

IMPERIAL COLLEGE LONDON

MENG INDIVIDUAL PROJECT

NoteEd

An Interactive Manuscript Tutor for Fundamental Music Theory

Author:

Peter HAMILTON

Supervisor:

Dr. Robert CHATLEY

Second Marker:

Dr. Tony FIELD

17th June 2014

Abstract

We present **NoteED** - an application that allows a child to practise their music theory and manuscript writing. The system automatically generates feedback which would otherwise require interaction with a human music teacher.

Using techniques from *computer vision*, *machine learning* and *domain knowledge*, **NoteED** recognises and analyses fundamental musical symbols handwritten on a tablet. This enables generation of useful feedback to the student in order to improve their written manuscript quality whilst reinforcing the learning of concepts from fundamental music theory.

Initial findings suggest that **NoteED** can be used as an effective for students to practice their notation and receive timely feedback. We also show that the basic level of feedback on a student's mistakes closely matches that of a human music teacher.

Acknowledgements

I would first like to thank Rob Chatley for his support throughout the ups and downs of this project and for his ideas, advice and helping me focus my goals on the more realistic targets. I'd also like to thank the students and teachers who helped me in gathering data and testing out **NoteED** , putting up with endless notation practice, hopefully it did you some good...

I also want to take the opportunity to thank my family who have supported me though my four years here at Imperial with constant encouragement, advice and occasional emergency food runs. I couldn't have done it without you.

Last but in no way least, I would like to thank Agnes, firstly for her expertise, which has proved invaluable, but also her constant support and encouragement, without you this project would never have even gotten off the ground, let alone landed safely.

Contents

1	Introduction	9
1.1	Context	9
1.2	Motivation	11
1.3	Objectives	12
1.4	Mistake Detection	12
1.4.1	Learning Improvement	12
1.4.2	Tablet Interface	12
1.4.3	Engaging Experience For Child	12
2	Background	14
2.1	Music Theory	14
2.1.1	The Staff	15
2.1.2	Bar Lines	15
2.1.3	Notes and Rests	16
2.1.3.1	Notes	16
2.1.3.2	Rests	16
2.1.3.3	Duration	16
2.1.3.4	Pitch	17
2.1.4	Accidentals	18
2.1.5	Clefs	18
2.1.6	Key Signatures	19
2.1.7	Time Signatures	20
2.2	Common Notation Mistakes	21
2.2.1	Gathering Professional Feedback	21
2.2.2	Note Heads	24
2.2.2.1	Ambiguous Position	24
2.2.2.2	Too Big/Small	25
2.2.2.3	Broken Note Heads	26

2.2.2.4	Messy Note Heads	27
2.2.2.5	Angled Heads	27
2.2.3	Note Stems	28
2.2.3.1	Stem Angle	28
2.2.3.2	Stem Straightness	28
2.2.3.3	Stem Direction	29
2.2.3.4	Stem Side	29
2.2.3.5	Stem Length	30
2.2.4	Quaver Tails	30
2.2.4.1	Quaver Tail Side	30
2.2.5	Rests	30
2.2.5.1	Position	30
2.2.6	Duration Dots	31
2.2.6.1	Wrong Side	31
2.2.7	Accidentals	32
2.2.7.1	Ambiguous	32
2.2.7.2	Wrong Side	32
2.2.7.3	Wrong Line	33
2.2.8	Key Signatures	33
2.2.8.1	Wrong Octave	33
2.2.8.2	Incorrect Order	34
2.2.8.3	Incorrect Pitch	34
2.2.9	Beats and Timing	35
2.2.9.1	Too Many Beats	35
2.2.9.2	Too Few Beats	35
2.3	Existing Music Theory Resources	35
2.3.1	Mobile Apps	35
2.3.2	Theory Workbooks	36
2.3.2.1	Taylor, E.R. and Associated Board of the Royal Schools of Music 1989	38
2.4	OMR Applications	39
2.4.1	Neuratron Photoscore	40
2.4.2	Audiveris	40
2.4.3	Gamera	40
2.4.4	Capella Scan	40

3	Technical Research	41
3.1	Previous Works	41
3.1.1	Taubman, Odest and Jenkins 2005	41
3.1.2	Ben-Dayana and Giloh 2013	42
3.1.2.1	Stem Detection	42
3.1.2.2	Note Head and Features Detection	43
3.1.3	Rebela et al. 2011	43
3.2	Generating Manuscripts	44
3.2.1	Professional GUI Tools	44
3.2.2	LilyPond	45
3.2.3	VexFlow	46
3.3	OMR Architecture	47
3.4	Segmentation	47
3.4.1	Connected Component Analysis	48
3.4.1.1	Recursive Labelling	48
3.4.1.2	Two Pass Labelling	50
3.4.2	Projections	51
3.4.3	Template Matching	53
3.4.4	Defects and Difficulties	54
3.4.4.1	Touching Objects	55
3.5	Component Features	55
3.5.1	Vertical and Horizontal Holes	55
3.5.2	Moments	56
3.5.2.1	Ordinary Definition	56
3.5.2.2	Zeroth Order Moments	56
3.5.2.3	First Order Moments	57
3.5.3	Run Length Encoding	57
3.6	Classification	58
3.6.1	Nearest Neighbour	58
3.6.2	KNN Editing	59
3.6.2.1	Wilson Editing	59
3.6.2.2	Genetic Algorithms	60
3.7	Scoring & Evaluation	61
3.7.1	Image Difference	61
3.7.2	Normalized Cross Correlation	61

3.7.3	Skeletonization	62
3.7.4	Watershed Segmentation	63
4	Techniques	65
4.1	Architecture	65
4.2	Identification	65
4.2.1	Segmentation	65
4.2.1.1	Initial Segmentation	66
4.2.1.2	Stem Removal	67
4.2.2	Feature Extraction	68
4.2.2.1	Resampling	68
4.2.3	Classification	68
4.2.3.1	K Nearest Neighbour	69
4.2.3.2	Neural Networks	74
4.2.4	Pitch	76
4.2.4.1	Sharp Centres	78
4.2.4.2	Flat Centres	78
4.2.4.3	Note Head Centres	79
4.2.5	Duration	80
4.2.6	Domain Knowledge	83
4.3	Scoring and Feedback	83
4.3.1	Pitched Notes and Accidentals	84
4.3.1.1	Position	84
4.3.2	Key Signatures	85
4.3.2.1	Wrong Octaves	85
4.3.2.2	Out of Order	85
4.3.2.3	Incorrect Accidental	86
4.3.3	Beats and Timing	86
4.3.4	Stems	87
4.3.4.1	Straightness	88
4.3.4.2	Angle	89
4.3.4.3	Direction	89
4.3.4.4	Side	91
4.3.4.5	Length	92
4.3.5	Quaver Tail	93
4.3.5.1	Side	93

4.3.6	Note Heads	93
4.3.6.1	Size	93
4.3.6.2	Messy Crotchet Heads	94
4.3.6.3	Broken Hollow Heads	94
4.3.6.4	Angled Semibreve	95
5	Implementation	96
5.1	Architecture	96
5.1.1	Client	97
5.1.2	Server	98
5.1.3	Processing Service	98
5.1.4	Core Database Schema	99
5.1.5	Entity Relationship Diagram	101
5.2	Input	103
5.2.1	Medium Selection	103
5.2.1.1	Flat Bed Scanner	103
5.2.1.2	Gestures	103
5.2.1.3	Tablet Input	104
5.2.2	Capturing Strokes	104
5.3	Data Storage and Retrieval	106
5.3.1	Stave Drawing	106
5.3.2	Components	107
5.4	Feedback	107
5.4.1	Listing Mistakes	107
5.4.2	Colour Coding	108
5.4.3	On Screen Correction	109
5.4.4	The Hybrid Approach	110
6	Evaluation	113
6.1	Mistake Detection	113
6.2	Learning Improvement	115
6.3	Engaging Experience For Child	116
7	Conclusions & Future Work	117
7.1	Conclusions	117
7.2	Future Work	118

Chapter 1

Introduction

1.1 Context

The earliest sources of Western European music recorded on manuscript can be dated back to the late 9th century¹ and most likely emerged from the need for a consistent framework by which music could be reproduced and shared, as opposed to just passed on by word of mouth through the generations.

Where once composers could only write and reproduce music by hand, even having to draw their own staff lines, the advent of pre-printed paper allowed rapid composition on a rigid and regular framework. Further advances in technology led to music typesetting and printed music scores which suddenly enabled rapid and faithful reproduction of musical works.

Since then, the advent of personal computers, tablets, scanners and home printers means that with the right software such as Avid's Sibelius², a composer can delegate the task of manuscript representation, notation, layout, and conformity to conventions to a piece of software, while they concentrate on the actual composition.

The benefits technology has brought to the communication of musical notation should not be understated, however in the learning stages of music theory, handwritten notation is a critical developmental tool which takes many students considerable time to master. I believe technology has not yet been used to its full potential in this area, something this

¹http://en.wikipedia.org/wiki/Music_manuscript, accessed 13th June 2014

²See <http://www.avid.com/US/products/Sibelius/features>

project aims to change.

A variety of software and applications (Section 2.3) now exists for learning advanced music theory concepts, composing scores, digitising existing handwritten music and a great many other uses. However in the realm of basic music theory, primarily taught to young children when they start their first instrument, it seems the available toolset is limited to the same exercise books, manuscript paper and repetitive practice that it always has been. Indeed in researching this paper, I discovered that in the 15 years since I first came across music theory myself, the same books and learning styles are still being applied.

Now, this doesn't mean they're in any way *ineffective*, indeed one could quite rightly assume the opposite, as these learning styles have demonstrably stood the test of time. In those same 15 years, as noted above, there have been significant advances in the use of technology to support the notation of music in other ways. However helping students to learn the basics of hand forming musical notation has been neglected.

Now that children are getting more and more used to computers, tablets, games and interactive learning styles, it is my aim to try and bring some of this technology into the area of fundamental music theory and manuscript notation.

1.2 Motivation

In the UK national curriculum for primary education, there is no requirement for children to learn to write or even read musical notation (DfE 2013). However, when they're learning an instrument and reach the point where they need to take grades³, this is something they must be able to do (Spanswick 2012).

Learning to write musical manuscript correctly and clearly takes time and practice, much as when children are learning normal handwriting skills. From informal research it is clear that most instrumental teachers would like their pupils to learn written music theory alongside their practical instrumental tuition.

However there often isn't an abundance of time to spend on learning and practising music theory and the options available to the student and teacher are therefore limited. Do you go over the topic during lesson time, eating into the instrumental tuition, or do you perhaps send the child away with some manuscript paper⁴ and 'homework', which can be marked during their next lesson, using yet more time. Sadly both of these are often ineffective methods of learning since without guidance and feedback on a regular basis, it's all too easy for a student to accidentally reinforce bad habits. **A faster, unsupervised feedback loop** is needed.

Whilst several solutions exist that can help children with academic theory as we can see in Section 2.3, the subject of handling handwritten notation and manuscript has so far not been tackled. **Optical Music Recognition (OMR)** is a field I draw on heavily in this dissertation. Commercial and research applications in this area typically focus on converting draft or rough handwritten notation into a digital format and are not primarily educational tools. With such applications, no feedback is provided and indeed extracting feedback would be hard given the technical methods used to analyse the musical scores.

In short, other than their music teacher, **nothing really exists which is capable of helping a child learn to correctly write musical notation whilst simultaneously providing feedback** This project aims to make a start towards filling that void.

³ABRSM Subjects, Exams and Prerequisites <http://gb.abrsm.org/en/our-exams/information-and-regulations/>

⁴An example of blank manuscript paper can be found at <http://www.blanksheetmusic.net/>

1.3 Objectives

1.4 Mistake Detection

Objective: Enable the detection of common mistakes in notation, specifically in symbols, pitch, time signatures and other musical features outlined in Table 2.4.

I intend to get a professional to identify mistakes in a catalog of musical samples I gather from children and other users and by building up a comparison table I will then be able to see if the application is firstly capable of spotting the same mistakes as a human and secondly providing contextualised, useful feedback to help them improve (Section 1.4.1)

1.4.1 Learning Improvement

Objective: Produce an application which can improve on and continue a child's learning outside of their music lessons

This will be most likely evaluated by logging a child's activity and looking for patterns in the number of mistakes they make over time, which we would hope to see decrease.

1.4.2 Tablet Interface

Objective: To design and create a scalable tablet compatible user interface which can provide a child with a smooth and effective writing experience in order to facilitate written notation.

This can be evaluated using user testing. Most likely I will have perform some optimisation to ensure the best experience and make sure that the 'manuscript paper' scales between devices properly

1.4.3 Engaging Experience For Child

Objective: Combine the tablet interface and mistake detection objectives to produce a streamlined experience which a student will happily engage with on a repeat basis.

Success should be evaluated by examining access and activity logs to determine how regularly a child engages with the system (particularly repeat usages).

Chapter 2

Background

2.1 Music Theory

“Right from the start, it is important to learn how to write down music clearly. As a musician, unclear manuscript can waste valuable rehearsal time and might lead to performance mistakes. In an exam, badly written work may be misunderstood and could lose you vital marks.” (Taylor, E.R. and Associated Board of the Royal Schools of Music 1989)

For early grades of music theory, the criteria which students need to meet in order to maximize their marks and musical potential is covered in quite specific detail. This makes sense as the majority of students taking the exam are likely to be young. Since they are also unlikely to have encountered much written music before, this is the most crucial time for teaching them good habits.

The following is a guide to the manuscript entities that a student aiming to take their grade 1 - 3 theory exams would most likely have to know about and/or write down, along with the relevant criteria for what constitutes acceptable and unacceptable manuscript, most of the examples below are taken from or based on the official documentation in the AB Guide to music Theory(Taylor 2008).

2.1.1 The Staff

A staff (or stave) is comprised of five horizontal parallel staff lines separating four staff spaces, together they form a framework within which music elements are placed.

The lines are numbered from bottom (*first line*) to top *fifth line* and spaces are labelled in a similar fashion. Notes are then placed on the staff and depending on the clef, similar locations can indicate different actual notes. Further information about the Clef is given in section (Section 2.1.5) below. The staff can also be used to notate untuned percussion, however this remains out of scope of this project for the time being.

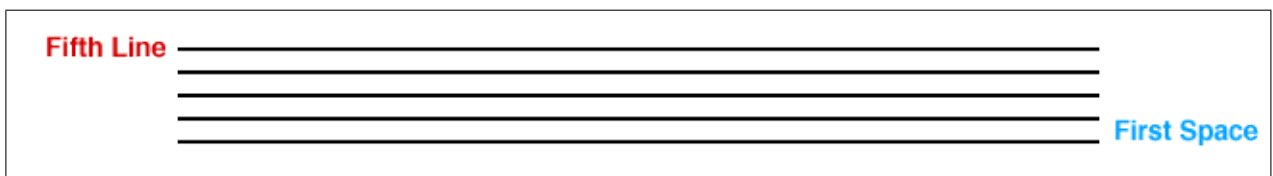


Figure 2.1: An example of a standard staff with 5 staff lines

2.1.2 Bar Lines

Aside from the staff, bar lines are one of the most important parts of the score. Bar lines divide the music up into smaller units of time called *measures* or *bars*.

There are multiple types of bar lines, some of which are shown in Figure 4.5 but the most important one is the single bar line which separates the bars. This is the only bar line we will consider in the scope of this project.

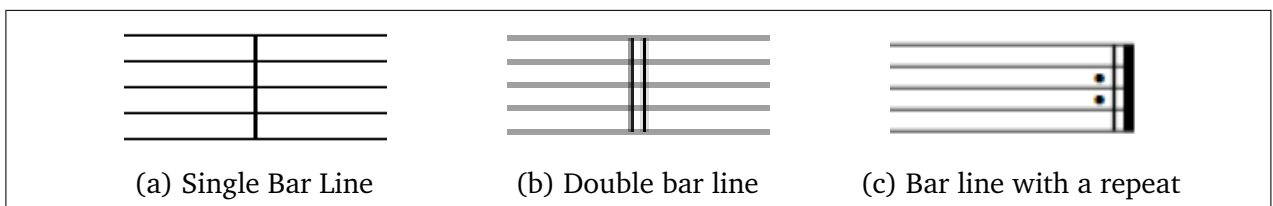


Figure 2.2: An example of single, double and repeated bar lines (Figure 2.2c)

2.1.3 Notes and Rests

2.1.3.1 Notes

Notes are used to represent duration (differentiated by their shape) and pitch (differentiated by where they're placed on the staff) of musical sounds. Most notes are comprised of a note head and a stem but may also have beams or tails in the case of notes shorter than a crotchet.

For reference, there are multiple variations in how note lengths are named and different papers refer to notes using the British and American terminology. I will be using british naming conventions throughout this project but for reference I have put both naming styles in Table [2.1](#).

2.1.3.2 Rests

Rests correspond directly to notes as you can see from Table [2.1](#) but unlike notes, rests can only indicate duration and do not have pitch. They're located centred around the central (3rd) staff line and each length is represented with a different symbol or a relative location to the staff line.

2.1.3.3 Duration

The note *length* or *value* or *duration* (number of beats a note lasts for) is determined by multiple features.

A semibreve lasts 4 beats, minims 2 and crotchets 1 beat. Further divisions for quavers and semiquavers are notated by the number of tails (in the case of single, isolated notes) or the number of beams (in the case of beamed notes), both of which are located on the note stem furthest from the note head.

In general, for multiple notes with a value less than 1 the convention is to beam them together and since multiple beams can be used in place of tails you are not limited to beaming only notes of the same duration

There also exist duration dots which are single dots placed to the right of a note head or rest. These indicate that the duration of the dotted note is equal to $1.5 \times$ the length of the

original note.

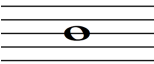
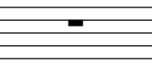
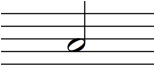




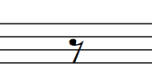

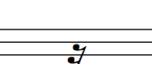

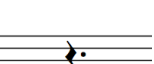
Name	Name (American)	Note Symbol	Rest Symbol
Semibreve	Whole Note		
Minim	Half Note		
Crotchet	Quarter Note		
Quaver	Eighth Note		
Semiquaver	Sixteenth Note		
Duration Dot			

Table 2.1: An overview of note and rest durations

2.1.3.4 Pitch

The absolute pitch of a note is determined by its vertical position on the staff relative to the staff lines taking into account the clef (Section 2.1.5), the key signature (Section 2.1.6) and any applicable accidentals (Section 2.1.4).

Pitch is notated by letter and number corresponding to the note itself and the octave. For example, the central C on the piano (the first ledger line on a treble clef) is noted by C4 and the next C one octave above it is notated by C5. A sequence of notes is shown labelled in Figure 2.3. For notes with sharps or flats applied, typically an additional lowercase character (s for Sharp and b for Flat) is added, for example Fs4, Bb4 etc.

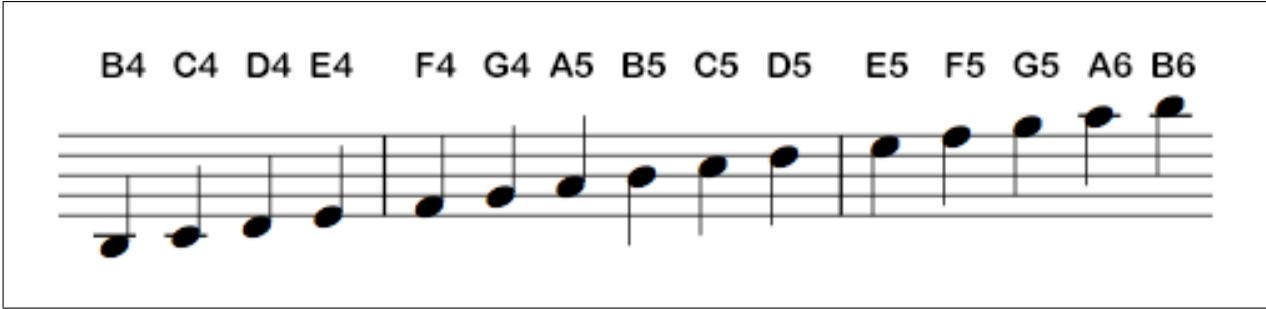


Figure 2.3: Pitch labelling for notes

2.1.4 Accidentals

When placed on staff lines or spaces, accidentals modify the note which they precede unless cancelled out by another accidental (often a natural). There are three types of accidental: sharps, flats and naturals, which are explained in Table 2.2

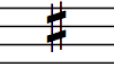
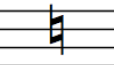
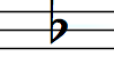
Name	Symbol	Description
Sharp		Raise the pitch of a note by one semitone
Natural		Cancels any previous sharp or flat which might have applied to a note, usually from the key signature
Flat		Lowers the pitch of a note by one semitone

Table 2.2: An overview of accidentals

2.1.5 Clefs

A clef is generally the first entity on a staff and defines the pitch range for the music, there are many clefs but within the scope of this project we will focus primarily on the most common two clefs, the treble clef and bass clef.



Name	Symbol
Treble Clef	
Bass Clef	

Table 2.3: An overview of common clefs

2.1.6 Key Signatures

Key signatures are found at the beginning of a stave next to the clef and are applicable to all musical notes which come after them, essentially setting the accidentals for the foreseeable future so we don't have to add them before every single note.

The key itself is defined by the number of sharps or flats in the signature and these follow a rigid structure and are added in a specific order left to right as shown in Figure 2.4 Sharps and flats are never combined in a key signature.

2.1. MUSIC THEORY

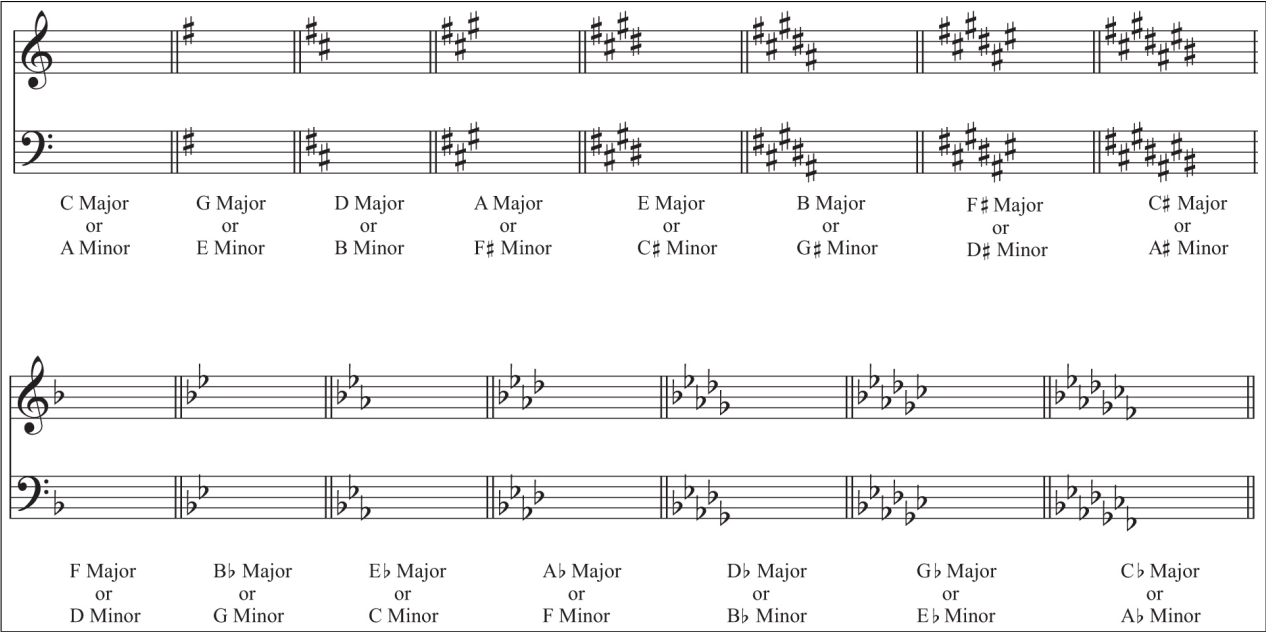


Figure 2.4: An overview of key signatures in treble and bass clef showing the number of accidentals and the positions on the staff, from <http://www.netplaces.com/singing/basic-music-theory/key-signatures.htm>, accessed 10/06/2014

2.1.7 Time Signatures

Time signatures can be found at the beginning of the staff or the start of a bar. The ‘numerator’ sets the number of beats per bar and the ‘denominator’ sets which note value is used to represent a beat for the music which follows. They can be split into several subcategories depending on the rhythm as shown in Figure 2.5.

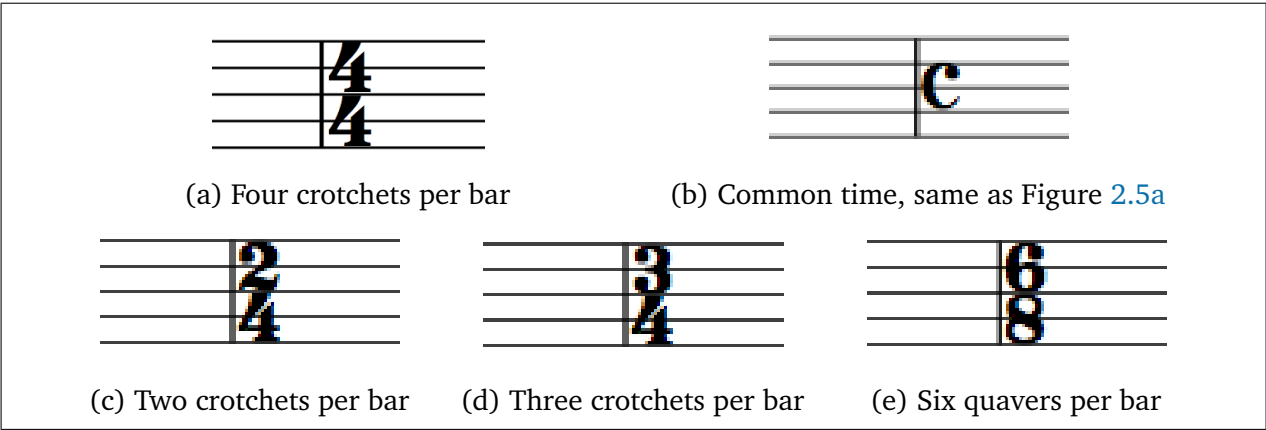


Figure 2.5: Examples of different time signatures

2.2 Common Notation Mistakes

Notation follows a strict set of rules and conventions, however there are a *lot* of them and as such it's perfectly natural that children struggle to remember them all, resulting in mistakes.

In order to focus the scope of this dissertation, I needed to establish exactly which mistakes were most likely to occur. In order to understand what these might be, I referred to two main sources.

The first is *Music Theory in Practice*, (Taylor 2008) This book, written specifically to assist those teaching music theory, has some good examples of mistakes a student should avoid.

The second is consultation with professional musicians and teachers. This is covered further in Section [2.2.1](#).

2.2.1 Gathering Professional Feedback

In order to build an application which can help to compensate for the absence of a tutor, we need to ensure that it can spot as many of the same mistakes as a tutor would.

The most logical way to establish precisely what these mistakes might be is to ask the tutors. When I did this, several of them found it difficult to express the mistakes they had seen children make without a piece of music in front of them.

I therefore devised the following method for gathering data to form a 'mistake database'.

1. Build the input section of the application such that a student can 'free-draw' on a blank music staff.
2. Set the student challenges and ask them to have a go at as many as they can
3. Print out the attempts (labelled) and get a music tutor to mark them freehand
4. Collate and summarise the feedback

To save time (and paper) the marking was done 3-UP on a portrait sheet of A4, an example of one of the tutor feedback sheets can be seen in Figure [2.6](#). It should be noted that in

2.2. COMMON NOTATION MISTAKES

some of the feedback tutors refer to errors differently, for example by using ‘the stem is on the wrong side’ to mean that either a note stem is up when it should be down or that it’s on the right when it should be on the left. Below are annotated examples of the mistakes **NoteED** needs to be able to identify.

1479

Sharps should come before the note.

1480

The stem is on the incorrect side of the note.

Below the middle line stems should be on the right side and above the middle line the left.

1481

The stem is on the incorrect side of the note

The image shows a handwritten musical manuscript with three examples of notation errors. Each example is enclosed in a rectangular box. Example 1479 shows a treble clef with a note on the second line (F) that has a sharp symbol (#) written after the note. A red circle highlights the sharp, and a red line points to it with the annotation 'Sharps should come before the note.' Example 1480 shows a treble clef with a note on the second line (F) that has a stem on the right side. A red circle highlights the stem, and a red line points to it with the annotation 'The stem is on the incorrect side of the note.' Example 1481 shows a treble clef with a note on the second line (F) that has a stem on the right side. A red circle highlights the stem, and a red line points to it with the annotation 'The stem is on the incorrect side of the note'. A general annotation in red ink between examples 1480 and 1481 states: 'Below the middle line stems should be on the right side and above the middle line the left.'

Figure 2.6: Example of tutor-annotated manuscript

2.2. COMMON NOTATION MISTAKES

Once this data was collected I had a sample of 83 attempts at various notation challenges, of which a number were annotated. I was then able to extract some of the recurring issues which **NoteED** should aim to recognise which I've summarised in Table 2.4. I present what is by no means an exhaustive list below and later on in Section 4.3 I go over how I actually identify, score and provide feedback to the student for these mistakes.

Element	Issue	Description	Analysis
Note Head	Ambiguous Position	Section 2.2.2.1	Section 4.3.1.1
Note Head	Broken Lines	Section 2.2.2.3	Section 4.3.6.3
Note Head	Angled	Section 2.2.2.5	Section 4.3.6.4
Note Head	Messy	Section 2.2.2.4	Section 4.3.6.2
Stem	Length	Section 2.2.3.5	Section 4.3.4.5
Stem	Straightness	Section 2.2.3.2	Section 4.3.4.1
Stem	Direction	Section 2.2.3.3	Section 4.3.4.3
Stem	Side	Section 2.2.3.4	Section 4.3.4.4
Stem	Angle	Section 2.2.3.1	Section 4.3.4.2
Quaver Tail	Side	Section 2.2.4.1	Section 4.3.5.1
Rest	Position	Section 2.2.5.1	Section 4.3.1.1
Key Signature	Incorrect Octave	Section 2.2.8.1	Section 4.3.2.1
Key Signature	Incorrect Ordering	Section 2.2.8.2	Section 4.3.2.2
Key Signature	Incorrect Accidentals	Section 2.2.8.3	Section 4.3.2.3
Beats & Timing	Too Many Beats	Section 2.2.9.1	Section 4.3.3
Beats & Timing	Too Few Beats	Section 2.2.9.2	Section 4.3.3

Table 2.4: A brief summary of some mistakes which emerged from the preliminary data gathering. 'FUTURE' indicates that analysis for the mistake hasn't yet been incorporated into the application

2.2.2 Note Heads

2.2.2.1 Ambiguous Position

The semibreve is the easiest note to draw and is simply an oval outline, however, the rules which govern its placement are quite specific as outlined in Figures Figure 2.7 & Figure 2.9

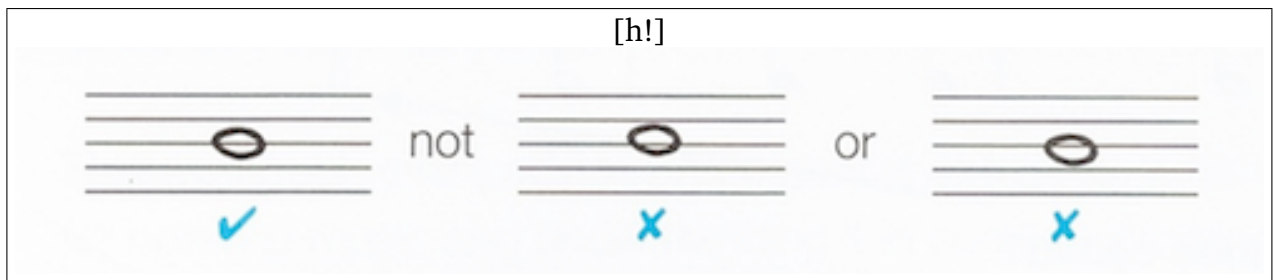


Figure 2.7: If it is drawn on the line, the line must go exactly through the middle of the semibreve

2.2.2.2 Too Big/Small

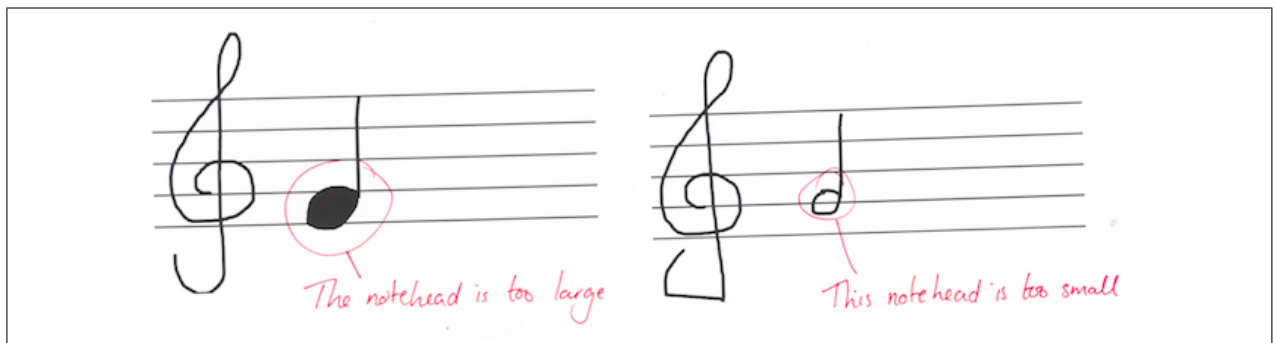


Figure 2.8: Note heads should be one staff space in size and not extend beyond a pair of staff lines

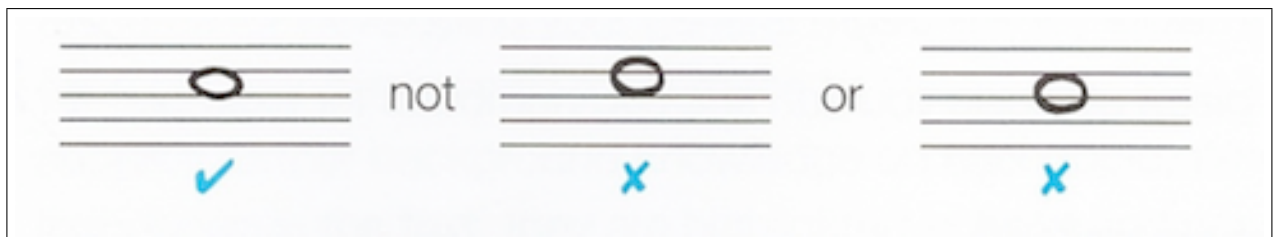


Figure 2.9: If it is drawn in the space, it should only cover half the space on either side

Similar rules govern the placement of crotchet and minim heads as seen in Figure 2.10.

2.2. COMMON NOTATION MISTAKES

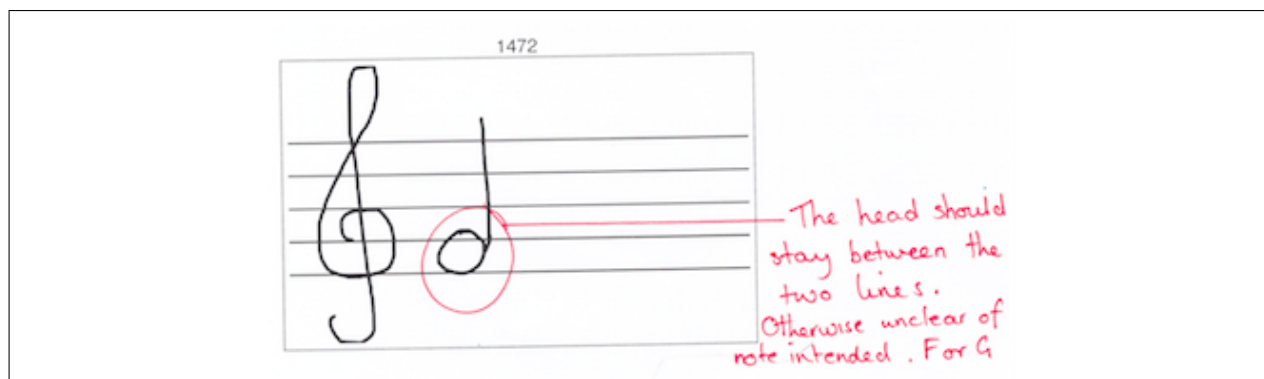


Figure 2.10: An example of an 'ambiguous' minim. It is placed both in a space and on a line

2.2.2.3 Broken Note Heads

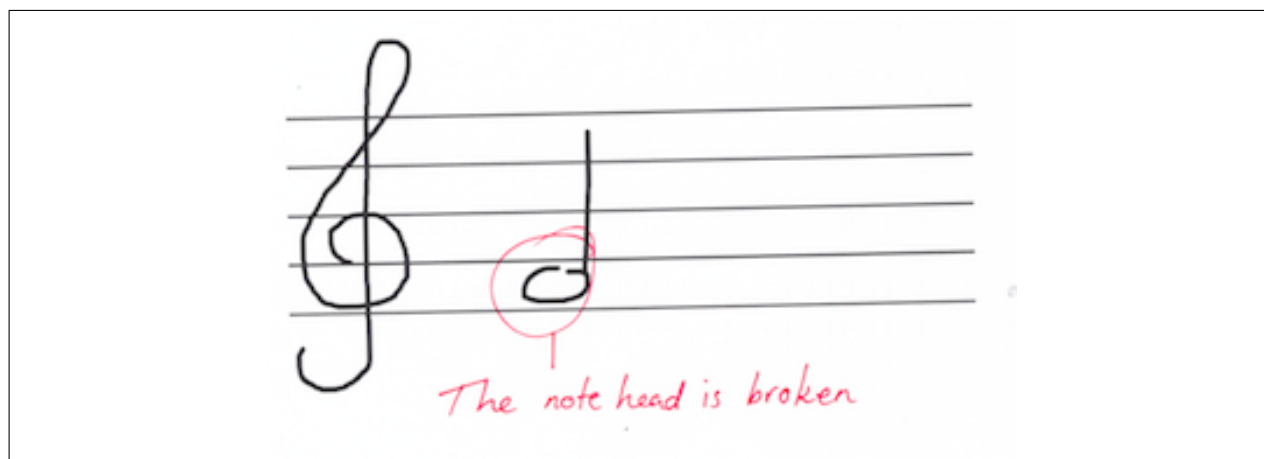


Figure 2.11: When writing semibreves and minims, it's important to form a full ellipse

2.2.2.4 Messy Note Heads

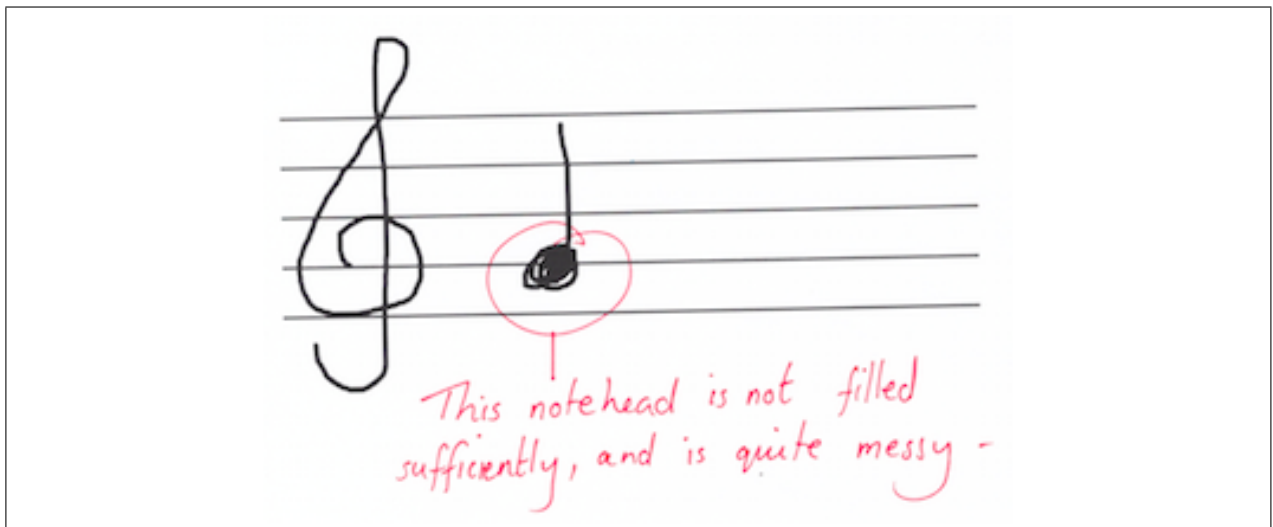


Figure 2.12: Several children drew solid noteheads rather haphazardly, leaving gaps and giving a messy scribbled effect to the note head

2.2.2.5 Angled Heads

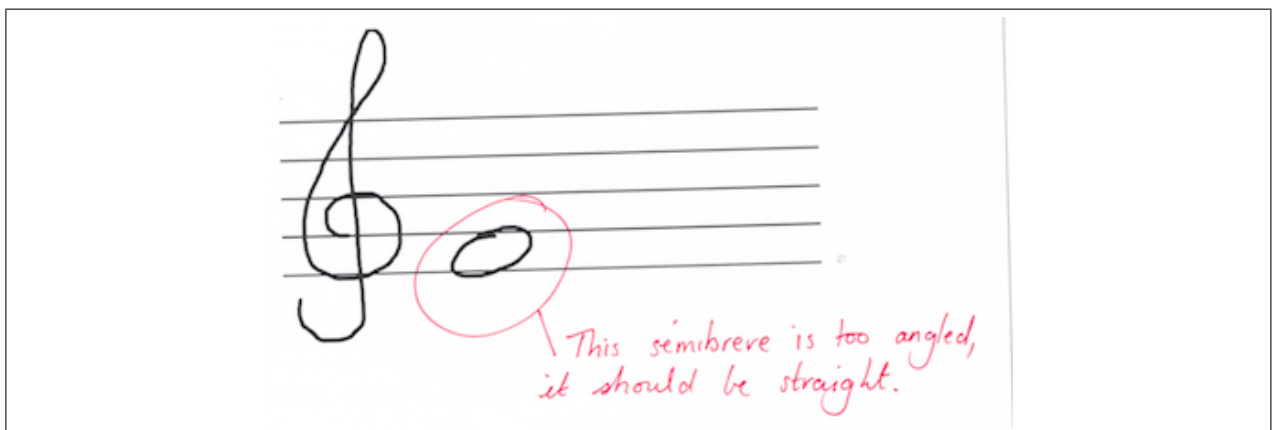


Figure 2.13: Semibreves are typically wider than minims so any deviation from horizontal appears more pronounced and should be avoided

2.2. COMMON NOTATION MISTAKES

2.2.3 Note Stems

2.2.3.1 Stem Angle

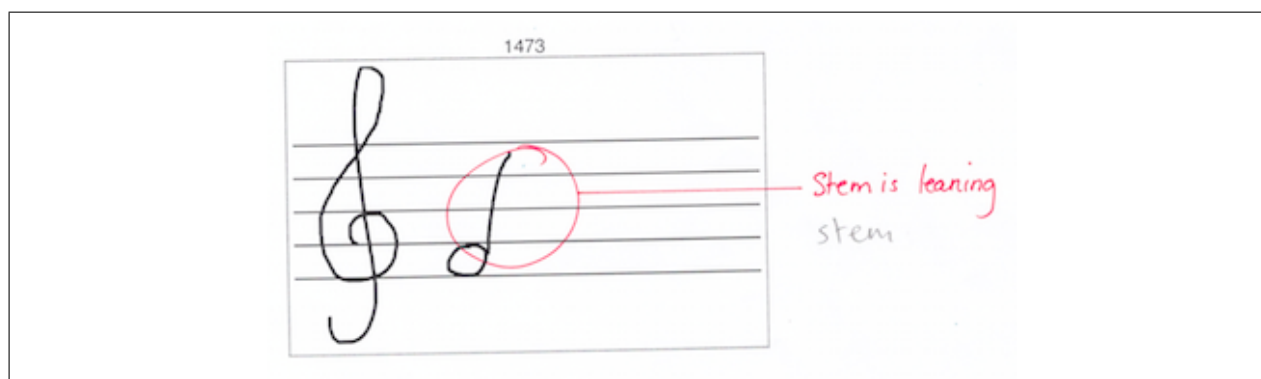


Figure 2.14: Stems should be vertical, unlike this one which is leaning too far to the right

2.2.3.2 Stem Straightness

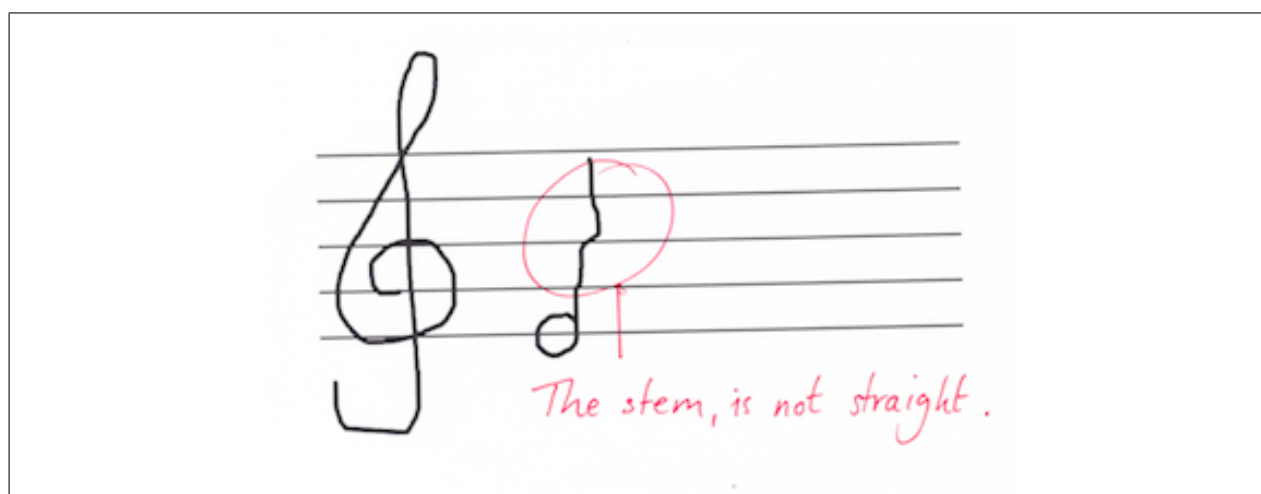


Figure 2.16

2.2.3.3 Stem Direction

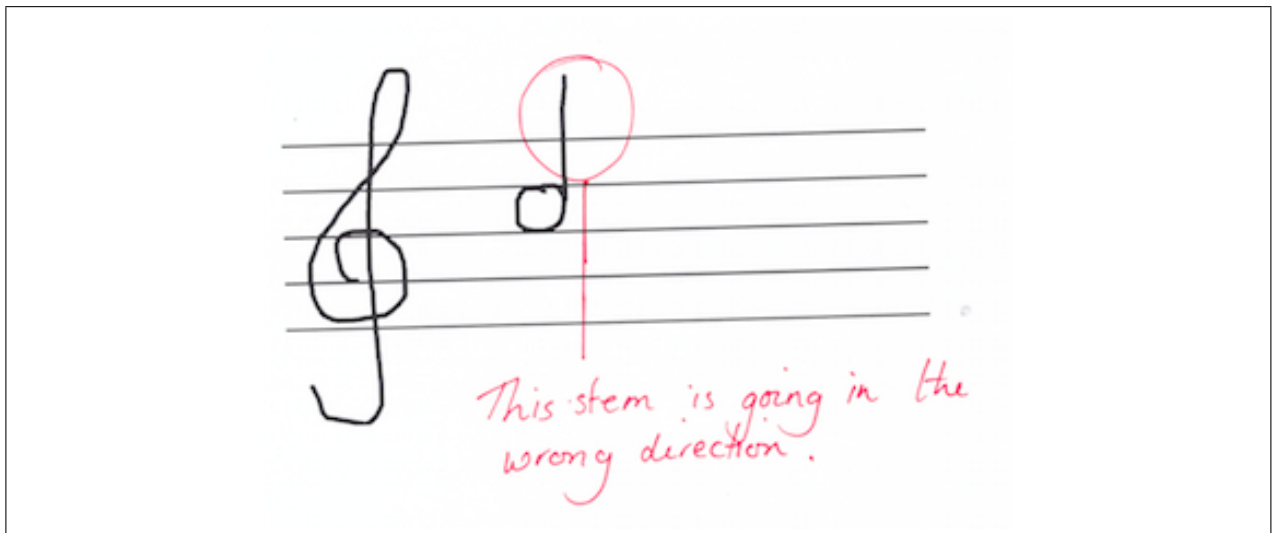


Figure 2.17: Stem direction is important and is based on which position on the staff the note head lies. You can see examples of which way stems should go in Figure 2.3

2.2.3.4 Stem Side

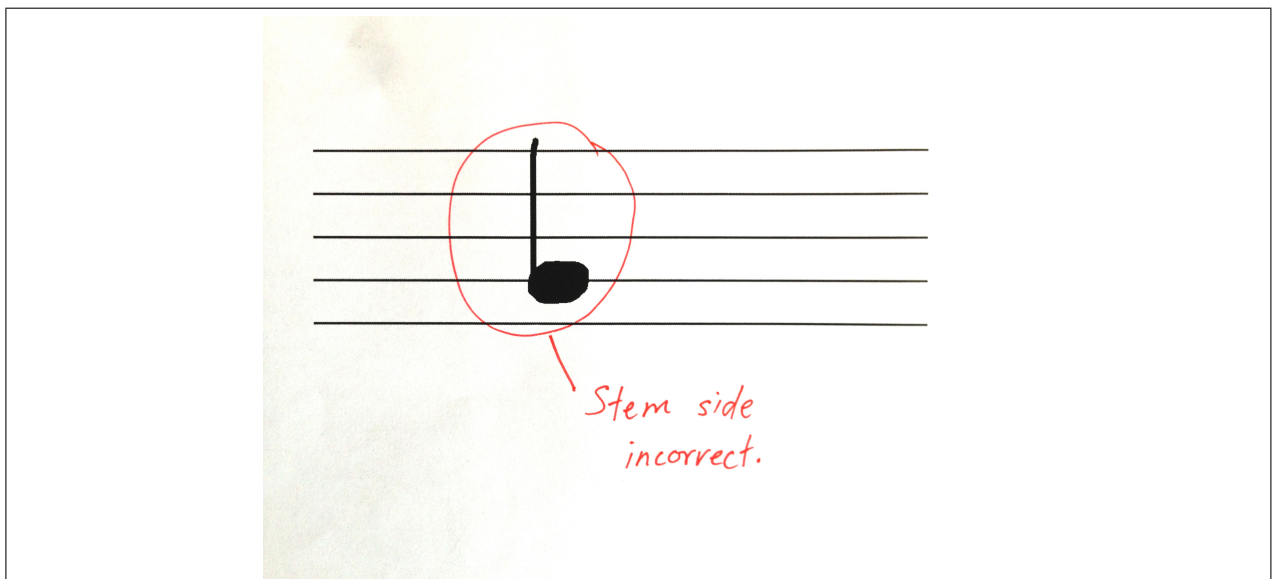


Figure 2.18: Again, as you can see from Figure 2.3 depending in the direction of the stem vertically, the side on which the stem goes also changes

2.2. COMMON NOTATION MISTAKES

2.2.3.5 Stem Length

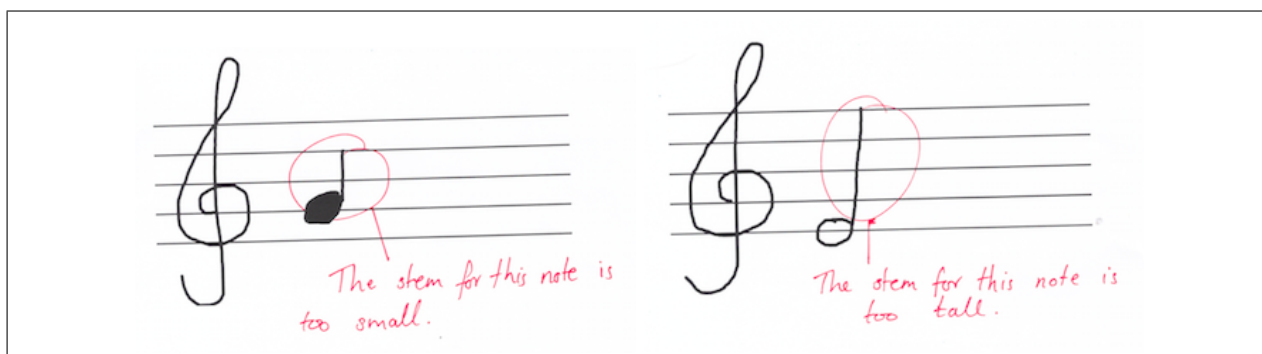


Figure 2.19: Here you see examples of a stem which is too short and another which is too long. Stems should be between 2 and 3 staff spaces in length

2.2.4 Quaver Tails

2.2.4.1 Quaver Tail Side

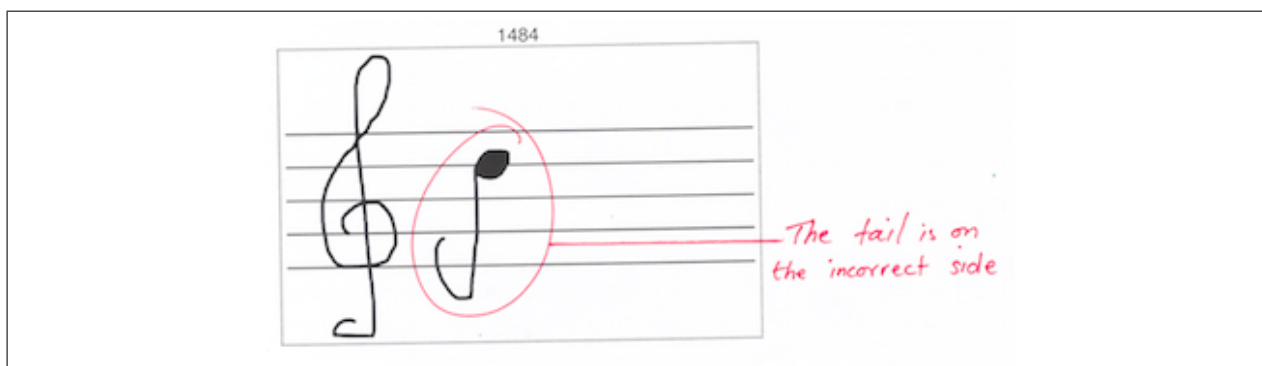


Figure 2.20

2.2.5 Rests

2.2.5.1 Position

Most rests should be centred around the middle staff line. However minim and semibreve rules differ slightly. Minim rests are placed sitting on top of the middle line and semibreve rests are placed sitting below.

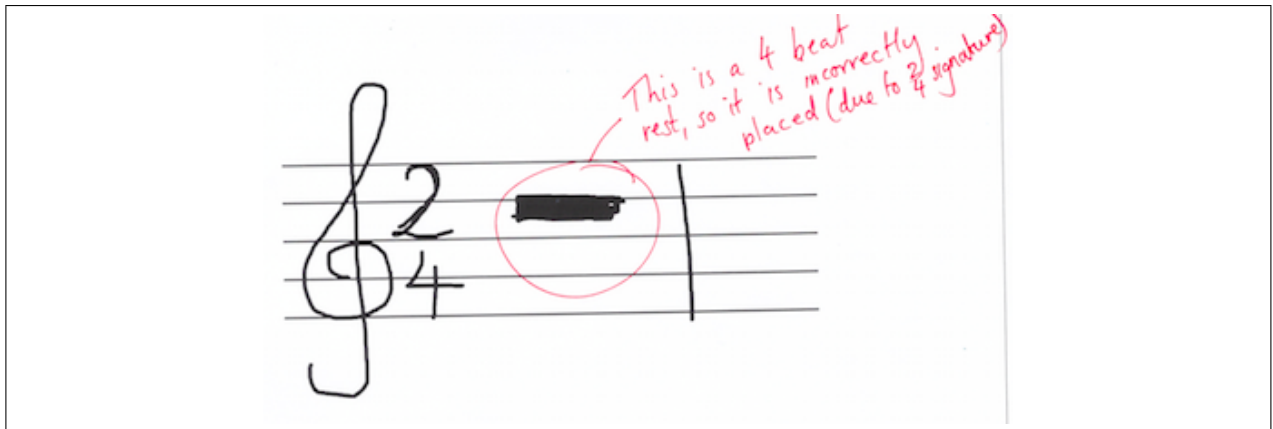


Figure 2.21

2.2.6 Duration Dots

2.2.6.1 Wrong Side

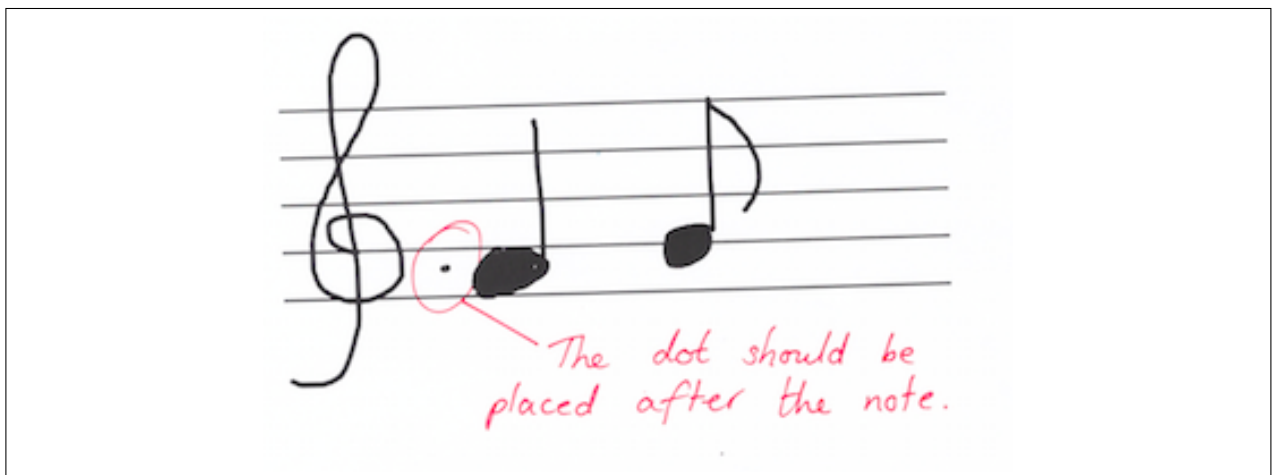


Figure 2.22: Duration dots should only be placed after the note

2.2. COMMON NOTATION MISTAKES

2.2.7 Accidentals

2.2.7.1 Ambiguous

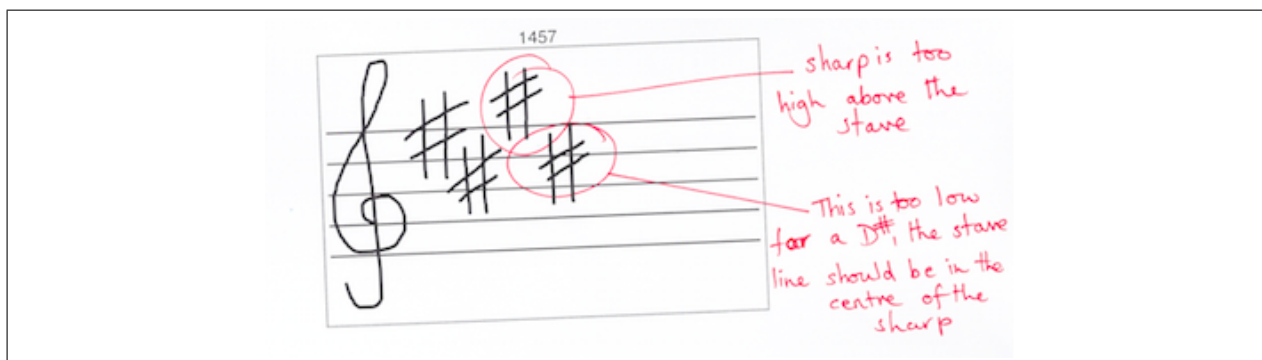


Figure 2.23

2.2.7.2 Wrong Side

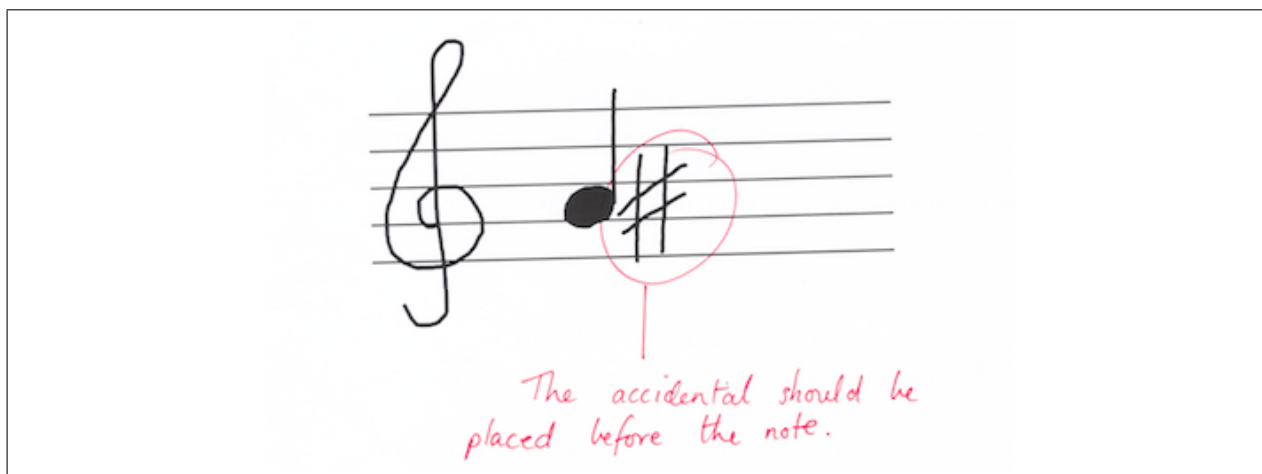


Figure 2.24: Accidentals should always appear *before* a note

2.2.7.3 Wrong Line

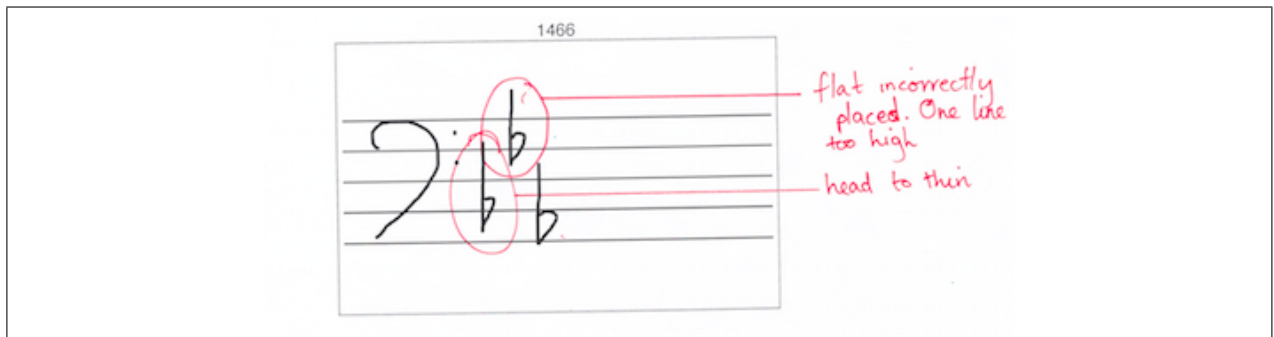


Figure 2.25

2.2.8 Key Signatures

2.2.8.1 Wrong Octave

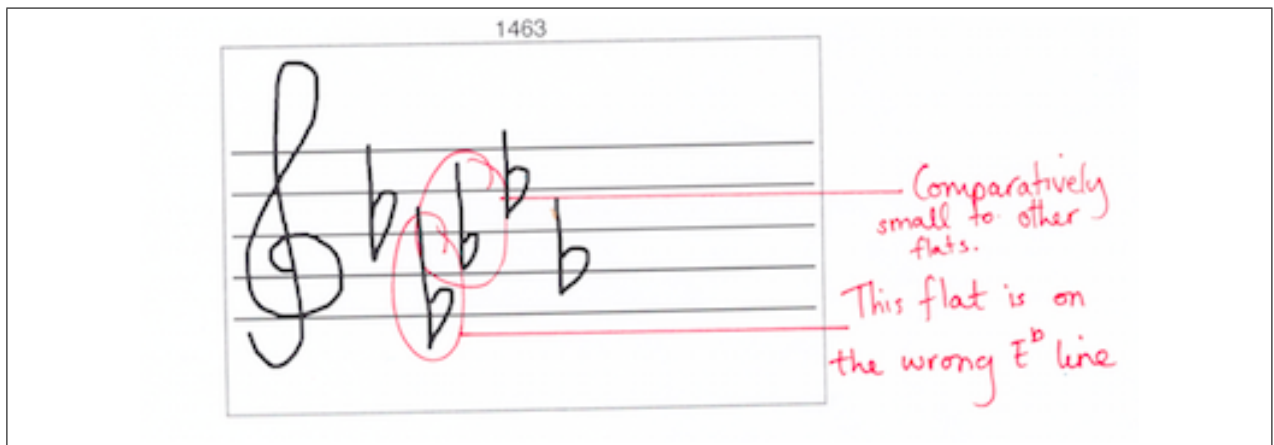


Figure 2.26

2.2. COMMON NOTATION MISTAKES

2.2.8.2 Incorrect Order

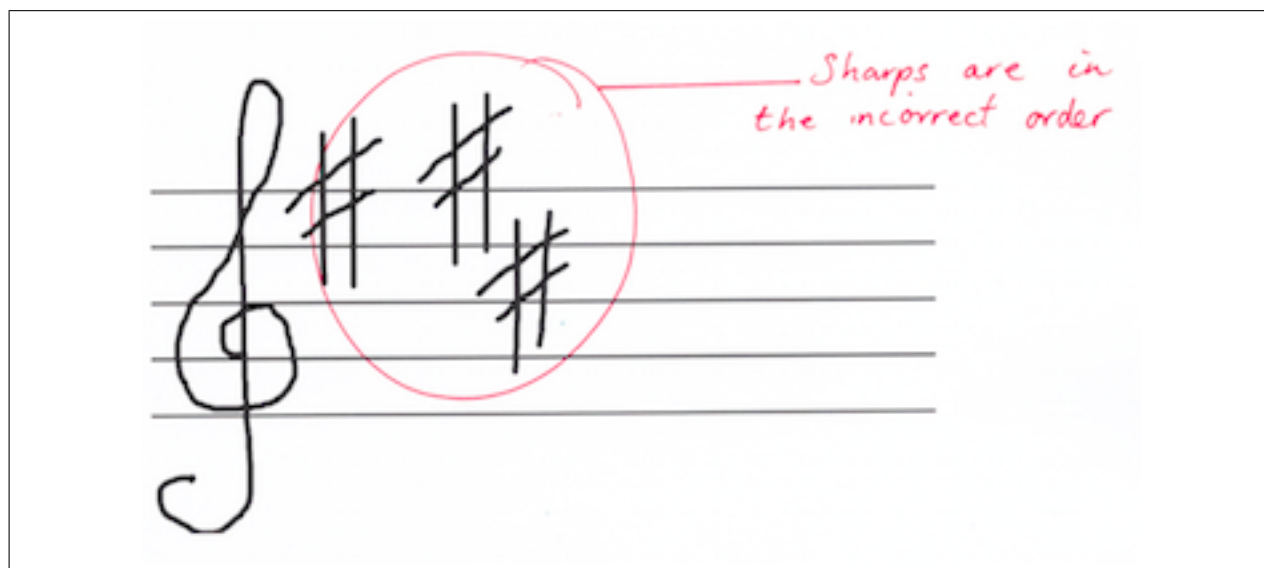


Figure 2.27

2.2.8.3 Incorrect Pitch

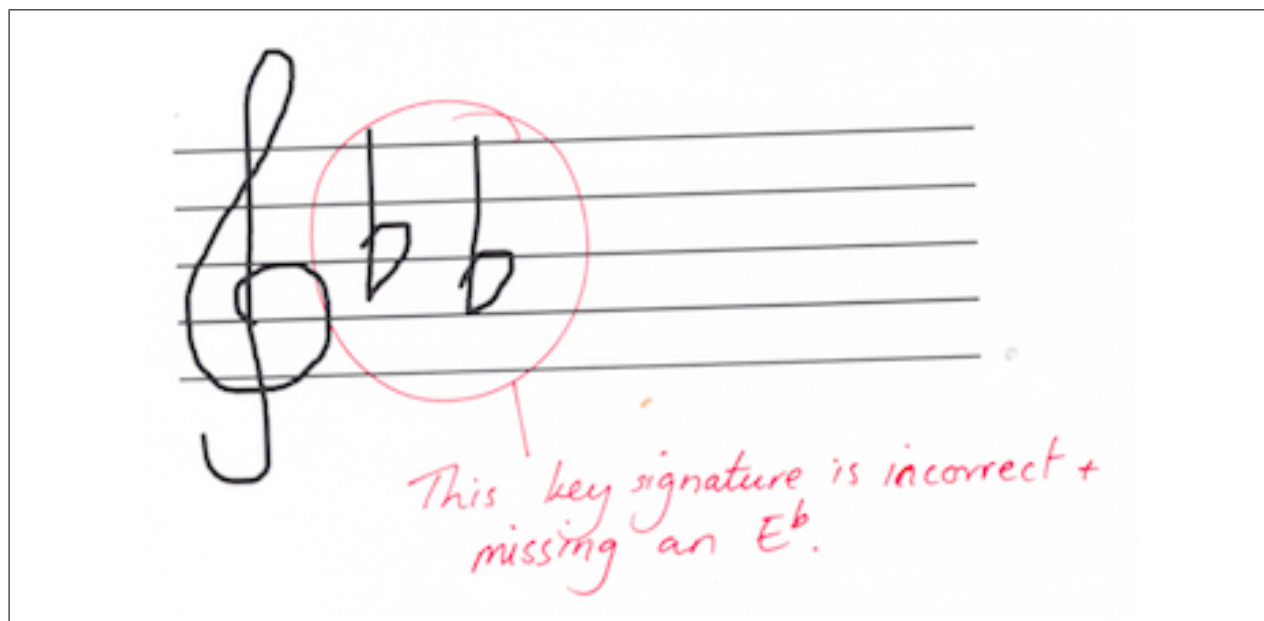


Figure 2.28

2.2.9 Beats and Timing

2.2.9.1 Too Many Beats

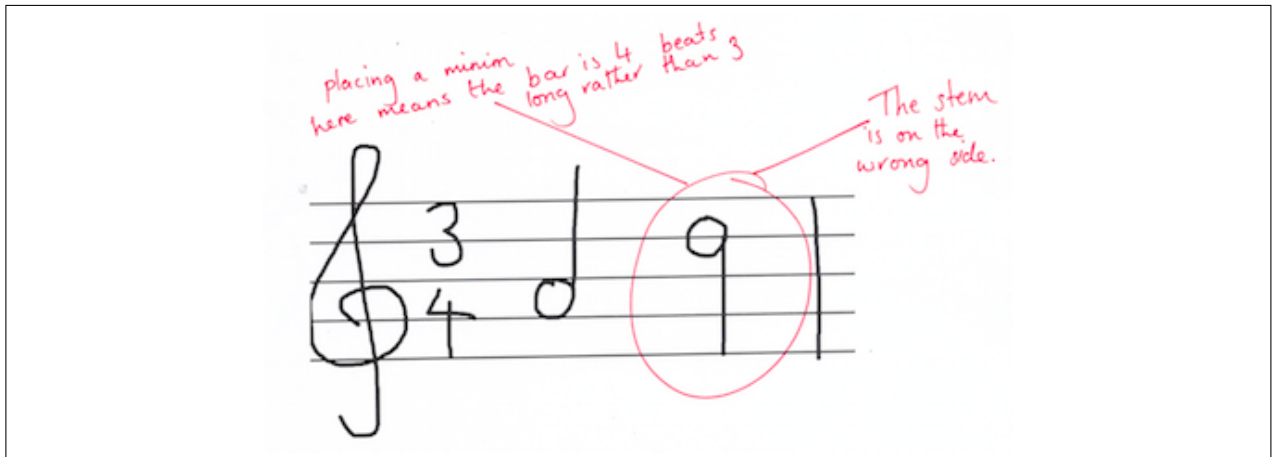


Figure 2.29

2.2.9.2 Too Few Beats

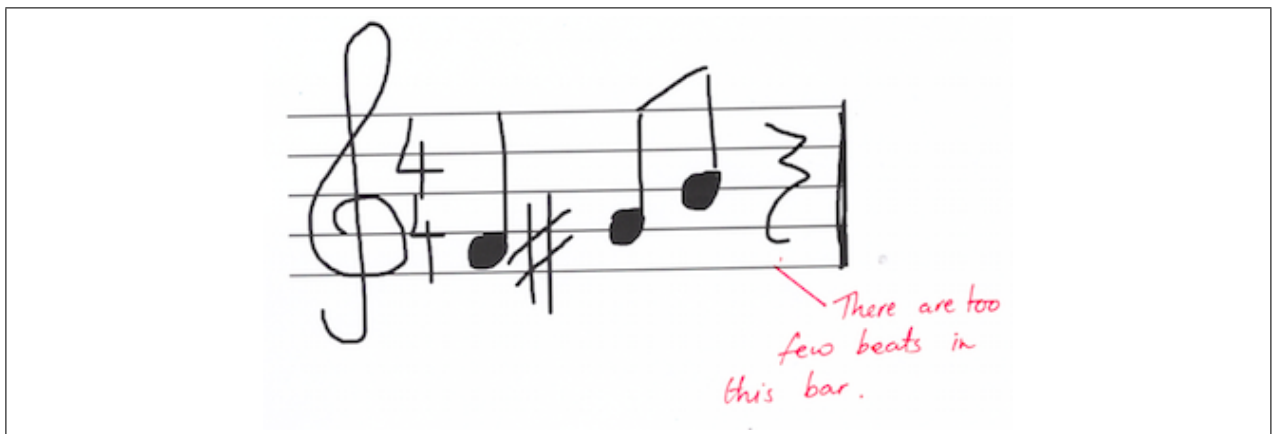


Figure 2.30

2.3 Existing Music Theory Resources

2.3.1 Mobile Apps

In order to establish what kinds of applications might exist in and around this music recognition and correction field, I researched online. The Apple “App Store” revealed

2.3. EXISTING MUSIC THEORY RESOURCES

several examples of applications which teach music theory, however none of these focus on writing music and are more based around ideas like flashcards and memorisation.

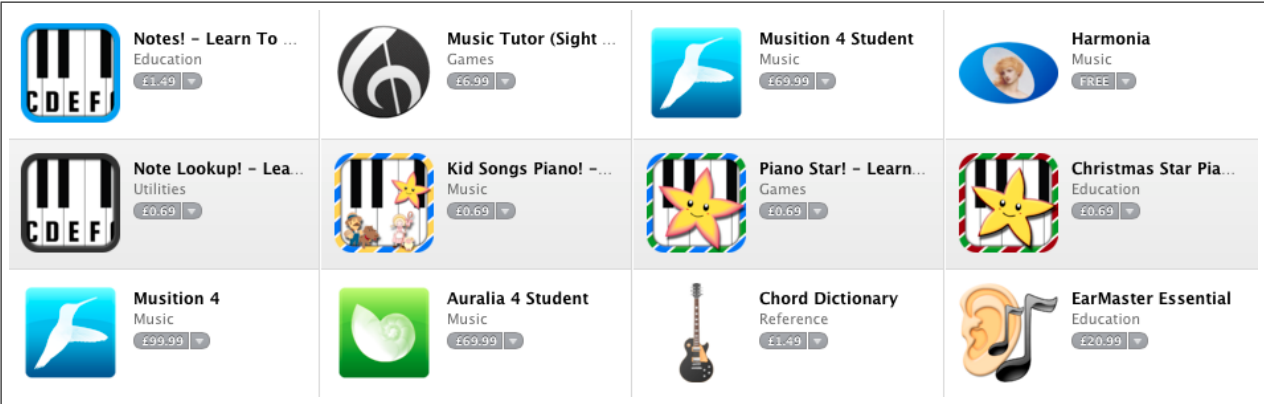


Figure 2.31: Music Theory Applications after searching “Music Theory” on the Apple App Store

Most of these apps however, are aimed at more advanced students and miss out a huge part of theory exams which is the written notation section so I won’t focus on them more in this project.

2.3.2 Theory Workbooks

The traditional teaching method typically uses template printed workbooks in which students write the answers to a series of exercises. The following scans provide examples of the kinds of exercises and lessons that the application might aim to replicate to be capable of in the long term.

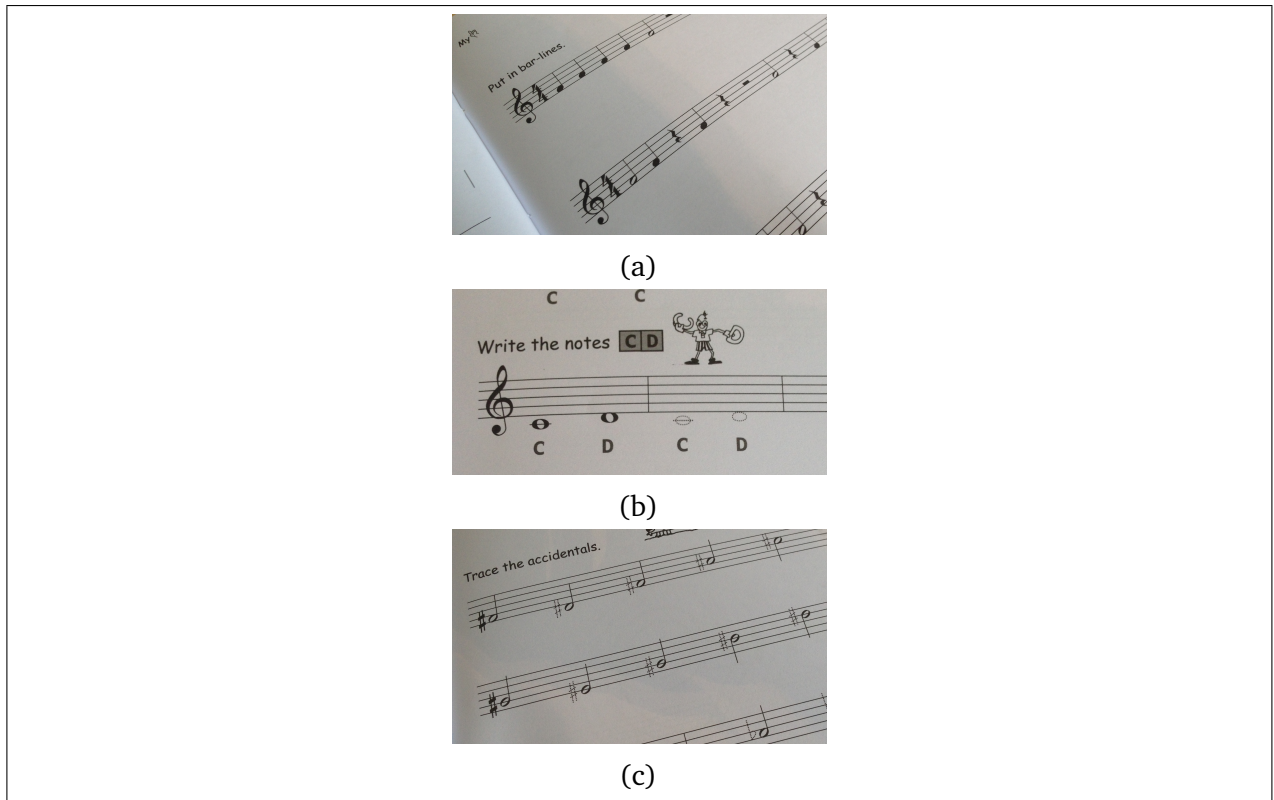


Figure 2.32: Examples of exercises in fundamental music theory, taken from Lina Ng 2001

From a wide list of potential exercises, I have narrowed down a small list of sample exercises types suitable for a student learning music theory at the equivalent of Grade 2-4.

2.3. EXISTING MUSIC THEORY RESOURCES



Figure 2.33: Some of the books on music theory targetted at the right age group

2.3.2.1 Taylor, E.R. and Associated Board of the Royal Schools of Music 1989

The AB Guide to Music Theory isn't specifically focussed around handwritten exercises but it gave a great reference for the more fundamental theory which I found very helpful whilst writing Section 2.1 as it give good examples on note divisions (Figure 2.34).

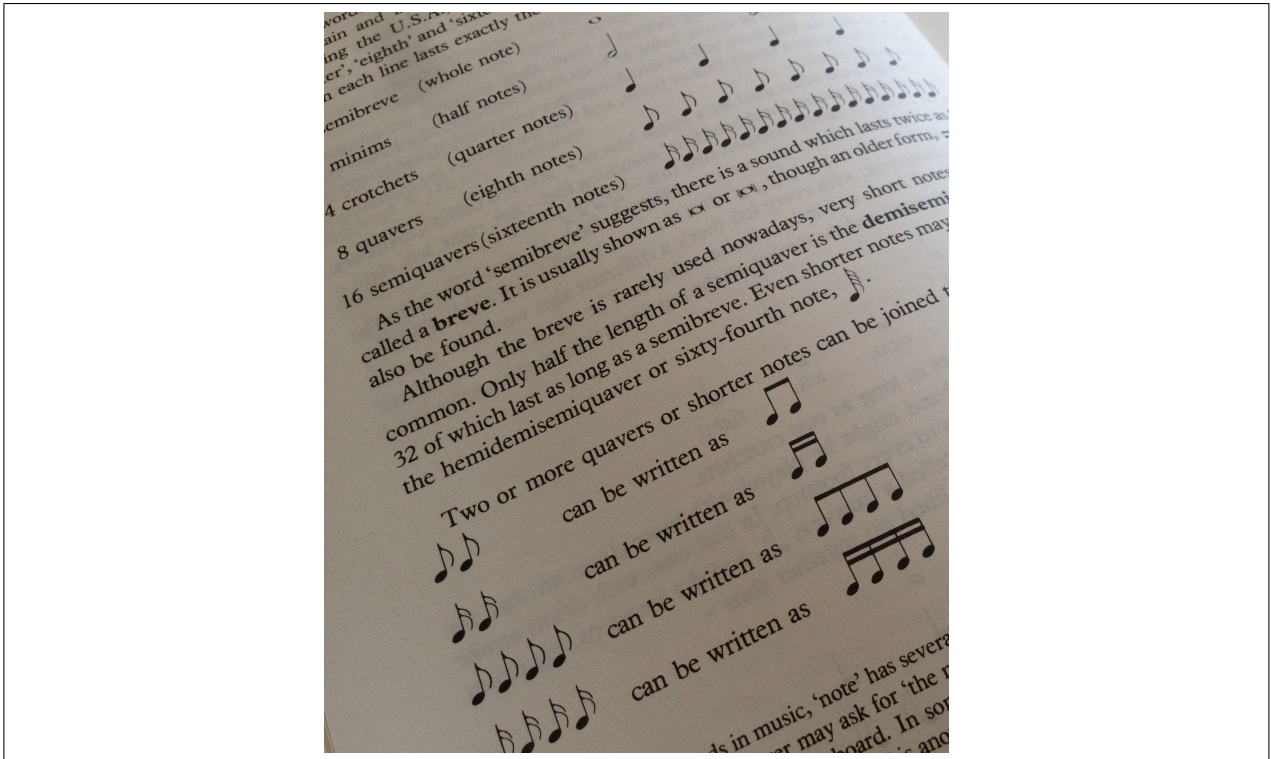


Figure 2.34: The AB Guide to music theory reminding me of how divisions and note beaming work

2.4 OMR Applications

This project heavily grounded technically in the field of [Optical Music Recognition \(OMR\)](#). Prior to embarking on a solution from scratch, research was done to gather an understanding of the existing OMR landscape and whether any tools already existed. Several software solutions were found which can assist in taking a printed score and process it to create a digital representation. However, the majority were commercial applications which required purchasing and almost all were designed to be used under highly supervised conditions inside a [Graphical User Interface \(GUI\)](#), doing an initial conversion and then guiding the user through the process of validating the application's 'best guess'.

Another issue with these applications in relation to **NoteED** is that most of them assume you just want accurate classification and you therefore lose the ability to examine and interact with the underlying components. Instead you just get out a MIDI file, MusicXML or another format.

2.4.1 Neuratron Photoscore

Neuratron Photoscore¹ is one of the market leading packages, features tight integration with several composition tools like Sibelius and Finale, outputs to a range of formats (MusicXML etc). Designed to be run by the end user in supervised conditions, it enables you to scan a handwritten musical sheet, interprets what you intended, and enables you to correct errors as you go.

2.4.2 Audiveris

“Audiveris is an open-source Optical Music Recognition software which processes the image of a music sheet to automatically provide symbolic music information in MusicXML standard.”²

At present it only supports high quality printed scores and operates by utilising a neural network which must be trained on samples provided by the end user.

2.4.3 Gamera

Gamera³ is primarily a structured document processing and symbol recognition tool (Mac-Millan, Droettboom and Fujinaga 2002) and was spun out of one of the authors’ previous thesis which focussed on OMR (Fujinaga 1996). Primarily used for academic purposes, one of the overlapping areas of focus of the NoteEd project is some of its supporting research into stave detection and removal⁴.

2.4.4 Capella Scan

As with most of the other applications mentioned, Capella Scan⁵ is primarily designed to convert old or dated music manuscripts into a more “preservable” digital format without needing to manually type all the music out again. However by the authors’ own admission it falls down slightly when it comes to handwritten music.

¹<http://www.neuratron.com/photoscore.htm>

²<https://audiveris.kenai.com/>

³<http://gamera.informatik.hsnr.de>

⁴<http://gamera.informatik.hsnr.de/addons/musicstaves/>

⁵<http://www.capella.de/us/index.cfm/products/capella-scan/info-capella-scan/>

Chapter 3

Technical Research

3.1 Previous Works

3.1.1 Taubman, Odest and Jenkins 2005

In this paper, Taubman attempts to improve on notation input for professional musicians, allowing them to input rough “sketched” music and convert it into a neatly printed form. The application created, “MusicHand” makes use of a graphics tablet connected to a desktop machine and records the individual strokes the user makes.

Strokes are captured individually but using a latency threshold which is then used to group the strokes together, adjusted on a per user basis. Taubman then makes use of statistical moments to classify these stroke groups, then making use of a K-NN algorithm to classify the object as one of several musical entities using the generated moments.

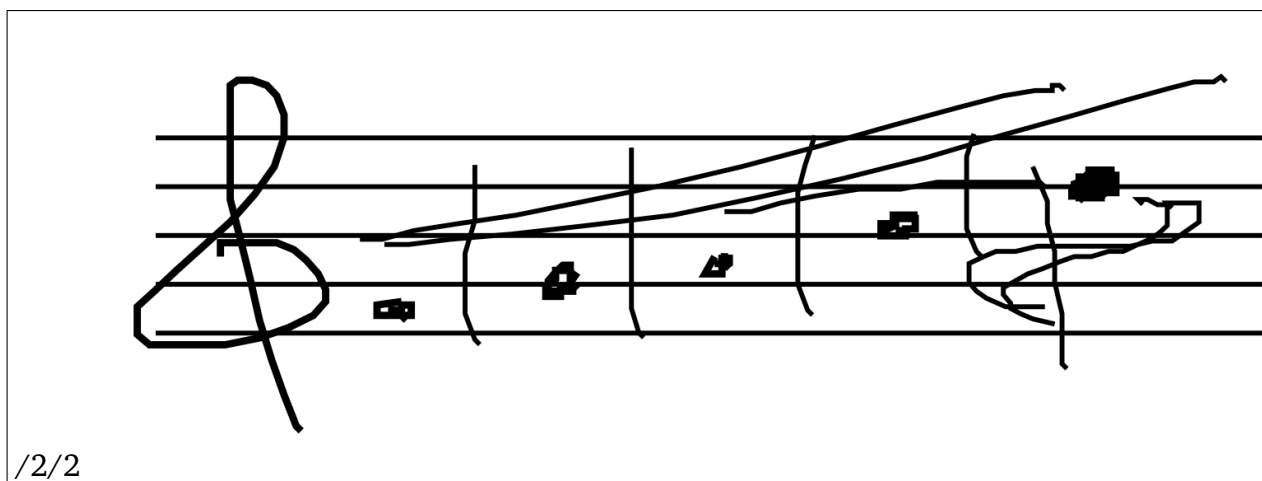


Figure 3.1: Example notation sketch from Taubman, Odest and Jenkins 2005

Since strokes are generally sketched as seen in Figure 3.1 the application also makes use of a great deal of domain expertise, covered further in Section 4.2.6, essentially running through a large flow chart in order to classify the entities further. However, the system notably relies on the compatibility of the generated moment “libraries” between users, a study of which was recommended for any future work but not done in this case. It is also suggested that instead of a graphics tablet, one might experiment with a more direct input method such as drawing onto a tablet directly with a stylus, the author specifically mentions the “Wacom Cintiq”¹, a “digital canvas” of sorts, which might allow a much more natural pen-paper feel.

3.1.2 Ben-Dayan and Giloh 2013

An interesting component of the work by Ben-Dayan and Giloh was their experiments with stem, note head and symbol detection during the segmentation phases.

3.1.2.1 Stem Detection

For stem detection, since handwritten stems are unlikely to be perfectly straight, the authors use vertical projections combined with a high pass filter to establish likely stems. These are represented by peaks in the vertical projection.

¹<http://uk.shop.wacom.eu/products/cintiq>

A bounding box is then generated and the note is split into an upper and lower region; the upper and lower parts being used in note head and beam detection later.

3.1.2.2 Note Head and Features Detection

The author extracts the head position by way of a distances transform and then examining the density of the pixels (distance from the nearest black pixel) the results of which you can see graphically in Figure 3.2. The authors also make several assumptions at this stage regarding the size of features (e.g. sharps are less than twice the stave space height) which enables them to separate notes from other components (clefs, accidentals etc).

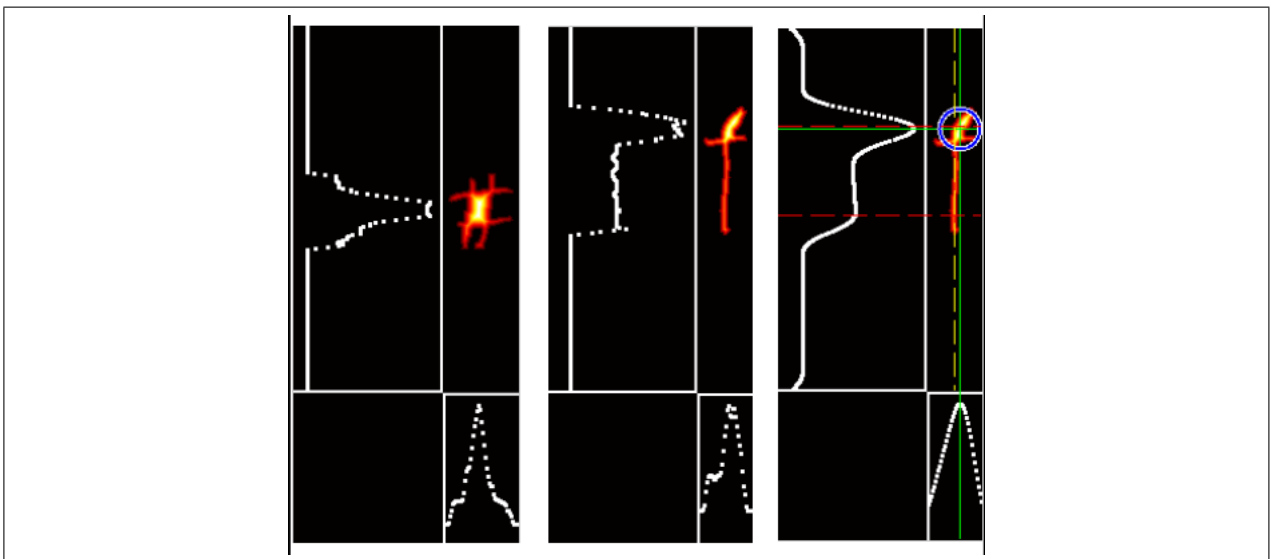


Figure 3.2: Finding component centres, (Ben-Dayan and Giloh 2013)

3.1.3 Rebelo et al. 2011

In this paper the authors discuss the use of domain knowledge in musical classification. In general the NoteEd project can't rely purely on standard musical conventions due to the potential mistakes which students are likely to make.

However, the flow diagram reproduced in Figure 3.3 which represents their musical extraction algorithm using domain knowledge provides an excellent reference for the order in which musical properties could be identified.

3.2. GENERATING MANUSCRIPTS

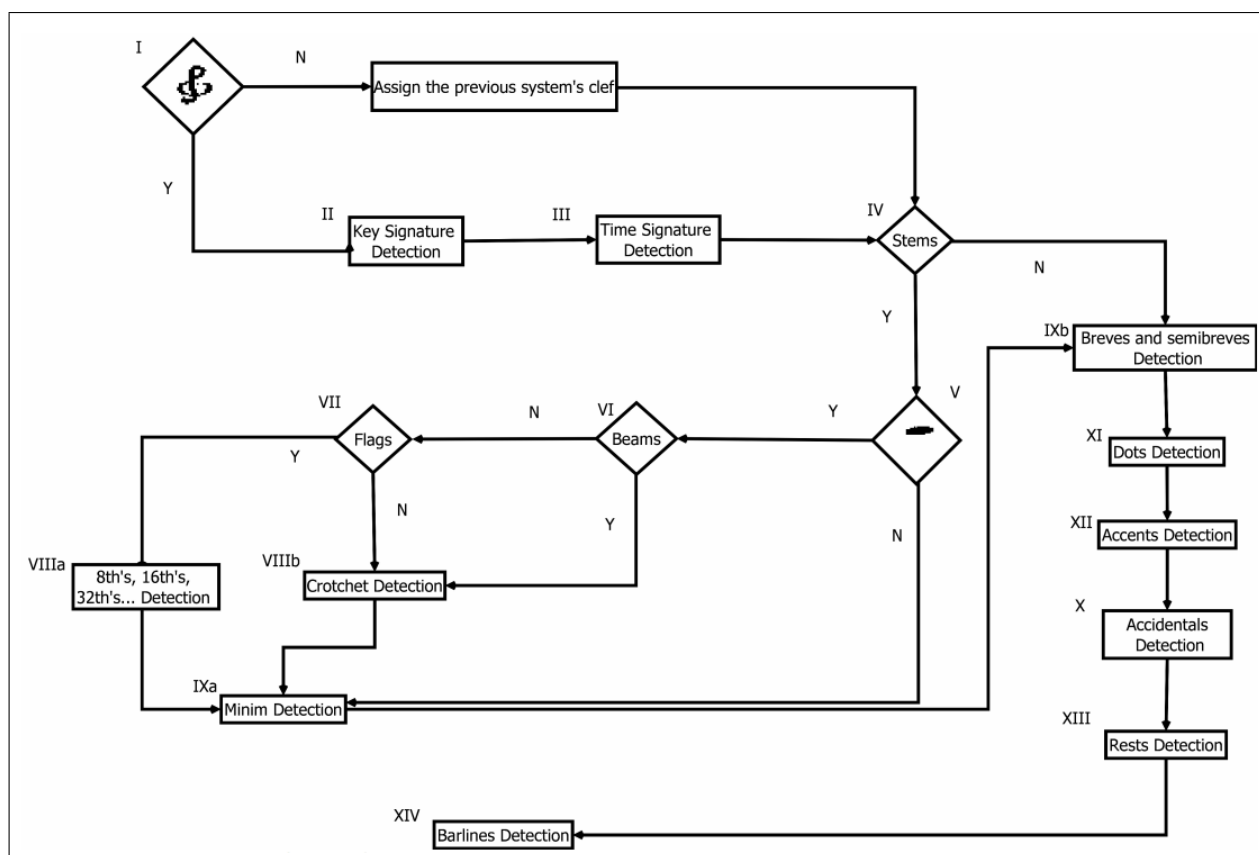


Figure 3.3: Musical symbols extraction algorithm from Rebelo et al. 2011

3.2 Generating Manuscripts

In order to produce enough musical examples and references for a student to use, some automated way of producing the reference manuscript is needed.

3.2.1 Professional GUI Tools

There are several professional tools which are used in industry to generate musical scores on the computer. The ones with most widespread usage are Sibelius² and Finale³ and more recently, NoteFlight⁴

²<http://www.avid.com/US/products/SibeliusFirst/overview>

³<http://www.finalemusic.com/products/finale/>

⁴<http://www.noteflight.com/>

They're worth mentioning as they're the "industry standards" for musical notation and composition, used by professionals and in education around the world, but their primary use case is manual input via a GUI so for our purposes they are not ideal.

3.2.2 LilyPond

LilyPond⁵ is a music engraver and serves as a 'modular, extensible and programmable compiler for producing high-quality music notation' (Nienhuys and Nieuwenhuizen 2003). Originally inspired by the efforts of projects like MusiXTeX⁶ which had aimed to 'be able to typeset complex polyphonic, orchestral or instrumental music' (Taupin 1999) in the same way that it was already renowned for beautifully typeset text and maths. It's a widely adopted tool but its flexibility with regard to formatting makes it difficult to learn.

The idea of Lilypond is that by entering or programmatically generating a formal representation of music which is designed to be easy to type, you can then use LilyPond to produce a manuscript engraving from that representation. Lilypond also supports conversion from other popular text-based music formats such as MusicXML⁷, or ABC⁸.

For example, given the following LilyPond syntax:

```
\relative c' {  
  c' d' e' f' g'2 g'2  
}
```

We can run LilyPond to produce the output you see in Figure 3.4

Using LilyPond and some basic algorithms around music theory we could easily generate textual representations of exercises and generate the necessary images from them. It's also free and accessible meaning that it can be easily installed on development machines and servers.

⁵<http://www.lilypond.org/>

⁶http://www.mab.jpn.org/musictex/musixtex_e.html

⁷Some good examples can be found at <http://www.musicxml.com/tutorial/hello-world/>

⁸Standards can be found at <http://abcnotation.com/>

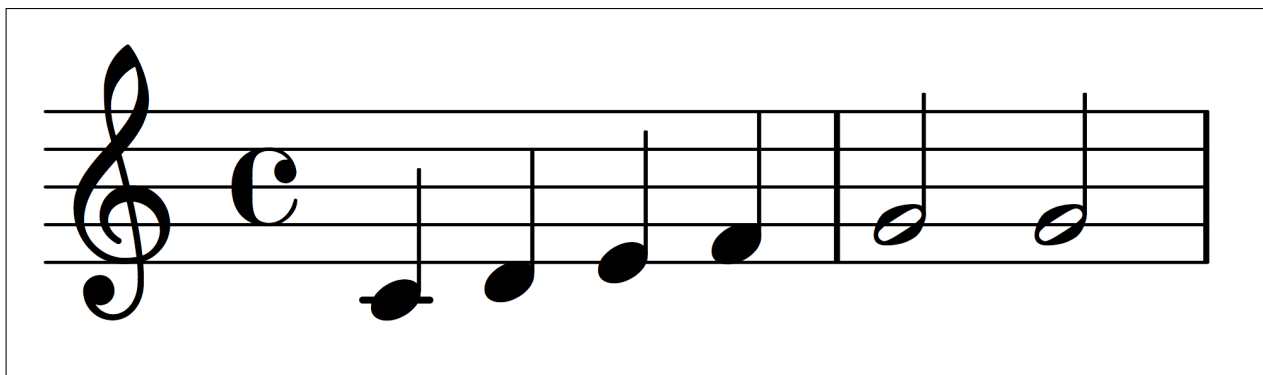


Figure 3.4: Lilypond Output Example

3.2.3 VexFlow

Vexflow⁹ is another music engraving application, but Vexflow is web based and makes use of HTML5 Canvas¹⁰ and SVG¹¹ so it can be used to generate manuscripts on the fly in a browser.

For example if you were to include the Vexflow library and then write the javascript outlined in Section 3.2.3 you will get the result in Figure 3.5

```
var canvas = $("#div.one div.a canvas")[0];
var renderer = new Vex.Flow.Renderer(canvas,
    Vex.Flow.Renderer.Backends.CANVAS);

var ctx = renderer.getContext();
var stave = new Vex.Flow.Stave(10, 0, 500);
stave.addClef("treble").setContext(ctx).draw();
```

An advantage Vexflow has over [LilyPond](#) is that with LilyPond, for use on the web, we would need to render the music remotely, then synchronously or asynchronously transport those resources (most likely images, which are expensive for web traffic) to the client. The downside of client side rendering is of course that for anything complex, since it increases the load on the end device, it's important to be aware on which devices the code will primarily be run to ensure a good user experience.

⁹<https://github.com/Oxfe/vexflow>

¹⁰<https://developer.mozilla.org/en/docs/HTML/Canvas>

¹¹http://en.wikipedia.org/wiki/Scalable_Vector_Graphics



Figure 3.5: Vexflow render, see <http://vexflow.com/docs/tutorial.html>

3.3 OMR Architecture

In general, the challenge of OMR can be decomposed into more defined sections. Although implementations obviously differ, an overview of some of the techniques I have come across for the various ‘common’ stages which I researched can be summarised by:

1. Pre Processing	2. Segmentation	3. Classification	4. Reconstruction
1. Level Adjustment 2. Binarization 3. Noise Removal 4. Handling Skew 5. Rotation 6. Stave Removal <ul style="list-style-type: none"> (a) Horizontal Projections (b) Hough Transforms 	1. Projections 2. Template Matching 3. Connected Components	1. K Nearest Neighbour (KNN) 2. Neural Networks 3. Support Vector Machines 4. Statistical Moments	1. Simple Heuristics 2. Grammar

Table 3.1: Typical OMR Stages

3.4 Segmentation

Segmentation is the challenge of taking a given image or scene and extracting individual objects or ‘components’.

Typically this can be a complex problem to tackle as stafflines connect almost all components together and must therefore first be removed. Also some musical entities are themselves comprised of multiple other entities. Good examples are the bass clef which is comprised of two dots placed above each other to the right of the curve (Section 2.1.5). Dotted notes would be another example (Table 2.1).

Particularly within the **NoteED** project, we also need to be able to break down the musical entities further into stems and note heads and so separating these from each other presents another problem which we must take into account.

3.4.1 Connected Component Analysis

Connected component analysis is a technique used to establish distinct regions within an image. An individual pixel in a binary image can possess two forms of connectedness, *4-neighbour* or *8-neighbour*. It should be noted that this evaluation only takes place for pixels with a value of 1 or “filled pixels”.

More formally, “A connected component labelling of a binary image B is a labeled image LB in which the value of each pixel is the label of it’s connected component” (Shapiro and Stockman 2001, pg 69)

To be **4-neighbour**, at least one of the pixels above, below or to the left or right (which we can refer to as the vertical and horizontal neighbours) of the pixel under investigation must have a value of 1.

In a similar fashion, an **8-neighbour** pixel is one where any of the surrounding 8 pixels (vertical, horizontal or diagonal neighbours) has a value of 1.

Regions for 4-neighbour and 8-neighbour connected pixels are outlined in Figure 3.6

There are two primary algorithms for establishing connected regions, the first is recursive and the second requires two scans.

3.4.1.1 Recursive Labelling

If we assume that the size of the image to be evaluated is small and we can fit it in memory (a reasonable assumption give the scope of the project and the hardware available) we can employ a recursive algorithm which can grow regions by visiting any pixel in the image



Figure 3.6: 4 and 8 Neighbour Regions and examples of connected pixels

using depth first or breadth first searching. An outline for a depth first algorithm is given in algorithm Listing 3.1

Listing 3.1: Recursive Connected Component Labelling (DFS)

```

let img be the binary image
let lblimg be the labelled image

lblimg = negate(image)
lbl = 0

label_components():

```

3.4. SEGMENTATION

```
for i = 0 to height:
    for j = 0 to width:
        if lblimg[i, j] == -1:
            lbl += 1
            label\_region(i, j)

label\_region(i, j):
    lblimg[i, j] = lbl
    for (i', j') in neighbours(i, j):
        if lblimg[i', j'] == -1:
            label\_region(i', j')
```

3.4.1.2 Two Pass Labelling

An alternate algorithm involves performing the labelling in two passes. Assuming 8-neighbour connectedness and that we will most likely be scanning left to right we inspect the 3 pixels above and the pixel to the left of the current pixel as seen in Figure 3.7.

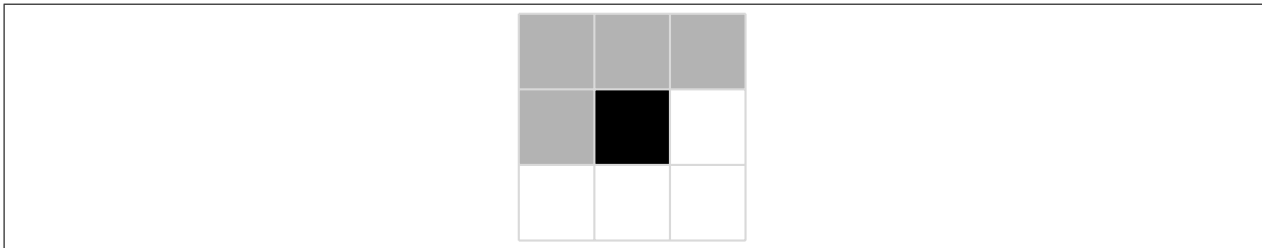


Figure 3.7: Pixels which are inspected during each row scan

We label each pixel according to these neighbours (if neighbours have multiple labels we just pick any of them and record that they were adjacent) and then finally reduce the number of labels by merging adjacent labels. An visual example of the process can be seen in Figure 4.11

Listing 3.2: Iterative Two-Pass Connected Component Labelling

```
let img be the binary image
let lblimg be the labelled image

lblimg = negate(image)
lbl = 1
```

```

equivalent_labels = []

first_pass():
    for i = 0 to height:
        for j = 0 to width:
            if lblimg[i, j] == 1:
                labels = neighbour_labels(i, j)
                if size(labels) == 0:
                    lbl += 1
                    lblimg[i, j] = lbl
            else:
                lblimg[i, j] = labels[0]
                if size(labels) > 1:
                    equivalent_labels << labels

second_pass():
    for labelgroup in equivalent_labels:
        firstlabel = labelgroup[0]
        otherlabels = labelgroup[1..]

        for label in otherlabels:
            relabelpixels(label, firstlabel)

```

3.4.2 Projections

Projections are regularly used in OMR during the preprocessing stage to detect and remove staff lines (Rossant 2002), but can also be used during the segmentation stage. An example section of score is shown in Figure 3.9 along with its horizontal and vertical projections.

The technique essentially involves projecting the manuscript in the x and y axes, collecting the pixels in either individual pixel lines or buckets in order to help establish information about the image.

Mathematically, if the image is represented as a 1 bit (2 colour) image $I(x_{\max}, y_{\max})$ of width x_{\max} and height y_{\max} , let $p_{xy} \in 0, 1$ denotes the value for a specific pixel at row y column x .

3.4. SEGMENTATION

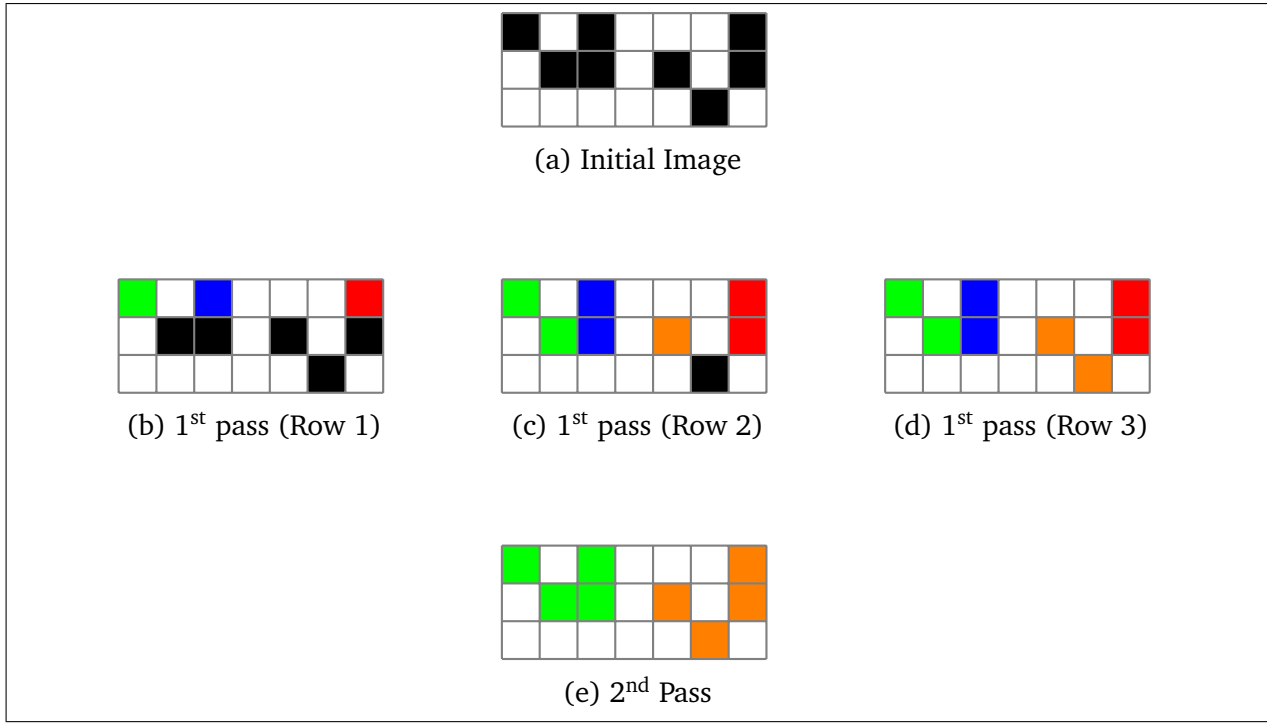


Figure 3.8: Step by step two pass connected component labelling

The horizontal and vertical projections can then be defined as:

$$P_{\text{horizontal}}(y) = \sum_{j=0}^{x_{\max}} p_{jy} \quad (3.1)$$

$$P_{\text{vertical}}(x) = \sum_{j=0}^{y_{\max}} p_{xj} \quad (3.2)$$

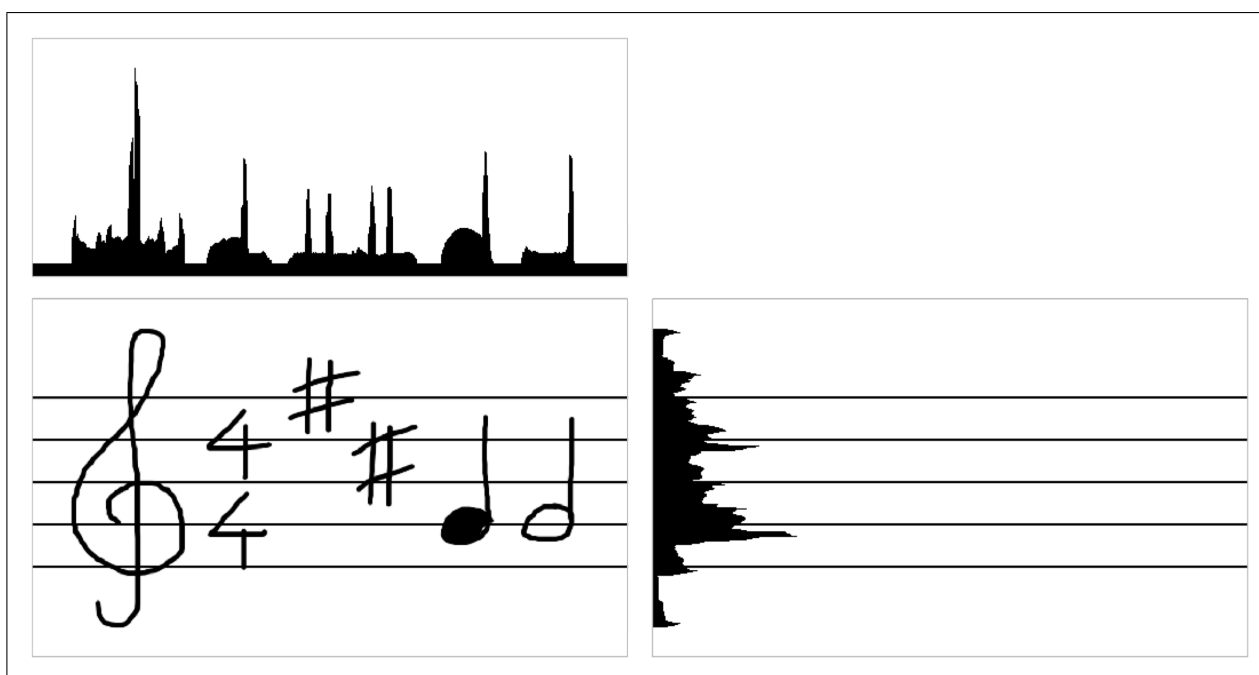


Figure 3.9: Horizontal and Vertical Projections of handwritten music excerpt

3.4.3 Template Matching

Image segmentation using template matching is not one of the main techniques used in segmentation due to its sensitivity to distortions but it has been the focus of a few OMR related papers. It receives particular attention in the segmentation phase in Rossant 2002 where different templates of reference components can be matched to the score, where a high correlation score in a certain position acts as both segmentation and classification.

In testing, I was able to reproduce this effect by extracting components like the note head from the whole note as seen in Figure Figure 3.10. However, as noted by Rossant this method is highly dependent on the font used in a printed score so for the purposes of handling handwritten notation it's unlikely to yield great results. Indeed, in preliminary testing, getting regular correct matches in a handwritten score proved troublesome.

More formally, for the example in Figure 3.10, template matching involves analysing each pixel in an image (or a region of an image) and comparing it to a reference pixel in a template image. The score for two $n \times m$ images x and y can be generated using Equation (3.3).

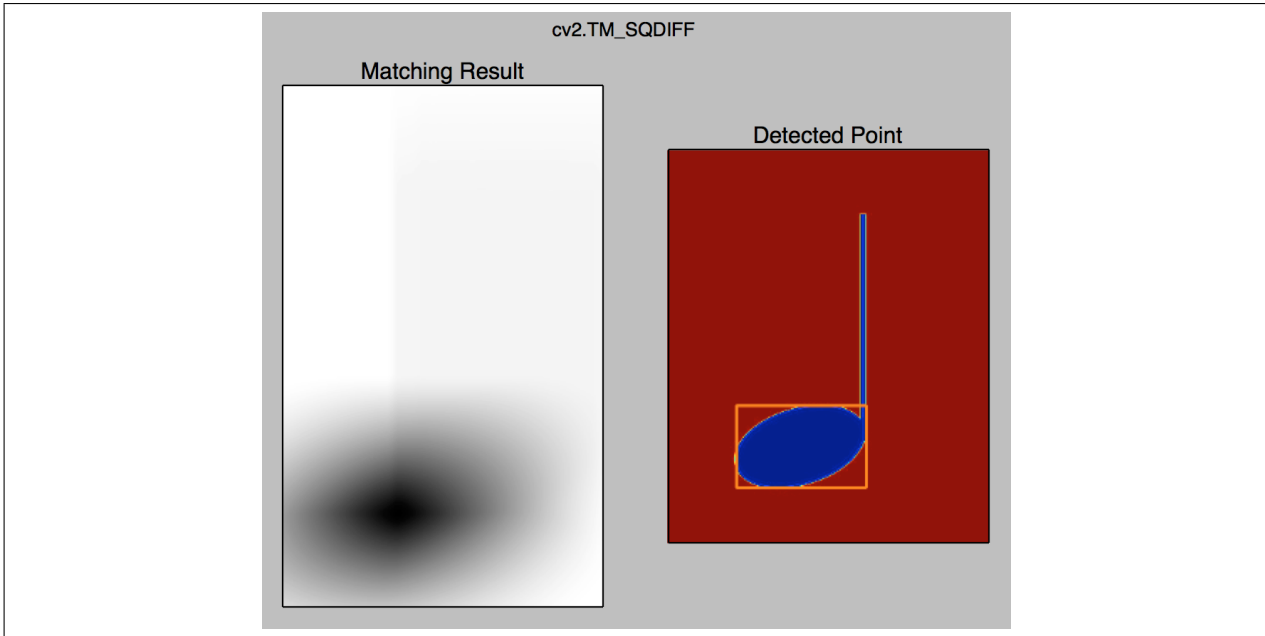


Figure 3.10: Example of extracting note heads using template matching utilising OpenCV and the sum of square differences scoring measure

$$SSD_{xy} = \sum_{i=0}^m \sum_{j=0}^n (x(i, j) - y(i, j))^2 \quad (3.3)$$

In the example given above, we're only looking for a subsection or a partial template so if image x is $m \times n$ and image y is $a \times b$ in size where $m \leq a \wedge n \leq b$, we can search each possible position (k, l) for y in x with Equation (3.4).

$$SSD_{xy}(k, l) = \sum_{i=k}^{m+k} \sum_{j=l}^{n+l} (x(i, j + l) - y(i, j))^2 \quad (3.4)$$

and whichever positioning gives us the highest score is likely to be the best match.

3.4.4 Defects and Difficulties

There are two regular types of error that can cause issues during segmentation, **touching objects** and **broken objects**. Since these are likely to occur a lot more regularly given the freedom allowed by handwritten music and the beginner's learning curve, it is important that the application is able to spot (and subsequently feed back on) these errors.

3.4.4.1 Touching Objects

Touching objects are where what was intended to be written as two separate objects actually touch and this can cause issues when segmenting components using connected component analysis Section 3.4.1.

In a printed score, you can use template matching as outlined in Section 3.4.3 in order to separate the components. When they are handwritten, it is more likely we will have to rely on using vertical and horizontal projections to find the point of minimal intersection as in Fujinaga 1996.

3.5 Component Features

Some features which can be extracted from a component for use in classification include (Fujinaga 1996): X Position, Y Position, Height, Width, Centroid X, Centroid Y, Extent, Aspect, Area, Bounding Box Area, Avg Vertical Holes, Avg Horizontal Holes and Moments

In some cases, x and y position on the staff are used in classification but for the purposes of the current project I have decided to perform classification assuming no prior knowledge. There might be exercises or free-drawing where a child doesn't want to have to follow the rules which apply to a normal score of music but rather to draw specific symbols.

3.5.1 Vertical and Horizontal Holes

The average number of vertical and horizontal holes is used by Fujinaga 1996 and are topological properties in that they're generally scale invariant and as defined by Burger et al. 2009 "do not describe the shape of a region in continuous terms; instead, they capture its structural properties".

Average vertical and horizontal holes can be calculated by analysing the columns and rows respectively and counting the number of segments which do not have a solid pixel in (the runs of 0s). For example, for average horizontal holes we would count the number of white segments per row and total them up, averaging by the number of rows. Similarly with columns for vertical holes.

3.5.2 Moments

In this project the concept of low level moments is used to extract information like region area and centroid coordinates, however I wasn't able to get to grips with some of the higher order moments used by Fujinaga 1996; Rebelo et al. 2011 and others to represent rotation, skew and other properties which may go some way to explaining my failed classifier in Table 4.1. It may be that my implementation was simply wrong, however since I was able to achieve good classification results using a resampled and flattened image, I decided that at least for the initial project iterations I would stick to that.

3.5.2.1 Ordinary Definition

As in Burger et al. 2009 a moment of the order p, q for an image or region $I(x, y)$ can be defined by Equation (3.5).

$$m_{pq} = \sum_{x,y \in \mathbb{R}} I(x, y) \cdot x^p y^q \quad (3.5)$$

When dealing with binary regions, since we only consider the pixels of value 1 we can simplify Equation (3.5) to Equation (3.6).

$$m_{pq} = \sum_{x,y \in \mathbb{R}} x^p y^q \quad (3.6)$$

3.5.2.2 Zeroth Order Moments

Zeroth order moments can be used to calculate the total sum of grey area of a region using Equation (3.7) and we can see the result is intuitive - the area is simply the count of the black pixels.

$$\text{area} = m_{00} = \sum_{x,y \in \mathbb{R}} x^0 y^0 = \sum_{x,y \in \mathbb{R}} 1 \quad (3.7)$$

3.5.2.3 First Order Moments

The first order moments m_{01} and m_{10} are use to obtain the centre of mass of a component at (\bar{x}, \bar{y}) .

$$\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}} \quad (3.8)$$

3.5.3 Run Length Encoding

Run Length Encoding (RLE) is something which is regularly mentioned in regard to OMR, it involves taking a pixel based image and converting what would be a huge amount of information into a more compact format by establishing “runs” of identical pixels which are in a contiguous block.

For a two dimensional binary image a run of pixels can be represented by it’s row, column, value and run length (Burger et al. 2009, p. 27-28) as seen in Table 3.2

Example	RLE (row , column , value , length)
$\begin{bmatrix} 1 & 2 & 2 & 3 \\ 3 & 3 & 3 & 1 \\ 1 & 1 & 5 & 5 \\ 5 & 5 & 2 & 2 \end{bmatrix}$	$[(0, 0, 1, 1), (0, 1, 2, 2), (0, 3, 3, 1), (1, 0, 3, 3), (1, 3, 1, 1), (2, 0, 1, 2), (2, 2, 5, 2), (3, 0, 5, 2), (3, 2, 5, 2)]$

Table 3.2: 2D Greyscale Image

Example	RLE (value , length)
$[1, 2, 2, 3, 3, 3, 3, 1, 1, 1, 5, 5, 5, 5, 2, 2]$	$[(1, 1), (2, 2), (3, 4), (1, 3), (5, 4), (2, 2)]$

Table 3.3: 1D Flattened Greyscale Image

3.6. CLASSIFICATION

Example	RLE $\langle \text{value}, \text{length} \rangle$
[0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1]	[0, 3, 3, 2, 4] (first bit sets ordering) [3, 3, 2, 4] (assuming initial bit is 0)

Table 3.4: 1D Flattened Binary Image

If you then reshape this 2D image into a one dimensional array (retrieving and reshaping it to it's 2D representation later), you can remove 50% of the compressed data (row and column) as seen in Table 3.3

If we are using binary data we can simplify this further, simply tracking the initial bit, then recording runs of alternating values as seen in ???. We can simplify this further with the additional assumption (Fujinaga 1996) that the sequence will start with a 0, removing the need for the initial bit. If the sequence begins with a 1 we just start with an entry of length 0. Since based on my research a lot of musical entities don't touch the top left pixel of an image (meaning it's value it almost always 0), this is the implementation I have used.

3.6 Classification

3.6.1 Nearest Neighbour

The [K Nearest Neighbour \(KNN\)](#) algorithm is an example of instance based learning and works by extracting a set of features for each sample and using this 'feature vector' to represent each sample in a multidimensional vector space.

Once this space has been generated, new samples may be classified by generating their feature vector and calculating the k closest or 'most similar' samples in the model by way of a distance calculation. By considering the k closest samples, a consensus can be reached and the new sample is classified according to the label with the greatest majority in the k nearest neighbours as seen in Figure 3.11.

The two most critical aspects of the KNN classifier are the distance method used and the number of neighbours considered. In order to maximise the accuracy of the classifier, I performed several experiments to this effect in Section 4.2.3.

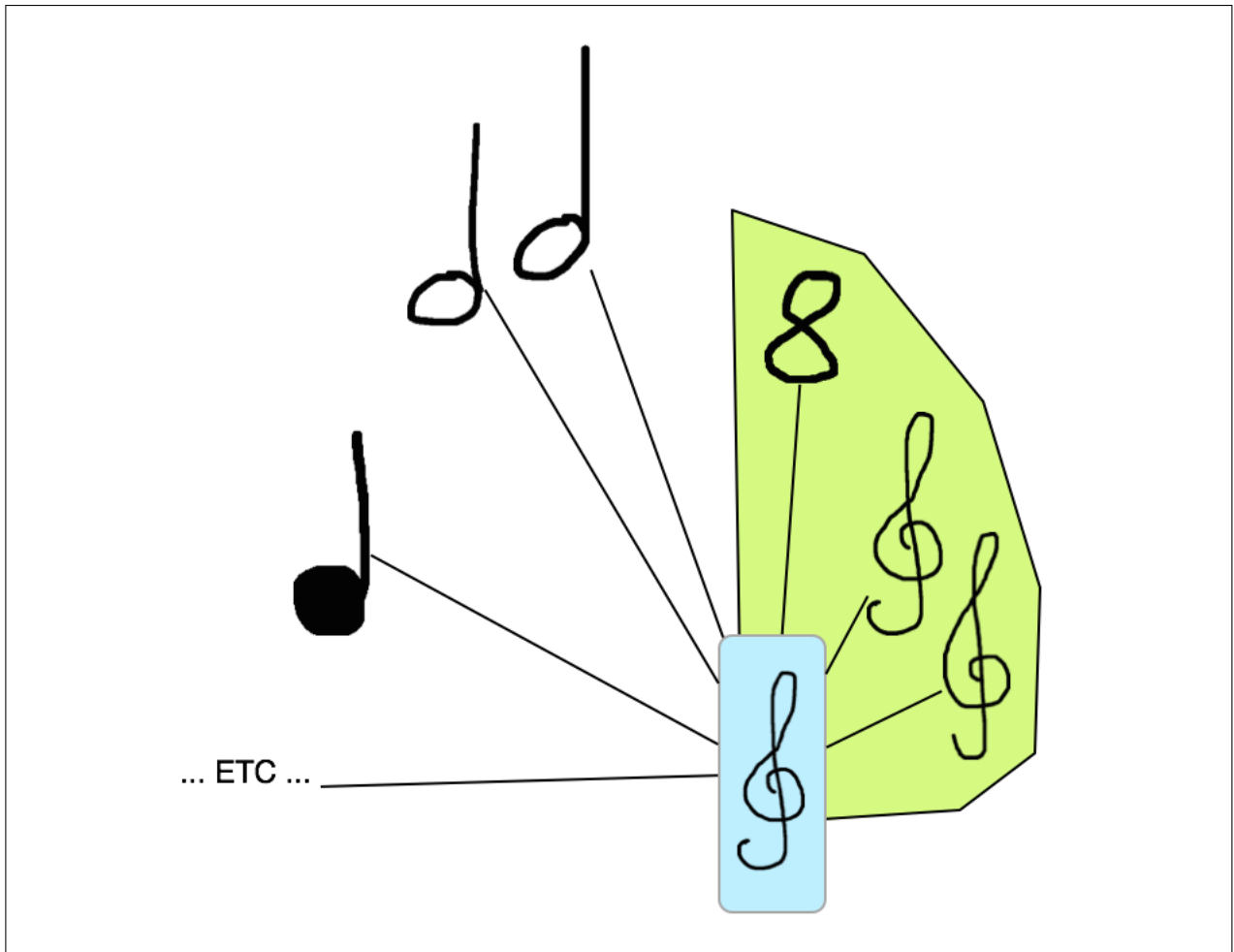


Figure 3.11: KNN example using musical symbols and a K value of 3. The new symbol being classified is highlighted blue and the three closest three entities in green. In this case, the sample would be classified as a treble clef

3.6.2 KNN Editing

Although KNN can produce good results, the need to store all the data makes it less attractive. To reduce the data needed in classifiers you can use an edited KNN algorithm where the idea is to try and remove “poor” samples which not only reduces the storage requirements but should also improve the accuracy of the classifier.

3.6.2.1 Wilson Editing

I originally came across the idea of edited KNN classifiers in Fujinaga 1996 in which he outlines the original KNN editing techniques from earlier work by Wilson 1972, (the

3.6. CLASSIFICATION

algorithm for this editing technique is outlined in Algorithm 1). You can further extend the algorithm to multiple iterations (referred to by most as the multi-editing algorithm), repeating the process until no more poor samples are removed, at which point you save the model for use in classification, hopefully at a reduced dataset size.

Algorithm 1 KNN editing algorithm

```
procedure EDITMODEL(model)
  trainingSamples = GETSAMPLES(model)
  for each sample in trainingSamples do
    classification = MODEL.CLASSIFY(sample)
    if classification  $\neq$  sample.classification then REMOVE_SAMPLE(sample)
    end if
  end for
end procedure
```

3.6.2.2 Genetic Algorithms

Although I decided not to actually employ the use of genetic algorithms in this project, they came up repeatedly in more recent research and so I felt it was worth mentioning. Genetic algorithms for feature selection is nothing new, indeed it's used in Fujinaga 1996 for selecting which features to use in a classifier; a traditionally tricky problem when you have lots of features (or indeed classifier variables such as the value of K in KNN classification) and need to work out which ones actually produce the best division of classes. Though there are other techniques to do this too like PCA¹². However in Kuncheva 1995 it is used specifically in editing a KNN classifier. The results suggested that it might perform comparably to wilson's technique and that it generally performed better than just using a random sample training set but relative to multi-editing technique the authors state that the results from genetic algorithm and multi editing could not be compared due to an inability to properly evaluate the power of the multi-edit algorithm.

¹²Principal Component Analysis is used to find the dimension/variable in a feature vector which provides the highest variance possible. Features which provide small variance are of little help in a classification problem

3.7 Scoring & Evaluation

3.7.1 Image Difference

If you're handling simple binary images, you can perform a pixel difference comparison which just takes the first image XOR'd with the second, $I_1(x, y) \oplus I_2(x, y)$. The result (shown in Figure Figure 3.12) is a simple highlighting of the difference between the two.

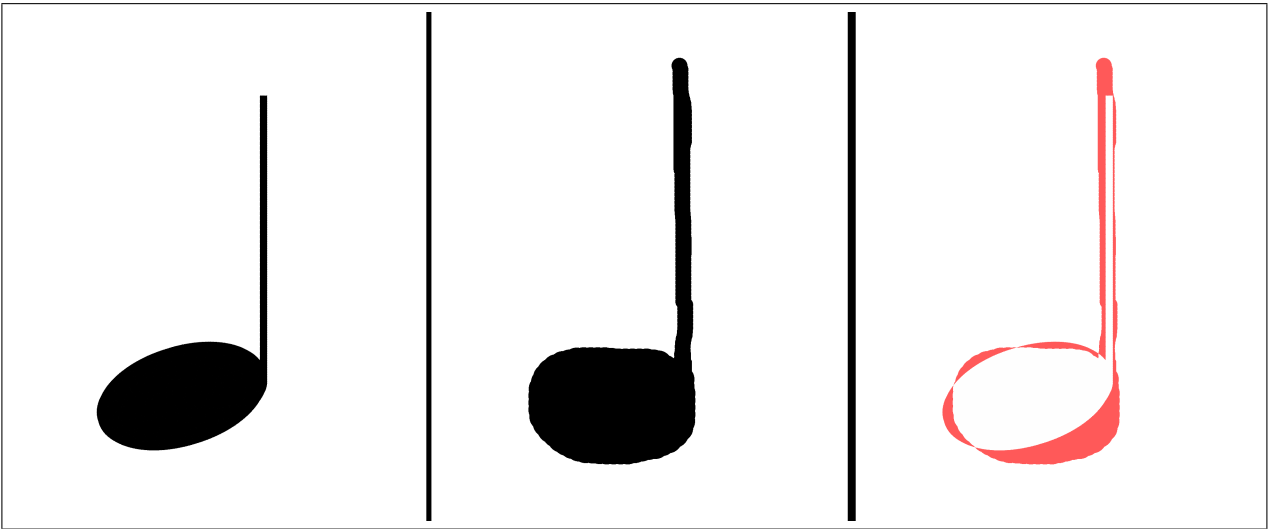


Figure 3.12: Highlighting differences between a perfect and a hand-drawn crotchet

I really liked the simplicity of this approach and as a visual representation of perhaps ‘going outside the lines’ it really gets the point across. However with some preliminary experiments it quickly became clear that it’s not very useful in the context of scoring components like the crotchet in Figure 3.12 where you can see by simple inspection that a small discrepancy in the scale of the note head contributes a much greater difference than a fairly significant error in the stem. If we therefore use the pixel difference outright we can’t reasonably compare one component to another with respect to how ‘correct’ it is.

3.7.2 Normalized Cross Correlation

Normalized cross correlation enables a pixel-wise comparison of two images, resulting in a score between $[-1, 1]$ where 1 indicates a perfect match e.g. the images are the same and -1 indicates the images are precisely the opposite of each other. The overall effect is to penalise the score for each pixel which is different in one image than the other. The

normalisation is designed to help in images where the brightness and light levels vary but it works fine on binary images too.

An NCC score between two binary images F and G of the same dimension can be calculated by using Equation (3.9).

$$\frac{1}{N} \sum_{x,y} \frac{(f(x,y) - \bar{f})(g(x,y) - \bar{g})}{\sigma_f \sigma_g} \quad (3.9)$$

Where $N = w \times h$ (the number of pixels in the image), $f(x, y)$ and $g(x, y)$ represent the pixel in row y column x for the respective images, \bar{f}, \bar{g} represent the average pixel values for each image and σ_f, σ_g are the standard deviations of the two images.

3.7.3 Skeletonization

Skeletonization is the process of reducing a component in a binary image to a single-pixel wide skeleton. The algorithm used in this project (as defined in Zhang and Suen 1984) works by making successive passes of the image, removing pixels on object borders. This continues until no more pixels can be removed as in Figure 3.13. The image is correlated with a mask that assigns each pixel a number in the range $[0...255]$ corresponding to each possible pattern of its 8 neighbouring pixels. A look up table is then used to assign the pixels a value of 0, 1, 2 or 3, which are selectively removed during the iterations.

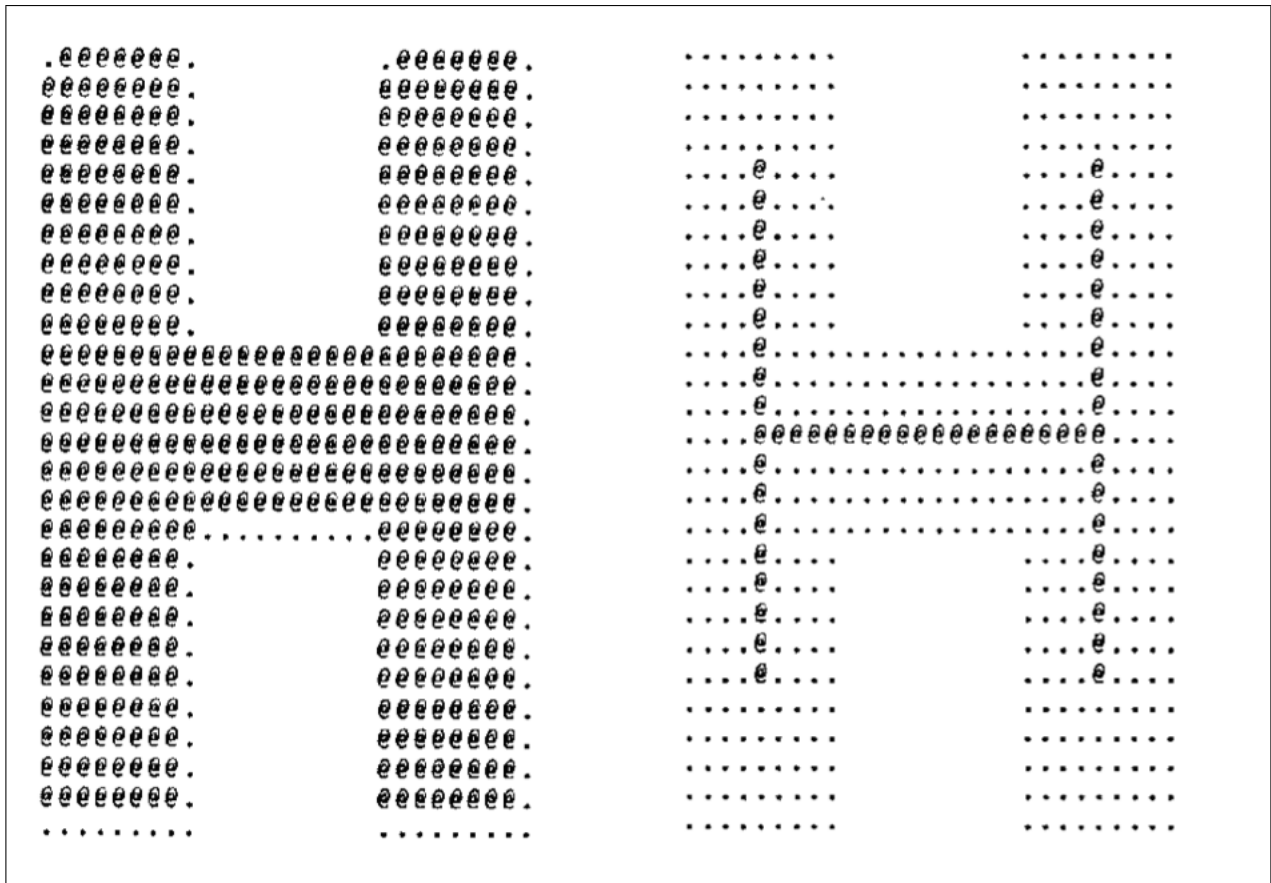


Figure 3.13: Results of the skeletonization algorithm from Zhang and Suen 1984

3.7.4 Watershed Segmentation

Watershed segmentation essentially allows us to segment an image by identifying starting points or ‘markers’ from which we grow segments, marking the boundaries where these segments meet. There are multiple ways to add more pixels to the markers, one is to flood fill outwards from the marker. Another way which I found success in the context of the **NoteED** project is to perform a distance transform on the (binary) image, where pixels are assigned values according to how far they are from the nearest background pixel and use local maxima and assign pixels to the maxima depending on which marker is reachable by the steepest gradient ascent.

A good example from Image is that of two overlapping circles we want to separate, shown in Figure 3.14. The first step shows the two overlapping circles, the second shows a map of the distance transform. Here the dark red indicates a background pixel and the gradient ends with blue indicating the maximum distance from a background pixel. Local maxima

3.7. SCORING & EVALUATION

are marked as starting points for the watershed algorithm. If we imagine the gradients as a 3D topographical map, and if we then assign pixels based on the steepest gradient ascent near them, they will group around the local maxima. Pixels which are equidistant from either peak result in a boundary forming which approximates the division between the original entities. We can now draw contours along these boundary lines to visually segment the image.

An example of watershed being used in **NoteED** to help find the centre of a broken flat is given in Figure 4.9.

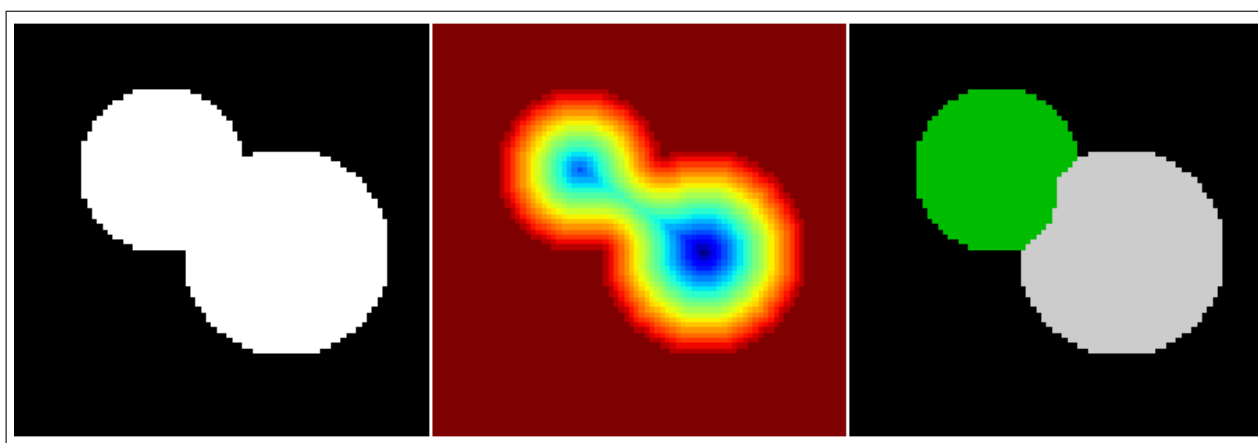


Figure 3.14: Separating two overlapping circles using watershed segmentation from Image

Chapter 4

Techniques

4.1 Architecture

NoteED is divided in a few core architectural components. Firstly,

4.2 Identification

As outlined in Chapter 4 above, a key challenge is identifying and isolating the various components which are going to be analysed. Section 4.2.1 outlines from the research conducted and which methods were selected for segmentation, feature extraction and allocation of pitch, duration and other properties.

4.2.1 Segmentation

In order to perform a decomposition of the manuscript, within **NoteED** I perform multiple stages of segmentation combined with classification. For the purposes of demonstration I will be following the identification process using the manuscript example in Figure 4.1.



Figure 4.1: The initial clean manuscript attempt

4.2.1.1 Initial Segmentation

Connected component analysis (see Section 3.4.1) is performed to isolate the individual components on the staff, resulting in the first stage of component segmentation seen in Figure 4.2.

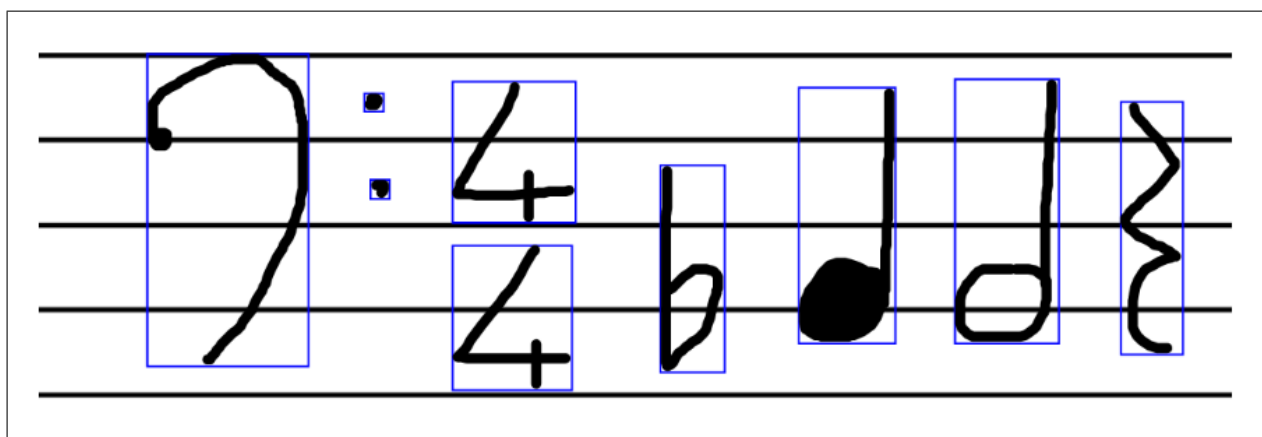


Figure 4.2: After initial segmentation of Figure 4.1

The components are then classified using the first of two KNN classifiers (KNN1 - discussed below in Section 4.2.3) which results in the basic labelling of the manuscript components seen in Figure 4.3.

The *note_complex* and *split_x/split_y* components are divided into sub-components using techniques such as stem removal (Section 4.2.1.2) and vertical projections (Section 3.4.2) respectively. The new set of components are classified using one of the specialised classifiers, the original KNN1 classifier in the case of split components or the KNN2 classifier for note subcomponents to produce the more detailed component labelling in Figure 4.4.

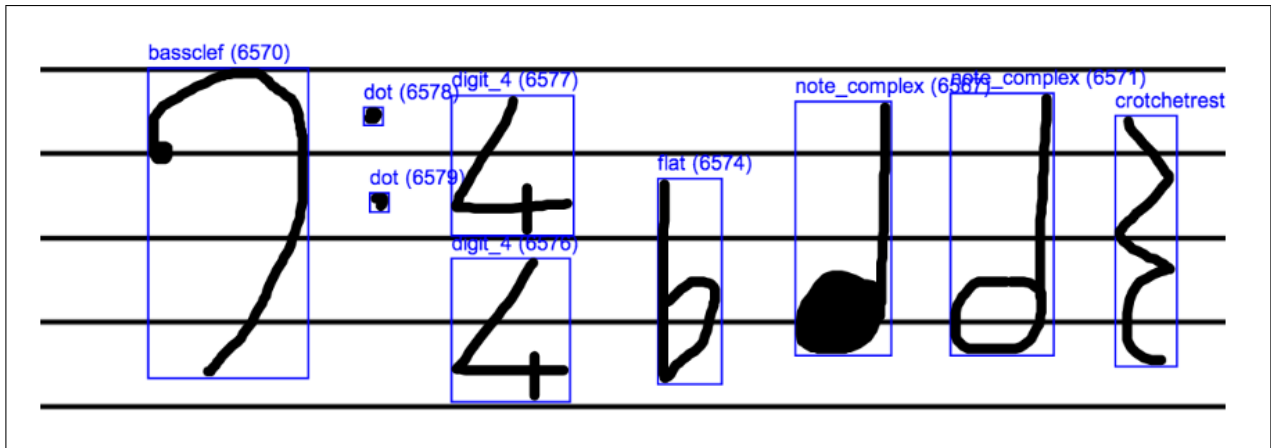


Figure 4.3: After first level classification of Figure 4.2

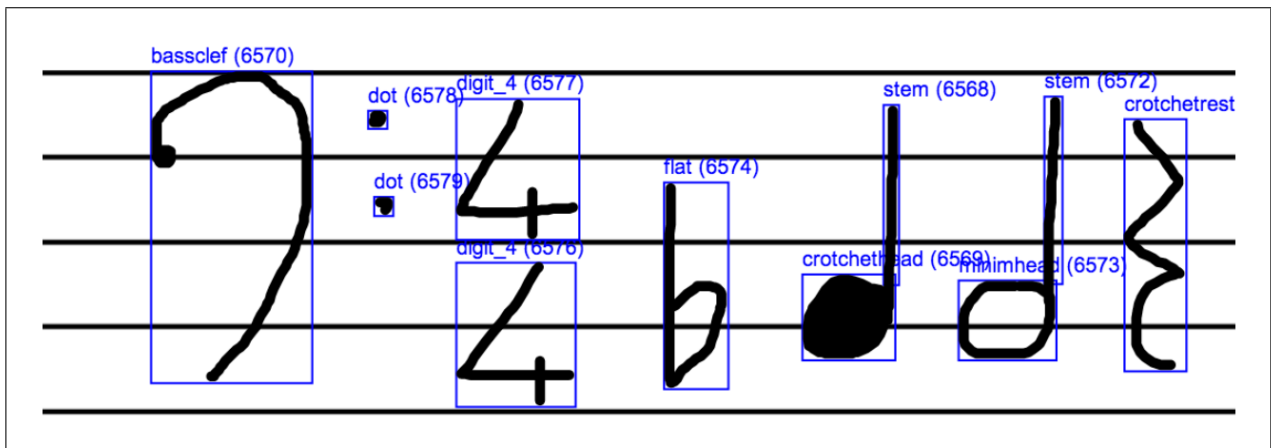


Figure 4.4: After second level segmentation and classification of Figure 4.3. Note the stems are now labelled

No further segmentation is performed on the components after this point.

4.2.1.2 Stem Removal

In order to split up a *note_complex* into heads, tails, beam, stem etc, the first step is to try and isolate the stems. We can do this by removing horizontal runs of black pixels which are above a threshold greater than the typical width of a stem. Since runs like this appear at the intersection of the stem with other components, the result is a large number of new regions in the image, one of which is likely to be the stem.

To establish which region is a stem and remove any noise, we look for regions which are within a set aspect ratio (I use 1 : 2, obtained experimentally and designed to catch angled

4.2. IDENTIFICATION

stems as well as very vertical ones) and a height above a minimum threshold (I use 40px, also obtained experimentally).

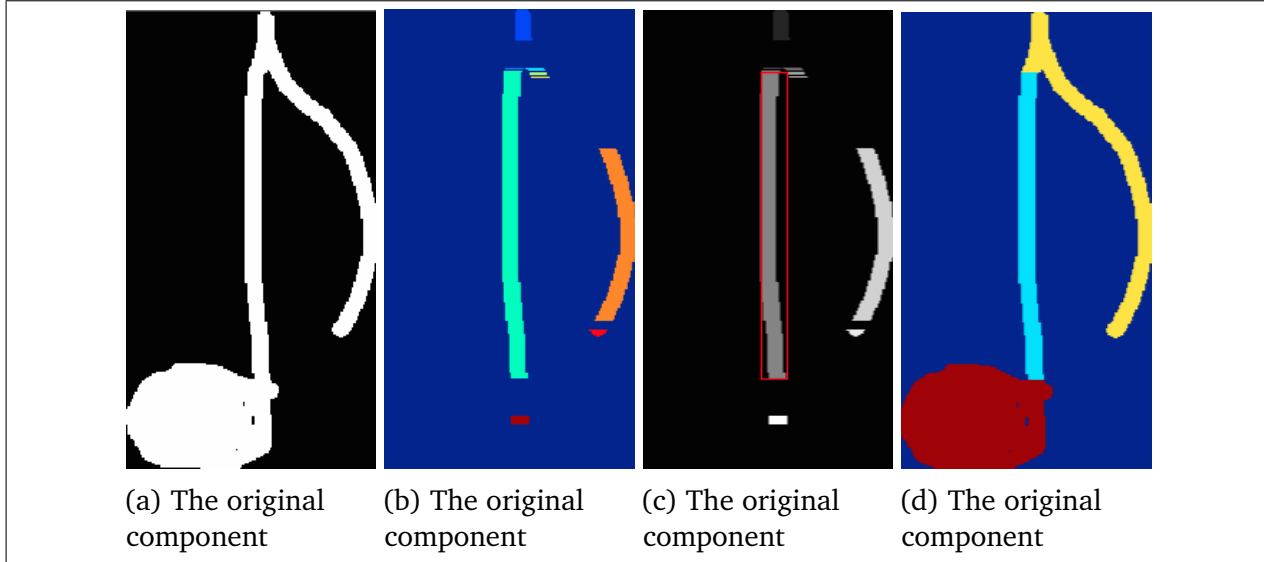


Figure 4.5: The stages in stem detection for sharps

4.2.2 Feature Extraction

4.2.2.1 Resampling

Using the pixels of an image as the features is an approach used in several of the papers I looked at Ben-Dayan and Giloh 2013; Rebelo 2012, however in order to be able to compare feature vectors, they need to be the same size which means we need to resize the images to a common dimension.

To do this, I resize the images down using bilinear interpolation to an experimentally obtained dimension of 20×50 px obtained in Section 4.2.3.1. The smaller dataset increases the speed at which a classifier can operate and I also found it to be comparable to if not more accurate than the feature vectors of larger images (Table 4.2).

4.2.3 Classification

Classifiers are created by taking a sets of previously labelled samples and building a model which provides the most accurate relation between the samples and their labels. This

allows new samples to be classified using the model to apply the most likely (and hopefully, correct) label.

In order to maximise the accuracy of my classifications, I ran multiple experiments on different classifiers before deciding on which one I would apply in my application.

4.2.3.1 K Nearest Neighbour

For the K Nearest Neighbour algorithm, I tried two different feature vectors, the first was using statistical properties and the second was a resampled binary image flattened into a 1D array.

4.2.3.1.1 Statistical Feature Vector An initial attempt at classification using statistical properties and a KNN classifier wasn't all that successful as you can see from the confusion matrix and accuracy scores in Table 4.1. Most components were incorrectly classified as crotchet heads though I was unable to come up with a good explanation as to why and it's something which it would be interesting to investigate further in future.

4.2.3.1.2 Image Feature Vector For the resampled image features, I ran a series of trials for different dimensions from 10px to 100px in both width and height (in 10px increments), averaging 3 repeats per dimension (each with a different training/testing split) and then generating a matrix of accuracies (which can be seen in full in Table 4.3).

Since most objects on a stove have a more vertical aspect ratio, I first graphed the average and maximum of the scores for each different height across all widths to see if there were any trends such as taller images producing higher accuracy.

The results can be seen in Figure 4.6 and it seemed that in general, very short images ($\leq 20\text{px}$) gave worse results than tall ones but there wasn't anything conclusive and the gains from very tall images as opposed to fairly small images were minimal. Since there wasn't a height which clearly stood out, an analysis of the full matrix was done to find the highest scoring height and width combinations.

The top ten experimental accuracies from the tested dimensions are listed in Table 4.2 and I eventually selected $20 \times 50\text{px}$ as my resampled dimension, the reason being that although larger dimensions did technically produce better accuracies, they were only *slightly* better

4.2. IDENTIFICATION

	crotchetrest	minimhead	note_complex	bassclef	beam_complex	sharp	semibreve	quaverrest	barline	quavertaildown	minimsemibreverest	flat	crotchethead	stem	trebleclef	digit_8	natural	digit_3	digit_2	digit_4	quavertailup	dot
crotchetrest	0	0	0	0	0	0	0	0	0	0	0	0	44	0	0	0	0	0	0	0	0	0
minimhead	0	16	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0
note_complex	0	0	32	0	0	0	0	0	0	0	0	0	87	0	0	0	0	0	0	0	0	0
bassclef	0	0	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0
beam_complex	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
sharp	0	0	0	0	0	6	0	0	0	0	0	0	46	0	0	0	0	0	0	0	0	0
semibreve	0	0	0	0	0	0	1	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0
quaverrest	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0	0
barline	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0	0
quavertaildown	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
minimsemibreverest	0	0	0	0	0	0	0	0	0	0	1	0	9	0	0	0	0	0	0	0	0	0
flat	0	0	0	0	0	0	0	0	0	0	0	2	50	0	0	0	0	0	0	0	0	0
crotchethead	0	0	0	0	0	0	0	0	0	0	0	0	133	0	0	0	0	0	0	0	0	0
stem	0	0	0	0	0	0	0	0	0	0	0	0	60	24	0	0	0	0	0	0	0	0
trebleclef	0	0	0	0	0	0	0	0	0	0	0	0	40	0	2	0	0	0	0	0	0	0
digit_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
natural	0	0	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0
digit_3	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	2	0	0	0	0
digit_2	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0
digit_4	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	7	0	0
quavertailup	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	3	0
dot	0	0	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	15

Accuracy	Precision	Recall
----------	-----------	--------

0.275	0.645	0.275
-------	-------	-------

Table 4.1: KNN classifier results using statistical features

and would have resulted in many more pixels in the feature vector, slowing down the process of building a classifier and testing samples.

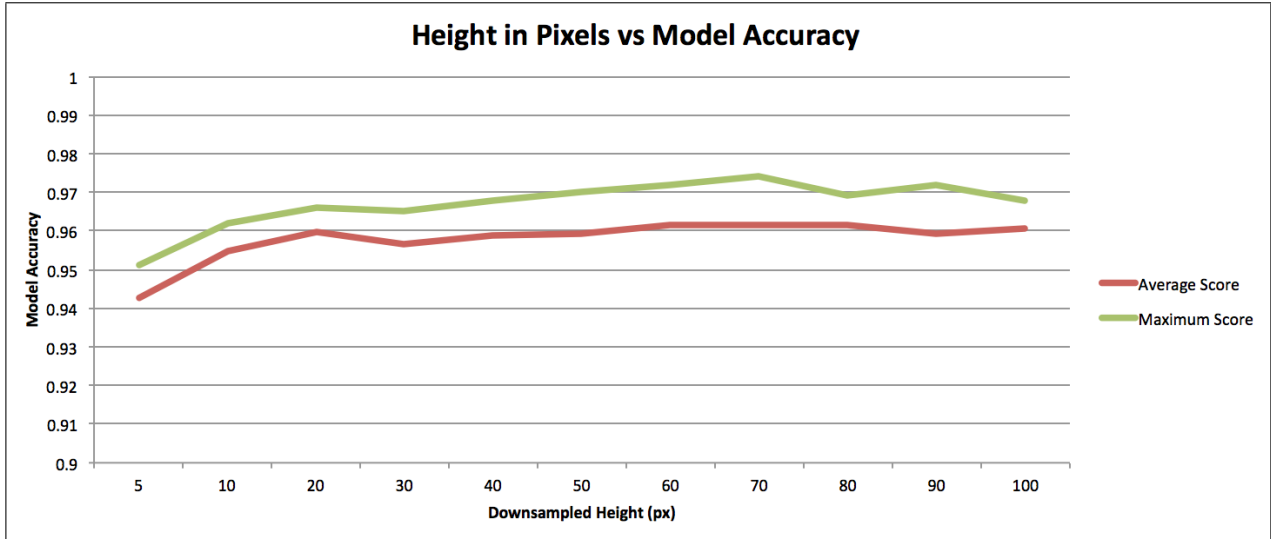


Figure 4.6: The effect of resampled image height on accuracy

Width	Height	Total Pixels	Accuracy
50	70	3500	0.974
20	60	1200	0.971
20	70	1400	0.97
→ 20	50	1000	0.969
30	70	2100	0.969
100	70	7000	0.969
20	40	800	0.968
70	50	3500	0.967
60	60	3600	0.967
20	100	2000	0.967

Table 4.2: Top 10 accuracy scores for different width and height combinations, the selection dimensions used in the classifier are highlighted and the full matrix can be found in Table 4.3

4.2. IDENTIFICATION

Height (px)	Width (px)									
	10	20	30	40	50	60	70	80	90	100
10	0.948	0.961	0.955	0.954	0.962	0.953	0.951	0.953	0.955	0.957
20	0.947	0.963	0.963	0.962	0.964	0.962	0.956	0.961	0.960	0.966
30	0.945	0.960	0.954	0.965	0.958	0.958	0.956	0.958	0.959	0.959
40	0.946	0.968	0.957	0.963	0.958	0.950	0.961	0.964	0.960	0.966
50	0.935	0.969	0.965	0.957	0.961	0.957	0.967	0.959	0.948	0.962
60	0.953	0.971	0.960	0.963	0.962	0.967	0.948	0.957	0.957	0.963
70	0.954	0.970	0.969	0.960	0.974	0.960	0.957	0.956	0.960	0.969
80	0.959	0.962	0.959	0.964	0.963	0.965	0.957	0.963	0.963	0.954
90	0.954	0.961	0.958	0.960	0.956	0.959	0.963	0.959	0.963	0.956
100	0.955	0.967	0.965	0.963	0.958	0.960	0.960	0.965	0.960	0.962

Table 4.3: Accuracy Matrix for various Height and Width Downsampling Combinations

Initial results on attempting to distinguish between all components were positive and I was able to achieve around 93% accuracy as seen in Table 4.4.

	flat	trebleclef	crotchetrest	digit_8	minimhead	crotchethead	digit_2	digit_4	quaverrest	semibreve	beam_complex	quavertailup	bassclef	sharp	digit_3	natural	barline	quavertaildown	dot	minimsemibreverest
flat	62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
trebleclef	0	53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
crotchetrest	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
digit_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
minimhead	0	0	0	0	43	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
crotchethead	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0
digit_2	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0
digit_4	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0
quaverrest	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0
semibreve	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0	0
beam_complex	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
quavertailup	0	0	1	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0
bassclef	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0
sharp	0	0	0	0	0	0	0	0	0	0	0	0	0	55	0	0	0	0	0	0
digit_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
natural	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0	0	0	0
barline	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0
quavertaildown	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0
dot	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	38	0
minimsemibreverest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
		Accuracy				Precision				Recall										
		0.933				0.922				0.933										

4.2. IDENTIFICATION

example of this is a note. Instead of trying to separate a note straight away into heads stems and beams, it's sufficient to simply classify it as a 'note_complex' entity and perform more detailed classification of components later.

Using a two-stage classifier much more satisfactory results were achieved, with an initial classification accuracy of 98.5% (Table 4.5) and a secondary classification accuracy of 100% (Table 4.6)!

	minimhead	crotchethead	quavertailup	quavertaildown
minimhead	51	0	0	0
crotchethead	0	118	0	0
quavertailup	0	0	23	0
quavertaildown	0	0	0	5

Accuracy	Precision	Recall
1.0	1.0	1.0

Table 4.6: 2nd level KNN Classifier Results

4.2.3.2 Neural Networks

Although neural networks are also common in OMR, I was unfortunately unable to extract any great results from them during my experiments. The best classification result I got was by using Hierarchical Classification, where the network achieved an accuracy of 80.38% for the 1st level classifier as seen in Table 4.7. I used trained the network using Backwards Propagation and used two hidden sigmoidal layers of 25 nodes each. As more potential classes were added, unfortunately the accuracy just decreased and so I chose to use KNN as my classification technique.

It should be noted that further depth of research in this area fell outside the scope of this project and is not an area in which I have specific expertise. Many people have great success

	flat	natural	crotchetrest	digit_8	note_complex	beam_one	digit_3	digit_2	trebleclef	digit_4	quaverrest	stem	semibreve	bassclef	dot	sharp	barline	minimsemibreverest
flat	62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
natural	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
crotchetrest	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
digit_8	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
note_complex	0	0	0	0	112	0	0	0	0	0	0	0	0	0	0	0	0	0
beam_one	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
digit_3	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
digit_2	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0
trebleclef	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0
digit_4	0	0	0	0	1	0	0	0	1	30	0	0	0	0	0	0	0	0
quaverrest	0	0	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0
stem	0	0	0	0	2	0	0	0	5	0	0	99	0	0	0	0	3	0
semibreve	0	0	0	0	0	0	0	0	0	0	0	0	67	0	0	0	0	0
bassclef	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0
dot	0	0	0	0	0	0	0	0	0	0	0	4	0	0	35	0	0	0
sharp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	74	0	0
barline	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	19	0
minimsemibreverest	0	0	0	0	0	0	0	0	0	0	0	4	0	0	2	0	0	1

Accuracy	Precision	Recall
0.95	0.948	0.95

Table 4.5: 1st level KNN Classifier Results

4.2. IDENTIFICATION

using neural networks for this and similar applications, achieving high accuracy rates and it's certainly something which is maybe worth coming back to again in future.

	crotchetrest	semibreve	dot	flat	digit_2	barline	trebleclef	sharp	quaverrest	digit_3	note_complex	natural	digit_4	bassclef	minimsemibreverest
crotchetrest	29	0	1	0	1	0	3	3	0	0	5	1	1	1	0
semibreve	0	45	0	0	0	0	0	0	0	0	1	0	0	0	0
dot	0	1	43	0	0	0	0	0	0	0	0	1	0	1	0
flat	0	0	0	43	0	0	0	3	0	0	8	1	1	0	0
digit_2	0	0	1	0	5	0	0	0	0	0	4	0	0	0	0
barline	0	0	10	0	0	28	2	1	0	0	0	0	2	0	0
trebleclef	1	0	0	1	0	2	29	3	0	0	0	0	1	0	0
sharp	1	0	2	0	0	1	1	46	0	0	0	0	1	0	0
quaverrest	0	0	0	0	0	0	0	0	32	1	1	1	0	0	0
digit_3	0	0	0	0	0	0	2	1	0	0	3	0	0	0	0
note_complex	0	0	0	0	1	0	1	0	3	0	102	1	0	0	0
natural	0	0	0	0	0	0	2	1	0	0	1	31	0	0	0
digit_4	1	1	0	0	0	2	3	4	0	0	4	0	10	0	0
bassclef	0	0	0	0	0	0	0	0	1	0	4	0	0	28	0
minimsemibreverest	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0

Accuracy	Precision	Recall
0.804	0.797	0.804

Table 4.7: 1st level Neural Network Classifier Results

4.2.4 Pitch

For note heads and accidentals, it's important to know exactly where the centre of the note lies in order to correctly identify it's position on the stave. Doing this roughly is usually sufficient for OMR, however, since we would like to be able to feed back to the student if

they aren't putting their notes in the right place or they're too big/small we need to be as accurate as possible.

Once the position has been determined, we can calculate the difference between that and the y coordinate of the various pitched components on the staff. Assigning the pitch which is closest using Algorithm 2 results in a pitch assignment for pitched components, an example of which can be seen in Figure 4.7.

Algorithm 2 Assigning a pitch to a component

```

1: procedure GETNOTEFORCOMPONENT(component)
2:   sum = GETCOMPONENTCENTRE(component)
3:   min_distance =  $\infty$ 
4:   assigned_pitch = None
5:   for each pitch in pitches do
6:     dist = ABS(pitch_coord - y)
7:     if dist < min_dist then
8:       min_dist = dist
9:       allocated_pitch = pitch
10:    end if
11:  end for return assigned_pitch
12: end procedure

```

An example of a manuscript where the pitches have been established can be seen in Figure 4.7

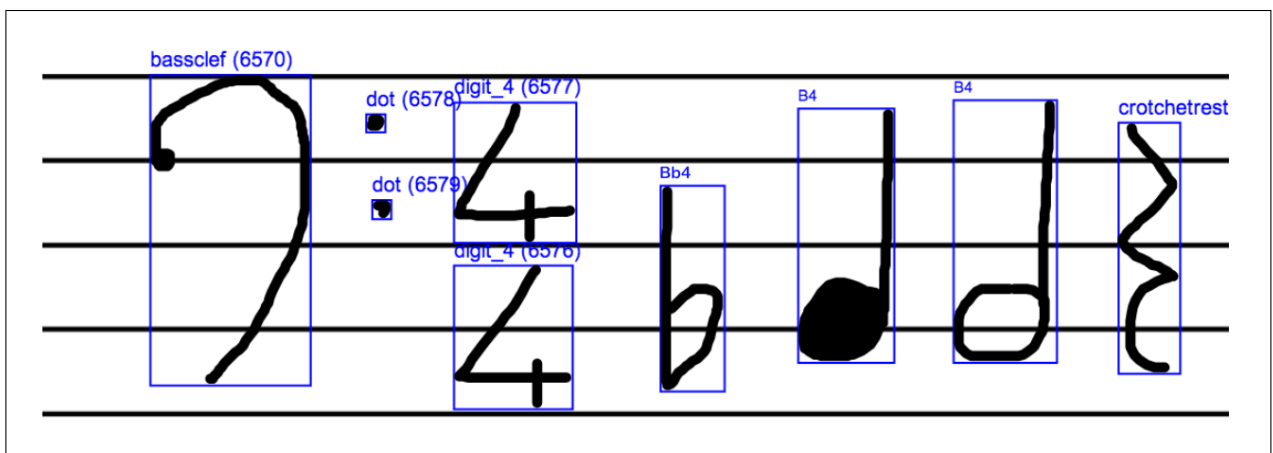


Figure 4.7: The manuscript after pitch analysis

For a rough estimate of note head position, the simplest method is to use a known position along the vertical axis, then use that as an estimate for the centre, indeed that's often what

4.2. IDENTIFICATION

is done in standard OMR. However we can get a more accurate position using alternative techniques.

4.2.4.1 Sharp Centres

For the sharp centre, the centroid proved less than satisfactory as it was easily affected by the length of the lines (see the blue centres in Figure 4.8), whereas in reality the ‘centre’ of a sharp is determined by the position of its island region in the middle.

By inverting the image and performing connected component segmentation, we can isolate the island region and after extracting the centroids from this region, we get a much more satisfactory identification of the sharp’s centre, show in Figure 4.8 by the intersecting red lines.

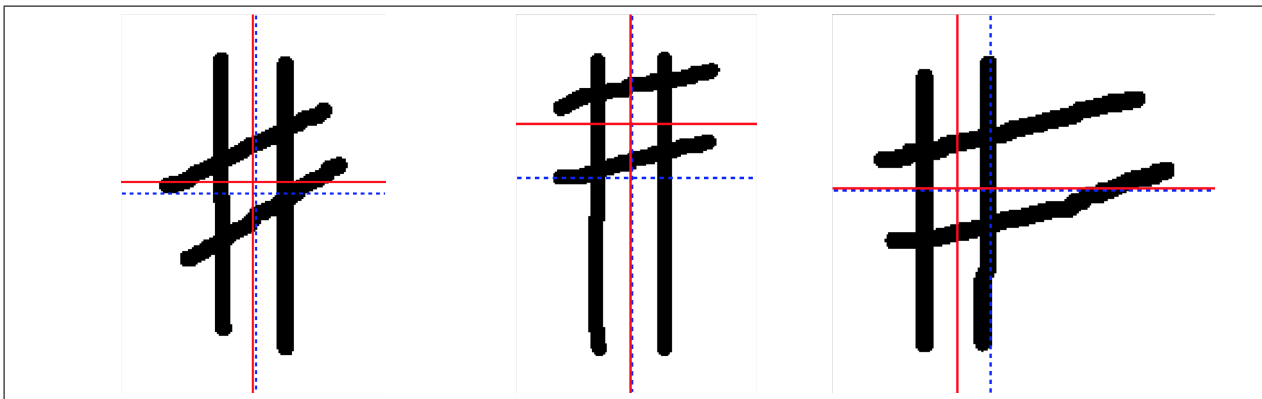


Figure 4.8: Identifying the true centres of the drawn sharps. Red intersecting lines show the true centres and blue intersecting lines show the centroid centres

4.2.4.2 Flat Centres

An unfortunate occurrence in flats during preliminary user testing was that of not joining the head up properly, an example of which can be seen in Figure 4.9a. Ideally we would like to perform a similar technique to sharps, however we need some reliable way to compensate for potential breaks.

The first experiment I ran was to perform watershed segmentation by computing a distance transform on the flat, then by using the local maximum peaks as a starting point watershed segmentation was performed, resulting in the initial segmentation seen in Figure 4.9c. From

there, by merging neighbouring regions from smallest to largest, we can identify the ‘island’ component (Figure 4.9d) and take the y coordinate of the centroid of this region.

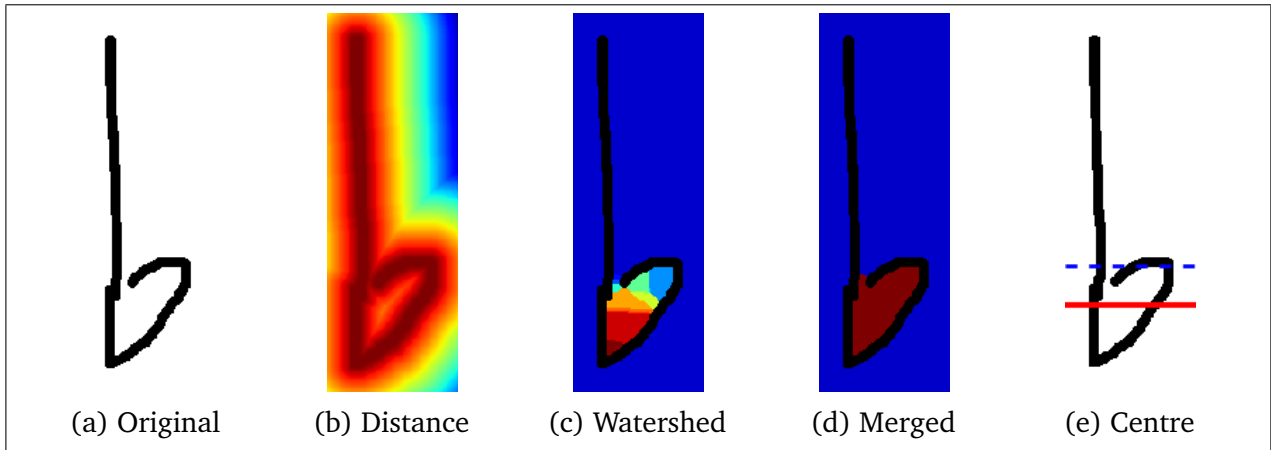


Figure 4.9: Identifying flat centre with watershed segmentation

An alternative solution uses dilations. This involves expanding the drawn image (and consequently shrinking the flat’s ‘head’ region) evenly until a distinct island region is formed (Figure 4.10b). Once that happens the vertical centre of the ‘head’ can be established in the same way as outlined previously, using the centroid.

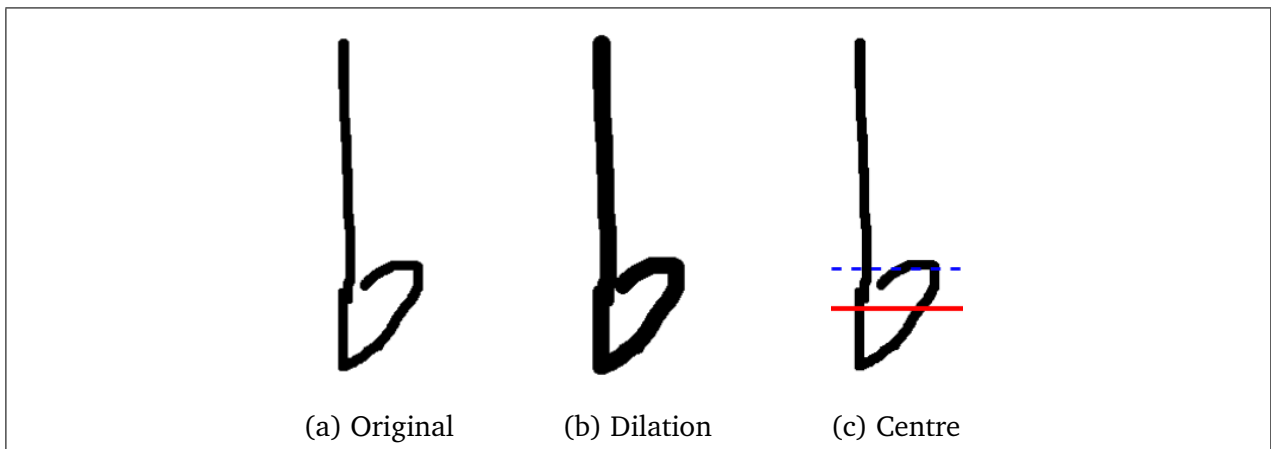


Figure 4.10: Identifying flat centre with dilation and connected component segmentation

4.2.4.3 Note Head Centres

For note heads, it was discovered that the centroid was a good representation of the centre, even in case of broken minims as seen in Figure 4.11e

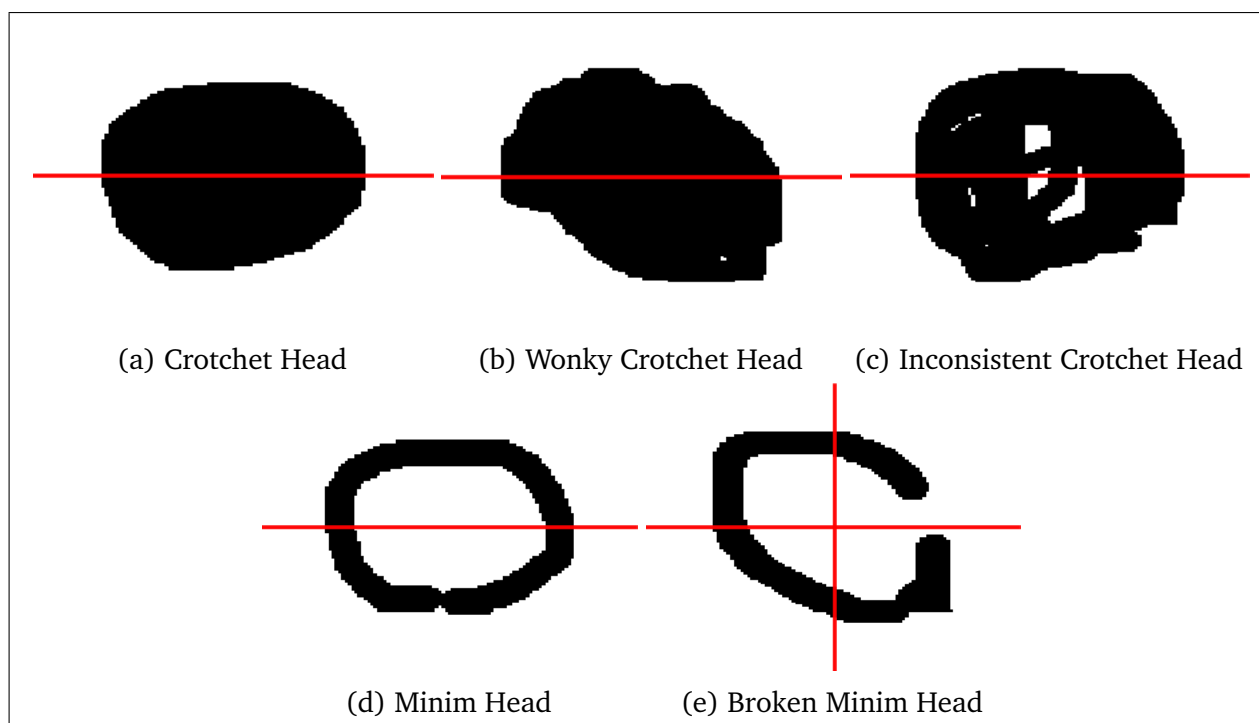


Figure 4.11: Filled and hollow note heads with their centres identified by two red intersecting lines

4.2.5 Duration

There are two main steps to the process for calculating note duration. The first is the extraction of the fundamental note value, is it a semibreve, minim, crotchet, quaver or semiquaver? For a semibreve, the note value will always be four, however for other notes, things are not so straightforward.

We first examine each note individually to ascertain the components which make up the note duration and then assign it a base value. For example we are particularly interested in the note head (is it solid or hollow?) and any tails or beaming. An outline of this heuristic can be seen in Figure 4.13.

The number of tails is fairly straightforward as they are attached to isolated notes or are joined in a more complex beam such as that in Figure 4.12. However, we need to have some way to work out which notes have which values. To do this, a section the width of $\text{Stave Space}/2$ is examined either side of the note's head. The beam is analysed in these segments for the maximum number of vertical black runs which represents the number beams.

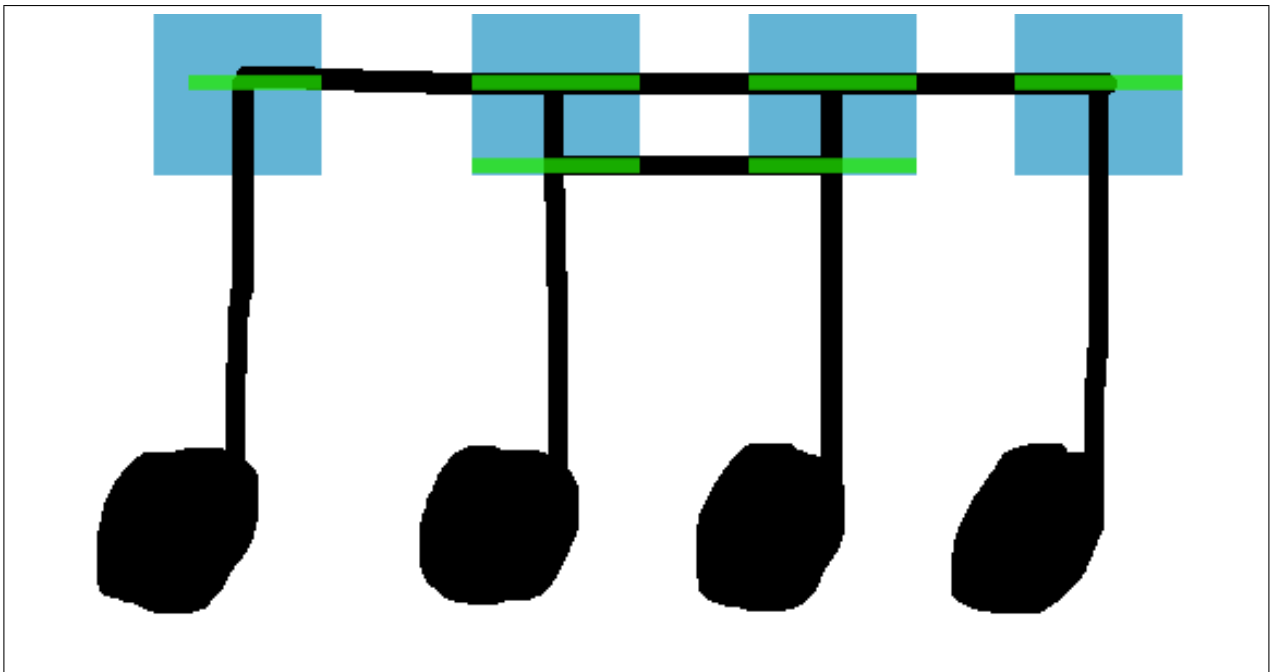


Figure 4.12: Calculating the number of beams to get note values. Blue areas show the regions scanned and green lines represent the maximum count of vertical black runs found

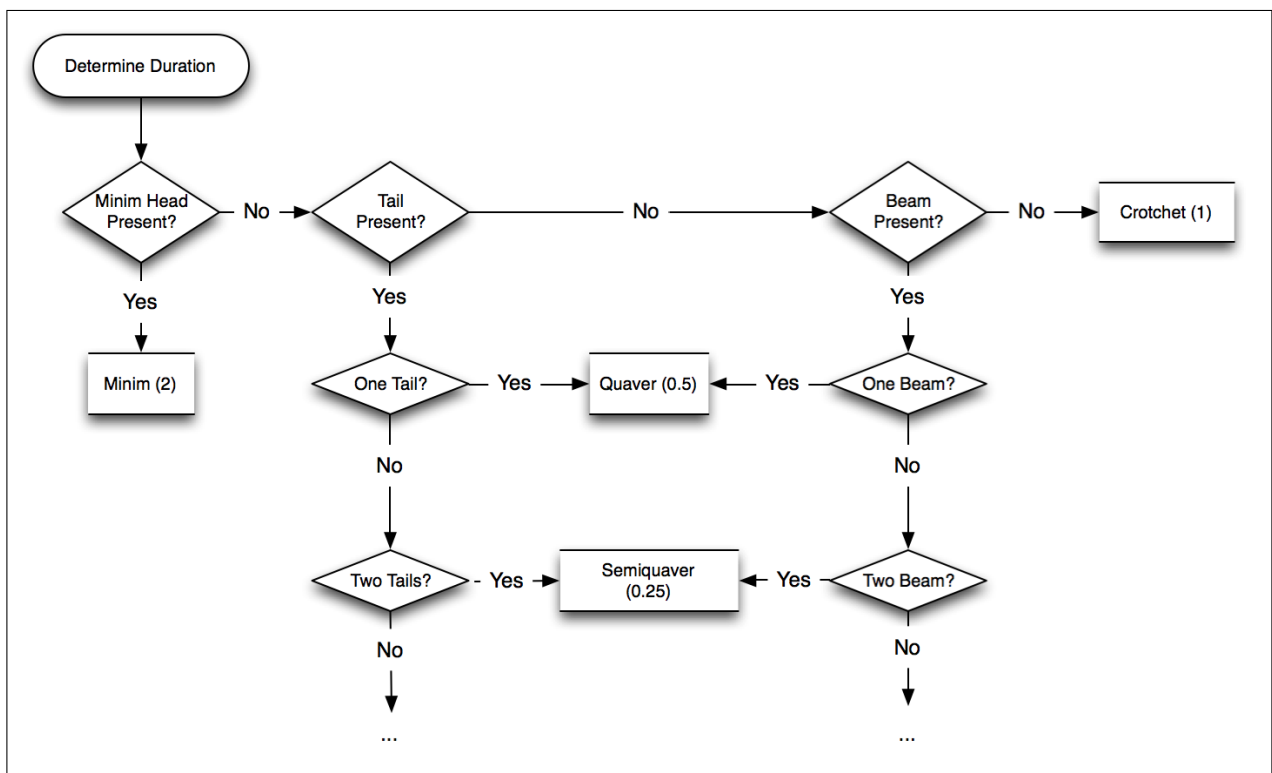


Figure 4.13: Flow chart for establishing base note duration

4.2. IDENTIFICATION

Note that with regards to tails and beaming, since every additional beam or tail present for the note divides it's value by two (examples can be seen in Table 2.1) we can generalise this section of the heuristic to deal with any number of beams and tails.

An example of a manuscript where the basic durations have been calculated can be seen in Figure 4.14

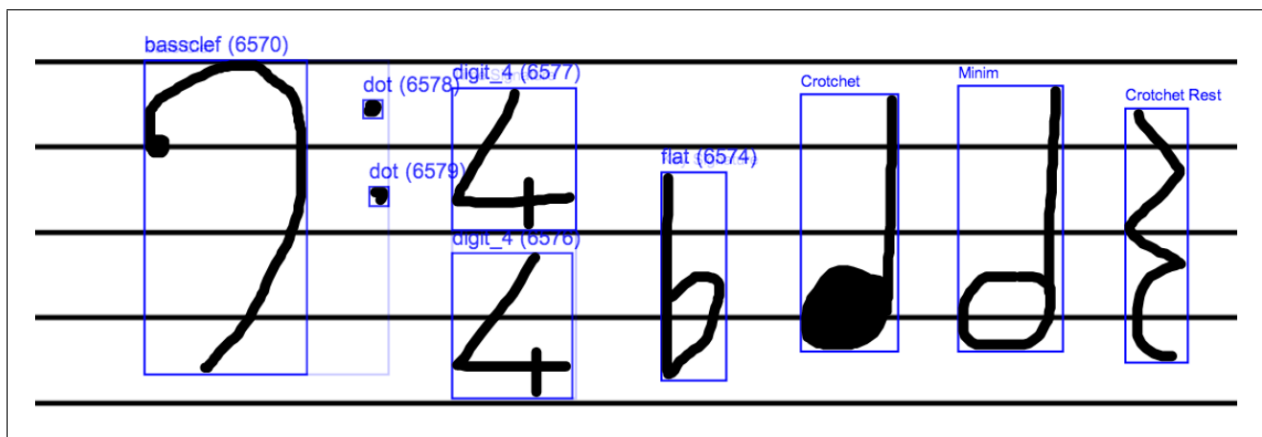


Figure 4.14: The manuscript after component duration analysis

Once the base value has been obtained, we perform a final check of the area surrounding the note centre for any ‘modifier’ dots. These extend the duration of the note by half, so for example, a dotted crotchet would last 1.5 beats as opposed to 1 beat without the dot. The region searched equates to half a staff space down and either side of the note as shown in Figure 4.15, anything outside of this region is ignored.

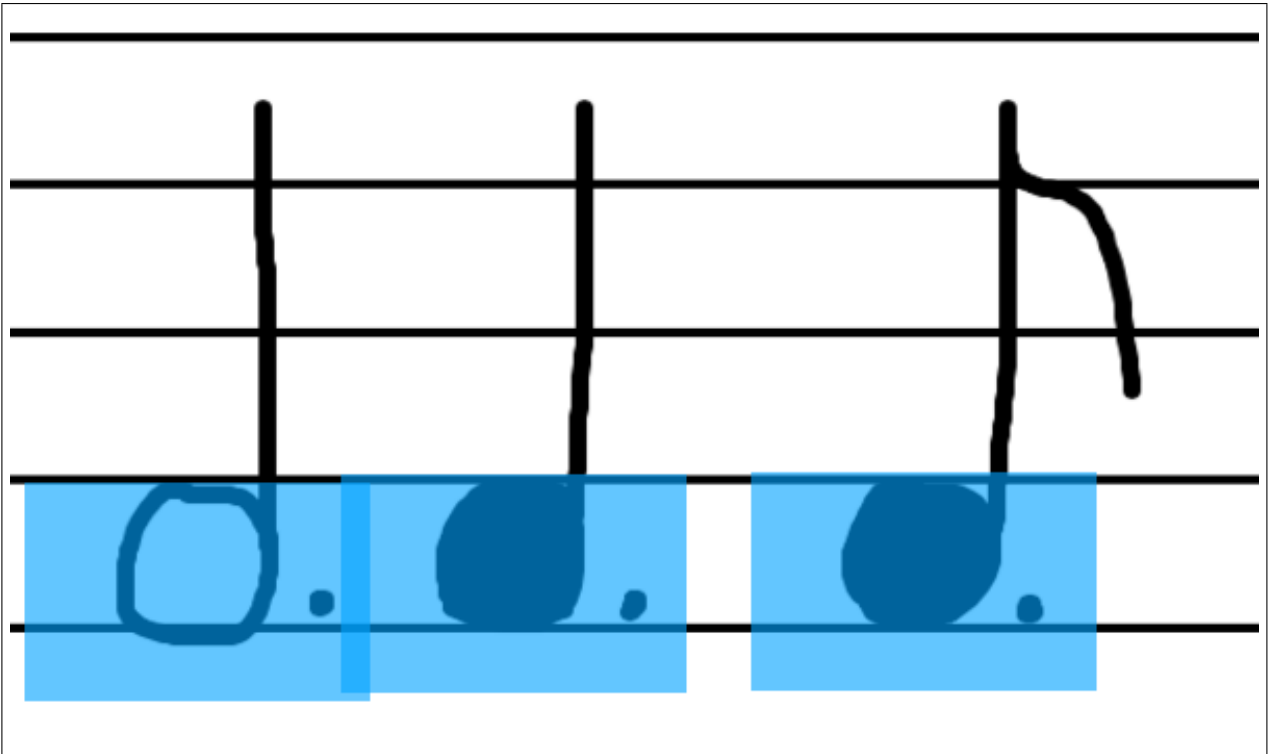


Figure 4.15: Searching for dots which would modify a note's length. Blue areas are the searched space for each note

4.2.6 Domain Knowledge

After all the segmentation, classification, pitch and duration analysis we can use domain knowledge as outlined in Sections 3.1.1 and 3.1.3 to apply musical rules (with looser thresholds and variations in control flow to account for the potential mistakes like those in Section 2.2) to group components as seen in Figure 4.16, enabling scoring Section 4.3 of the manuscript.

4.3 Scoring and Feedback

In this section I cover some of the techniques used to score and evaluate the now-identified musical entities along with their components.

Some of the rules and heuristics are very simple, but I mention them here for completeness.

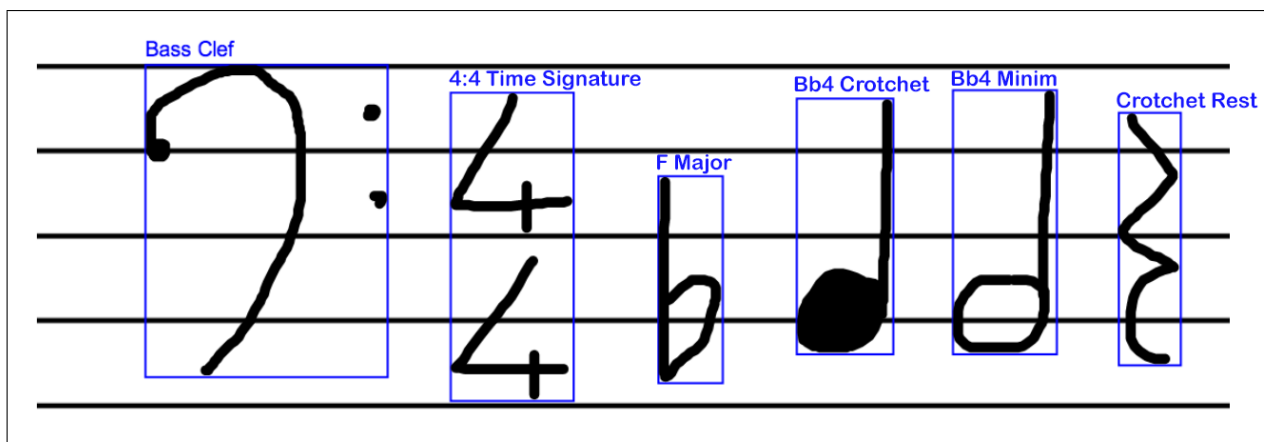


Figure 4.16: The manuscript after application of domain knowledge. Note the time signature, key signature and clef components have been grouped correctly

In general, I applied heuristics and thresholds based on dimensions and features of the musical score as well as using knowledge from the theory books Section 2.3.2 but in future it would be interesting to try and use the mistake data gathered in Section 2.2.1 and algorithms or machine learning to automatically adjust the thresholds either globally or maybe even on a per user level (for example, making judgement more lenient in the beginning and stricter as a student progresses, or vice-versa).

4.3.1 Pitched Notes and Accidentals

4.3.1.1 Position

For pitched notes and accidentals, we compare their vertical centres as calculated in the identification (Section 4.2) stages to positions of the surrounding stave lines and spaces.

We designate the attempted stave position as being the one which is closest to this center (the same position which was used for calculating pitch) and then measure the distance from it. If the distance is greater than a given threshold we then designate this a potentially ambiguous entity.

4.3.2 Key Signatures

A key signature consists of several sharps or flats which are required to be in a certain order and position on the staff.

Before we perform any analysis, we need a reference copy of what we expect the accidentals in the key signature to look like. We therefore count the number of drawn accidentals and then produce the set of accidentals we would expect to see for comparison. Once this has been done there are several analysis methods we can use to further assess the key signature which are summarised in Table 4.8.

Mistake	Feedback
Wrong Octave	Which sharps were affected and whether they're an octave too high or too low
Incorrect Order	Which sharps are in the wrong place and where they should be
Incorrect Accidentals	Which accidentals are wrong and what they should be

Table 4.8: Mistakes we need to look for in key signatures

4.3.2.1 Wrong Octaves

Sometimes the student gets the right accidental, but just in the wrong octave. This can be identified by checking to see if the actual note (C, D, E) of the drawn accidental rather than a specific pitch (C4, D4, E4) matches that of the expected accidental.

4.3.2.2 Out of Order

Sometimes a key signature contains all the sharps required, but not in the right order. This case can be established by taking the number of accidentals drawn, establishing what would be the correct accidentals for that key signature and comparing the list of pitches to an ordered list of the drawn pitches. If they match, the student has mis-ordered some of the accidentals so we note which ones are out of order and where they should be.

4.3.2.3 Incorrect Accidental

In the case of an Incorrect Accidental, the other three conditions having been checked, the student will have got the right number of sharps/flats but not the right accidentals meaning it's not a permutation or ordering issue and it's not that an accidental is in the wrong octave as the result of a clef mistake. Comparison with the correct reference example for the given number of accidentals enables us to identify specifically which accidentals are incorrect.

4.3.3 Beats and Timing

There are really only two things we need to check for when scoring beats and timing, either too few or too many beats in a bar as outlined in Table 4.9.

Mistake	Feedback to relay
Too few beats in a bar	Which bar, how many beats we count and how many we expect based on the time signature
Too many beats in a bar	Which bar, how many beats we count and how many we expect based on the time signature

Table 4.9: Mistakes we need to look for in beats per bar

Regardless of which mistake we're checking for the algorithm is essentially the same and can be checked at the same time as demonstrated in Algorithm 3

Algorithm 3 Searching for incorrect beats per bar

```

procedure CHECKBARS(bars)
  for each bar in stave.bars do
    expectedBeats = GETEXPECTEDBEATSINBAR(bar)
    actualBeats = GETACTUALBEATSINBAR(bar)
    if actualBeats > expectedBeats then
      Log the mistake
    else if actualBeats < expectedBeats then
      Log the mistake
    end if
  end for
end procedure

procedure GETACTUALBEATSINBAR(bar)
  totalBeats = 0
  for each note in bar.notes do
    totalBeats += GETNOTELENGTH(note)
  end for
  return totalBeats
end procedure

```

4.3.4 Stems

Stems are a very common component of manuscript and as such will be drawn regularly. It is important to have detailed checking to weed out any bad habits early and so we check for all the potential mistakes in Table 4.10.

Mistake	Feedback
Straightness	Whether it's just a bit uneven or really wonky
Angle	Whether the stem was leaning just a little bit or severely
Direction	Which direction the stem should have been pointing
Side	Which side the stem should have been on
Length	Whether it was too long or too short

Table 4.10: Mistakes we need to look for in note stems

4.3.4.1 Straightness

Given a drawn note stem, we wish to be able to determine a measure of ‘straightness’ which we can threshold to discern a badly drawn stem shown in Figure 4.17a from a straight one as shown in Figure 4.17b.

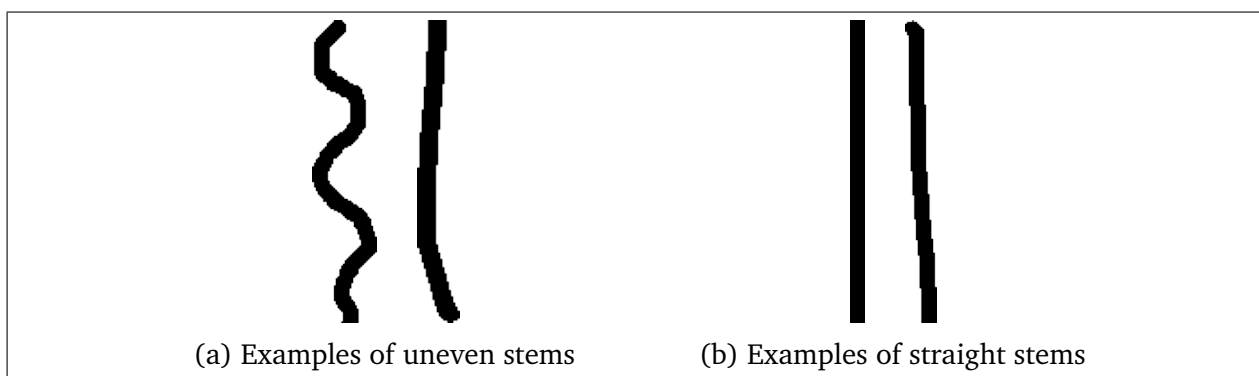


Figure 4.17: Examples of straight and uneven stems

Stem straightness is different to the stem angle because intuitively even if the stem is angled, it can still be straight. Therefore, we need to establish a technique which gives a measure irrespective of the stem angle.

To do this, we take the original stem (Figure 4.18a) and generate it’s skeletal representation using techniques outlined in Section 3.7.3 which approximates a line following the center of the stem (Figure 4.18b). If we treat this skeleton as a plot of points, we can draw a line of best fit through them to approximate what a perfectly straight version of the stem (Figure 4.18c).

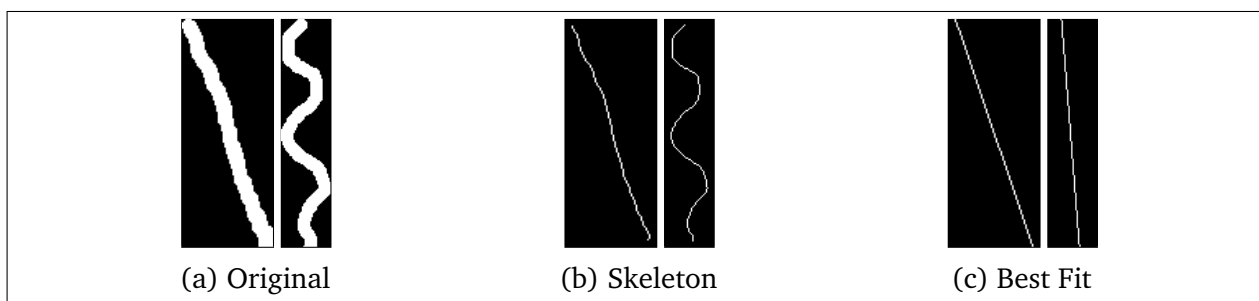


Figure 4.18: Examples of stem skeletons

We now have a skeleton line Equation (4.1) and a best fit line Equation (4.2) we can calculate the difference at each point Equation (4.3).

$$L_{\text{skeleton}}(x) = (a_0, a_1, a_2, \dots, a_n) \quad (4.1)$$

$$L_{\text{ref}}(x) = (b_0, b_1, b_2, \dots, b_n) \quad (4.2)$$

$$R(x) = L_{\text{skeleton}}(x) - L_{\text{ref}}(x) = (r_0, r_1, r_2, \dots, r_n) \quad (4.3)$$

$$\text{Straightness} = \sigma = \sqrt{\frac{\sum_{i=1}^n (r_i - \bar{r})^2}{n-1}} \quad (4.4)$$

After experimenting with using the standard deviation Equation (4.4) of the residuals as the straightness measure the results turned out to be positive upon visual inspection. As you can see in Figure 4.19 when ranked according to their ‘straightness’ measure, the stems do indeed appear to be ordered according to what one would visually define as being ‘straight’.

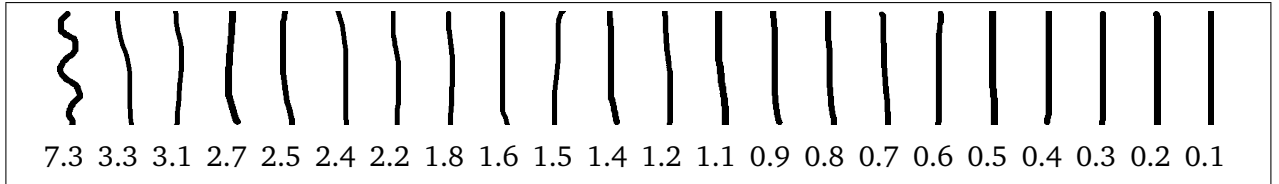


Figure 4.19: Ranking of stems according to their ‘straightness’ score

4.3.4.2 Angle

In order to determine the angle of a stem, we first need establish the line of best fit which most accurately reflects the direction of the stem, we can then examine this line $y = mx + c$ and compute $\arctan(m)$ to get the angle relative to vertical.

4.3.4.3 Direction

To establish the stem direction which I will refer to as S_d , the vertical position (y coordinates) of the head and the stem are compared.

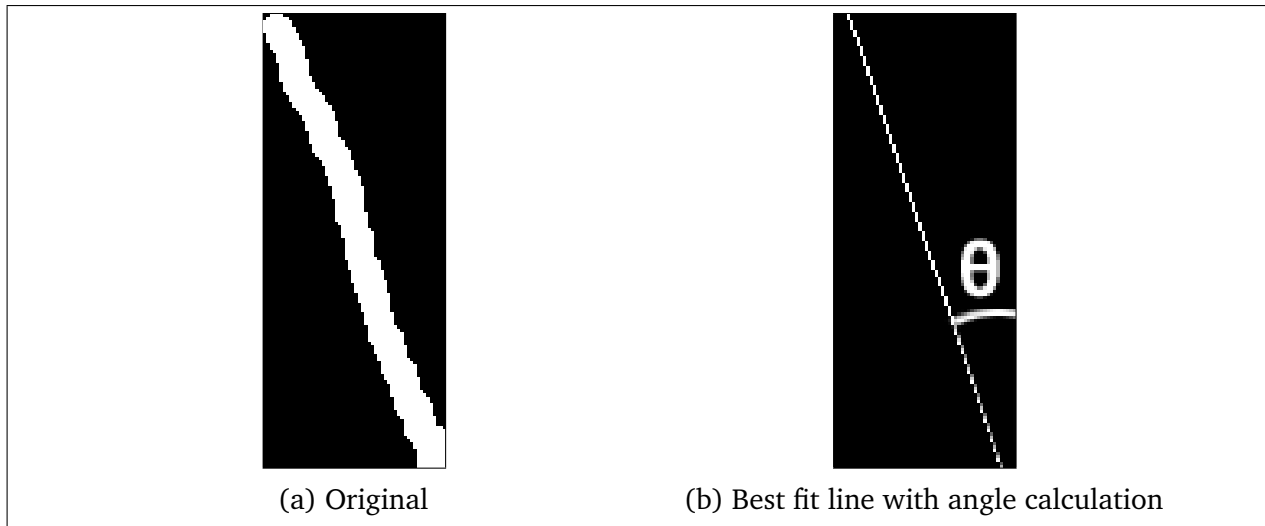


Figure 4.20: Examples of stem skeletons

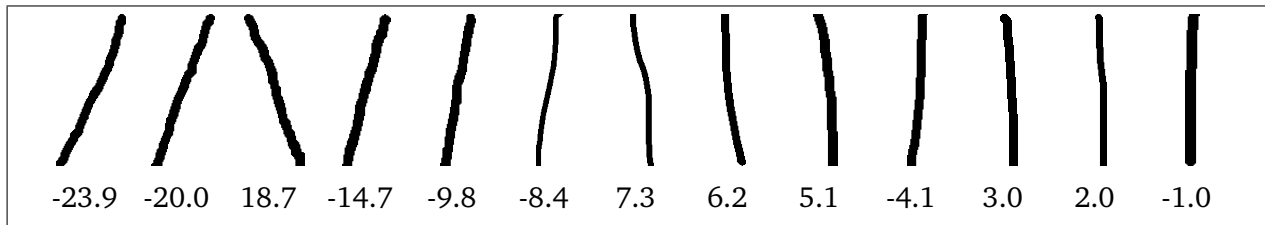


Figure 4.21: Ranking of stems according to their absolute angle (in degrees)

If the stem is located at $(x_{\text{stem}}, y_{\text{stem}})$ and the note head at $(x_{\text{head}}, y_{\text{head}})$, knowing that the coordinate axes have an origin starting from the top right of a given image, we can establish the following classifications:

$$S_d(y_{\text{stem}}, y_{\text{head}}) = \begin{cases} \text{up} & \text{if } y_{\text{stem}} < y_{\text{head}} \\ \text{down} & \text{if } y_{\text{stem}} > y_{\text{head}} \end{cases}$$

The case where a stem has the same y coordinate as its head isn't possible due to the way stems are extracted from a note complex as for this to happen a stem would need to be extracted *out* of a note head. Since note heads are removed in order to identify stems as in Section 4.2.1.2, this is impossible.

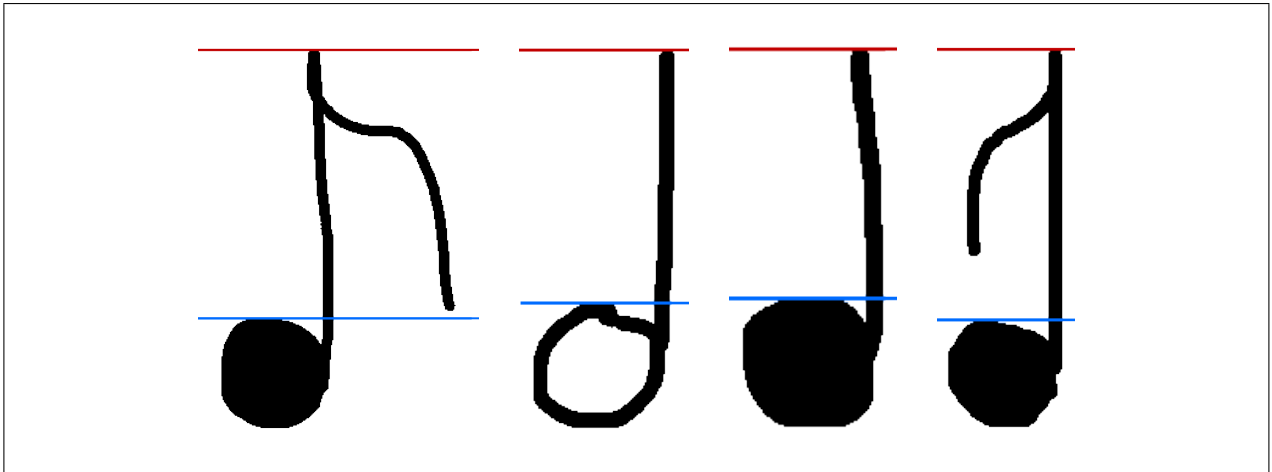


Figure 4.22: Identifying upward stems, the red line represents the stem y coordinate and the blue represents the head's y coordinate

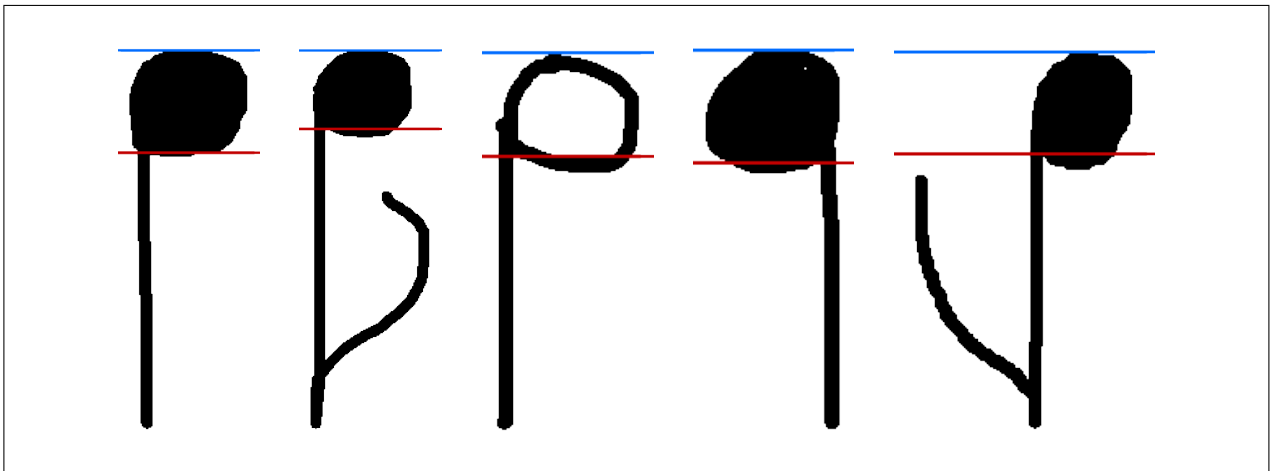


Figure 4.23: Identifying downward stems, the red line represents the stem y coordinate and the blue represents the head's y coordinate

4.3.4.4 Side

To establish the stem side which I will refer to as S_s , we do a similar operation to in determining the direction, however, since it is perfectly feasible that the stem and head have the same x coordinate, we can't just use the x coordinate directly.

Instead, we compare the x coordinate of the stem x_{stem} to the centroid of the note head $\mathcal{C}x_{\text{head}}$

Again, given that the coordinate axes start from the top left, we can now establish the stem x offset

$$S_{\text{xoff}} = x_{\text{stem}} - cx_{\text{head}}$$

Using this, we can establish the side classification as

$$S_s(S_{\text{xoff}}) = \begin{cases} \text{left} & \text{if } S_{\text{xoff}} < 0 \\ \text{right} & \text{if } S_{\text{xoff}} \geq 0 \end{cases}$$

4.3.4.5 Length

Stem length may seem like a simple case of measuring the length of the extracted stem, but it's actually a little more complicated. In order to get the best results, we need to take into account the fact that the stem has been extracted from the note head and that this separation point could occur anywhere from near the bottom to the top as shown in Figure 4.24.

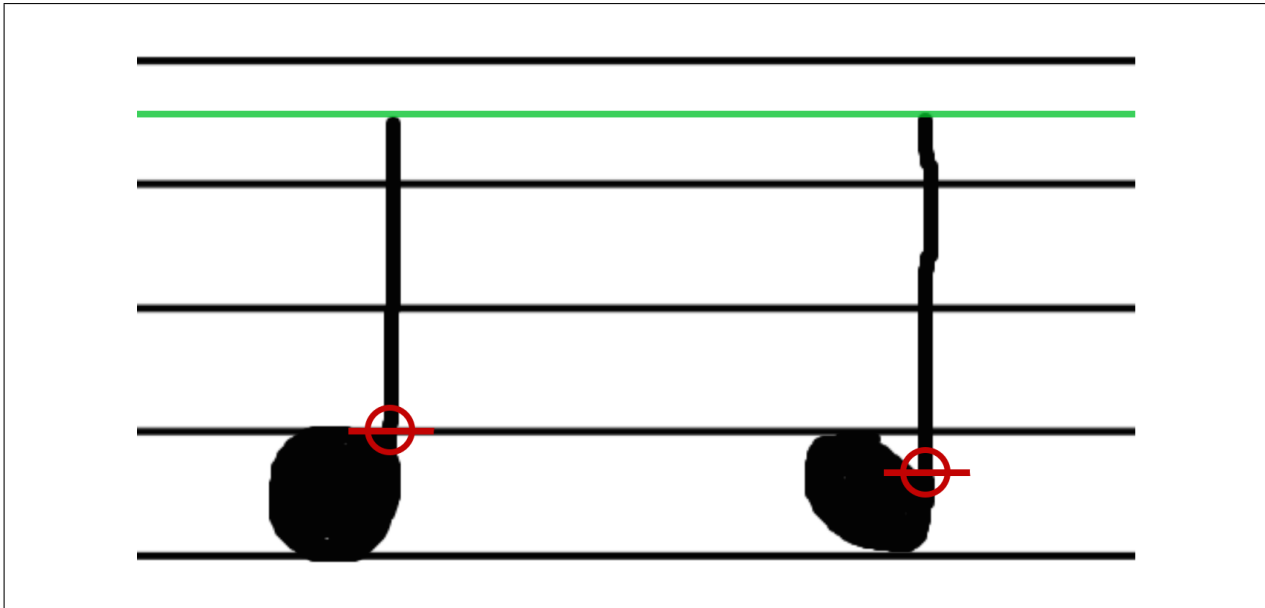


Figure 4.24: An example of the effect stem and note head intersection can have on length. Note that the right hand note would, if the raw stem was extracted, have a longer stem (potentially perceived as *too* long) but in actual fact both stems reach the required height

Similarly to the straightness and angular measures, we can establish the raw length L_R by drawing a line through the perceived centre of the stem and measuring its length using trigonometry. To be valid a stem it must lie between 2 and 3 stave spaces in length, any less and it's too short, any more, too long.

4.3.5 Quaver Tail

4.3.5.1 Side

To determine on which side a quaver tail lies, left (Figure 4.25a) or right (Figure 4.25b), initially I attempted to utilise vertical projections, much as in segmentation (Section 3.4.2).

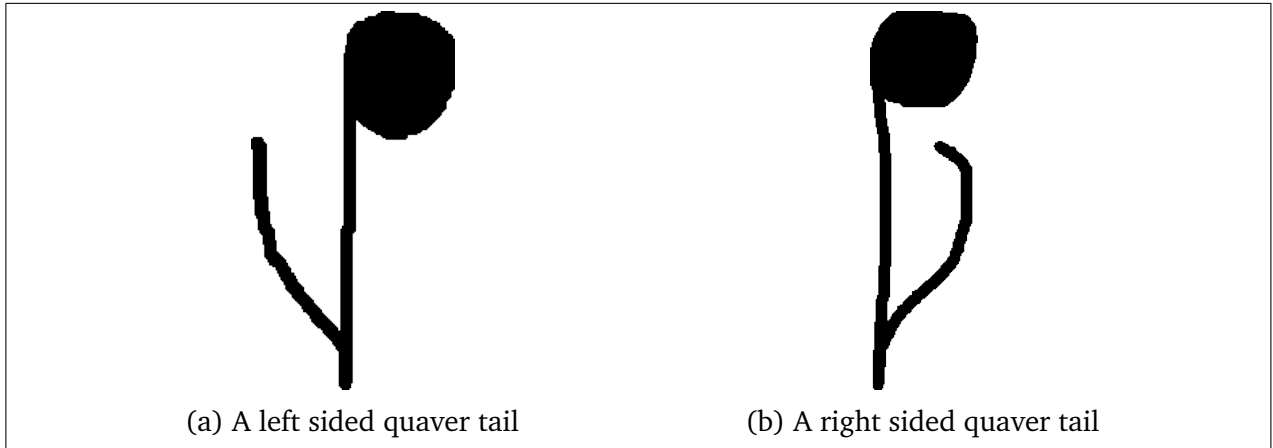


Figure 4.25: Quaver tail sides

After taking a vertical projection of the quaver tail, the maximum peak in the projection is taken and by establishing on which side of the centre the peak lies, this told us which side the ‘flick’ of the quaver is on.

Another method I investigated was to divide the image into two horizontally, then calculate the average black pixel column coordinate in each half as seen in Figure 4.26. By working out which half has the highest average pixel position, we can then establish the side on which the tail lies. In practice this proved the simpler method out of the two and the outcomes equally successful, when tested with the same sample data.

4.3.6 Note Heads

4.3.6.1 Size

Note heads must be approximately one staff space in height and up to 1.5 staff spaces in width. If it’s larger than these dimensions we consider it to be too large.

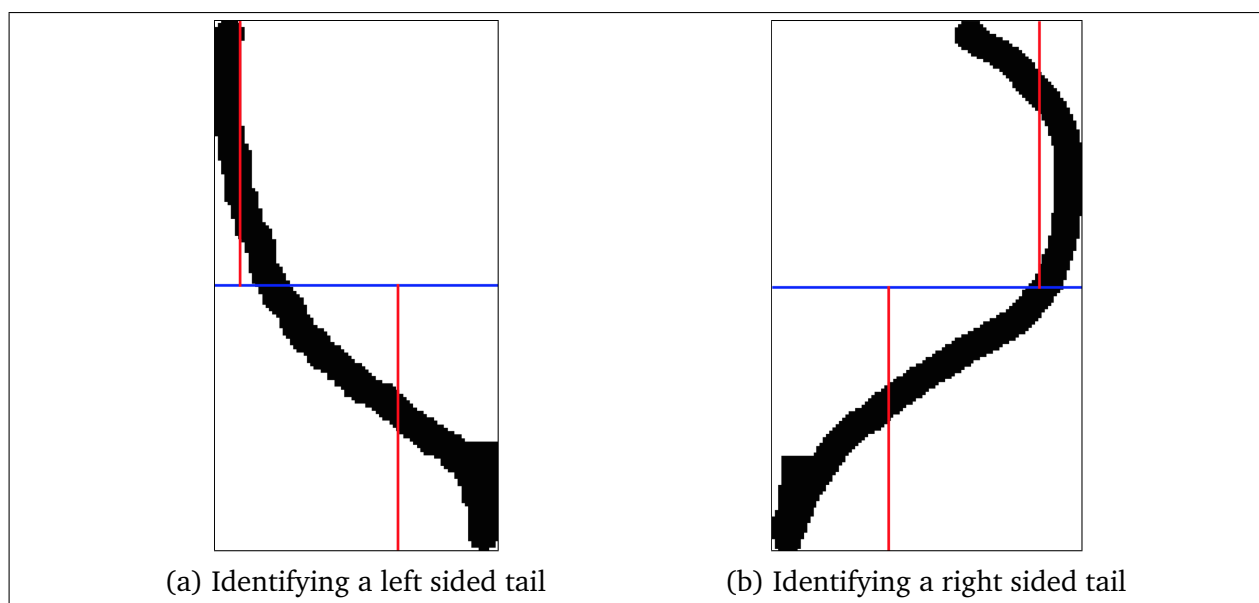


Figure 4.26: Calculating the quaver tail side with pixel averages

4.3.6.2 Messy Crotchet Heads

Some crotchet heads are badly drawn in that they're quite clearly not filled in all the way and have been left full of holes. I have called these 'messy crotchets' and I identify them by finding all the white regions inside the filled head and counting the number of them with an area above a threshold value which I periodically update as the application gathers more data, but is generally between 5-10 pixels. If the number of these 'holes' exceeds four (also obtained experimentally), then the head is marked as 'messy'.

It may be interesting in future to perform a comparison with a different heuristic (like total percentage of area which is filled) or perhaps by altering the thresholds automatically to bring the classification more in line with the professional dataset and improve the results.

4.3.6.3 Broken Hollow Heads

Broken minims and semibreves can be spotted trivially. Once they have been classified, we simply pad the image, resulting in a black ring (hopefully unbroken) surrounded by a white background with a white centre. By performing a connected component analysis on the white regions, if we only get one region, the shape is broken.

4.3.6.4 Angled Semibreve

We've already seen the zeroth and first order moments and to work out the angle of an ellipse, we can use the second order moments $m_{2,0}$, $m_{0,2}$ and $m_{1,1}$ to calculate the major axis of a region. We can then store the angle between this axis and the horizontal, the *orientation* form which we can later establish whether the semibreve is at an incorrect angle.

Chapter 5

Implementation

5.1 Architecture

The architecture of **NoteED** consists of 3 primary components, a client application Section 5.1.1, a server application Section 5.1.2, and an image processing and a processing service responsible for the image analysis and machine learning Section 5.1.3. The latter two both access a backing store in the form of a Postgres¹ database as seen in Figure 5.1.

¹<http://www.postgresql.org/>

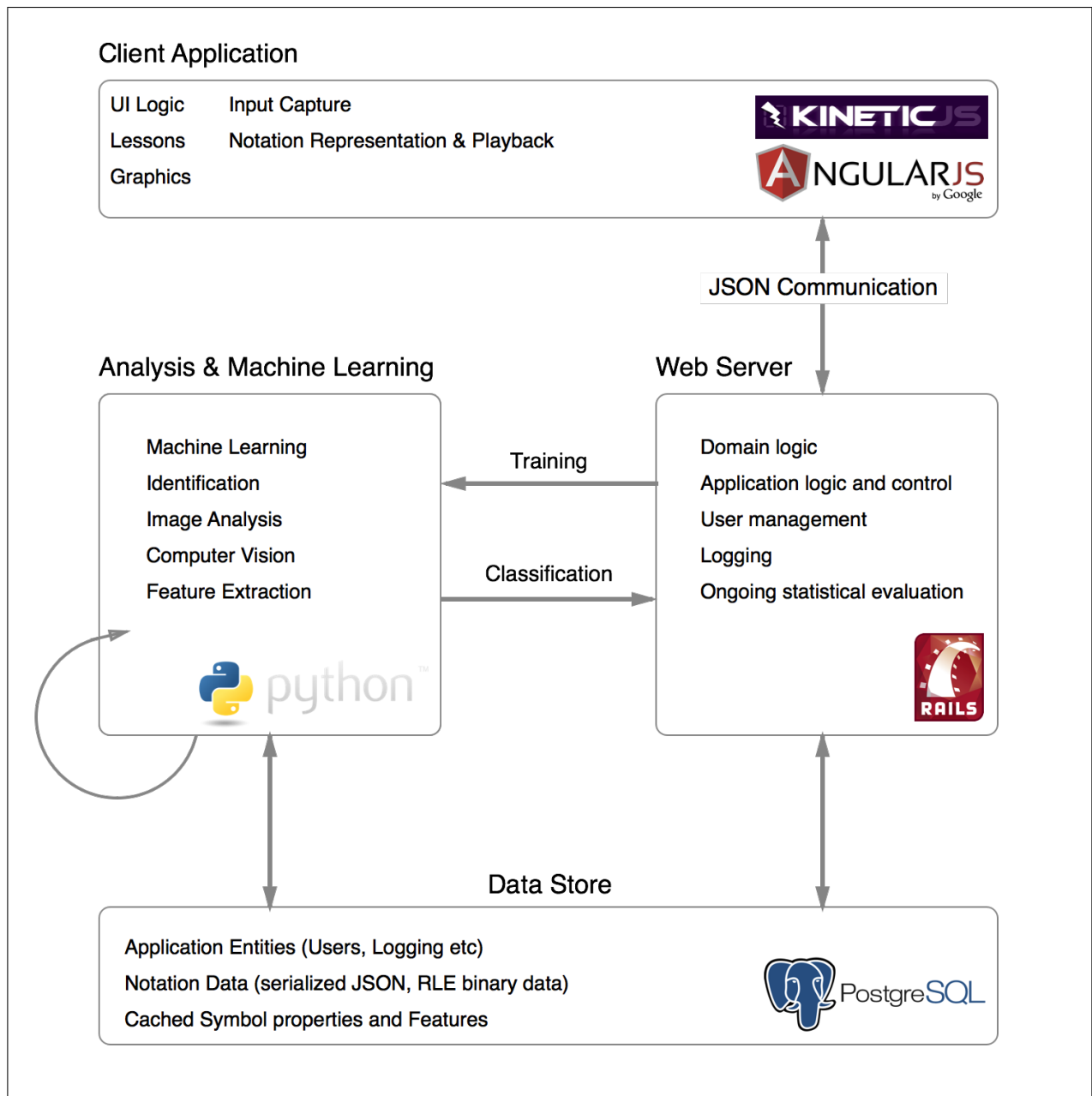


Figure 5.1: Overview of the system implementation architecture

5.1.1 Client

The main component with which the user interacts is the client application. The client application takes an MVC² approach, is an entirely browser based application written using

²Model View Controller

5.1. ARCHITECTURE

Google's AngularJS³ framework which communicates with the server via lightweight JSON data. This means the browser only ever needs to fetch from the server once to get all the templating information (aside from data) and the UI is then completely driven by the data.

5.1.2 Server

The server is written in Ruby on Rails⁴, a web development framework. This server handles the logic behind all the application entities like users, notation attempts, combining components to form more advanced musical entities and marshalling data to send to the client application.

The server is also responsible for both feeding data into the processing service and also for classifying new data against it before relaying feedback to the client application once it has been generated by the processing service.

5.1.3 Processing Service

The processing service takes an image and performs the techniques necessary to segment and identify fundamental components using a number of image processing techniques as seen in Section 4.2. To do this, the system periodically generates the two KNN classifiers seen in Section 4.2.3.1 using existing data⁵ and stores them to disk using *Pickle*⁶.

When a new staff needs classifying, the processing service segments the image into components, loads the first classifier model from disk and runs the new components through it. It then performs further segmentation and analysis on any higher level components requiring further decomposition, before running them through the level 2 classifier.

The processing service also extracts a component's features such as x and y position on the staff, width, height, area and component-specific features like straightness and angle for stems or messiness for crotchet heads. After the processing, segmentation and classification have taken place, the service stores the components in a database, linking them to the

³<https://angularjs.org/>

⁴<http://rubyonrails.org/>

⁵I make use of the Pedregosa et al. 2011 python module to assist with this step

⁶<https://docs.python.org/2/library/pickle.html>

original drawing using a database schema which maps directly to a model in the web application.

At any point, we can request a ‘reclassification’ or a ‘re-evaluation’ of a component to regenerate all its features if we later want to generate a different feature set, perform improvements (for example the location of the ‘center’ for different components) or add additional evaluation variables.

5.1.4 Core Database Schema

Column	Type
id	integer
email	character varying(255)
created_at	timestamp without time zone
updated_at	timestamp without time zone
Indexes	
users_pkey	PRIMARY KEY, btree (id)

Table 5.1: Components Table

Column	Type
id	integer
user_id	integer
file	character varying(255)
binary_file_data	bytea
created_at	timestamp without time zone
updated_at	timestamp without time zone
json_data	json
Indexes	
drawings_pkey	PRIMARY KEY, btree (id)

Table 5.2: Drawings Table

5.1. ARCHITECTURE

Column	Type
id	integer
drawing_id	integer
features	json
created_at	timestamp without time zone
updated_at	timestamp without time zone
manual_class	character varying(255)
parent_component_id	integer
auto_class	character varying(255)
feedback	json
Indexes	
components_pkey	PRIMARY KEY, btree (id)

Table 5.3: Components Table

Column	Type
id	integer
type	character varying(255)
drawing_id	integer
created_at	timestamp without time zone
updated_at	timestamp without time zone
Indexes	
entities_pkey	PRIMARY KEY, btree (id)

Table 5.4: Entities Table

Column	Type
id	integer
user_id	integer
entity_id	integer
mistake	character varying(255)
created_at	timestamp without time zone
updated_at	timestamp without time zone
Indexes	
entity_mistakes_pkey	PRIMARY KEY, btree (id)

Table 5.5: Entity Mistakes Table

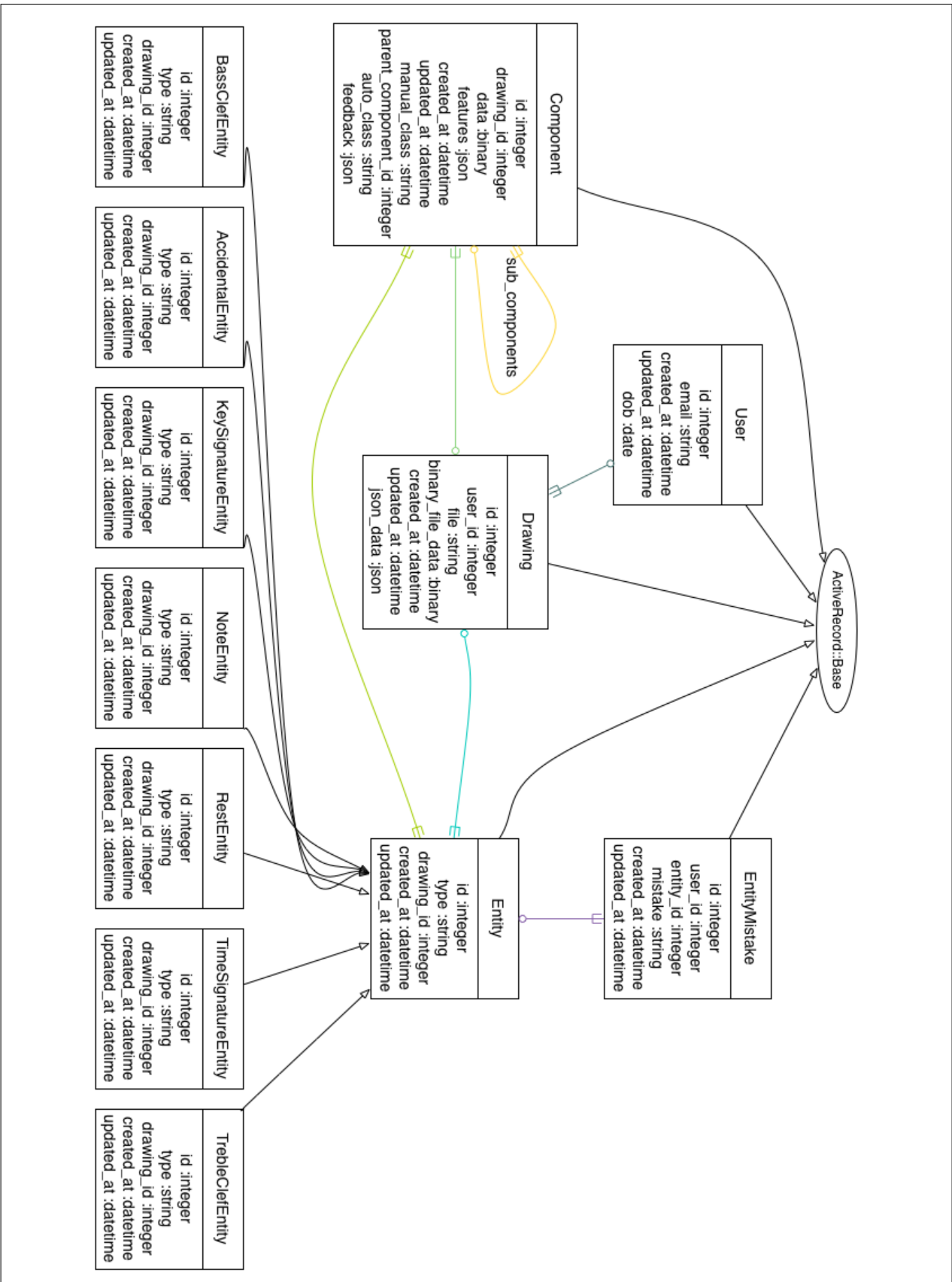
Column	Type
id	integer email character varying(255)
created_at	timestamp without time zone
updated_at	timestamp without time zone
dob	date
Indexes	
entity_mistakes_pkey	PRIMARY KEY, btree (id)

Table 5.6: Entity Mistakes Table - This is where we log a professional's observations for existing musical entities

5.1.5 Entity Relationship Diagram

The following is a summarised view of the core entities that make up **NoteED** and how they're related to each other.

5.1. ARCHITECTURE

Figure 5.2: **NoteED** Entity relationship diagram

5.2 Input

5.2.1 Medium Selection

The first step in this project was to establish the best method for a student to interact with the application. As seen in Section 3.1 different mediums for OMR have been tried, however which one would be most suitable for a student to use wouldn't necessarily correlate with which was best for quality of scanning, speed of analysis etc.

5.2.1.1 Flat Bed Scanner

The most simple of all the input methods, this would involve a student writing on a sheet of manuscript paper, then using a flat bed scanner (the type commonly found as a standalone device and in multi-function printers) to input the sheet into the computer. This method allows for a scan with high and consistent light levels (minimising noise and light-dependent artefacts), minimal distortion due to paper curvature (as the scanner usually flattens the sheet) and a high resolution end image, typically 300-600dpi.

The application would process the sheet using traditional OMR techniques and then provide feedback to the student. This technique would therefore require ownership of both a scanner and a computer on which to install and run the application. Alternatively, the student could upload the scanned image to a web based service for analysis, removing the need to install software.

5.2.1.2 Gestures

Taking input on a tablet enables the capturing of individual strokes. I decided to leverage more traditional OMR techniques by first rendering the strokes as an image but there is potential for future research to see if using statistical features of the strokes themselves in a similar way to the methods described by Taubman, Odest and Jenkins 2005 and George 2003. I also looked at character recognition using multi-stroke gestures and \$ N-Stroke recognisers and variations such as \$ N-Protractor in Anthony and Wobbrock 2012.

My main reason for not investigating gesture based recognition further at present is that the recognition techniques seemed to rely on users having a consistent stroke structure for the different note entities. The most promising work is that of Taubman, Odest and Jenkins

2005 but even then, the project used a very rigid structure of which entities were expected to be drawn in which order.

There also seemed to be a small number of users in the Taubman, Odest and Jenkins 2005 study and the author notes that user testing and experiments on the transferability of models between users was not done. Sadly I can't perform a lengthy calibration stage where I can trust the user's input which makes a per-user model difficult.

5.2.1.3 Tablet Input

Although I decided not to use gestures, I did opt to use a tablet as my primary method of input.

In testing it proved a fairly natural experience for the children who trailed **NoteED** using the freedrawing environment. Interestingly although most people initially said they wanted to use a stylus, when it came down to it they regularly interchanged between the two⁷. Both finger and stylus work well as input methods and although I didn't gather data on the quality of drawing versus the input method, there were no immediate differences of problems encountered by using one over the other for children.

Interesting, adults seemed to prefer the stylus, my hypothesis for this (which was then confirmed in further conversation) was that since adults have larger fingers, it's harder for them to accurately track the point at which their line is being drawn on the canvas.

It was also suggested to me that if the application could be accessed through a web browser and scale according to the browser, many schools now have smartboards⁸ which are in effect large tablets.

5.2.2 Capturing Strokes

To capture strokes drawn by the user, several listeners are attached to the HTML5 canvas element on which the manuscript is rendered. When the user presses their mouse down or initiates a touch event, a new array of line points is created and the initial point of interaction is stored in that array. From there, any mouse or touch movement whilst the

⁷This is a classic example of when watching what your users do is much better than asking them what they want, a technique I used quite a few times in this project

⁸<http://smarttech.com/Home+Page/Solutions/Education+Solutions/Products+for+education>

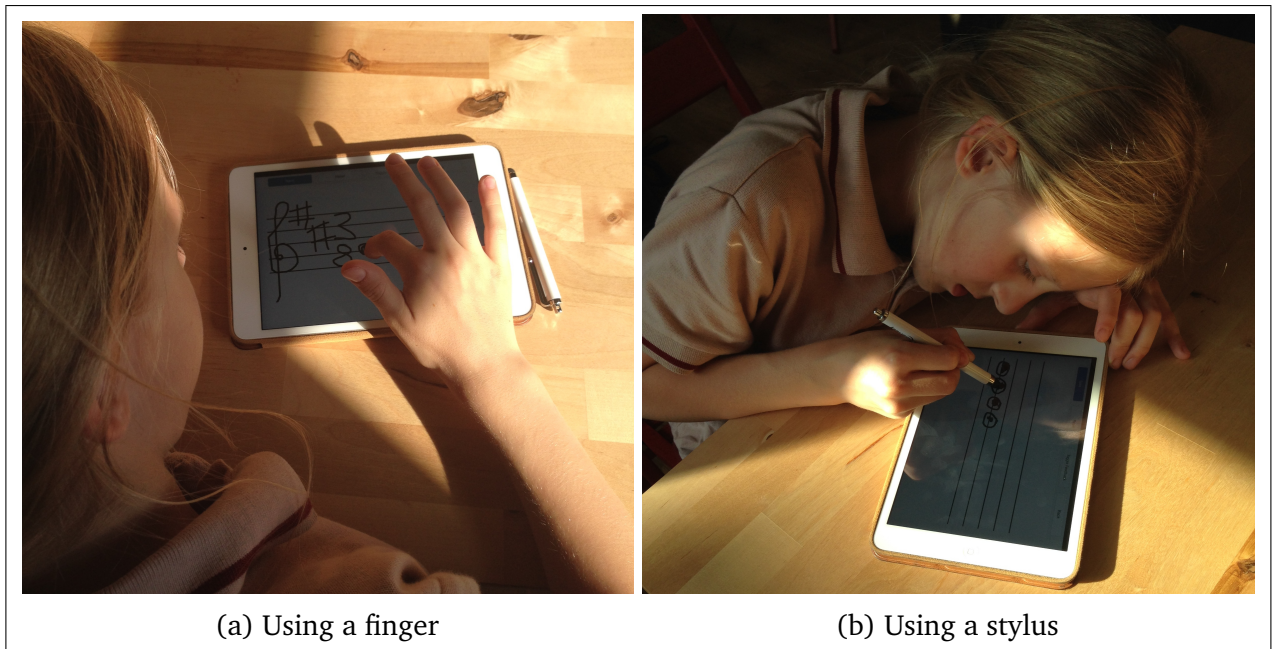


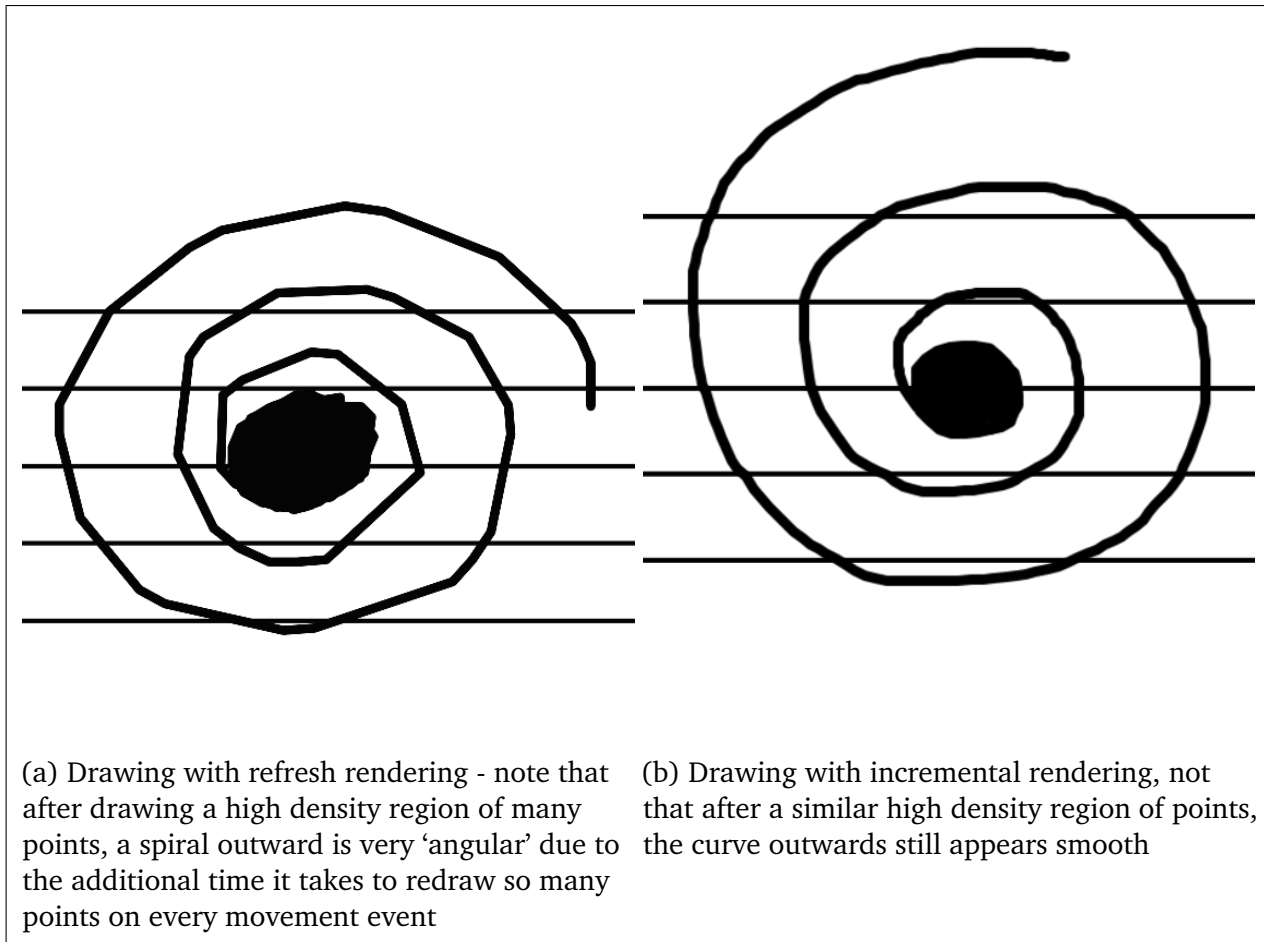
Figure 5.3: Examples of a student trying different tablet input methods

touch or ‘mousedown’ event is active triggers a new point recording, until the mouse button is released or the touch event ends. In this way, an array of points is built up which represent a drawn line.

Each time the line changes via the addition of new points, a redraw event is triggered which connects all the points together and renders the line onto the canvas.

In initial experiments with simple manuscript entities, this worked well, however as children began to experiment with more complex entities (requiring more lines and therefore more points) the length of time required to redraw the lines on the canvas became prohibitively long. Eventually a lag occurred between ‘pen’ movement and lines being rendering on the canvas, the result of which was that as more lines were drawn, the time between new points being registered increased. This increase led to the drawing experience feeling (as one user described it) ‘really clunky’ and long straight lines appearing instead of smooth curves (Figure 5.4a).

To counteract this, I modified the code such that it renders new line segments on the fly, rather than re-rendering all the lines every time a new point is added. Although this requires some more logic to render the line segment, overall the experience turned out much smoother as the small increase in the length of the code path had much less impact than re-rendering the drawing every time (Figure 5.4b).



5.3 Data Storage and Retrieval

5.3.1 Stave Drawing

I store the data gathered in Section [5.2.2](#) in two ways.

Firstly, I store the serialised JSON of the stroke data in the database which is postitive for several reasons pertaining to future work as well as immediate benefits for **NoteED** . It enables more potential experiments in future in the area of gesture based segmentation and recognition (see Section [5.2.1.2](#)) and could support showing a student corrections by transforming points in the strokes as opposed to transformations on the image. It also facilitates some great UI features like playing back the drawing for the tutor so they can see exactly how the student approached the problem.

Secondly, I store the entire rendered canvas in the cloud using an Amazon S3 bucket, as a

‘reference copy’ in case I need to check the JSON against the original image at a later date or re-analyse images after updating my classifiers, features or analysis techniques.

5.3.2 Components

Once components have been extracted from the drawing using the techniques outlined in Section 4.2, I store both their features and [Run Length Encoding \(RLE\)](#) representation of the whole component in the database for retrieval later during any further image processing.

Although 3000 components uses around 1GB in data when raw, using compression we are able to lower this considerably using run length encoding as shown in 5.7.

Metric	Without RLE	With RLE	Improvement
Storage Required	759 MB	22 MB	91.7%
Total Retrieval Time	1639 ms	42 ms	97.4%

Table 5.7: To improve the speed of my application, I utilised [Run Length Encoding \(RLE\)](#) (covered in Section 3.5.3) to improve storage and retrieval times during feature extraction and classification by up to 97% (Table 5.7).

5.4 Feedback

I experimented with several ways of presenting feedback to the user leveraging all the information extracted in previous stages but there were only three which came across reasonably well. I outline them briefly below before explaining my eventual feedback technique.

5.4.1 Listing Mistakes

Using knowledge of what the user has done wrong on which entities, we can display a simple list of feedback or a "well done" message if they have no mistakes.

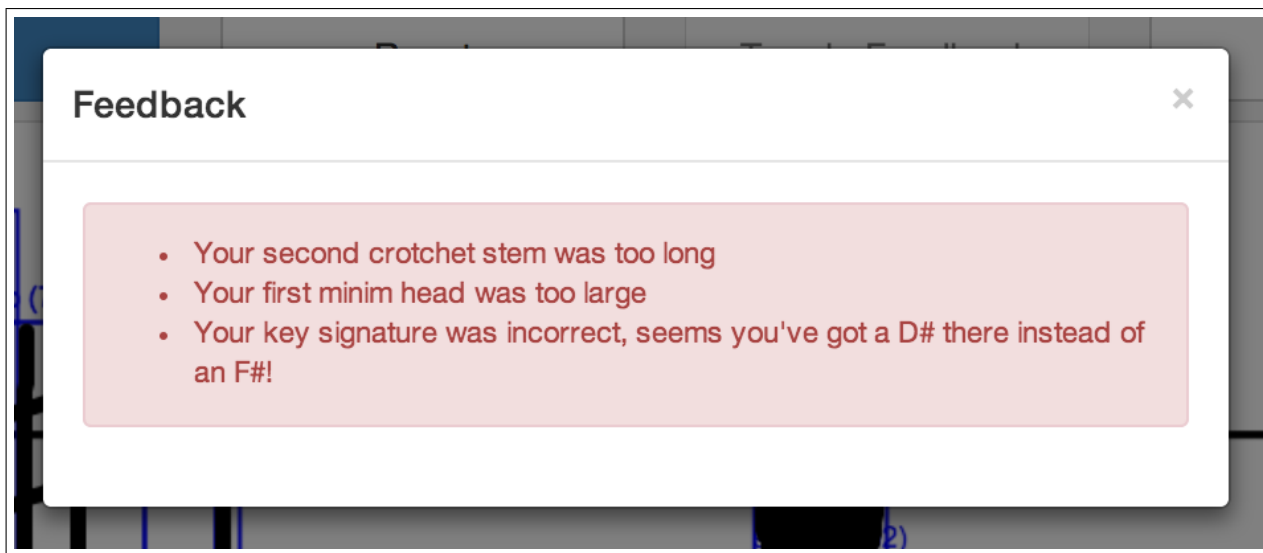


Figure 5.5: Feedback in list form

The advantages to the list-style feedback were:

- It's explicitly stated what's wrong
- It's relatively easy to generate and you can use music theory to provide hints as to what they might do to fix the problem

However in testing some of the negative feedback was that:

- It's a little daunting if you're just starting out
- It's quite boring
- It takes a minute to work out which components you're referring to and you keep having to toggle the feedback on and off while you check
- It won't always mean much unless the person already knows some music theory - not great for beginners

5.4.2 Colour Coding

The next method I tried was simple colour coding. If something was wrong or had a problem, it got coloured in.

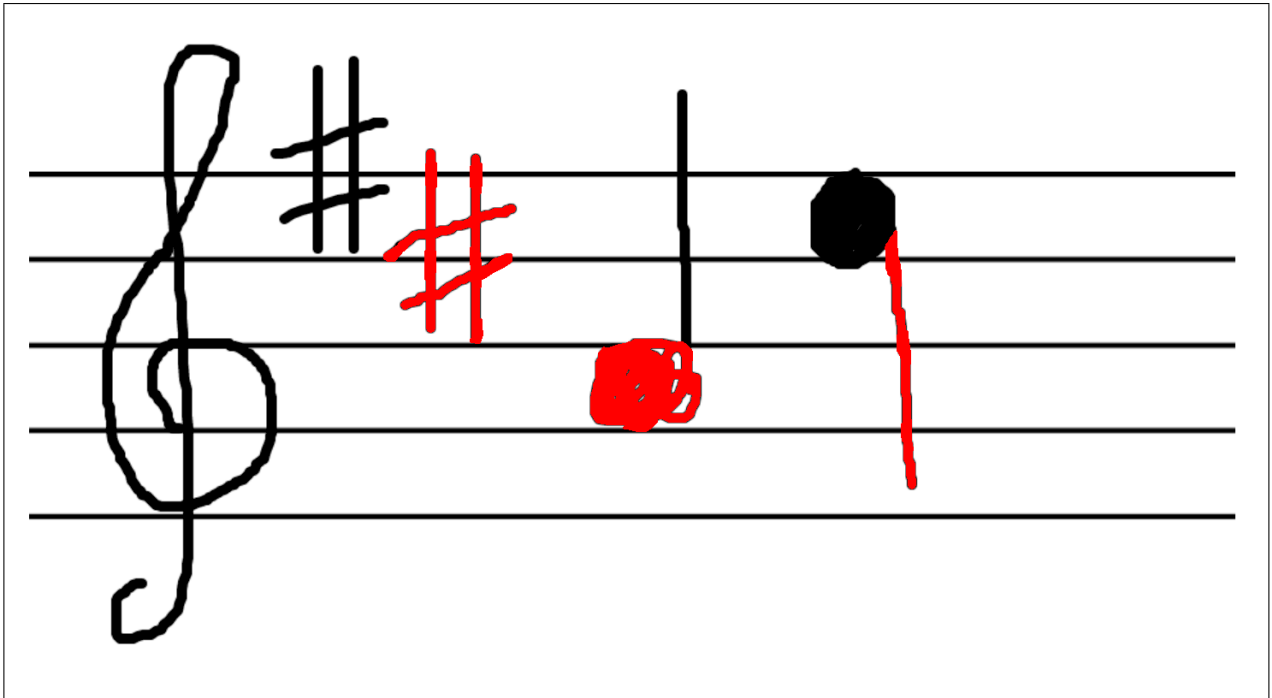


Figure 5.6: Feedback in colour coded form

The obvious disadvantage to this was that while you know exactly which components are wrong, you can't be 100% sure what exactly the problem is. Usually if you've made the mistake you're unlikely to immediately jump to the right conclusion. For example, thinking you're being marked down/highlighted for drawing a crotchet messy while actually you're putting the stem on the wrong side would be very frustrating.

5.4.3 On Screen Correction

I toyed briefly with the idea of on screen correction using scaling, rotations and transformations. For example, if a stem is at an angle, correcting it by using an animation in front of the user worked really well if done component by component and made it immediately obvious what the problem had been. However this style of correction couldn't be performed on all components and greatly increased the complexity of implementation. I therefore decided to postpone pursuing this avenue, perhaps to revisit at a later date once more fundamental functionality was in place.

5.4.4 The Hybrid Approach

My actual feedback mechanism brings together multiple components to form what I believe to be a superior method of feedback.

1. Show a simple aggregate rating and instructions for how to load entity-specific feedback . The aggregate score is very simple right now, just taking the number of entities and the number of mistakes and using that to compute a percentage of how many errors there are, before mapping it to a rating system.
2. highlight the components which have had mistakes identified, as we did in Section 5.4.2 and allow click events on those components.
3. On click, show detailed feedback for the component as in Section 5.4.1
4. Give the option to see an animation or guide about how to correctly write that bit of notation

For example, in the case of a bad crotchet the interface looks something like this:

In Figure 5.8 an example of a student receiving her feedback for a job well done!

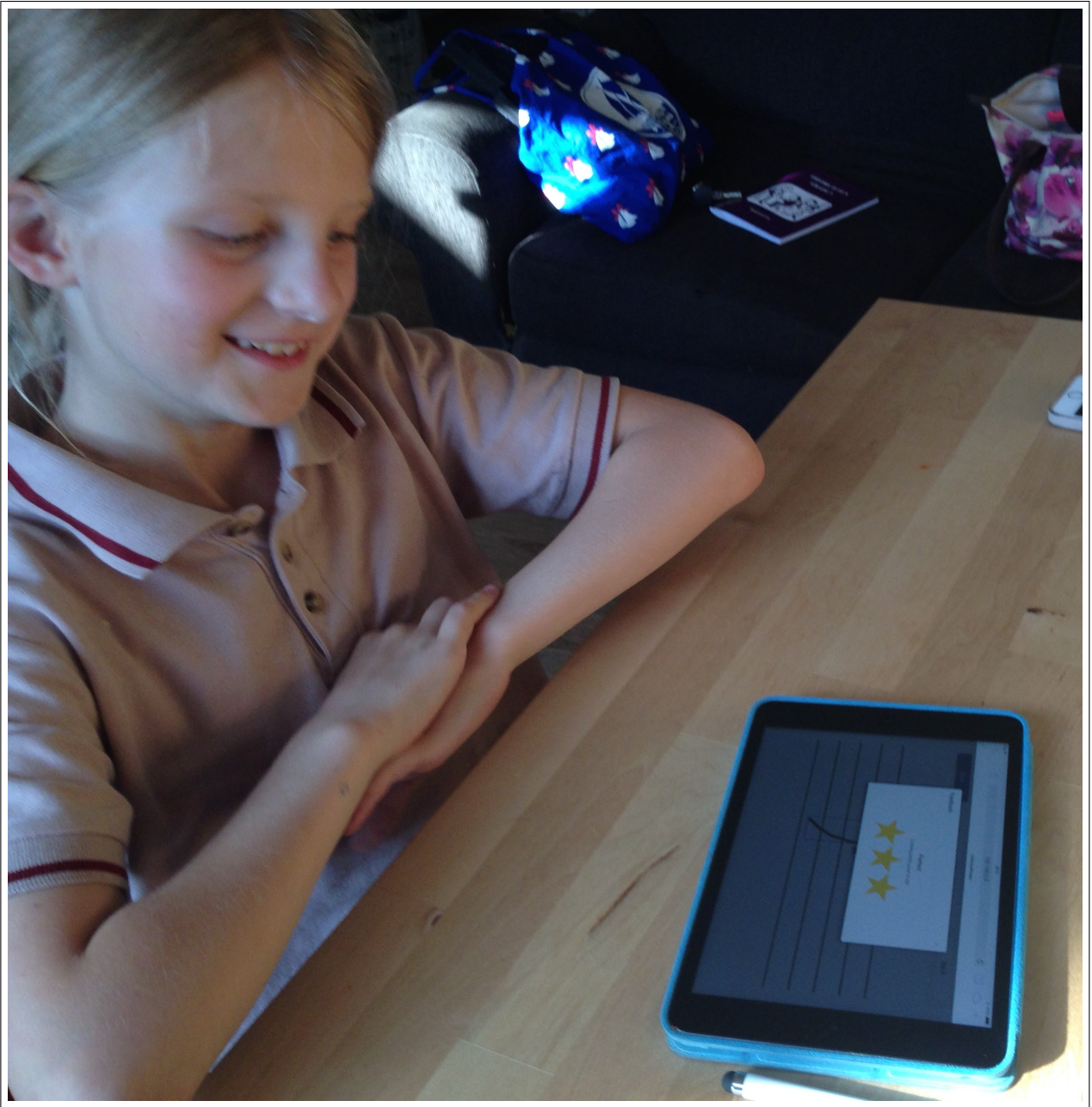
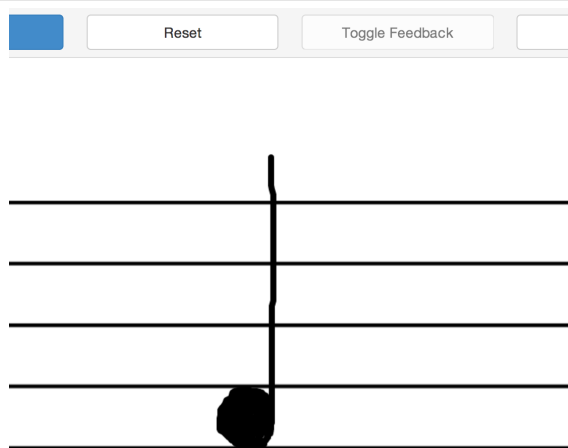
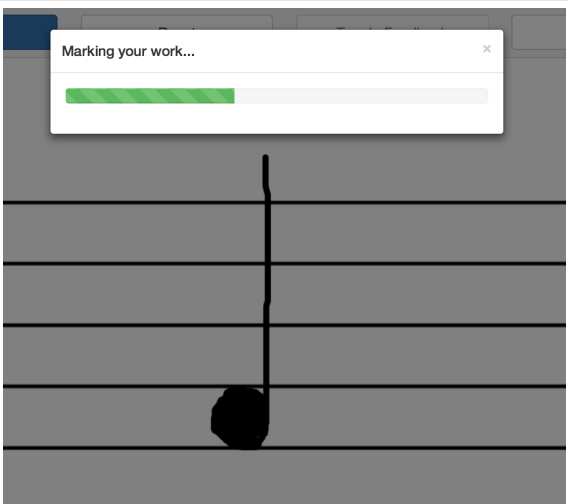


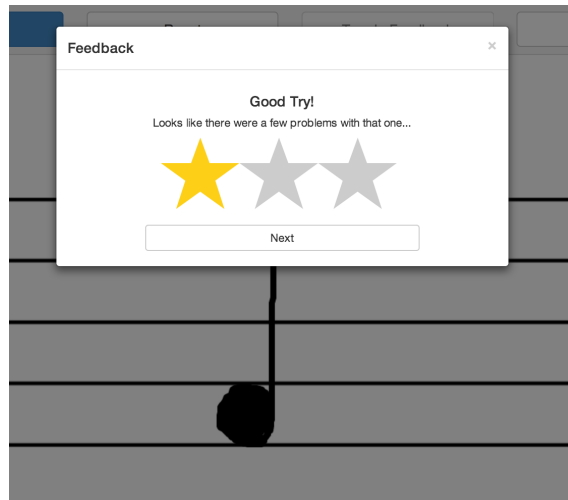
Figure 5.8: Child receiving simplified graphical feedback



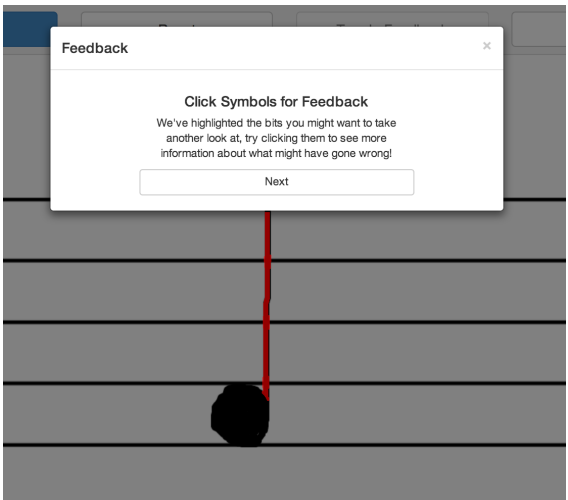
(a) The notation attempt



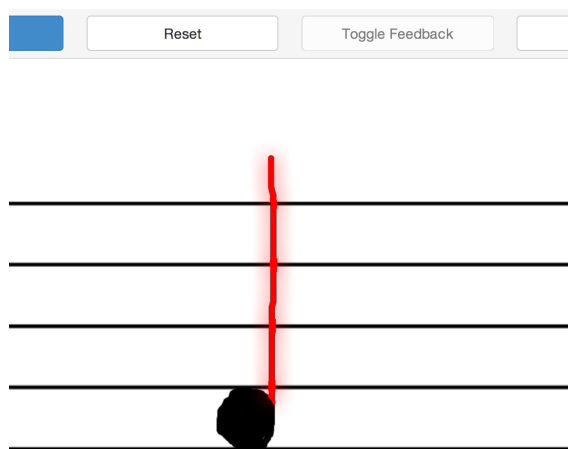
(b) Keeping the user updated...



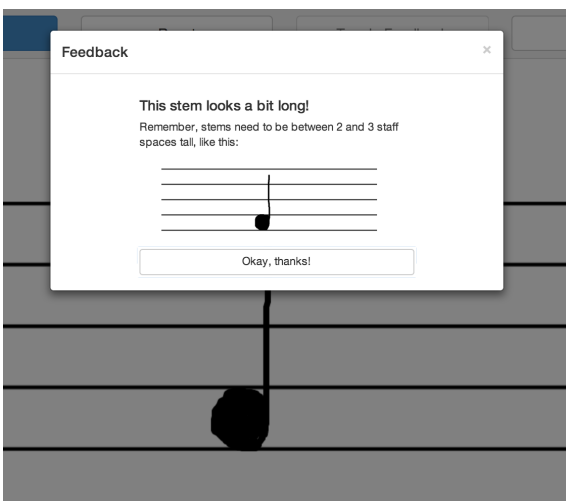
(c) Showing an aggregate piece of feedback



(d) Showing instructions for detailed feedback



(e) The clickable entity with a mistake



(f) The detailed feedback (the image is actually animated and shows a note being drawn on loop)

Figure 5.7: The feedback process for a bad crotchet length

Chapter 6

Evaluation

6.1 Mistake Detection

Objective: Enable the detection of common errors in notation symbols, pitch, time signatures and other musical features outlined in Table 2.4.

This is one of the most important evaluation components, can we in fact spot the same mistakes which a teacher would?

In short, yes, I believe we can and although long term data gathering and trials haven't been performed, preliminary data is positive.

Once the likely mistakes had been established (Table 2.4), I developed an application which assisted in rapid assignment of relevant mistakes to score entities by a human. I also used this over the original data gathered from students and ongoing data from participants and experiments.

The result was a database of entities and their mistakes which I then used to adjust thresholds and parameters in Section 4.3. It would be interesting to modify these values according to the results of scoring against this database automatically and I talk more about this in Section 7.2.

By comparing my heuristics against the 'true' data provided by professionals and competent musicians I generated confusion matrices for each of the mistakes.

The full dataset of results for the implemented mistake heuristics and algorithms can be seen in Table 6.1. Notice that for the majority of mistakes, we obtain a great accuracy,

however since most samples actually do *not* get labelled with a mistake, the accuracy is unfairly weighted by the large number of true negatives, meaning it isn't necessarily the best performance criteria. Instead, I think that it makes more sense to judge our system by its predictive power or *Precision* (the proportion of predicted positives which are actually positives). If we do this, the results still appear successful.

An alternative score which provides a weighted measure between recall (The number of actual positives correctly identified) and precision is the $F1$ measure defined in Equation (6.1).

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6.1)$$

Mistake	TP	TN	FP	FN	Accuracy (ACC)	Precision (PPV)	Sensitivity (TPR)	Fall Out (FPR)	False Discovery Rate (FDR)	Miss Rate (FNR)	F1
note-head-ambiguous	12	325	2	0	0.994	0.857	1.0	0.006	0.143	0.0	0.923
note-head-broken	24	314	1	0	0.997	0.96	1.0	0.003	0.04	0.0	0.98
note-head-angled	33	273	25	8	0.903	0.569	0.805	0.084	0.431	0.195	0.667
note-head-messy	7	327	0	5	0.985	1.0	0.583	0.0	0.0	0.417	0.737
note-head-too-big	31	298	9	1	0.971	0.775	0.969	0.029	0.225	0.031	0.861
note-head-too-small	8	325	0	6	0.982	1.0	0.571	0.0	0.0	0.429	0.727
note-head-wrong-type	2	337	0	0	1.0	1.0	1.0	0.0	0.0	0.0	1.0
stem-length-short	20	317	1	1	0.994	0.952	0.952	0.003	0.048	0.048	0.952
stem-length-long	20	313	3	3	0.982	0.87	0.87	0.009	0.13	0.13	0.87
stem-straightness	16	317	3	3	0.982	0.842	0.842	0.009	0.158	0.158	0.842
stem-direction-wrong	11	322	6	0	0.982	0.647	1.0	0.018	0.353	0.0	0.786
stem-side-wrong	15	315	9	0	0.973	0.625	1.0	0.028	0.375	0.0	0.769
stem-angle	21	302	14	2	0.953	0.6	0.913	0.044	0.4	0.087	0.724
dot-wrong-side	10	329	0	0	1.0	1.0	1.0	0.0	0.0	0.0	1.0
accidental-wrong-side	19	316	4	0	0.988	0.826	1.0	0.013	0.174	0.0	0.905
accidental-wrong-line	12	323	3	1	0.988	0.8	0.923	0.009	0.2	0.077	0.857
keysig-octave	0	29	2	1	0.906	0.0	0.0	0.065	1.0	1.0	0.0
keysig-order	1	29	1	1	0.938	0.5	0.5	0.033	0.5	0.5	0.5
keysig-incorrect	14	9	9	0	0.719	0.609	1.0	0.5	0.391	0.0	0.757

Table 6.1: The results of **NoteED** vs a professional's marking

6.2 Learning Improvement

Objective: Produce an application which can improve on and continue a child's learning outside of lessons

This is one of the most difficult outcomes to measure, initial in person feedback and conversation with students after performing various notation tasks suggest that **NoteED** is doing a good job so far.

However, since a conversation is not a quantifiable measure of how well a child's learning has been improved, there is also logging throughout the system for use over a longer period of time, data such as average mistakes per task or mistakes compared to how many times a task has been done before can therefore be analysed.

We propose that in order to best establish a reliable outcome for learning improvement, a minimum of one month's observation is needed and preferably closer in the region of 6 months if possible.

6.3 Engaging Experience For Child

Objective: Combine the tablet interface and notation analysis objectives to produce a streamlined experience which a student will happily engage with on a repeat basis.

Whilst talking with students was highly valuable in the design and implementation stages, what people say and what they do are often not the same¹. Therefore simply asking a student 'would you use this again' wasn't necessarily going to prove anything. Instead, we keep track of student actions and we can subsequently ascertain from these, more reliable readings how often the student is interacting the the application.

Preliminary data (gathered through initial trials) suggests that a student will usually perform a task 4-5 times before wanting to move on, usually because they got it right and got bored or in a few early cases, because the feedback and analysis wasn't being very helpful which was a useful discovery in itself. To keep the experience engaging, we propose that in further iterations, **NoteED** supports varying levels of difficulty by way of 'lessons' where users are given specific challenges to complete. Since lessons focus on very targetted criteria, they would also reduce the amount of feedback and criticism that a new student has to deal with.

¹<http://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/>

Chapter 7

Conclusions & Future Work

7.1 Conclusions

In this chapter we summarize what this project has achieved and where it might go next, providing suggestions for future work and experiments.

Before this project, no application existed which was capable of helping a child learn to correctly write musical notation whilst simultaneously providing feedback, other than their music teacher. This project has now made some progress towards solving this challenge.

A summary of core achievements on the way to solving this problem can be summarised as follows:

- We have created an application which allows a student to input a musical notation attempt by way of a graphics tablet and can send that input to a server for processing (Section [5.2.1.3](#))
- A modified OMR system was implemented within the application which we have shown to be capable of classifying the student's handwritten notation with a high degree of accuracy whether or not it is written in the 'correct way' (Section [4.2.3.1](#))
- We have leveraged this OMR system, along with domain expertise, to further identify the musical entities a student draws and analyse them individually and in relation to each other (Section [4.2.6](#))
- The analysis of the musical entities is capable of spotting mistakes in the notation and

feeding them back to the user in a smooth and intuitive fashion via the same interface they input their notation Section 5.4.4

- We assessed the implementation of the identification and feedback system and found that the mistakes identified by **NoteED** closely match those found by a professional music teacher, this was our primary goal and the results are promising, further research and development are recommended ()

In conclusion, we have succeeded in building an application which, give a notation attempt by a student on a tablet, is capable of providing meaningful feedback in a fast turnaround time, without intervention from a human and with a quality which closely resembles the essential feedback they would get from a music teacher.

7.2 Future Work

During the course of this project, a lot of work was put into the computational techniques behind the notation identification and analysis, leaving less time for user interface polish and experimentation. In preliminary testing, already pieces of feedback have been shared which I believe warrant further investigation and development time.

- **Horizontally scrolling manuscript** - Students are unable to really take full advantage of some of the features around beats, time signatures and measures since they can only effectively write one bar at a time. The most pressing next step is
- **Automatic adjustment of scoring thresholds** - Though I originally based my thresholds for spacing, neighbouring search areas and more on what I read in theory books, in reality, small changes made a big difference to the accuracy of the system and it seems how a professional teacher marks their students work can sometimes be more strict and sometimes more lenient. It would be interesting, therefore to see if we could generate thresholds based off of the EnityMistake data we gathered for testing in Section 7.1. Perhaps it would even be possible to generate unique “marking style” for different teachers.
- **Better features for teachers** - The teachers whom I showed **NoteED** to were very excited about it’s potential but some were also disappointed it was so focussed on the students. As teachers the feedback I got was that being able to see students’ work and track their progress was a highly desired feature.

- **More advanced OMR** - Whilst the OMR in **NoteED** performs well, it does so on a limited subset of traditional music. For example dynamics and note modifiers such as staccato, slurs etc. are not currently recognised. Experiments into whether these components can be added to the system are recommended.
- **Alternate feature sets** - Currently the resampled image is a small but not insignificant amount of data and it loses a lot of information when resized. Sadly my attempts at using statistical features weren't all that successful Section [4.2.3.1](#) but perhaps with further investigation some could be found.

Bibliography

- [1] DfE. *Attainment target level descriptions*. 2013. URL: <http://www.education.gov.uk/schools/teachingandlearning/curriculum/primary/b00199150/music/attainment> (visited on 24th Jan. 2014).
- [2] Melanie Spanswick. *Why is Grade 5 Theory so important?* 2012. URL: <http://melaniespanswick.com/2012/08/12/why-is-grade-5-theory-so-important/> (visited on 27th Jan. 2014).
- [3] Taylor, E.R. and Associated Board of the Royal Schools of Music. *The AB Guide to Music Theory*. The AB Guide to Music Theory pt. 1. Associated Board of the Royal Schools of Music (Publishing) Limited, 1989. ISBN: 9781854724465. URL: <http://books.google.co.uk/books?id=h7hZSAAACAAJ>.
- [4] E.R. Taylor. *Music Theory in Practice*. Simon & Schuster, 2008. ISBN: 9781860969461. URL: <http://books.google.co.uk/books?id=SeNlNwAACAAJ>.
- [5] Lina Ng. *My First Theory Book*. 2001.
- [6] Karl MacMillan, Michael Droettboom and Ichiro Fujinaga. 'Gamera: Optical music recognition in a new shell'. In: *Proceedings of the international computer music conference*. 2002, pp. 482–485.
- [7] Ichiro Fujinaga. 'Adaptive optical music recognition'. PhD thesis. McGill University, 1996.
- [8] Gabriel Taubman, A Odest and C Jenkins. 'Musichand: A handwritten music recognition system'. In: *Undergraduate Thesis, Brown University* (2005).

- [9] Barak Ben-Dayana and Ilai Giloh. ‘Optical Music Recognition’. In: (2013).
- [10] Ana Rebelo et al. ‘A method for music symbols extraction based on musical rules’. In: *Proceedings of Bridges 2011: Mathematics, Music, Art, Architecture, Culture*. Tessellations Publishing. 2011, pp. 81–88.
- [11] Han-Wen Nienhuys and Jan Nieuwenhuizen. ‘LilyPond, a system for automated music engraving’. In: *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*. Citeseer. 2003, pp. 167–172.
- [12] Daniel Taupin. *MusiXTEX – Using TEX to write polyphonic or instrumental music, version 1.93 edition*. 1999.
- [13] Linda G Shapiro and George C Stockman. ‘Computer Vision Prentice Hall’. In: *Englewood Cliffs, NJ* (2001).
- [14] Florence Rossant. ‘A global method for music symbol recognition in typeset music sheets’. In: *Pattern Recognition Letters* 23.10 (2002), pp. 1129–1141.
- [15] Wilhelm Burger et al. *Principles of Digital Image Processing*. Springer, 2009.
- [16] Dennis L Wilson. ‘Asymptotic properties of nearest neighbor rules using edited data’. In: *Systems, Man and Cybernetics, IEEE Transactions on* 3 (1972), pp. 408–421.
- [17] Ludmila I Kuncheva. ‘Editing for the k -nearest neighbors rule by a genetic algorithm’. In: *Pattern Recognition Letters* 16.8 (1995), pp. 809–814.
- [18] TY Zhang and Ching Y. Suen. ‘A fast parallel algorithm for thinning digital patterns’. In: *Communications of the ACM* 27.3 (1984), pp. 236–239.
- [19] Scikit Image. *Watershed segmentation – skimage v0.11dev docs*. (Visited on 15/06/2014).
- [20] Ana Rebelo. ‘Robust Optical Recognition of Handwritten Musical Scores based on Domain Knowledge’. PhD thesis. 2012.
- [21] F. Pedregosa et al. ‘Scikit-learn: Machine Learning in Python’. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

- [22] Susan E George. ‘Online pen-based recognition of music notation with artificial neural networks’. In: *Computer Music Journal* 27.2 (2003), pp. 70–79.
- [23] Lisa Anthony and Jacob O Wobbrock. ‘\$N-protractor: a fast and accurate multistroke recognizer’. In: *Proceedings of Graphics Interface 2012*. Canadian Information Processing Society. 2012, pp. 117–120.