tracking is resumed at a position that is shifted up or down by this resulting value (see Fig. 3.11). For children’s tunes, a differentiation by the extent of their horizontal (beam) or vertical (hook) travel was sufficient. A vertical travel  $> 1.5*staffspace$  indicates a hook while a horizontal travel  $> 3*staffspace$  identifies the element as a beam.

It is a property of both beam and hook that the duration of the attached note is divided by  $2^n$  where  $n$  stands for the number of detected elements per note. If a beam is present, this effect is propagated to the following note, reducing its duration value to the same extent.

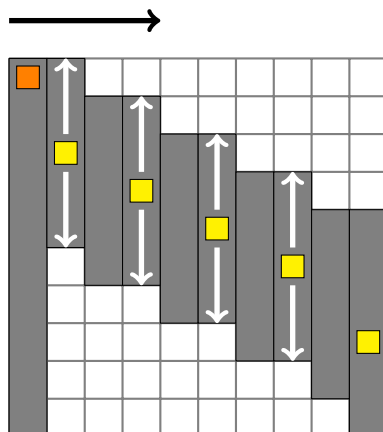


Figure 3.11: Detection of beams through a line tracking algorithm with the seed point coloured in orange. For each position  $x$ , the new adjustment parameter is computed. On the left, a segment of a note stem can be seen to where the beam candidate is connected to.

With simple children’s tunes being the scope of this thesis, recognition of notes shorter than sixteenth notes was not implemented. Only hooks or beams up to a number of two per single note are detected.

### 3.3.5 Key signatures

Commonly, a key signature—if present at all—is the very first item following a clef viewing the notation from left to right. Occuring either as a series of up to a total of seven sharp and/or flat accidentals, their vertical locations affect the tonal properties of many notes due to their scope. For this reason, the identification of key signatures is an essential task for preserving the musical sensation intended by the author of a tune. While the key of a song can be subject to change in certain musical pieces, e.g. from G major to D minor, this special case is disregarded in the present work. Furthermore, much like during the location of the clef, only the first stave of a page is computed.

In contrast to accidentals, which might occur at any location in the score, the matching of key signatures is restricted to a narrow area near the clef, bound by the first occurrence of a note on its right side. Therefore, this step is logically placed after note detection but for reasons of clarity and comprehensibility, I am providing deeper insight into the algorithm here. In order to differentiate between sharp and flat accidentals, an approach based on template-matching, similar to [15] was chosen. As pointed out early, there is some degree of variance even in western music notation. Therefore, an accurate detection of certain symbols cannot be guaranteed by matching of one template per symbol alone. For this difficult task, this work employs the familiar concept of pro-





Figure 3.12: Flat (one peak) and sharp (two peaks) accidentals can be derived from the number of peaks in their vertical projection.

jections which allows for distinguishing between the two prominent accidentals by the number of vertical peaks.

In [42], the authors made use of template matching to detect key signatures or accidentals. To account for different shapes of primitives among scores of different heritage, templates are made from primitives in the score. This work, however, makes use of the relatively convenient and fast projection-based approach. For the main purpose of this thesis, this technique has provided good results.

Except for a time signature, accidentals are the only symbols located in the space between clefs and notes. We perform a search at each horizontal position  $i$  on the interval given by the specified boundaries. If the local value on the x-projection exceeds a threshold, zero-crossing detection is run in order to determine both the local minima and maxima. Based on these values, the number of peaks in this area can be derived, from which we can deduct whether a flat (one peak) or a sharp (two peaks) is present.

Since changes of tonality occur at a much lesser extent in children's tunes compared to other musical pieces, the algorithm was not developed to provide means of distinguishing a sharp from a natural symbol. The detection algorithm resumes until a vertical blank space  $> 1.5 * staffspace$  has been processed. The result of this operation is forwarded to the semantics stage.

### 3.3.6 Accidental detection

While both key signatures and accidentals share the very same appearance, they are distinguished by their position. The occurrence of accidentals (for reasons of simplicity, only sharp, flat and natural are considered) is coupled with note heads.

Basically, an accidental is located within a certain area next to a note head. There-

fore, it is reasonable to restrict the lookup algorithm to relatively small fields. As we have already determined the positions of the note heads, they provide us with a good starting point for the matching process. For each note, a window slightly larger in both height and width denotes the matching area. A height of  $2.5 * \text{staffspace}$  and a width of  $\text{staffspace}$  has delivered accurate results. Considering that the appearance of all accidentals shows a distinct height of around two times the staff height, knowing the projection profile helps us minimizing the computational effort. Consequently, we only need to perform the lookup algorithm if the projection of any x-coordinate within the defined region equals or exceeds the pre-defined threshold. The core of the matching algorithm follows the scheme for key signatures, which was presented earlier.

### 3.3.7 Detection of other elements

While we have already referred to the main musical characters above, a musical sheet written in modern staff notation is made up of many other elements. Symbols affecting matters such as dynamics, expression and tempo are very common in more elaborate musical pieces but as they rarely play a role in children's tunes, their detection was not taken care of in the development process.

#### Prolongation dots

Aside from accidentals, there are other elements that are not directly attached to a stem or note head but which affect the properties of the note next to it. Dots are very common symbols in musical notation, increasing the duration of the note to its left by one half of its original value. For every note that was recognized in an earlier stage, an algorithm is run to identify prolongation dots. Since these elements regularly occur in a certain region beneath the note head, the search window is allowed to remain relatively small. Its horizontal extent measures  $1.5 * \text{staffspace}$  with the window starting at a distance of  $0.5 * \text{staffspace}$  from the right edge of the previously located note head. Likewise, upper and lower border are directly affected by the value of the aforementioned reference lengths. We are now taking into consideration that ideally, the center prolongation dot is placed at the same y-coordinate as the center of the note head. This allows us to shorten the vertical extent of the search window on the upper and on the lower boundary by exactly  $0.5 * \text{staff height}$  compared to the area that is spanned by the note head.

The template matching process either returns a positive result—in case there is a sufficiently high vote for a dot candidate—or a negative one if no match has been found. A prolongation dot extends the duration of a note by half of its original value.

## Rests

Musical pieces of a higher complexity than those covered by this thesis may or may not show a multitude of different manifestations of symbols given a certain class. As we have already learned, this applies to notes and accidentals but also to the class of rests. This module is carried out at the end of the symbol recognition framework with staff lines, notes and accidentals having already been eliminated from the scoresheet. By the updated projection profile we obtain the positions of candidates which, in turn, are tested for occurrences of rests. For each of the different types, a separate recognition process is run with the appropriate template. The actual matching follows the same scheme as presented earlier when matching note heads.

As opposed to notes, where the difference in shape is minimal, the three most common types of rests (minim, crotchet, quaver) are of a distinct appearance. Therefore, good recognition results have been obtained by the framework.

## Bar lines

Bar lines are vertical segments that separate a staff into short blocks. As a key property, the number of beats within each pair of two bar lines matches the top number of the time signature that is denoted at the beginning of the musical piece. The special case of a time change at some point in the musical piece was not taken care of in this thesis but future work could see optimizations in this area. There are different types of bar lines of which the standard and double line are of lesser interest to us. Thicker vertical lines, however, not only indicate the start or the end of a piece of music. Coupled with a repeat sign, which is made up of two dots that stand on top of each other, a bar indicates that a certain section has to be repeated. The boundaries of this section are denoted by the repeat sign both at the end and at the start.

Listing 3.3: Bar line extraction

---

```
for each staff
  compute projection profile (vertical)
  for x = 0 : width;
    if profile(x) > 4*staffspace
      bar line candidate found
      remove bar line from image
      save position to boundaries[]
    end
  end
end
return boundaries[]
```

---

Locating bars as such is a straight-forward process that only involves computing the vertical projection of a staff (see Listing 3.3) If, at some point in the notation, the threshold is surpassed in consecutive x-coordinates ( $0.5 * \text{staff space}$  would be a suitable value), one can safely assume that a bar line has occurred. A disambiguation between

note stems and bar lines due to their similar vertical extent is prevented by removing notes and their adjacent stems from the music sheet after each iteration of the note matching algorithm.

In some cases, bar lines span across several staves which is typical to music scores with at least two voices or instruments. Assuming that a bar line of this size was detected in a music sheet, the two or more staves involved are grouped by the bar, meaning that the music that is contained herein will not be played consecutively but simultaneously.

## 3.4 Musical Semantics and Transformation

### 3.4.1 Symbol Interpretation

As a final step in our music recognition environment, musical semantics are extracted from the symbols that were detected in the previous stage. We have made it clear earlier that the position of symbols is important in order to correctly interpret a given music score. Moreover, symbols must not be seen isolated but meaning is affected by the relationship between individual shapes. Many works are based on grammars that synthesize symbols into more complex forms. While works like [35] referred to a graph-based grammar, this thesis is based on a low-level structure.

The simplicity and the resulting low density of childrens' tunes allow for breaking down the complexity of music recognition and interpretation into a one-dimensional scheme: Every symbol detected is stored in an array, using their horizontal position as an index. As illustrated above, vital contextual information (vertical position, pitch and duration) is part of each data record which we refer to as *Symbol* in the following. Since music will be interpreted serially, elements in the database are initially sorted by their position on the x-axis. In order to derive the pitch of a certain note, its vertical position relative to the staff lines needs to be analyzed. This will be discussed in detail in the following subsection.

#### Key

While not needed in the process of converting optical sources of music into their audible representations, the key of a given song can be derived from the previous process of symbol recognition. As described earlier, each piece of music is written in a particular key. While more elaborate music like classical music quite often has different sections with contrasting keys, children's tunes commonly are based on exactly one key. In the process, we need to distinguish between *major* and *minor* keys, that are commonly referred to as "happy" or "sad" keys as the structure of the scale affects the emotion of a musical piece.

Having obtained the key signature of our song in a previous step, we can easily determine the key, applying a rule-based system, which is described in Fig.3.13


Figure 3.13: Obtaining the key from the number of sharps and flats in the key signature.

Number of $\sharp$	Number of $\flat$	Key (major / minor)
-	-	C / a
1	-	G / e
2	-	D / b
3	-	A / f $\sharp$
4	-	E / c $\sharp$
5	-	B / g $\sharp$
6	-	F $\sharp$ / d $\sharp$
7	-	C $\sharp$ / a $\sharp$
-	1	F / d
-	2	B $\flat$ / g
-	3	E $\flat$ / c
-	4	A $\flat$ / f
-	5	D $\flat$ / b $\flat$
-	6	G $\flat$ / e $\flat$
-	7	C $\flat$ / a $\flat$

### 3.4.2 Audio Synthesis

#### The MIDI Protocol

Being widely used on various platforms and applications, the multi-purpose *Musical Instrument Digital Interface (MIDI)* protocol has proven to be a viable route for storing and interpreting musical information for this prototype. As opposed to a plain audio format, MIDI enables the developer/user to edit notational information at a later point. Moreover, the amount of data is a magnitude lower than even a compressed audio file. Several operating systems for mobile phones provide means of interpreting MIDI com-



Command	$\Delta$	Velocity
note on	0	127
note off	16	127
note on	32	127

Figure 3.14: Example of two quarter notes, separated by a quarter rest in Western Music Notation (left) and its pseudo-MIDI representation (right)

mands and of course, Android OS is no exception. The task of audio synthesis from MIDI, however, was complicated by the developers' choice to remove the appropriate *javax.sound.midi* library from the android platform, making "live" synthesis of MIDI commands impossible. There was an attempt to solve this issue by *android-midi-lib*, a Java implementation of the MIDI file format, which unfortunately was not available to programmers during the time of development of this prototype.

Above we have already referred to the *Symbol* class as means of temporarily storing music information and this construct was chosen with regards to the structure and functionality of MIDI. This protocol is roughly built around events for the playback of individual notes. For this prototype, the two commands of *note-on* and *note-off* have been of particular interest. Contrary to the *Symbol* class, however, a 3-byte MIDI dataset does not include *duration* as a property. Non-intuitively, MIDI is based on the *on-time* and *off-time* values instead of *duration* that might be more familiar through the different note types. While musicians refer to *tempo* as number of quarter notes played every minute or *beats per minute (BPM)*, MIDI features the opposite approach. Tempo, as understood by MIDI, is defined as *amount of microseconds per quarter note* and is processed in commands of three bytes. Consequently, a "natural" tempo of 120 BPM translates to a MIDI tempo of 500,000 ms per quarter note.

Another value to be aware of when dealing with MIDI commands is  $\Delta$  which is referred to as time between two individual events measured in timebase ticks. While a *rest* symbol is an explicit way of describing a pause between two notes in musical notation, this information is conveyed in an implicit way in MIDI commands. A  $\Delta=0$  means that a note is played at the next possible time after the previous MIDI event. If, after a *note off* command with  $\Delta=16$ , the following *note on* command is accompanied by  $\Delta=32$ , it implicitly mimicks a quarter rest between these two notes (see Fig. 3.14).

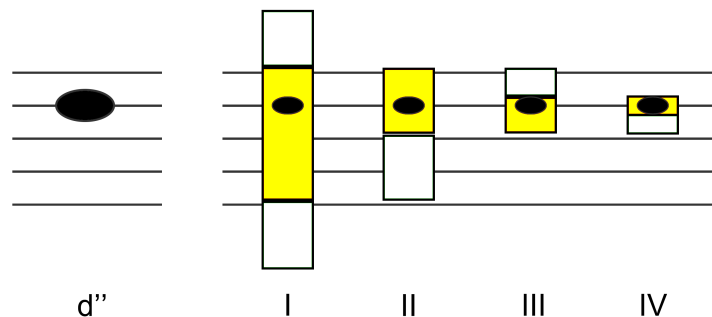


Figure 3.15: Determining pitch from the vertical position relative to the staff lines by pseudo-binary search.

### OMR-to-MIDI Mapping

Present thesis incorporates a two-tiered method of audio synthesis. First, each of the above mentioned symbols has to be translated into MIDI commands. The staff extraction algorithm has provided us with the positions of the staff lines which play a crucial role in determining the pitch of a note. For each of the notes detected, the y-coordinate is available through the *Symbol* class.

In order to determine the pitch from the positional values, a *pseudo-binary search* algorithm was invented (see Fig. 3.15). In the first step (I), we check whether or not the note lies in between the top-most and the lower-most staff line. If that applies, the search window is further divided and the algorithm is resumed in the respective subframe (II-IV). If in (I) a negative response was obtained, the area above the first and below the lowest staff line needs to be processed. Depending on the region of the staff the center of a note is located in, a distinct MIDI note value is returned by the method. Likewise, the duration of a singular note needs to be translated. In the previous section we have already learned that, according to the note type that was detected (quarter, half, whole, quaver or semiquaver) and considering the occurrence of symbols that have an effect on the duration of a note (hooks, beams, prolongation dots), a distinct duration value is saved.

The actual OMR-to-MIDI translation used in this prototype was developed without the aid of the Java Sound API [7]. It does, however, provide a more intuitive way of inserting notes in MIDI files through which the complicated use of delta values is bypassed (see Lst.3.4 and 3.5). Eventually, the recorded MIDI commands are written to a MIDI file, which in turn is played back by the Android Media Player.

---

Listing 3.4: Sample MIDI command with the Midifile.class

---

```
mf.noteOnOffNow ((int)(sym.getDuration()*0.66), pitch+12, 127);
```

---

---

Listing 3.5: Sample MIDI command with rest - velocity is set to zero

---

```
mf.noteOnOffNow ((int)(sym.getDuration()*0.66), 60, 0);
```

---





# Implementation

## 4.1 Platform and tools

The prototype described in this thesis was developed for the Android operating system. Not only the total number of handheld devices being based on Android has seen a continuous rise since its introduction in 2007, but also low-end devices have more and more become equipped with the open source operating system. This becomes important considering that the software presented mainly addresses non-professionals. Providing comprehensive tools for developers, Android OS was found suitable to serve as a basis for our OMR system.

During development of the prototype, the Eclipse IDE was used as the main environment while the necessary API libraries and developer tools were provided by the Android SDK (22.3 at the time of development). Complete testing and evaluation was carried out on Android 4.4.2 (API 19). Throughout the development process, runs of the program were performed on a mobile phone (see below).

## 4.2 Hardware

The handset used during the later part of development and testing is a Sony Xperia ZL (C6503) smartphone that features a Quad-core (1.5 GHz) processor and 2GB of RAM. The prototype has been targeted for Android version 4.0 and later and likewise, testing was carried out on several other handsets that revert to recent versions (API levels 15-19).

In order to obtain satisfactory results, pictures need to be taken at a minimum resolution of 300 dpi [15]. Given the size of a common notation sheet (DIN A4, 8.27 x 11.69”), we would need a camera with a capability to work at about 9 megapixels.

However, at the time of writing, this was not feasible in the assumed environment due to hardware - most of all, memory - limitations.

## 4.3 perfOMR

In the following, *perfOMR* denotes the software prototype that was developed over the period of the thesis. Focusing on providing a quick and logical means of sheet music detection and audio synthesis, the author came up with the idea of naming the prototype *perfOMR*, a composition of the word *perfect* and the acronym for optical music recognition, *OMR*. *perfOMR* onomatopoeically resembles the noun *performer*, which, in turn, is a reference to the feature of music playback.

### 4.3.1 User Interface

Targeted at a non-professional audience, user interface development presented a couple of challenges. Generally speaking, focus was on a functional interface that does not provide for means of altering settings (these are hard-coded), leaving the end user with the options presented in Fig. 4.1.

Regarding the origin of the music sheet that is to be processed, we need to distinguish between two main sources: a) a file that is already stored in the local file system or b) capturing an image through the built-in camera. Eventually, information on the musical piece that was obtained during the processing of the notation is displayed (see Fig.4.2). Above the playback controls, a staff is displayed for debugging purposes. A more elaborate (graphical) user interface is outside of the scope of this thesis.

### 4.3.2 Symbol templates

As proposed earlier, *perfOMR* reverts to pre-defined symbol templates during template matching. While logically, performance of the system suffers in case of slightly diverse appearances of the respective symbols, this approach provides a decent tradeoff between implementational and computational effort on the one hand, and performance on the other hand. Present work uses characters contained in the Unicode Standard, Version 6.3. Symbols were extracted from the template, then cropped and resized (e.g. 26x22 for the quarter note head, 36x22 for the full note head, 22x22 for the dot symbol). For a comprehensive list of symbols see Figure 3.8.

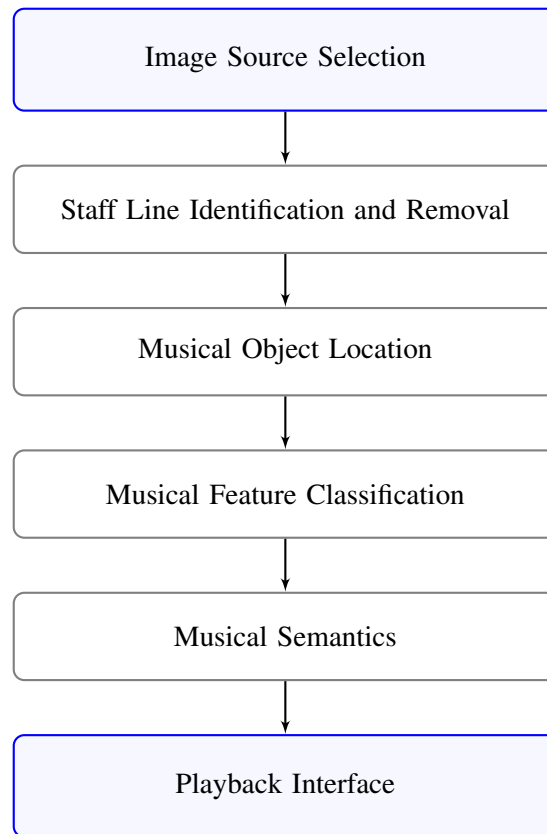


Figure 4.1: Control flow in *perfOMR* - only first and last stage are accessible through the user interface.

### 4.3.3 Helper Classes

While the core of algorithms used by the software was invented and implemented by the author, the prototype makes use of helper classes and external libraries, which will be explained in the following.

#### Android

The routine for providing the input image makes use of methods inherent to the Android Framework. As described earlier, we need to distinguish two cases: capturing photos through the in-built camera or selecting a locally stored image. In order to allow our prototype to take a photo with the in-built camera, it is required to set the corresponding permission tag in Android's manifest file (see Lst. 4.1).

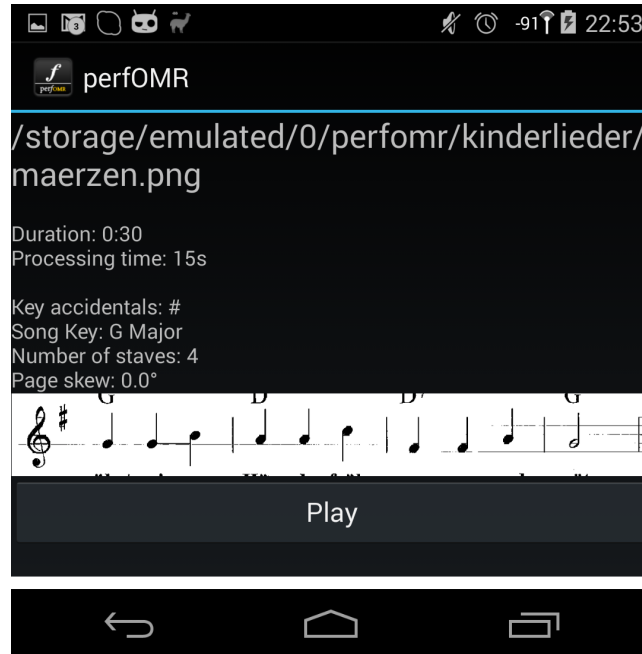


Figure 4.2: Output of the recognition process (simplified): The user is provided with information on the notation processed and the playback interface.

#### Listing 4.1: Setting Camera permissions in AndroidManifest.xml

---

```

<manifest ...>
  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="19"/>
  <uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>

```

---

In the following, a new `Android.Provider.MediaStore.ACTION_IMAGE_CAPTURE` intent is created which calls the appropriate Android activity. As a result, the requested image is returned and then forwarded to the recognition routine by invoking a new intent that encapsulates the appropriate class. Likewise, the same intent is launched in case of the user choosing an image stored in the file system of the device. It needs, however, to be stated that only a reference to the respective image by means of a *Uniform Resource Identifier (URI)* is used within the new intent call. Another prominent call involving the `Android.Media.MediaPlayer` class is described in the following subsection.

## OpenCV

OpenCV (Open Source Computer Vision Library) has proven beneficial during the development process due to its emphasis on image processing and computer vision. The version of the library used during the development of the prototype (2.3.1) enables development of applications on most major platforms including Windows, Linux, MacOS and iOS. Use on the Android OS is facilitated by a customised OpenCV4Android SDK.

During the image preprocessing stage, as described in Section 3.1, methods for image conversion, binarisation and contour finding as well as methods carrying out morphological operations represent the main part of the image cropping algorithm.

## MidiFile

As we have laid out in detail earlier, Android currently does not provide support for dynamically generating and playing MIDI commands at the same time. Although there are stand-alone libraries (e.g. `android-midi-lib`), actual audio playback or device interfacing has not been implemented. This issue has been tackled by dynamically writing commands to a MIDI file that is then written to the device storage.

The `MidiFile` class [7] was used for setting up a new MIDI file, including file header, footer and a `Vector` of play events where the previously detected notes are stored into (see Listings 3.4 and 3.5).

### 4.3.4 Audio Rendering

Having a MIDI file at disposal, we are left with the task of playing back the result of the aforementioned stages of the OMR framework. In detail, the file now serves as `FileInputStream` for our `MediaPlayer` object. Audio feedback is then provided using a familiar layout, allowing the user to pause and resume play at any moment or at any position. Apart from the purpose described, the file may be used as an input for digital audio workstations or music notation programs, taking advantage of the communication interfaces provided by the Android OS.

### 4.3.5 Bugs and Limitations

Firstly, we shall refer to errors of the software by a strictly technical means - shortcomings that are related to choice of algorithms will be discussed subsequently.

#### a) Android Gallery

As per Android version 4.4.1 (API Level 19), Google introduced several new features, which not only comprise camera enhancement for several devices but which also affect handling of photos and images. As we have already learned, already

taken images as well as the built-in camera feature as sources for the OMR process. While aforementioned changes in Android OS left the workflow of taking photographs of notation sheets and forwarding them to the recognition engine untouched, we encounter a difficulty when choosing a locally stored notation sheet as input. Currently, Android 4.4.1 provides a coexistence of the newly integrated *Google Photos* application and the long-known *Gallery* application. When choosing a local file as image source in perfOMR via *Photos*, perfOMR crashes unexpectedly when operated on 4.4.1 or above. As a workaround, the user must enter the *Gallery* from the drop-down menu in the top left corner, selecting the notation sheet in the following step.

While this thesis is concerned with how to make use of existent techniques in order to develop performant OMR systems under various conditions, abstraction was chosen for the process of prototype development. The sample notations as shown in 5.2 do not only serve as a test set but also as a reference throughout the process of development. Functions necessary for processing sample notations were implemented while other, more complex features are subject to future work. As a consequence, the following issues arise when providing material with different properties to the recognition algorithm:

**b) Region of interest**

Section 3.1 has given insight in the approach used for extracting a region of interest (ROI) from a given image. The cropping algorithm used fails on complex images, defining only a subregion of the notation sheet as region of interest. A possible step to overcome this deficiency is to introduce a degree of interactivity. An adapted interface could allow the user to point at the region showing the notation sheet or even to define the boundaries of the ROI.

**c) Notation**

Even though there is an understanding that Western Standard Notation is the common form of graphically depicting music, there are minor differences as we have already mentioned in Section 2.1. This presents challenges to the way that templates are matched with the ground data. Focusing on SAD metrics and given templates, means are not provided to accommodate the algorithm in order to work with considerable changes in appearance of music symbols, let alone handwritten notes. See [34] for an extensive analysis of elaborate algorithms.

**d) Polyphony**

Due to the specialisation on simple children's tunes, it was obvious to have the software recognise each sequence of staves as one line of a song. Given a notation sheet that not only provides for one but for two or more voices/instruments, the algorithm will recognise each subsystem as subsequent lines. Hence, the melody, which is intended to be polyphonic, will be played as a sequence instead of simultaneously. In order to deal with this issue, we could introduce an algorithm that measures the gap between each set of staves in order to derive related sets of different voices. Another feature of related staves is the property of having barlines reaching from the top of the first staff to the lowest staff line of the staff that depicts the last voice.

Another way of representing a polyphonic musical piece where the respective voices are identical in phrasing is to attach more than one note head to a stem. That particular case sees the detection algorithm recognising the leftmost note that is connected to the stem while skipping recognition on the other side of the stem. Consequently, the algorithm design would need to be adjusted in order to satisfy the needs of such types of notation.

**e) Clefs**

Children's tunes, or more generally speaking, notation for vocal music, commonly has a G- or treble clef indicating the pitch of the notes placed on the staff lines. Yet, the prototype, which we refer to herein, does not accommodate for special cases (e.g. B- or bass clef). Consequently, this results in a wrong interpretation of pitch whereas temporal attributes of the sheet music are not adversely affected.

**f) Measure length**

Similar to the issue quoted above, the problem of note and pause lengths within a given measure and thus, the correct number of beats per interval, can be tackled by incorporating additional semantics. Since time signature is not derived through template matching but only intrinsically, the majority of measures could be used to determine the number of ticks per measure. Yet, current implementation does not provide for intelligent means for automatic correction of note lengths or likewise, an introduction of pauses in cases when the total length of notes and pauses within a measure does not match the time signature. Consequently, as only notes or pauses that have distinctively been recognised by the algorithm are written to the MIDI file and played back, variations in measure length are effectively ignored.



### 4.3.6 UML

Using Unified Modeling Language (UML) as a tool for visually mapping the software design, the project can be roughly divided up into the three main classes described below. For a detailed illustration of classes present by means of a UML diagram see Fig. 4.3. Attributes have been omitted in order to ensure readability.

#### a) **HelloAndroid**

Providing for an entry point to the software prototype, this class bears similarities to the tutorial published on the Android developers' page. The main layout is created with reference to a previously specified layout resource that defines the architecture for the user interface in an Android Activity. Generally, user action is coupled with the buttons available. Here, listener methods launch the respective `Views` on the respective user command.

Logically, the user is allowed to either load an image from memory, take a photograph using the in-built camera or, as the final option, exit the program. If the user chooses to take a picture himself, `MediaStore.ACTION_IMAGE_CAPTURE` is launched, leading the user to the standard Android camera interface. After the image has been taken, the appropriate `Intent` is created and the *Uniform Resource Identifier (URI)* of the image chosen is stored inside the `Intent` through its `putExtra()` method. Subsequently, a new `Activity` is started by passing the `Intent` described above.

#### b) **Precomp**

After an instance of `Precomp` has been created the extra data are queried by the new `Activity` and used to load the image into a `Bitmap`. As a final preparatory step, the correct rotation value is retrieved from the *EXIF orientation tag*, thus leading to an optional rotation of the input image. If locally stored images serve as foundation for the music recognition algorithm, `Intent.ACTION_GET_CONTENT` is called through a new `Activity`. This enables the user to select a particular image, which, in turn, is retrieved via the `getBitmap` method, using the resource identifier that was returned by the activity.

At this point, the input bitmap is forwarded to the preprocessing stage as described in Section 3.1. In case of a positive result (i.e. if staves have been detected by the engine), staves are then forwarded to the recognition engine via the `computeStaves` and `identifySymbols` methods. The latter can be described as the core of the framework, retrieving possible locations of symbols through projections and detecting notes by the means of the `matchNotes` method, which is called independently for each of the different note lengths. Templates `t` are loaded by instancing

the `Templates` class. The result of the template matching algorithm is stored into `ArrayList<Symbol>` after the correct MIDI note values have been retrieved through `getNoteMIDIVal`. Here, location values (position on the y-axis relative to the staff) are matched with the according MIDI note value. After the MIDI file has been compiled (see below), an instance of `android.media.MediaPlayer` provides for the playback routine.

During all of the aforementioned processes, a `Handler` provides the user with feedback regarding the current stage, with error messages describing the issue, and, eventually, the result in terms of information on the notation processed.

#### c) **MidiFile**

In order to provide for auditive feedback, the symbols recognised during the recognition stages are sequentially used in a `MidiFile.noteOnOffNow` method, which results in a collection of MIDI commands that finally is written to a MIDI file in `MidiFile.writeToFile`.

## 4.4 License

The program `perfOMR` is free software. Redistribution and/or modification is therefore allowed under the terms of the GNU General Public License, Version 3 (GPLv3), as published by the Free Software Foundation. Source code can be obtained from: `git@github.com:hiace/performr.git`. A copy of the GNU General Public License is available under <http://www.gnu.org/licenses>.

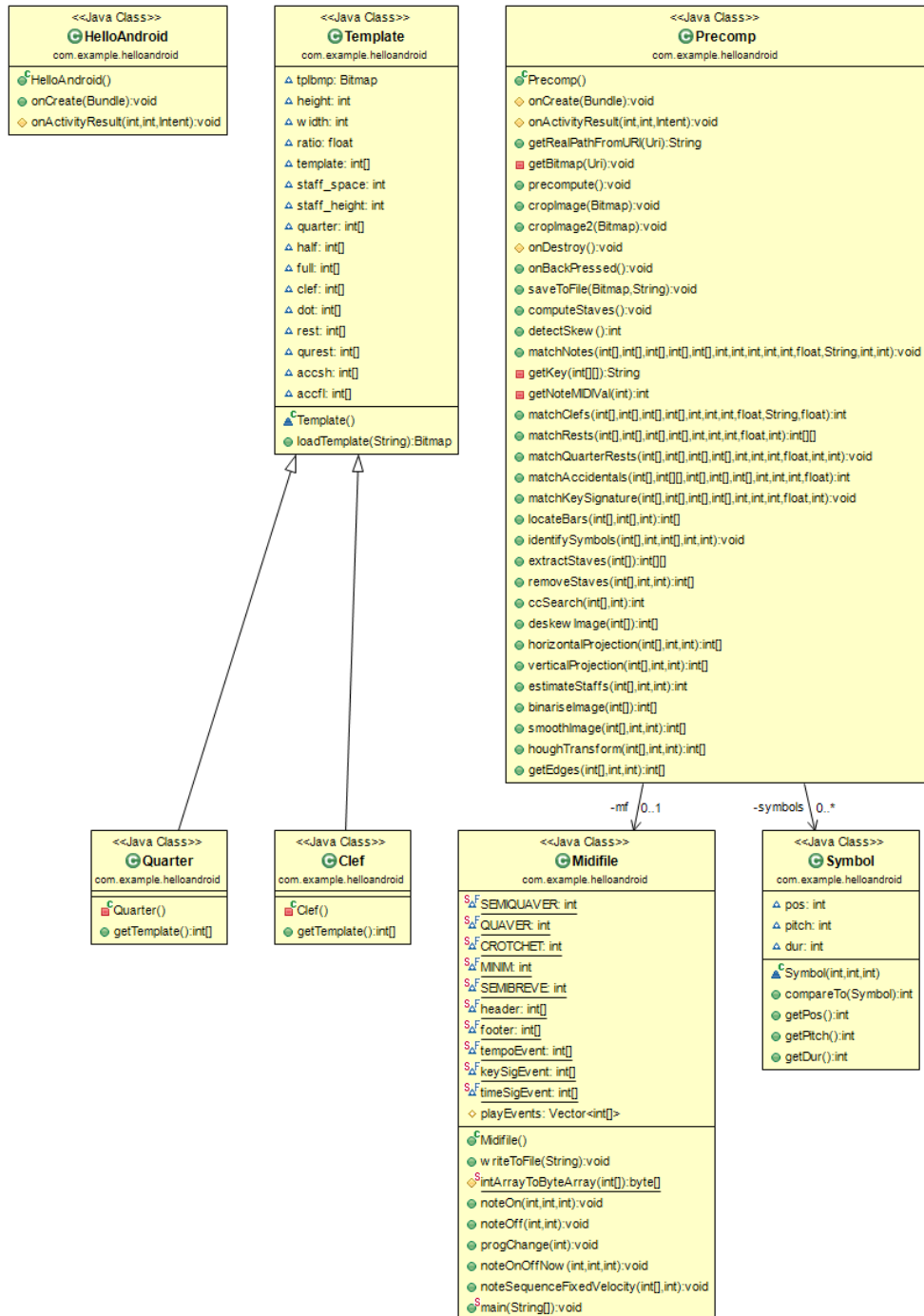


Figure 4.3: UML structure diagram of the software prototype.

# Evaluation

## 5.1 Methodology and Metrics

As stated in [11], the lack of a standard in measuring makes it difficult to compare different OMR systems with regard to their performance. Observing that the appearance of the very same music symbol can be subject to change when dealing with different music sheets underlines the aforementioned thesis. This chapter deals with a description of the methods used to evaluate perfOMR and it will provide an in-depth view on overall performance and accuracy of the prototype.

Testing the results of OMR against ground-truth data can be a highly time-consuming task, let alone the difficulty of finding a proper means of comparing this data. An interesting approach was presented in [40] who addresses the problem by measuring differences between the reference score and the tested score, while both objects are stored in the MusicXML format (see Sec.2.3.3). Since this project primarily serves scientific purposes, this section does not comprise a benchmark test of perfOMR against other systems, be they scientific or commercial. Instead, we will focus on how variations of the input material affect the output.

In the field of pattern recognition and classification, the following metrics are commonly defined:

- true positives (TP)
- false positives (FP)
- false negatives (FN)
- true negatives (TN)

We define *TP* as symbols that were correctly classified as belonging to a particular class (e.g. note heads, hooks, rests) while *FP* denote elements where the recognition

process has resulted in a misclassification (e.g. quaver note detected as half note). In our testing environment,  $FN$  could be symbols that were possibly removed during pre-processing. Bellini et al. have observed that the number of true negatives can not be quantised since it includes data that the OMR system has chosen not to consider as valid symbols [6]. Furthermore, the authors have acknowledged that the relevance of symbols is different across the different classes. As an example, the scope of a key signature is considerably broader than that of an accidental, despite having the same appearance (disregarding their position within the framework of staves). For that reason, a more elaborate metric was introduced, providing categories for different classes of symbols where the number of expected symbols per class  $i$  is defined as follows:

$$NEBS_i = NTBS_i + NFBS_i + NMBS_i \quad (5.1)$$

For a class, the number of symbols expected is denoted by  $NEBS_i$ . This value is obtained by adding up the number of correctly identified and categorised symbols of this class  $NTBS_i$ , the number of incorrectly recognised symbols  $NFBS_i$  and symbols that were not detected at all by the algorithm  $NMBS_i$ .

## 5.2 Template Dataset

Evaluation of the system does not include actual photographs taken with the internal camera in order to provide for an environment that is as controlled as possible. Instead, the underlying test set of images consists of 15 synthetic scores of (monophonic) children's tunes that have been scanned at a resolution of 2480 x 3507 pixels. An example is given in Fig. 5.1. As opposed to the captured images of the mobile's camera, this test set solely contains images that are already binarised. This, however, does not affect the approach described in Chapter 3, with regard to the binarisation process.

As a further important property, standard music notation is used for which the prototype has been tuned accordingly. Ranging from three to five staves each, the notation covers all elements that are common to simple tunes. A higher degree of difficulty is introduced by textual (e.g. title, lyrics) as well as non-textual artefacts. The latter are either part of the musical sheet itself or they might have emerged during the capturing process.

### 5.2.1 Image Deformation

For the prototype it is vital to come up for several deficiencies that occur when capturing images of notation through the mobile phone's camera. Along with the aforementioned obsolete data, the system needs to be robust against rotation of the camera relative to the music sheet. As outlined earlier, the template data set consists of 15 scanned images that have not been normalised ahead of computation. Thus, scans show a natural skew,

**Alle Vögel sind schon da**  
Volksweise (aus Schlesien)  
Text: H. von Fallersleben

1. Al - le Vö - gel sind schon da, al - le Vö - gel  
al - le. Welch ein Sin - gen, Mu - si - zieren, —  
Pfei - fen, Zwi - schern, Ti - ri - liern! Früh - ling will nun  
ein - mar - schieren, kommt mit Sang und Schal - le.

2. Wie sie alle lustig sind,  
flink und froh sich regen!  
Amsel, Drossel, Fink und Star  
und die ganze Vogelschar  
wünschen dir ein frohes Jahr,  
lauter Heil und Segen.

**Im Märzen der Bauer**  
Volksweise (aus Mähren)

1. Im Mär - zen der Bau - er die Röss - lein an - spannt. Er  
setzt sei - ne Fel - der und Wie - sen in - stand. Er  
pflü - get den Bo - den, er eg - get und säht und  
rührt sei - ne Hän - de früh - mor - gens und spät.

2. Die Bäuerin, die Mägde, sie dürfen nicht ruh'n;  
sie haben im Haus und im Garten zu tun.  
Sie graben und rechen und singen ein Lied,  
und freu'n sich, wenn alles schön grünet und blüht.

Figure 5.1: Binary images of two children's tunes as used in the dataset. Errors introduced by the scan as well as ornaments can be seen near the edges.

typically in the range between  $-1^\circ$  to  $+1^\circ$ . Further tests have shown that the system successfully recognises rotation for notation sheets that have been cropped to show only the staves. Due to preprocessing that has been introduced to remove excess data (e.g. ornaments, shadows) near the borders of the image, positive results are only achieved in cases of minor rotations of up to 5 degrees. A more robust preprocessing method would be desirable for these cases as we will learn in the following.

For results listed in the following sections, no preprocessing was run prior to the algorithm proposed in this thesis. The introduction of random defects or curvature as suggested by [11] was omitted.

## 5.3 Results

The underlying framework has performed satisfactory on the test images on subjective tests. A brief overview over its performance will be given in the following. Broadly,

	True	False	Sum	Rate
Staves	52	1	53	0.981

Table 5.1: Recognition of Staves

testing was based on the following objectives: as a basic benchmark, the number of staves detected will be measured. Then, for the larger scope of vertical objects, performance of note and note length detection will be tested. Concludingly, classification rates of other symbols present in the staff system will be computed, while another focus will be on calculating the classification rate of pitch. For all of the above mentioned cases, numbers will be measured against ground truth data, which was manually computed. While it has been outlined earlier that the data set comprises 15 notation sheets, one particular piece has been found unsuitable, as unlike the vast majority of children's tunes, this particular score was written for two voices.

### 5.3.1 Staff line detection

We have already learned that the detection of staff lines is an important concept to any OMR system, as in music notation the meaning of symbols is highly dependent on their position relative to the staff lines. Furthermore, the fact that staff detection (and removal) is embedded in a very early stage of the OMR algorithm, makes this process critical. A system that does not incorporate any means of dealing with undetected lines (e.g. with information retrieved in later stages [21]) will inevitably bring about poor results, independent of how staff lines are dealt with afterwards by the respective systems. Consequently, performance of the line detection algorithm applied is of critical interest.

The approach chosen, namely focusing on projection profiles, has proved very robust, as it shows a detection rate of 0.981 on the test set (see Table 5.1). Considering stave detection at a larger scale (i.e. including preprocessing), the prototype has failed to detect staves in one single case. This failure is due to the edge detection algorithm, which classified the lowermost staff as background. Here, improvement could be made by applying a different rule to the preprocessing scheme. Ignoring potentially misleading areas near the border of the scanned image, one could try to introduce morphology transformations directly to the binarised image, trying to detect long horizontal and parallel lines. An opening transformation is expected to yield the appropriate areas where staves are likely to be located.

### 5.3.2 Symbol Recognition

Having obtained a horizontal strip consisting of a set of five staff lines and a range of symbols, vertical objects are located via projection profiles. One of the many difficulties

Prediction	True			Sum
	b	#	Other Symbols	
b	5	0	0	5
#	0	14	0	14
Omitted	0	0	0	0
Sum	5	14	0	19
Rate	1.000	1.000	-	1.000

Table 5.2: Confusion matrix listing the classification rate of accidentals before the inclusion of musical rules.

that arise processing this data is to discriminate between objects that are worth examining and those that can be easily omitted without losing information. Generally speaking, finding and classifying clefs, accidentals and notes is fundamental.

### Clefs

During early benchmarking, template matching for clefs yielded subpar results. Lowering the threshold for recognising a clef, however, had an even more adverse effect, as it led to false positives (e.g. note heads found at the horizontal position of the treble clef). Moreover, this threshold lowering had a follow-up effect on key signatures, which, being commonly positioned as first symbol(s) beneath the clef symbol, were not recognised as such anymore.

Instead of the template-based approach, a projection-based recognition scheme was found suitable for providing a solution to this problem. While a clef symbol occurs in different shapes conveying different meanings, this prototype was tuned specifically for the treble clef. For our test set, clefs were found with a success rate of 1.000.

### Key signature and accidentals

Aside from accidentals accompanying individual notes, this evaluation considers recognition quality for the tonality of the musical piece, which is derived from the key signature. As Table 5.2 shows, evaluation of accidentals for given input images have yielded perfect results due to a good set of rules that distinguish between *flats* and *sharps*.

It needs to be clarified that broadening the scope to more general notation sheets will obviously lead to imperfections due to the similar shape of *sharps* and *naturals*, which might occur in less simple scores than children's tunes. Here, musical rules could provide for means of fixing the problem of a potential misclassification between the respective types of accidentals.



Prediction	True							Sum
	1	2	3	4	5	6	Other	
(1) Quarter	214	0	0	0	0	0	0	214
(2) Half	0	12	1	0	0	0	0	13
(3) Full	0	0	2	0	0	0	0	2
(4) Eighth	0	0	0	283	0	0	0	283
(5) Sixteenth	0	0	0	0	2	0	0	2
(6) Dotted note	0	0	0	0	0	4	0	4
Omitted	1	1	0	4	2	2	0	10
Sum	215	13	3	287	4	6	0	528
Rate	0.995	0.923	0.667	0.986	0.500	0.667	-	0.979

Table 5.3: Confusion matrix listing the classification rate for note lengths.

## Notes

A thorough preprocessing stage combined with a delimiting of the search window by detecting clefs and key signatures prior to notes ensures a straight-forward detection of vertical objects. This is performed in the fashion of a thresholding on the vertical projection profile. For the total of 52 staves (recognition rate: 0.981) detected in our sample set, detection rate for filled notes (i.e. quarter, eighth, sixteenth notes or dotted notes) was 0.998. From Table 5.3 we observe, however, that percentage for note lengths is considerably lower at 0.979, which is primarily caused by deficiencies in half or full note detection. Even though statistics regarding full notes being misclassified are not significant, the high error rate of 0.333 corresponds well with other works that outline this particular problem.

Whereas properties regarding note length are indicated by different shapes of note heads in many cases, note length is often affected (i.e. reduced) by hooks or, connecting individual notes, by beams. The line tracking algorithm introduced by the author has not only proved capable of cleanly discriminating between beams and hooks, but it also provides for a decent overall detection rate of 0.986. As for hooks, half of the number filed under *omitted* is due to double hooks not being recognised (see Table 5.3).

### 5.3.3 Other symbols

After successfully eliminating bar lines, notes, clefs and accidentals, which all are characterised by peaks in the vertical projection profile, only few classes of symbols remain to be dealt with, among the likeliest of which are rests. These have in common a vertical position central to the set of staff lines, between second and fourth stave. Taking advan-

Prediction	True			Sum
	1	2	Other	
(1) Beams	9	0	0	9
(2) Hooks	0	264	0	264
Omitted	0	4	0	4
Sum	9	268	0	277
Rate	1.000	0.985	-	0.986

Table 5.4: Confusion matrix listing the classification rate of beams and hooks.

Prediction	True						Sum
	1	2	3	4	5	Other	
(1) Notes	518	0	0	0	0	0	518
(2) Accidentals	0	19	0	0	0	0	19
(3) Clefs	0	0	14	0	0	0	14
<b>(4) Crotchet Rests</b>	0	0	0	26	0	0	26
<b>(5) Quaver Rests</b>	0	0	0	0	5	0	5
Omitted	3	0	0	0	4	0	7
Sum	521	19	14	26	9	0	0
Rate	0.994	1.000	1.000	<b>1.000</b>	<b>0.556</b>	-	0.988

Table 5.5: Confusion matrix listing the classification rate of rests compared with other vertical objects (without bar lines).

tage of this property, template matching can be restricted to this very class. Here, the appearance of the crotchet rest is very different to that of the whole or half rest, which, in turn, are very similar. For our test set of notation sheets, a discrimination between crotchet and quaver rests was sufficient.

The difficult shape of the quaver rest and the shape variations between the music sheets makes this symbol prone to non-classification. On the test set, recognition of these quaver rests shows a relatively low success rate of 0.556 (see Fig. 5.5), which provides for an overall recognition rate of 0.988 regarding vertical objects. Possible workarounds would be lowering the threshold for this particular symbol or the inclusion of musical knowledge. For the latter, the duration of symbols detected within a measure (made up by the space that is delimited by two bar lines) could be summed up and compared to the most common duration of other measures.

In an earlier stage, the notation sheet was divided up into horizontal strips, which

Prediction	True							Sum
	C	D	E	F	G	A	B	
C	58	0	0	0	0	0	0	58
D	0	66	0	0	0	0	0	67
E	0	0	28	0	0	0	0	28
F	0	0	0	71	0	0	0	72
G	0	0	0	0	87	0	0	87
A	0	0	0	0	0	117	0	117
B	0	0	0	0	0	0	92	92
Omitted	0	1	0	1	0	0	0	0
Sum	58	67	28	72	87	117	92	521
Rate	1.000	0.985	1.000	0.986	1.000	1.000	1.000	0.996

Table 5.6: Confusion matrix listing the true pitch horizontally, and the predictions vertically.

were then cropped as a preparatory step for vertical projection. For this reason, objects between these strips are not processed. Thus, in order to detect chords (identified by a set of individual letters and optional elements such as accidentals or numbers), the algorithm would need to be altered.

### 5.3.4 Pitch Recognition

We have laid out earlier that one out of 8 notes are typically assigned to a musical pitch. Notes, on the other hand, can appear in different octaves with double or half their frequencies. The recognition scheme of the prototype is designed to work with notes between *E2* and *B4*. As confusions between the same notes of different octaves are very unlikely due to their vertical distance, it is sufficient only to discriminate between the following 7 distinct notes: *C*, *D*, *E*, *F*, *G*, *A*, *B*. Under good to perfect conditions, the pitch recognition rate is satisfactory and the algorithm applied succeeded in classifying the pitch at a rate of 0.996. As depicted in Table 5.6, errors stem exclusively from note heads that could not be recognised at all.

As pitch height is highly dependent on the previously computed location of staff lines, confusions between two neighbouring pitches are possible and such effects are more likely in cases of warped staff lines, for which the algorithm would need to be strongly tuned. As a workaround, vertical positions of staff lines could be computed for each note head candidate.

## Conclusion and Future Work

The author has presented a framework for music recognition through an optical source, based on projection methods and template matching. The prototype has proved suitable for processing simple children's tunes and it provides a basis for further research in the field of OMR on handheld devices.

During preprocessing, a range of image manipulation methods deal with deficiencies that may or may not occur in the input notation. As a basis for further computation, a set of two reference length parameters is calculated from a horizontal projection profile. As staff lines are already eliminated after the first preprocessing, detection of objects resident in these vertical positions is simplified. For means of feature classification, a scheme based on template matching working with a fixed set of templates for individual notes and rests has been chosen. On the other hand, symbols of a more difficult shape are dealt with either by means of the aforementioned projection methods or by line tracking algorithms. The problem of determining the pitch of individual notes and the scope of accidentals is solved by comparing the vertical position of the center of note heads to the previously computed location of staff lines. Finally, the set of notes along with their parameters are sent to a translation engine, which, in turn, yields a MIDI file that can be played back by the user.

In order to raise quality of the recognition algorithm, two prominent issues could be targeted by follow-up research: first of all, instead of considering the vertical positions of staff lines as absolute numbers, which are detected in a preliminary step, one could track and store line positions during the task of staff line removal. This could help reducing the effect of image warping, which occurs particularly during the capturing process on mobile devices.

Another great challenge in OMR, which has already been laid out in this thesis, is the choice of templates. Working on a fixed set of templates, which were extracted from the Unicode database, the proposed prototype obviously deals well with uniform

scaling. This is done by means of resizing the template according to the previously discovered staff line spacing. As, despite all efforts made in the history of written music, publishers still use different layouts and musical symbols, a wise choice of (universal) features would greatly help dealing with different appearances. These could be derived by taking into account individual symbols of different shapes, leading to a set of features for each symbol class.

By the time of finishing this thesis, great progress has been made in offering OMR software to a broader audience, both as a practical tool for work and educational purposes. Applications available to end users have added capability for optical music recognition that has led to a shift in workplace. Now, musicians and composers alike are allowed to write or capture music notation independent of their current location. The challenge posed by non-perfect sources has been highlighted in this thesis, making a wise choice of methods necessary for obtaining good results. At the same time, this problem is still one of the open questions left, as the algorithms chosen by the author have found to be limited in their capabilities. Moreover, results could be further improved by adopting an interactive approach, where both user and recognition framework benefit from each other.

# Bibliography

- [1] Audiveris - Open Music Scanner. January 2013. Website. <http://audiveris.kenai.com>.
- [2] Baba, T.; Kikukawa, Y.; Yoshiike, T.; Suzuku, T.; Shoji, R.; Kushiyama, K.; Aoki, M. 2012. Gosen: A Handwritten Notational Interface for Musical Performance and Learning Music. *ACM SIGGRAPH 2012 Emerging Technologies*
- [3] Bainbridge, D.; and Bell, T.C. 1996. An Extensible Optical Music Recognition System. *Nineteenth Australasian Computer Science Conf.* pp. 308-317.
- [4] Bainbridge, D.; and Bell, T.C. 1997. Dealing with Superimposed Objects in Optical Music Recognition. *Proc. Sixth Int'l Conf. Image Processing and Its Applications* pp. 756-760.
- [5] Bainbridge, D.; and Bell, T. 2001. The Challenge of Optical Music Recognition. *Computers and the Humanities* 35. pp. 97-98.
- [6] Bellini, P; Bruno, I; Nesi, P. 2007. Assessing Optical Music Recognition Tools. *Computer Music Journal*, 31:1 pp. 68-93.
- [7] Boone, K. *Generating simple MIDI files using Java, without using the Java Sound API*. January 2013. Website. <http://kevinboone.net/javamidi.html>.
- [8] Bresenham, J.E. 1965. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal* 4 (1) pp. 25-30.
- [9] Carter, N.P.; and Bacon, R.A. 1992. Automatic Recognition of Printed Music. *Structured Document Image Analysis*, Baird, H.S.; Bunke, H.; Yamamoto, K., eds. pp. 454-465.
- [10] Coüason, B. 1997. Segmentation and Recognition of Documents Guided by a priori Knowledge: Application to Musical Scores. *PhD Thesis, IRISA*.
- [11] Dalitz, C; Droettboom, M.; Pranzas, B.; Fujinaga, I. 2008. A Comparative Study of Staff Removal Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 5. 755.

- [12] Fahmy, H.; Blostein, D. 1991. A Graph Grammar for High-Level Recognition of Music Notation. *Proc. First Int'l Conf. Document Analysis and Recognition* pp. 70-78.
- [13] Fremerey, C.; Müller, M.; Kurth, F.; Clausen, M. 2008. Automatic Mapping of Scanned Sheet Music To Audio Recordings. *Proc. Ninth Int'l Conf. Music Information Retrieval* pp. 413-418.
- [14] Fujinaga, I. 1988. Optical Music Recognition using Projections. *Master Thesis, Faculty of Music, McGill University, Montreal.*
- [15] Fujinaga, I. 2004. Staff Detection and Removal. Visual Perception of Music Notation: On-Line and Off-Line Recognition, George, S., ed.. *Idea Group Inc, Hershey* pp. 1-39.
- [16] George, S.E. 2004. Visual Perception of Music Notation: On-Line and Off-Line Recognition. *IRM Press* 112.
- [17] Goecke, R. 2003. Building a System for Writer Identification on Handwritten Music Scores. *Proc. Int'l Conf. Signal Processing, Pattern Recognition and Applications* pp. 250-255.
- [18] Kilmer, A. D.; and Civil, M. 1986. Old Babylonian Musical Instructions Relating to Hymnody. *JCS* 38. pp. 94-48.
- [19] Kurth, F.; Müller, M.; Fremerey, C.; Chang, Y.-H.; Clausen, M. 2008. Automated Synchronization of Scanned Sheet Music with Audio Recordings. *Proc. Eighth Int'l Conf. Music Information Retrieval* pp. 413-418.
- [20] Martin, P.; and Bellissant, C. 1991. Low-Level Analysis of Music Drawing Images. *Proc. First Int'l Conf. Document Analysis and Recognition* pp. 417-425.
- [21] McPherson, J.R. 2002. Introducing Feedback into an Optical Music Recognition System. *Proc. Third Int'l Conf. Music Information Retrieval* pp. 259-260.
- [22] Miyao, H. 1996. Note Symbol Extraction for Printed Piano Scores using Neural Networks. *IEICE Trans. Inf. Syst. E79-D* pp. 548-554.
- [23] Miyao, H. 2002. Stave Extraction for Printed Music Scores. *IDEAL 2002, LNCS 2412, Yin, H. et al. eds.* pp. 562-568.
- [24] MusicXML - Tutorial: Hello World January 2014. Website. <http://www.musicxml.com/tutorial/hello-world/>.
- [25] Neuratron NotateMe - Music Handwriting App for Phones and Tablets October 2014. Website. <http://www.neuratron.com/notateme.html>.

- [26] Ng, K.; Cooper, D.; Stefani, E.; Boyle, R.; Bailey, N. 1999. Embracing the Composer: Optical Recognition of Handwritten Manuscripts. *Proc. Int'l Computer Music Conf.* pp. 500-503.
- [27] Niblack, W. 1986. An Introductino to Digital Image Processing. *Prentice Hall* pp. 115-116.
- [28] Otsu, N. 1979. A Threshold Selection Method from Gray-Level Histograms. *IEEE Trans. Systems, Man and Cybernetics* pp. 62-66.
- [29] Pavlidis, T. 1982. Algorithms for Graphics and Image Processing. *Computer Science Press*.
- [30] Prerau, D. 1988. Optical Recognition of Music Symbols. *A Critical Survey of Music Image Analysis*, Blostein, D.; and Baird, H.S., eds., Springer-Verlag, Heidelberg, 1992 pp. 405-434.
- [31] Pugin, L. 2006. Optical Music Recognition of Early Typographic Prints using Hidden Markov Models. *7th Int. Conference on Music Information Retrieval*. 1.
- [32] Randriamahefa, R.; Cocquerez, J.P.; Pepin, F.; Philipp, S. 1993. Printed Music Recognition. *Proc. Second Int'l Conf. Document Analysis and Recognition* pp. 898-901.
- [33] Rebelo, A.; Capela, G.; Cardoso, J.S. 2010. Optical Recognition of Music Symbols. A Comparative Study. *Int'l Journal Document Analysis and Recognition*, vol. 13, 1 pp. 19-31.
- [34] Rebelo, A.; Fujinaga, I.; Paszkiewicz, F.; Marcal, A. R. S.; Guedes, C.; Cardoso, J. S. 2012. Optical Music Recognition - State-of-the-Art and Open Issues for Handwritten Music Scores. *Int'l Journal of Multimedia Information Retrieval*, 1-18. 3.
- [35] Reed, K.T.; Parker, J.R. 1996. Automatic Computer Recognition of Printed Music. *Proc. Int'l Conf. Pattern Recognition* pp. 803-807.
- [36] Roach, J.W.; and Tatem, J.E. 1988. Using Domain Knowledge in Low-Level Visual Processing to Interpret Handwritten Music: An Experiment. *Pattern Recognition*, vol. 21 pp. 33-44.
- [37] Stückelberg, M. V.; and Doermann, D. 1999. Markov Source Model for Printed Music Decoding. *Journal of Electronic Imaging*, vol. 5, no. 1. 1.
- [38] Stückelberg, M. V.; and Doermann, D. 1999. On Musical Score Recognition using Probabilistic Reasoning. *Proc. Fifth Int'l Conf. Document Analysis and Recognition*. 1.



- [39] Suzuki, S.; Abe, K. 1985. Topological Structural Analysis of Digitized Binary Images by Border Following. *Computer Vision, Graphics and Image Processing* 30 pp. 32-46.
- [40] Szwoch, M. 2008. Using MusicXML to Evaluate Accuracy of OMR Systems. *Proc. Fifth Int'l Conf. Diagrammatic Representation and Inference* pp. 419-422.
- [41] The Unicode Standard, Version 6.2.0. January 2013. Website. <http://www.unicode.org/versions/Unicode6.2.0/>.
- [42] Toyama, F.; Shoji, K.; and Miyamichi, J. 2006. Symbol Recognition of Printed Piano Scores with Touching Symbols. *Proc. 18th Int'l Conf. Pattern Recognition* (2). 1.
- [43] Williams, C. F. A. 1903. The Story Of Notation. *Charles Scribner's Sons, New York*. 1:1-2.