

# **Piano Music Transcription Based on Computer Vision**

by

**Robert McCaffrey B.A. (Mod.)**

**Dissertation**

Presented to the

University of Dublin, Trinity College

in fulfilment

of the requirements

for the Degree of

**Master of Computer Science**

**University of Dublin, Trinity College**

Supervisor: Dr. Kenneth Dawson-Howe

May 2017

## Declaration of Authorship

I, Robert McCaffrey, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

---

Robert McCaffrey

May 17, 2017

---

## Summary

---

This dissertation explores a solution for automating piano transcription through computer vision in OpenCV. Transcription of music is here defined as the process of analysing a video feed so as to write down the musical parameters of the sounds that occur in it.<sup>1</sup> The composition performed on a piano shall be transcribed to sheet music using computer vision. The video will be captured by mounting a device with video capability at an angle overhead, while a pianist performs on the piano below. Computer vision estimation based models are then used to create an automated piano transcription software. Through computer vision, the objective of this dissertation is to explore the process of transcription via a different methodology, thus avoiding the difficulties associated with audio signal processing (where most contemporary research in transcription is focusing on).

At the beginning of this project, the goal of this dissertation was to design and implement a prototype automated piano transcription software that could correctly transcribe a piece of music played on the piano to sheet music from only a video feed. The approach for solving this goal was achieved through the use of computer vision algorithms on the captured video feed of the pianist, by placing a stationary camera at an angle from the side of the piano during their performance. This was a challenging task as at the beginning of the project there was only one piece of published literature available on the problem, due to most pub-

licly available research concentrating on audio signal processing for piano transcription. A number of components were developed to achieve the scope of this application. These components included location of the keyboard in a scene, identification of the individual keys and the key values on the piano, a digital reference to the beat of the performance, identifying the key change events in real-time, and transforming this into a digital representation that could be rendered into music notation (commonly known as sheet music, or a score).

The lack of research meant no publicly available third party test data-set existed. Hence the test data-set used had to be recorded and developed from scratch. The testing portion of the application has been achieved through the creation of an interface. This enabled a more efficient method of obtaining the ground truth from the videos, rather than visually writing out each key change event by hand. To get an extensive evaluation of how well the algorithms perform, this dissertation used the grading system of the Associated Board of the Royal Schools of Music (ABRSM). The ABRSM grading system provides a structured framework for progression from beginner to advanced pieces. Using this system, I hypothesised an algorithm which performs well in grade 1 pieces, and steadily decreases as it moves up the grades to more advanced pieces. However, no clear relationship was identified of the complexity of the piece to the performance of the application. But, the performance of the system worked best on an acoustic piano, and when the rhythm of the piece was fast. The average precision and recall was 78.72% and 93.57% respectively, with two outliers dramatically affecting precision.



---

## Acknowledgments

---

Firstly, I would like to thank my supervisor Kenneth Dawson-Howe for all his help and guidance throughout the year. I would also like to thank my family who also deserve significant thanks for their continued support and encouragement throughout the year.

---

---

# Piano Music Transcription Based on Computer Vision

Robert McCaffrey, Master of Computer Science

University of Dublin, Trinity College, 2017

Supervisor: Dr. Kenneth Dawson-Howe

Current research in the area of automated transcription technology focuses primarily on audio signal processing. The precision of audio signal processing is significantly below that of a human expert due to the presence of polyphonic tones (several sounds occurring simultaneously), and the difficulties of successfully parsing such signals given current algorithms. The aim of this paper is to investigate the application of computer vision to piano transcription. Transcription of music is here defined as creating a piece of music notation, in this case by analysing a video feed, so as to extract the musical parameters of the sounds that occur within the performance.<sup>1</sup> The video feed to be analysed shall be captured through a stationary mounted device that is angled at the side of the piano, while a pianist performs on the piano below. The transcription shall be accomplished purely through computer vision techniques, which shall examine key changes at each piano key in real-time. The results of the computer vision application create an industry standard digital representation of an instrument's performance known as a MIDI (Musical Instrument Digital Interface) file. The MIDI file can then be rendered to written music notation known as sheet music, through freely available external software. The performance of the system worked best on an acoustic piano, and when the rhythm of the piece was fast. The average precision and recall were

78.72% and 93.57% respectively, with two outliers dramatically affecting precision.

---

## Contents

---

<b>Declaration of Authorship</b>	<b>i</b>
<b>Summary</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Goals . . . . .	2
1.3 Dissertation Overview . . . . .	4
<b>2 Literature Review of Piano Transcription</b>	<b>8</b>
2.1 History of the Piano . . . . .	9
2.2 Keyboard Technique . . . . .	9
2.3 The Basic of Rhythm and Tempo . . . . .	10
2.3.1 Time Values . . . . .	10
2.3.2 Time Signatures . . . . .	10
2.3.3 Tempo . . . . .	11

2.3.4	Rhythm . . . . .	11
2.4	Pitch . . . . .	11
2.5	Octave . . . . .	12
2.6	Staccato . . . . .	13
2.7	Fermata . . . . .	13
2.8	Musical Instrument Digital Interface . . . . .	14
2.8.1	General . . . . .	14
2.8.2	Generating Music Notation . . . . .	14
2.9	Current Approaches . . . . .	16
2.9.1	Audio Signal Processing . . . . .	16
2.9.2	Computer Vision . . . . .	19
<b>3</b>	<b>Overview of the Approach</b>	<b>21</b>
3.1	Approach . . . . .	21
3.2	Method Overview . . . . .	23
<b>4</b>	<b>Location of the Keyboard and Keys</b>	<b>27</b>
4.1	Finding the Keyboard . . . . .	27
4.1.1	Algorithm for Finding the Keyboard . . . . .	29
4.2	Finding the Keys . . . . .	36
4.2.1	Locating the Black Keys. . . . .	36
4.2.2	Locating the White Keys. . . . .	38
4.2.3	Associating the Keys to their Respective MIDI Values. . . . .	39
4.3	Reference Beat of the Music . . . . .	42
4.4	Conclusion . . . . .	43
<b>5</b>	<b>Converting Video Feed to Sheet Music</b>	<b>44</b>
5.1	Median Background Model . . . . .	45
5.2	Create a Mask of the Hands . . . . .	47
5.3	Finding the Key Changes . . . . .	51
5.4	Convert the Key Changes to MIDI . . . . .	57
5.4.1	Method . . . . .	57
5.5	Convert the MIDI to Sheet Music . . . . .	60
5.6	Conclusion . . . . .	60
<b>6</b>	<b>Evaluation and Results of the System</b>	<b>62</b>
6.1	System Evaluation . . . . .	62

6.1.1	Apparent Evaluation . . . . .	63
6.1.2	Intended Evaluation . . . . .	63
6.1.3	Proposed Evaluation . . . . .	64
6.2	System Results . . . . .	66
6.2.1	Precision and Recall . . . . .	68
6.2.2	Recall . . . . .	73
6.2.3	Success of the Project . . . . .	75
6.2.4	Limitations in the Proposed Framework . . . . .	75
<b>7</b>	<b>Conclusion and Future Work</b>	<b>77</b>
7.1	Conclusion . . . . .	77
7.2	Time Spent . . . . .	78
7.3	Future work . . . . .	78
7.3.1	Investigate Different Background Model . . . . .	78
7.3.2	Refine Classification with Audio Processing for a Greater Precision . .	79
7.3.3	Neural Networks to Predict Key Changes . . . . .	79
7.3.4	Augmented Reality MIDI-to-Piano Learning Assistant . . . . .	80
7.4	An Improved Skin Detection Algorithm . . . . .	81
	<b>Bibliography</b>	<b>82</b>
	<b>Appendices</b>	<b>84</b>
<b>A</b>	<b>Metric for similarity in music</b>	<b>1</b>

---

## List of Tables

---

6.1	Performance Metrics . . . . .	68
-----	-------------------------------	----

---

## List of Figures

---

1.1	Capturing the video through a webcam mounted on a stand using a computer device . . . . .	3
1.2	Blue box illustrating the four sides of the piano found using the "find keyboard" algorithm. . . . .	4
1.3	The geometric transformation of the keyboard using the perspective transformation matrix found in the "find keyboard" algorithm . . . . .	5
1.4	Purple 'blobs' illustrate the result of finding the black keys using the feature detection algorithm. . . . .	5
1.5	Result of inferring the line (in green) that segregates two white notes. Only the top of the white key is required to be explained in Section 4.2 . . . . .	5
1.6	A visual representation of the hands been found labelled in red. . . . .	6
1.7	A visual representation of the key changes being found labelled in blue. . . .	6
2.1	Location of the keyboard on a piano. . . . .	8
2.2	Modern piano keyboard with key values . . . . .	9
2.3	Musical notation of the marching feet in sheet music with explanation of time signatures . . . . .	11
2.4	MIDI number representation of notes . . . . .	12
2.5	Illustration of the distance of an octave on the piano. . . . .	12
2.6	Symbols of a staccato and fermata on sheet music. . . . .	13
2.7	MIDI number representation of notes . . . . .	14
2.8	Modern piano keyboard with associated MIDI values . . . . .	16
2.9	Signal created with a period of 13 and 19, with 25% noise. <sup>2</sup> . . . . .	18



2.10	Output of M-Best algorithm using $M=10$ . <sup>2</sup>	18
3.1	Capturing the video through a webcam mounted on a stand using a computer device	22
3.2	Architecture for the initial setup required before my application can begin converting video feed to sheet music	25
3.3	Architecture for converting the piano video feed to sheet music.	26
4.1	Perspective transformation model. <sup>3</sup>	28
4.2	The corners top left (A), top right (B), bottom right (C), and bottom left (D) to be found by the algorithm as the four observations.	28
4.3	Result of binary edge image from Canny Edge Detector using threshold from Otsu's thresholding algorithm.	30
4.4	Line responses from the Hough line transform. Each line has been labelled in a different colour for visualisation purposes.	30
4.5	Peak line responses from the Hough line transform. Each line has been labelled in a different colour for visualisation purposes.	31
4.6	Vector cross product for a line 'a' and 'b'.	32
4.7	Pair of lines matched based on their slope, with minimum distance enforced to remove repeated responses. Lines labelled in the same colour for visualisation purposes.	33
4.8	Using the four observations $p$ , to determine coefficients $p$ , to then create the perspective transformation matrix. <sup>3</sup>	35
4.9	Modern piano keyboard with key values.	36
4.10	Black keys in purple 'blobs' depicting the result of algorithm 1.	37
4.11	Modern piano keyboard with key values.	38
4.12	Result of using midpoint to infer the line (labelled in green) that segregates two white keys.	38
4.13	Modern piano keyboard with its MIDI values.	39
4.14	Unique pattern of the C key on the piano. This pattern is used to find C on a keyboard.	40
4.15	Unique pattern to find C applied in blue/green: the location of middle C is in green found using equation 4.2.	41
5.1	The mask of the hand found at each stage of the algorithm overlaid on the frame it is analysing for illustration.	48
5.2	Zone of the keyboard where the key changes are classified.	51

5.3	Illustration of the background subtraction based method applied on the background and foreground image to create the two difference images. . . . .	53
5.4	Anatomy of a key within a piano used for illustrating the action of a key when it is struck by a finger. <sup>4</sup> . . . . .	54
5.5	Octaves are played normally at the edge of the keys which leaves only a small region of skin that can be found . . . . .	55
5.6	The box in green illustrates the zone of the keyboard that is examined for key changes by using the hand's proximity to exclude impossible note change. This zone is found by moving one key(blue) from the hands(red) on each side	55
5.7	Simplified structure of the MIDI file that is created using header information and MIDI events . . . . .	57
5.8	Relationship from musical theory to duration to MIDI ticks . . . . .	58
5.9	Transferring of the MIDI file to the Musescore application . . . . .	60
6.1	Architecture and brief illustration of the interface for creating ground truth from the videos captured. . . . .	65
6.2	Visual explanation of precision and recall <sup>5</sup> . . . . .	67
6.3	Performance metric of precision graphed with recall. Results from the six dataset videos with the sheet music taken from the first six grades of ABRSM	68
6.4	Key value and amount of occurrences off the false positive key changes within the grade 1 video . . . . .	69
6.5	Position of the camera relative to the keyboard for the recording of the Grade 1 piece. . . . .	70
6.6	Difference of the neighbouring exposed note for a C and F for an acoustic piano (wood keys) and digital piano (plastic). . . . .	71
6.7	Purple 'blobs' illustrate the result of finding the black keys using the feature detection algorithm. Notice the distance between black notes decreases as you get farther from the camera (Notice this camera was positioned on the left) .	73
6.8	Result of inferring the line (in green) that segregates two white notes. Notice the distance between black notes decreases as you get farther from the camera (Notice this camera was positioned on the left of the piano) . . . . .	73
6.9	Omission cases 1 on the left and case 2 on the right . . . . .	74
A.1	Input and output of the edit distance algorithm . . . . .	3

### 1.1 Motivation

A trained musician or other music professional tasked with creating sheet music from a recording transcribes the notes that make up a piece of music to music notation using either their sense of relative or absolute pitch. With advanced musical training, an individual can transcribe a piece of music note-for-note purely by ear; some after they have heard the music only a few times. Mozart, who was known to be a child prodigy, demonstrated this ability by recreating note for note Miserere after hearing it only once (this was secret sheet music written by Allegri for Pope Urban VIII for exclusive Tenebrae service in the Sistine Chapel). Musicians study for years to truly master this ability. What if technology could share this ability with an amateur musician?

Current research in automating this process through audio signal processing has not yet reached the necessary precision. The limitations of this process are due to the presence of polyphony, noise, variations of pitch, and variations of note duration. Consequently, to record a performance into a computer, currently the most effective technology is to use an electronic instrument with MIDI (Musical Instrument Digital Interface) capability, or an acoustic instrument fitted with infra-red, or other sensors; both have their limitations and weaknesses. Electronic instruments lack the rich timbre and depth of touch of an acoustic instrument, and are therefore not preferred by experienced musicians (other than for convenience or for novel effects and timbres). Infra-red sensors that monitor individual strings at a time are

expensive and cumbersome to install. However, even with the availability of both of these technologies they still require some computer device to look after the processing. Hence, a proposed application that uses only the capabilities of a modern portable device (laptop, mobile, tablet) gives us an easily accessible solution.

This dissertation explores a solution for automating piano transcription through computer vision in OpenCV. Transcription of music is here defined as the process of analysing a video feed so as to write down the musical parameters of the sounds that occur in it.<sup>1</sup> The composition performed on a piano shall be transcribed to sheet music using computer vision. The video will be captured by mounting a device with video capability at an angle overhead, while a pianist performs on the piano below. Computer vision estimation based models are used to create an automated piano transcription software. Through computer vision, the objective of this dissertation is to explore the process of transcription via a different methodology, thus avoiding the difficulties associated with audio signal processing, where most current research focuses on. The ambition is to extend this application to other instruments, giving a variety of suitable musicians access to sheet music transcription with their portable device. To get an extensive evaluation of how well the algorithms perform this paper will use the grading system in the Associated Board of the Royal Schools of Music (ABRSM). The ABRSM grading system provides a structured framework for progression from beginner to advanced pieces. Using this system, I expect an algorithm which performs well in grade 1 pieces, and steadily decreases as it moves up the grades to more advanced pieces.

During research, there is a need to review the ontology of the field. Through understanding the various domains relationships to one another, I can better answer the questions I seek and better question the answers I obtain. The multimedia ontology provides a knowledge infrastructure for the computer vision domain, while the musical ontology provides a knowledge infrastructure for music theory. The domain specific parts support content layer analysis of the video material. In the video domain, the signal layer is necessary for the representation of data, before using the music theory domain to model the conversion algorithms to MIDI and sheet music. These conversion algorithms will be explored in the method of this dissertation.

## 1.2 Goals

The goal of this dissertation was to design and implement a prototype automated piano transcription software, to correctly transcribe a piece of music played on the piano captured only through video. A composition performed on a piano is thus transcribed to sheet music

using computer vision. The video is captured such as in Figure 1.1 by mounting a device with video capability at an angle from the side of the piano; a pianist then performs on the piano below. Computer vision based models are used to create an automated piano transcription software. Through computer vision the objective is to locate the piano's keyboard in the scene, identify the keys that are pressed along with the duration they are held, and render a piece of sheet music in PDF format. The performer will be able to use this sheet music to recount what they have played and distribute to other performers. However, while this is the primary goal, there are endless possibilities for music education and professional composition for such a technology, which I will shall discuss at the end of this dissertation in my future works.



Figure 1.1: Capturing the video through a webcam mounted on a stand using a computer device

### 1.3 Dissertation Overview

In Chapter 2, I outline the properties of the piano such as a description of a keyboard in Section 2.1, techniques that have evolved to play the instrument in Section 2.2, and how the action of striking keys can produce a collection of tones that make the piano such a diverse instrument in Section 2.4. Through understanding the ontology of the piano, I have modelled the computer vision algorithms. For the reader to understand how these ontologies were used for modelling my algorithm, Chapter 2 gives a broad background on the piano in Section 2.1, music theory in Section 2.3 — 2.4, and a standard for representing musical instruments digitally within a computer in Section 2.8. To understand the complexity of this research in audio signal processing, Section 2.9.1 gives a brief overview of how processes within this domain are modelled and why multiple concurrent tones negatively affect the algorithms within this field. Chapter 2 also outlines what research has been carried out in the video domain, and the faults I have discovered and intend to improve upon within Section 2.9.2.

Given the goal outlined above, a webcam was mounted at an angle from the side of the piano, to capture the video feed of a pianist while they performed on a piano. For the computer vision to work, the application required the location of the keyboard in the scene so that it can identify what keys in each frame the pianist has struck. Therefore, the keyboard of the piano had to be located, this task was achieved using algorithms described in Chapter 4.1. The result is an estimation of where the keyboard lies in the scene, Figure 1.2 outlines the results of the algorithms in Chapter 4.1; performing the estimation on a captured video feed.



Figure 1.2: Blue box illustrating the four sides of the piano found using the "find keyboard" algorithm.



Figure 1.3: The geometric transformation of the keyboard using the perspective transformation matrix found in the "find keyboard" algorithm

The estimation is used to transform the view of the keyboard into a perspective where the keyboard is returned to its original rectangular shape with the keys vertically aligned; Figure 1.3 illustrates the result of this operation known as geometric transformation outlined in Chapter 4.1. With the keyboard in this perspective, it was analysed to retrieve information on what sound each key on the piano in the scene represented, so that it could be ported into a standard that a computer could understand. Section 4.2 describes in full the algorithms that were used to perform this task. The result of these algorithms extracting the black keys as features within the image is illustrated as purple blobs within Figure 1.4. The white keys inferred from these features are illustrated in Figure 6.8 as green lines.



Figure 1.4: Purple 'blobs' illustrate the result of finding the black keys using the feature detection algorithm.

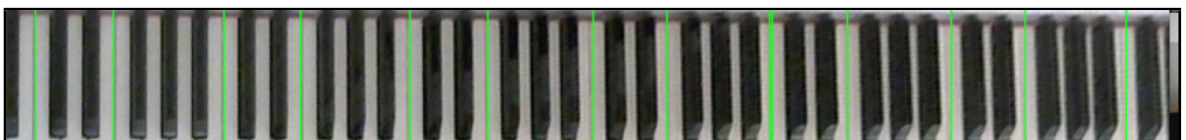


Figure 1.5: Result of inferring the line (in green) that segregates two white notes. Only the top of the white key is required to be explained in Section 4.2

With the keyboard and the location and values of the keys found, the next stage is preparing for identifying key changes. A potential solution could have been using pose estimation of the hands and fingers to identify key changes. However, pose estimation is difficult unless



multiple camera angles or an infra-red camera such as Microsoft Kinect is applied. Moreover, I wanted a readily available application that can be used on any conventional video camera. In response to this, I decided to ignore the hands completely and focus only on the key movements on the keyboard. Consequentially, the transcription software completely ignores the section of the keys where the pianist's hands are present, Section 5.2 outlines the algorithms that were used to find the hands within the scene. Figure 1.6 show the hands illustrated in red to signify the result of the skin detection algorithm used to find the hands.



Figure 1.6: A visual representation of the hands been found labelled in red.

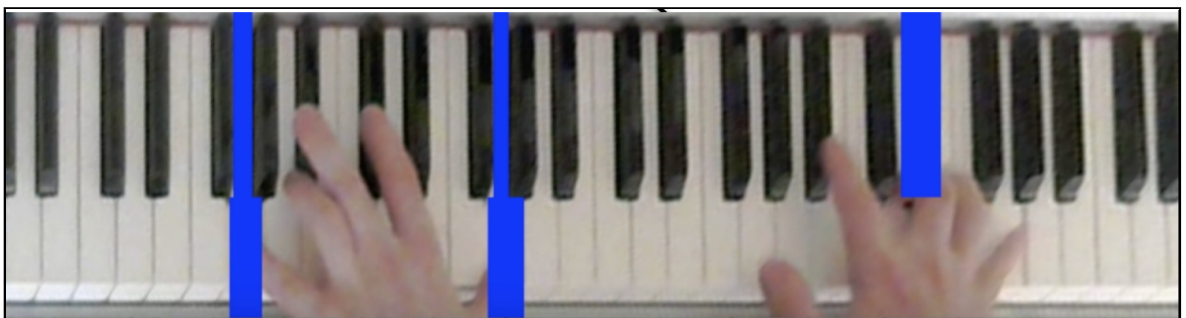


Figure 1.7: A visual representation of the key changes being found labelled in blue.

Given that every key's location has been identified and the hands have been removed from the frame, key changes can now be detected using the algorithm outlined in Section 5.3. Figure 1.7 shows an example of the key changes on a frame: the keys coloured in blue illustrate the result of the key detection algorithm on this particular frame.

The results of the key detection algorithm are passed to an interface presented in Section 5.4 which communicates with an external MIDI library. The interface handles the creation of a MIDI file that stores a digital representation of the pianist's performance. This MIDI file is then used to create sheet music in Section 5.5, at the end of the execution of the program.



---

After the application is executed it is worth noting that the MIDI file created throughout the run-time is a playable recording of the pianist's performance, found purely through computer vision.

Evaluations and tests were performed on the results of this transcription system. The feasibility and the functionality of the proposed evaluation frameworks are analysed in Chapter 6.1. The proposed evaluation framework offers two performance metrics: precision and recall. These metrics are applied and analysed in Chapter 6.2 where an in-depth review of the advantages and the limitations of the proposed framework (as well as of the successes of the project) is presented.

## CHAPTER 2

---

### Literature Review of Piano Transcription

---



Figure 2.1: Location of the keyboard on a piano.

## 2.1 History of the Piano

"The story of the piano begins in Padua, Italy in 1709, in the shop of a harpsichord maker named Bartolomeo di Francesco Cristofori (1655-1731). Many other stringed keyboard instruments preceded the piano and led to the development of the instrument as we know it today."<sup>6</sup> It gradually surpassed the popularity of the harpsichord due to its ability to vary dynamics, and also the clavichord, given its ability to produce more sound pressure level (particularly as the instrument developed\*). The piano is played via a keyboard that contains rows of black and white keys, with each key representing a different pitch (fundamental frequency). The action of hitting a key invokes a hammer within the piano to then strike the strings and produce a note (pitch) of varying volume (depending on the velocity exerted by the pianist). The piano comes in many different shapes and sizes, however, the keyboard and pattern that exists is consistent throughout models, with only the number of keys varying. The modern piano's keyboard contains 88 keys of which 52 are white keys and 36 are black keys. Figure 2.2 depicts a modern piano's keyboard and Figure 2.1 the location of the keyboard on the piano.

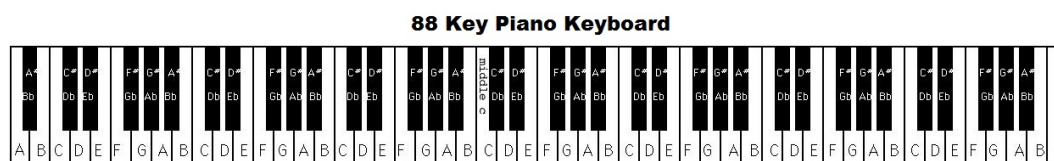


Figure 2.2: Modern piano keyboard with key values

## 2.2 Keyboard Technique




From Bach to Beethoven, throughout the eighteenth century while exploration were emerging across the East and Europe, so too was the technique of the keyboard. By the end of that century a true pianist style had emerged.<sup>7</sup> It can be said that the foundations of modern piano technique had been laid down by the end of the eighteenth and early nineteenth century.<sup>7</sup> The principles techniques for playing the piano such as posture, finger-action, hand and arm movement were used to create empirical methods for modelling the semantics of the algorithms in this dissertation. For example, with the knowledge of the playing technique required for the works of the great composer-pianists such as Bach, Beethoven, Chopin, etc., I formed empirical methods for identifying key changes in Section 5.3.

\*Note that the modern concert grand can compete with an entire orchestra in the hands of an accomplished player, a cursory listen to a piano concerto by Rachmaninoff would suffice to illustrate.

## 2.3 The Basic of Rhythm and Tempo

Throughout this dissertation there will be references to music theory and how through computer vision I have produced rules based on music theory to achieve an automated transcription software. The following analogies and figures are either my own, or are taken from *The AB Guide to Music Theory* by Eric Taylor. The book is intended to help beginners learn to read, write, and understand written music. The book is published by the Associated Board of the Royal Schools of Music (ABRSM); an examinations board and registered charity based in London. It is an essential book for providing an introduction to the basic elements in harmony and musical structure for any student carrying out the ABRSM examinations and is often used in undergraduated music courses. Therefore, it was a suitable text for pinching analogies to explain the basic musical concepts required for a non-musically trained reader.

### 2.3.1 Time Values

When soldiers march along to a band, their footsteps are absolutely regular and even.<sup>7</sup> The goal is to keep the soldiers marching together in the strictest possible synchronicity, and as an analogy the footstep provides a simple explanation of time values through a single beat. A single beat in music theory is determined by the time signature, in the case of 2/4 as a crotchet, that is two crotchets (or quarter notes) to a bar. The time signature of LEFT RIGHT, would thus be 2/4, that is two distinct beats before the pattern repeats. The simple beat produced for a company of soldiers, LEFT RIGHT LEFT RIGHT, where a beat is struck on every footstep can be denoted in musical notation as . A double beat is referenced as a minim, where the beat is struck every second footstep, and would be denoted in musical notation as . Finally, a half beat is referenced as a quaver where two strikes of the drum equates to a footstep .

### 2.3.2 Time Signatures

The analogy used above of the soldier marching feet is an example of a succession of equal beats.<sup>7</sup> Although they all carry the same beat. Marching feet, for example naturally make groups of two beats - LEFT right LEFT right. Which in musical notation would separate the groups of two beats by vertical lines known as bar lines.

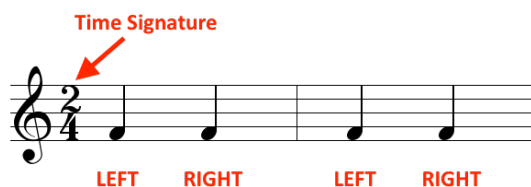


Figure 2.3: Musical notation of the marching feet in sheet music with explanation of time signatures

Each group is called a bar, at the beginning of a piece of music there is a symbol known as a time signature. In the example above the time signature would be  $\frac{2}{4}$ , which means that between each bar there must be a quantity of notes (or rests) that equal two crotchets, and that the strong beat is the first crotchet of the bar. Fig. 2.8 represents 2 bars with a time signature of  $\frac{2}{4}$  where each bar line separates a total note value of two crotchets.

### 2.3.3 Tempo

Time signatures indicate how many beats are in a bar, while tempo denotes the speed at which the beats should be played. A funeral march, for instance, would have a slow tempo. A device used to create or measure tempo is known as a metronome, it works by making a steady ticking sound that can be adjusted to any speed one wishes. The metronome can be set at any given number of beats per minute. Therefore, it is often annotated to b.p.m. For instance, the pieces in my test plan will be played at 80 b.p.m, which indicates that the speed of the music will equate to 80 beats per minute.

### 2.3.4 Rhythm

'Rhythm' refers to the way in which sounds of varying length and accentuation are grouped into patterns.<sup>7</sup> The most simplistic way to understand it is to imagine your favourite song, you could not play it by tapping on a table but you could tap out the rhythm. While you can change the tempo by tapping out the beat faster or slower, the rhythm will stay the same.

## 2.4 Pitch

Striking any key on the piano produces a particular fixed fundamental frequency. The frequency produced is referred to as the pitch I.e. 440Hz is referred to as A, or, specifically, A above middle C; in scientific notation (Helmholtz pitch notation) it would be signified by a'..

Each key on the piano produces its own pitch. Fig. 2.4 depicts the key values on the piano and how it is represented on the staff for sheet music.

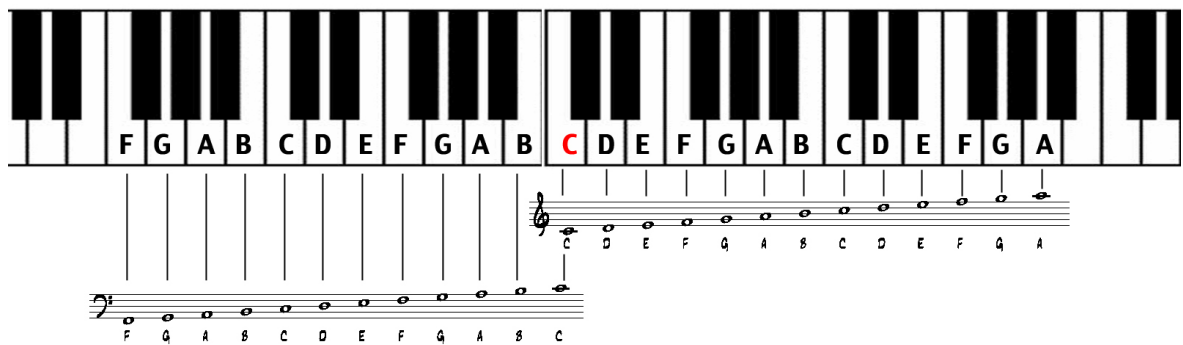


Figure 2.4: MIDI number representation of notes

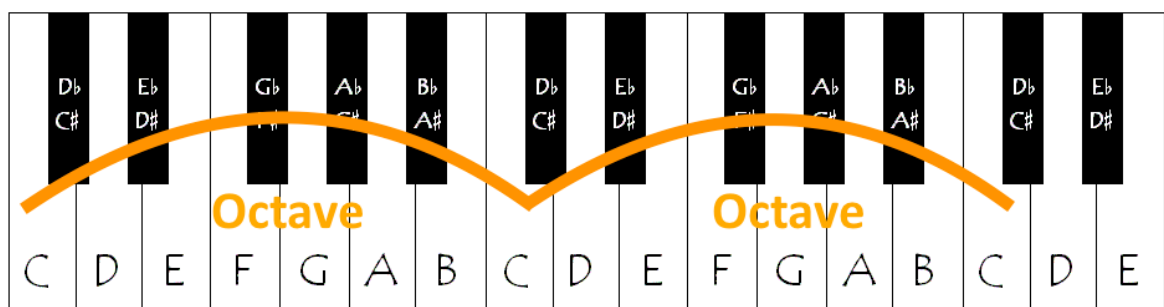


Figure 2.5: Illustration of the distance of an octave on the piano.

## 2.5 Octave

The distance between any note and the next one of the same key value is known as an octave, technically this would be when the fundamental frequency of the note is multiplied by two (e.g. given  $A=220\text{Hz}$ ,  $A$  one octave above would be equal to  $440\text{Hz}$ ). For example, in Fig. 2.5 the distance between the  $C$  farthest to the left to the  $C$  in the middle is one octave and

the distance between the C farthest to left and the C farthest to right would be a distance of two octaves.



Term	Symbol	Meaning
<i>staccato</i>		Play the note with a quick key strike, not holding the note to blend with the next
<i>fermata</i>		Hold the note longer than its full length

Figure 2.6: Symbols of a staccato and fermata on sheet music.

## 2.6 Staccato

A staccato symbol is a small dot placed beneath (or above a note, depending on the direction of the stem), that indicates that the note should be shortened in duration, and not connected to the subsequent note. The degree of shorting is contextual, and is very much dependant on the tempo of the music. Staccato would be the opposite of legato; legato indicates that a note should be held for its full metric duration and joined to the subsequent note.

## 2.7 Fermata

A fermata symbol indicates that one should pause upon a particular note (or series of simultaneous notes, as in a chord) for an unspecific period of time. The amount of time one adds to the written duration is contextual, and will depend on many factors, including tempo, the written duration, but also the position within the piece of music (i.e. a fermata at the end of a piece would be held for longer than one that occurs during a piece, even if written over the exact same note/chord and duration).

## 2.8 Musical Instrument Digital Interface

### 2.8.1 General

Musical Instrument Digital Interface (MIDI) was created as a communication standard for electronic music equipment. MIDI files are not a music file, but rather a file that describes music by a series of events - essentially a kind of virtual sheet music. MIDI files are made up of a *header* and a sequence of *events* that describe a song. There are many different types of events but the most common are *note-on* and *note-off* events. Note-on events describe when a key has been pressed, and note-off events describe when it has been released. Using these two events one can determine what notes were played and their duration. Time is measured using *ticks*. Tick values are symbolic time units, such as durations in music notation. Each event has a tick value that indicates the number of ticks to wait from the last event in the track before performing the current event. Tempo and ticks-per-quarter-note information contained in the header allow the song to be played at the correct speed. Notes are represented as numbers in the range 0 to 127 (as shown in Figure 2.7).

Note Names / Numbers (C3=60)												
Octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-2	0	1	2	3	4	5	6	7	8	9	10	11
-1	12	13	14	15	16	17	18	19	20	21	22	23
0	24	25	26	27	28	29	30	31	32	33	34	35
1	36	37	38	39	40	41	42	43	44	45	46	47
2	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71
4	72	73	74	75	76	77	78	79	80	81	82	83
5	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107
7	108	109	110	111	112	113	114	115	116	117	118	119
8	120	121	122	123	124	125	126	127	—	—	—	—

Figure 2.7: MIDI number representation of notes

### 2.8.2 Generating Music Notation

A MIDI file can be converted to music notation using postscript based software. However, integrating MIDI and musical notation is not as simple as it sounds. "The core of the problem is that the correlation between music notation and how a piece sounds is very loose and open to interpretation",<sup>8</sup> while the data sequence in a MIDI file is concrete and how it sounds is



not open to interpretation. There are many problems that arise when a computer tries to re-enact the different aspects of a performance. Take following two examples from a book "MIDI for the Professional" by Paul D. Lehrman.

(a)"A computer cannot readily differentiate between a staccato quarter note and an ordinary eighth note."<sup>8</sup>

(b)"Or consider the case of a fermata over a whole note. A human musician might interpret this by holding the whole note for one extra beat". If such a performance were played into a computer, it would simply be interpreted as a five-beat note. Thus, every subsequent measure would be off by one beat". Therefore, converting a MIDI file to sheet music involves second guessing what the performer intended.<sup>8</sup>

Furthermore, for a MIDI file to work it requires reference to the tempo the performer is playing. Most applications use a metronome (or click track) so that they can accurately transcribe the performance to sheet music. However, to play metronomically is mechanical and thereby impossible for a human without the aid of a metronome or click track. Only a concert pianist could even get close to playing at a mechanically perfect tempo, and even then there would always be a minute variation that would accumulate from bar to bar. These discrepancies cause a great challenge for manufacturers to automate sheet music transcription. Current applications such as Finale, Sibelius and Logic X, some of the biggest players in industry as of 2017, both leave it to the user to correct these discrepancies through visually adjusting the MIDI file for the desired sheet music representation. Logic X, however, has an interesting feature known as Beat Mapping, this allows the user to manually add bar lines and beat markers to a MIDI recording that was done without a click track. Logic then creates a tempo map to preserve the flow of the performance. Using this feature one can overcome the problem mentioned above, however, it is very laborious as one must at the very least add most bar lines by hand.

The limits of interpreting a performance perfectly into music notation and domain specific problems have governed how I designed my project while also impairing features and changing my aspiration for testing from a metric of similarity of music to computer vision performance metrics.

Keyboard	Note name	MIDI number
	C8	108
	B7	107
	A7	106
	G7	105
	F7	104
	E7	103
	D7	102
	C7	101
	B6	100
	A6	99
	G6	98
	F6	97
	E6	96
	D6	95
	C6	94
	B5	93
	A5	92
	G5	91
	F5	90
	E5	89
	D5	88
	C5	87
	B4	86
	A4	85
	G4	84
	F4	83
	E4	82
	D4	81
	C4	80
	B3	79
	A3	78
	G3	77
	F3	76
	E3	75
	D3	74
	C3	73
	B2	72
	A2	71
	G2	70
	F2	69
	E2	68
	D2	67
	C2	66
	B1	65
	A1	64
	G1	63
	F1	62
	E1	61
	D1	60
	C1	59
	B0	58
	A0	57
		56
		55
		54
		53
		52
		51
		50
		49
		48
		47
		46
		45
		44
		43
		42
		41
		40
		39
		38
		37
		36
		35
		34
		33
		32
		31
		30
		29
		28
		27
		26
		25
		24
		23
		22
		21

Figure 2.8: Modern piano keyboard with associated MIDI values

## 2.9 Current Approaches

### 2.9.1 Audio Signal Processing

Audio to score transcription involves utilising audio signal processing to convert an audio recording of a performance on the piano to sheet music. In monophonic signals, where at most one note is sounding at a time, audio signal processing for estimating the pitches and durations occurred by an acoustic piano can be considered solved. "Automatic transcription of polyphonic music has been the subject of increasing research interest during the last 10 years".<sup>1</sup> In polyphonic signals, several sounds occur simultaneously within the signal, this results in a signal that needs to be separated before the estimation can even begin. Sound separation is extremely complex, especially with signals containing different pitches at similar levels, it is difficult for an algorithm to determine the exact point at which one pitch begins and another pitch ends. However, it can be accomplished to a degree through widely used techniques in audio processing known as multiple  $F_0$  estimations and periodicity transforms.<sup>9</sup>

In audio processing, the fundamental frequency is known as the lowest wavelength and the lowest frequency in a periodic waveform.<sup>10</sup> Therefore, the fundamental frequency is usually abbreviated as  $F_0$  counting from 0.  $F_0$  is the first harmonic of the waveform and is defined as the inverse of the period. In music, the fundamental frequency is defined as the pitch which in our case refers to the sound created by a note from the piano. Hence  $F_0$  estimation is the process of identifying the pitch from an audio waveform. As the piano is a polyphonic instrument, it deals with multiple  $F_0$  estimations, the process of concurrently identifying multiple pitches from the waveform. Multiple  $F_0$  estimation identifies the pitch produced, however, another process must also decompose the rhythm from the waveform in a process known as rhythm parsing. Multiple  $F_0$  analysis and rhythm parsing will typically occur separately; based on what we know about the modularity of music processing in the human brain, this is justified not only in technical artefacts.<sup>1</sup>

**1) Multiple  $F_0$  analysis:** Most multiple  $F_0$  analysis algorithms operate with only musical (harmonic complex) tones. Tones produced by a musical instrument consist of the fundamental and a series of overtones; overtones are frequencies which are higher and often, but not always, directly related to the fundamental frequency. A sound is harmonic when the overtones are periodic, i.e.,  $(F\ 2F\ 3F)$ . For example, when we hear the note A4, it is composed of harmonics with frequencies 440, 880, 1320, etc., continuing up through the harmonic series. Therefore, if the waveform displays harmonicity, i.e. has harmonic overtones, it can be said to be periodically oscillating. The human ear uses these harmonics overtones to differentiate these sounds. The piano is an example of a periodically oscillating instrument. Therefore, we can use multiple  $F_0$  estimation algorithms to decompose the waveform to its relevant pitches. In 2004, Klapuri describes in "Automatic Music Transcription as We Know it Today" the following techniques for decomposing monotonic  $F_0$  such as spectral location, spectral interval and periodicity of the time-domain amplitude envelope. However, the "aim of multiple- $F_0$  estimation is to find the  $F_0$ s of all the component sounds in a mixture of signals"<sup>1</sup> which Klapuri notes increases the complexity significantly and is closely related to sound separation. Therefore, the problem with music transcription using an audio signal is it is inherently difficult. Even with this difficulty, multiple  $F_0$  estimation only finds the note value but does not find the duration it is played for. Therefore, the process becomes even more complicated by the need to incorporate another technique known Rhythm parsing with Multiple  $F_0$  estimation to find the notes duration.

**2) Rhythm Parsing:** There are many meter estimation systems available for audio input. The most widely used is periodicity transforms by William A. Sethares and Tom Staley.<sup>9</sup>

Periodicity transforms decompose a data sequence into a sum of simple periodic sequences by projecting onto a set of periodic sub-spaces, leaving residuals whose periodicities have been removed.<sup>2</sup> The decomposition is done directly in periodic sequences and not frequency or scale; therefore the representation is linear in period. Fig. 2.9 shows two periodic sequences added together, one with a period of 13 and the other with a period of 19; there is approximately 25 percent of noise added.

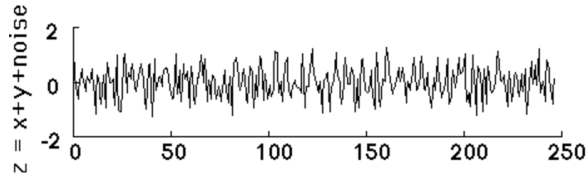


Figure 2.9: Signal created with a period of 13 and 19, with 25% noise.<sup>2</sup>

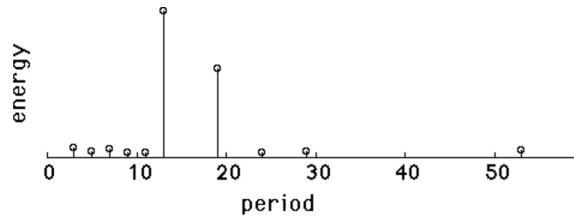


Figure 2.10: Output of M-Best algorithm using  $M=10$ .<sup>2</sup>

Applying the periodicity transform called the 'M-Best algorithm' one can search for  $M$  largest periodicities. In this example  $M=10$ , therefore, the 10 largest periodicities will be found as in the Fig. 2.10. As you can see the two periods, 13 and 19, are clearly visible with the other small periods reflecting minor accidental patterns that occur from noise.

The mechanism by which these two process work together can be imagined in a time-frequency plane where time flows from left to right on a vertical axis with different  $F_0$  arranged in an ascending order. On top of this plane, a multiple- $F_0$  estimator produces horizontal lines which indicate the probabilities of different notes to be active as a function of time. Rhythm parsing, in turn, produces a framework of vertical 'grid lines' which can be used to segment the note activation curves into discrete note events and to quantize their timing.<sup>1</sup>

In using this technique, there is still a need to explore the convergence properties of the algorithms, its robustness to noises, and to the interactions between periodicities; frequencies and sampling rates further expanding the complexity of the problem. Due to the presence of polyphony in music, with each concurrent note having harmonic complex tones, the precision

of recognition has not reached the accuracy required for real-world applications. However, due to shared problems in speech and audio signal processing, as technology advances and increasingly sophisticated tools are developed and researched by global multinationals for speech recognition software like Siri and Google Voice, the future possibilities for audio transcription software are not unforeseeable. "An important fact about music transcription using audio signal processing is that it is difficult. The problem is at best comparable to automatic speech recognition which has been studied for 50 years and is only now becoming practically applicable".<sup>1</sup> However, with current methods, my proposed solution through computer vision provides an opportunity for a more feasible solution to automating piano transcription using a readily available video recording device.

### 2.9.2 Computer Vision

In the field of computer vision, research into automated music transcription is in its preliminary stage. There is currently one paper available "claVision: Visual Automatic Piano Music Transcription" by Akbari and Cheng which claims a very high accuracy (over 95%) using conventional computer vision principles. The algorithms explored do not cover the requirements of having a reference to the beat that the performer is intending to play. Therefore, it seems that a reference to the beat of the music may be hard-coded in the solution as a default value. The aim of my solution is to incorporate the requirement of the transcription software to have this reference to the beat of the music and have it adjustable by the user, this is presented in Section 4.3.

In addition, I would like to explore a more extensive evaluation of my algorithms. The test plan within<sup>11</sup> only considers varying tempos on a simplistic piece: the nursery rhyme "Twinkle Twinkle Little Star". Upon evaluation of the piece used I observed the presence of no black keys in the piece. While the paper does discuss methods used to identify black key changes; the evaluation gives no indication into whether it, in fact, does take into account black key changes. Evaluation problems as such constantly arise in the field of computer vision due to the lack of external ground truth. Too often, the researchers are both the creator and the user of the new technology and do not use any external data, or known theories to show their technology conforms to some underlying model or requirement.<sup>12</sup> In recognising these errors, my evaluation will use a known theory of evaluation in piano studies for assessing the difficulty of a piece. I will use the grading system in the Associated Board of the Royal Schools of Music (ABRSM). Using this grading system, I can have a performer

---

play a piece randomly from each grade and then evaluate the precision of the solution using performance metrics. Given that the difficulty of the grades increase reasonably linearly: the complexity of the piece, its covering/range of the keyboard, the variety of finger work and hand movements etc., should provide a concrete linear progression for revealing the latent use of my solution across different standards of musicians.

### 3.1 Approach

To rid my research of ambiguity, opinion, and undelineated work, I have chosen to construct my scientific method on the four precepts of French philosopher Rene Descartes. In his work "Discourse on the Method", Descartes said "fewer rules are more strictly observed"<sup>13</sup> therefore I have chosen to adhere to his four precepts:

1. "Avoid haste and prejudice."
2. "Divide up each of the difficulties."
3. "Commence with objects that are the most simple."
4. "In all cases make enumerations so complete and reviews so general that I should be certain of having omitted nothing."

Following these precepts, initially, I doubted how plausible this research could be. To accept nothing as true, I ensured every output of the project was possible. Due to the presence of only one research paper available on the video processing of piano transcription: this was the most ambiguous output of the project. Therefore, it was necessary to investigate the

amount of information that could be found on a conventional camera with no depth perception. To get a quantitative result of the amount of observable note changes, an experiment using a video of a musician was analysed. The experiment sequentially compared each frame to its predecessor analysing the pixel changes that occurred during key presses and releases. The results were assuring enough to advocate further research using the proposed technique of computer vision. The video for these experiments was captured as shown in Fig. 3.1. The same procedure was be used in the final application.



Figure 3.1: Capturing the video through a webcam mounted on a stand using a computer device



## 3.2 Method Overview

During research I used advice from Descartes' second precept to divide up each task. I divided the work into tasks and subdivided the states in each task to ensure that the product is a defensible dissertation within the time limit. I divided my method into the following two tasks and states:

A. Location of the Keyboard and Keys:

- (1) Find the keyboard in the scene.
- (2) Classify the keys on the keyboard to their relevant key value.
- (3) Identify a reference to the beat of performance.

B. Converting the Video Feed to Sheet Music:

- (1) Identify and remove the hands from the scene.
- (2) Find the key changes as they occur in the video feed.
- (3) Convert the key changes to a MIDI file.
- (4) Convert the MIDI file to sheet music.

Before the transcription can begin, the system requires three attributes to be defined in what has been named the initial setup. Upon execution of the application, the video that is fed to the system will contain a pianist and a single piano within a relatively static background. There are no limitations within the algorithm that specify any non-permitted objects in the scene, unless they resemble a piano or they are blocking the camera's view of the piano in question. Once the initial setup has completed, the pianist will be notified by the application that they may begin performing (approximately 5 seconds). The first step of this setup is the location of the piano's keyboard within the scene. A keyboard is a rectangular shape when it is viewed face-on from above. But the required position and angle of the camera for my application results in a perspective where the keyboard looks like an irregular quadrilateral that is not rectangular. The aim of my first initial setup algorithm is to find the perspective matrix that transforms this non-rectangular object back to its original rectangular shape. Transforming it to its original shape allows simpler classification of key values and key changes, however, placing the camera in a face-on position from the beginning would negatively affect the precision of key changes for this system—this shall be discussed later.

This perspective matrix only needs to be found once, due to the specifications of my application the camera should not be moved after the initial setup has completed (hence every consecutive frame can make use of the same matrix). After the first perspective transformation has occurred, the resulting image will be used for finding the individual key locations and key values, i.e., where they are in the scene and what MIDI number they represent. Likewise, the algorithm used to find the key's location and value shall not change within the run-time of the application and therefore only needs to be found once. The last and final attribute required in the initial setup is a reference to the beat of the music that the pianist intends for their performance. As mentioned in 2.3.4, the rhythm of a piece of music is invariant to the tempo chosen for the performance. In theory, if two pianists play the exact same piece of music, the resulting sheet music should be equivalent, regardless of whether one pianist's performance was at a faster or slower tempo. This can be perceived by imagining your favourite song, think of the rhythm of that song and imagine yourself tapping it out on a table. Now, if you increase the tempo at which you tap out that song on the table, the rhythm of the song does not change, only the tempo at which you tapped it.




For my application to be able to interpret and transcribe the performance of a pianist correctly, it requires a reference to the tempo at which the pianist intends to play. A possible solution which most MIDI applications utilise is having a metronome (or click track) which a performer can configure and play along to during the performance. In these applications, the performer can hear when the beat is supposed to occur, and the application which is creating the sound of the beat can document when it occurs. However, the sound of the metronome can be off-putting for a musician during a performance. Therefore, my application allows the pianist to 'count themselves in' which alleviates the distraction of a ticking sound. 'Counting yourself in' indicates that the musician has to strike the middle C key at the same intervals as they would set a metronome. The application can then interpret the beat of the music to be used throughout the performance, by counting the interval of these strikes as a duration of frames. Hence, in the initialisation setup, after locating the keyboard and its keys, the application will instruct the pianist to count themselves in. The result of this calculation is what will be used in the transcription to represent the duration of a single beat. If we recall from Chapter 4.3 a single beat is represented as a , by having a reference to the number of frames that equal this note, all other note values can easily be inferred from this; e.g., twice the frame count would result in a transcription duration of , half the frame count would result in a transcription duration of , etc., etc.

Figure 3.2 outlines a visual representation of this initial setup for my application.

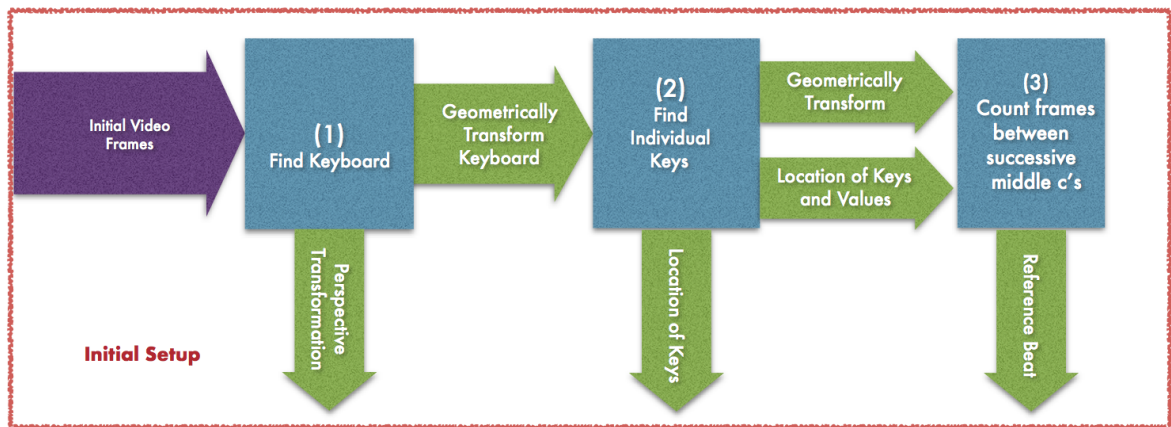


Figure 3.2: Architecture for the initial setup required before my application can begin converting video feed to sheet music

Upon completion of the initial setup, the transcription begins. Three attributes have been defined at this point: the perspective matrix, the location and value of the keys, and the reference beat of the performance. All of these attributes are vital for the transcription algorithms to work. Every frame that is fed into the application at this point is geometrically transformed using the perspective matrix. The resulting image of this geometric transformation is used in every component until a key change is found: the background model, removal of the hands, and identifying key changes.

The process by which I identify key changes is through the use of background subtraction. Background subtraction is the process of subtracting a foreground image from a background image, to create a foreground mask of a moving object in the scene. The moving object in my application is the key changes that are to be found. A model is required for this background image so that it can adjust to rapid changes of lightening, and the introduction of shadows of hands into the scene by the pianist. The model is created and periodically updated from the resulting image of the geometric transformation of the keyboard from the video feed.

The hands pose a problem for my background subtraction as they will be included in the mask. However, the only moving objects that should be present in the foreground mask are the keys that are being pressed and released. Therefore, the hands are ignored and consequently removed from the difference image, this is achieved by identifying skin pixels that are within either the background or foreground image, and removing the location of those pixels from the foreground mask.

Finally, with key change events being the only moving objects in the mask, the difference image is analysed and the criteria by which a key can be classified as pressed or released is applied. With the results of the key change events, which includes the MIDI number and the pressed and released frame number, the reference beat is used to convert the events stored in memory to a MIDI file.

The final step of this algorithm is through the use of an open source library known as Muscore. Through Muscore, a PDF version of the sheet music can be rendered via a scripting tool that passes the MIDI file created by my application as input to Muscore, where it is processed and outputs the final product of my application. With all components of the transcription software completed, the pianist can now view the sheet music of their performance in PDF format and distribute it as they please. Figure 3.3 outlines a visual representation of the architecture for converting the video to sheet music.

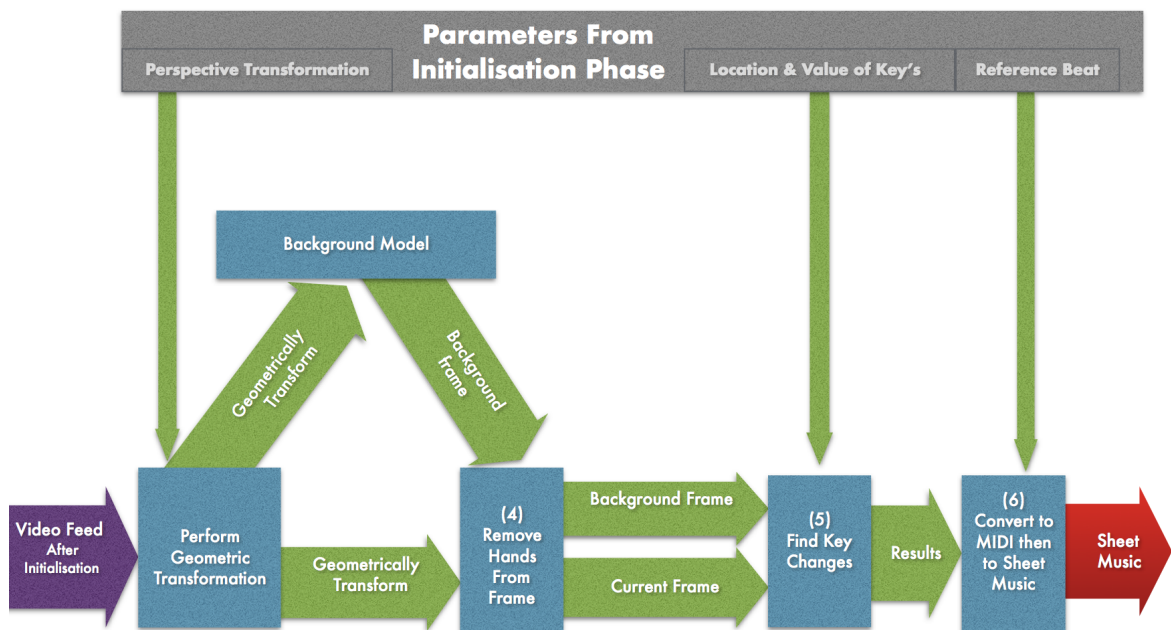


Figure 3.3: Architecture for converting the piano video feed to sheet music.

## 4.1 Finding the Keyboard

As mentioned in Chapter 3, the first step of the initial setup is to establish the location of the piano's keyboard within the scene. Recall that a keyboard is a rectangular shape when it is viewed face-on from above. But, the angle and position that the camera needs to be in for my application results in a perspective where the keyboard looks like an irregular quadrilateral that's not rectangular. The aim of the first initial setup algorithm is to find the perspective matrix that transforms this non-rectangular object back to its original rectangular shape. This perspective matrix needs to be found only once, due to the specifications of my application; the camera should not be moved after the initial setup has completed, hence every consecutive frame can make use of the same matrix. After the keyboard has been found the resulting geometric transform will produce a keyboard where the keys are vertically aligned. The geometric transformation used for this object is a perspective transformation. A perspective transformation is a linear transformation that is required when the planar surface is not parallel to the image, in this case, for example, an affine transformation would not properly correct the view of the planar. To perform a perspective transformation I require four observations. The problem is to determine eight coefficients  $p$  for the transformation based on these four observations.<sup>3</sup> Fig. 4.1 depicts the matrix used to perform perspective transformations.

$$\begin{bmatrix} i \cdot w \\ j \cdot w \\ w \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

Figure 4.1: Perspective transformation model.<sup>3</sup>

The four observations of this technique are the four corner points of the piano; top left (A), top right (B), bottom right (C), and bottom left (D) as illustrated in Fig. 4.2. As can be seen in Fig. 4.2, for my application, the piano ends at the last black note, while this is a design choice to find the pose of the keyboard using the orientation of the black keys, it is not a significant limitation. Very few composers use the far reaches of the piano, and even fewer the last three white keys that are lost. Even when these notes are used they are nearly always a result of doubling at the octave\* in which case we would have the upper or lower note within our recognition zone. For instance, at the time of Beethoven, the piano only contained 6 octaves, recall from Chapter 2.5 the synopsis of an octave, while a modern piano like this contain 8 octaves. Therefore, there are an extra 12 keys on each side of this piano that would never be used while playing any of Beethoven's works.



Figure 4.2: The corners top left (A), top right (B), bottom right (C), and bottom left (D) to be found by the algorithm as the four observations.

The outline of the algorithm to find the corners is discussed below, each step in the outline will be explained in detail thereafter.

---

\*Doubling at the octave occurs frequently in piano music, this is when a pianist plays the same note an octave apart often with the same hand; often a melody or bass-line is "doubled" as such to add texture and emphasis.

**Algorithm to find the corners of the piano:**

- A. Find lines in the scene using the Hough Line Transform.
- B. Separate the lines into two categories based on their slope.
- C. Order the lines using the vector cross product.
- D. Match parallel lines into bins.
- E. Find a quadrilateral object that satisfies the predefined property using the lines.
- F. Verify the object by searching for black keys within it.
- G. Analysis if an improved representation of the piano can be found.
- H. Find the pose of the keyboard using the orientation of the black keys
- I. Confirm that the corners are suitable using a verification function and finally create and return the perspective transformation matrix.

**4.1.1 Algorithm for Finding the Keyboard*****A) Find lines in the scene using the Hough Line Transform.***

A piano's keyboard is a recognisable object, it has black keys on the upper  $\frac{2}{3}$ rd's and white keys on the lower  $\frac{1}{3}$ rd. This criterion means the upper  $\frac{2}{3}$ rd's contain a mixture of black and white keys, and the lower  $\frac{1}{3}$ rd contains only white keys, therefore, the upper average pixels value will be darker than the lower. Using this property, a quadrilateral object located in the scene with the upper  $\frac{2}{3}$ rd's pixels darker than the lower  $\frac{1}{3}$ rd, can be concluded to be a piano's keyboard. To find a quadrilateral object the top and bottom lines of the piano which are parallel are found to form this shape by joining the edge of the lines through geometry. An algorithm for finding straight lines in an image is used to find these lines.

The Hough line transform is used to detect straight lines. The transform works on an image that has been pre-processed using an edge detection algorithm such as Canny edge detector to create a binary edge image; Fig 4.3 depicts this binary edge image where the threshold was found using Otsu's thresholding<sup>†</sup>. The Hough line transform works by finding an edge point in this binary image, and increments the cells in the Hough space such that it corresponds to every possible line through that edge point. Interpretation of the accumulator

---

<sup>†</sup>Otsu's thresholding is an algorithm that calculates every possible threshold value in order to find a value where the spread of the pixel values on each side of the threshold is minimised.

in the Hough space yields lines of infinite length. The interpretation is done by thresholding and conversion of infinite lines to finite lines. Figure 4.4 shows the result of the Hough on this binary edge image in Figure 4.3.

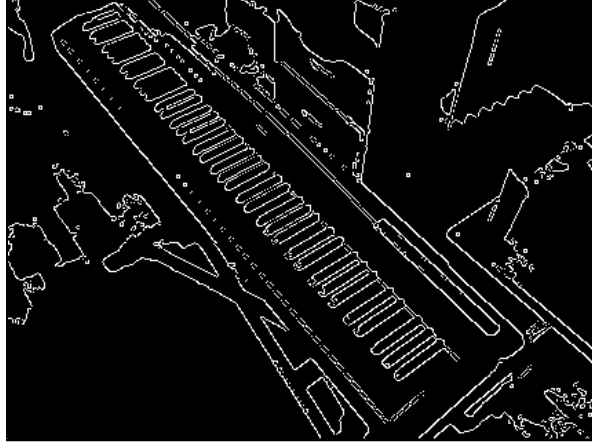


Figure 4.3: Result of binary edge image from Canny Edge Detector using threshold from Otsu's thresholding algorithm.



Figure 4.4: Line responses from the Hough line transform. Each line has been labelled in a different colour for visualisation purposes.

The problem with the Hough transform is that it is a computationally expensive operation. Therefore, I often try to apply methods to reduce the computational cost. The following restrictions are applied by my specification to reduce the cost, and also create a restriction on the responses given by the Hough space:

- Image sampling is changed for the input image to this algorithm to reduce the number



of pixels. This is due to the resolution from the Logitech HD webcam being far higher than is required for this operation, therefore, having a full resolution would significantly slow down the process. The resolution is reduced by a factor of 3 from  $960 \times 1280$  to  $320 \times 426$ .

- The minimum number of points that can form a line is set to 50 points. This will avoid small objects from being interpreted as lines, such as the black keys seen in Fig. 4.3.
- To reduce the amount responses of the Hough Space, only lines with the greatest peak are further processed.



Figure 4.5: Peak line responses from the Hough line transform. Each line has been labelled in a different colour for visualisation purposes.

Fig.4.5 depicts the set of lines with greatest peak<sup>‡</sup> from the Hough. As seen from Fig. 4.4 the amount of responses have significantly been reduced, while not removing the line at the top of the piano. However, the weak responses at the end of the white keys have been lost, but, as the bottom of the piano will always be parallel with the bottom of the white keys, this will not effect my application based on its requirements.

***B) Separate the lines into two categories based on their slope.*** The two categories are a slope of greater than 90 degrees and a slope of less than 90 degrees. Depending on the camera view, the keyboard will be in one of the categories, but not both. By separating the lines into two categories, the category with the most lines can be processed first and the second category can be ignored if the piano is found, thereby reducing the complexity.

<sup>‡</sup>local maxima in the accumulator space of the Hough Line Transform

**C) Order the lines using the vector cross product.** To interpret if a line is potentially the top or bottom of the piano, some ordering is required for the lines. Lines are ordered by sorting them based on if they are above or below one another. This ordering is achieved using the vector cross product. First, lines are converted to their vector representations, and then using the property of the vector cross product as shown in Figure 4.6 it can be determined if a is above b or vice versa.

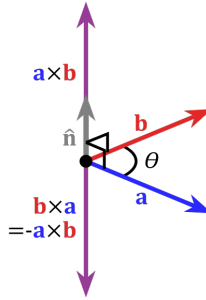


Figure 4.6: Vector cross product for a line 'a' and 'b'.

**D) Match parallel lines into bins.** After the lines have been sorted they are matched into bins based on their slope, allowing for a small deviation of the slope; this deviation is due to lines rarely being perfectly parallel. To eliminate lines that have more than one response, the lines in each bin are iterated through and it is calculated if their perpendicular distance from each other is at least the minimum perpendicular distance. This removes peak lines that have multiple responses.

If two lines do not meet the minimum perpendicular distance, the shorter line is removed. Figure 4.7 depicts the result of this process: in this example three lines have been matched based on their slope where the perpendicular distance is greater than the minimum requirement. Comparing Figure 4.7 to Figure 4.5, the minimum perpendicular distance has removed the small pink line as it was a repeat response of the blue line at the bottom of the piano's keyboard.



Figure 4.7: Pair of lines matched based on their slope, with minimum distance enforced to remove repeated responses. Lines labelled in the same colour for visualisation purposes.

***E) Find a quadrilateral object that satisfies the predefined property using the lines.***

Next, the bins of matching slopes are iterated through, ignoring bins with less than two lines present. Pairs of lines that are above and below one another are geometrically transformed. Geometrically transforming simplifies the implementation by not having to deal with a rotated quadrilateral when calculating pixels values. Following the geometric transformation the average pixel value is computed in the upper  $\frac{2}{3}rd$ s and the lower  $\frac{1}{3}rd$  of image. If the average of the upper  $\frac{2}{3}rd$  is darker than the average of the lower  $\frac{1}{3}rd$  the image is processed further. This criterion is used due to the structure of a piano, the upper  $\frac{2}{3}rd$ s contains a mixture of black and white keys, and the lower  $\frac{1}{3}rd$  contains only white keys; therefore, the upper average pixels value will be darker than the lower.

***F) Verify the object by searching for black keys within it.*** To eliminate false positives, black keys are tested for within the geometrically transformed image. This ensures the wrong quadrilateral object has not been found in the scene who has the same property as the piano, i.e., the upper  $\frac{2}{3}rd$  is darker than the lower  $\frac{1}{3}rd$ . Since the keyboard contains black keys within a white background (white keys), there is a high contrast between them. Therefore, the separation between each black key is well defined due to this high contrast. For white keys, this is not the case, as there is only a small region of separation which may not be evident under bright lighting conditions. Therefore, finding the black key through applying region labelling to the inverse of the binary threshold image is the easier approach. An inverse of the binary threshold image is applied to focus on the black features rather than

the white features. Note that if the line found at the top of the piano from the Hough Line Transform has any errors that cause it to include some of the piano outside of the keyboard, then the black keys can become one single connected component if the colour of the piano beyond the keyboard is dark. If this is the case, then these failures need to be removed by excluding a portion of rows from the top of the image to adjust the line properly to the top of the keyboard. Then the following algorithm is reapplied for finding black keys.

First the inverse threshold of the image is found using an Otsu's thresholding algorithm. The inverse binary threshold image focuses on the black keys for feature detection. By performing feature detection for blobs with a similar area, individual black keys are identified. If the number of black keys (blobs) found is greater than or equal to half the number of black keys on a standard piano: the lines of the top and bottom of the piano keyboard have been found, and the next stage of the algorithm begins. An in-depth discussion of the algorithm for finding the black keys is outlined in Chapter 4.2, when this same algorithm is applied in part for retrieving the location and values of the keys (within the geometrically transformed image that can be found upon completion of this component).

***G) Analysis: if an improved representation of the piano can be found.*** Depending on the performance of the Hough Line Transform, a portion of the keyboard in view may still not be found. To find the missing keyboard, I extend both sides of the piano separately. Each side is incrementally extended, and the new portion of the keyboard must match the criterion that the average of the upper  $\frac{2}{3}rd$  is darker than the average of the lower  $\frac{1}{3}rd$ . This process is repeated until the test fails, at which point the new portion is thrown away, as the end of the keyboard has been found on that side in the previous increment.

***H) Find the pose of the keyboard using the orientation of the black keys.*** The piano's keyboard will be distorted at this point if the video feed is not taken face-on. To remove this distortion, the angle of a number of black keys are found. By finding the slope of a number of black keys on either side of the piano's keyboard, the orientation of the black keys can be found, allowing us to retrieve the pose of the keyboard in the image, which ensures the perspective transformation matrix vertically aligns the keys.

***I) Confirm that the corners are suitable using a verification function and finally create and return the perspective transformation matrix.*** At this point the four corners of the keyboard have been found. Using these four observations the perspective transformation matrix is created, and it is performed on the video feed image. The final step

is a verification function that tests the perspective transformation: a), the pose estimation is correct (i.e, the black keys orientation is 90 degrees), and b), the number of black keys is greater than the last pass of this whole algorithm. These two properties ensure that the keyboard representation has the correct pose, and that the current representation of the keyboard has got a better result than is previously known for the keyboard in the scene. Both of these properties must be true or else the perspective transformation matrix is thrown away for this frame, and the perspective transformation matrix in the previous frame will be used instead.

As mentioned in (I) this algorithm is given a couple of passes with different video frames before the perspective matrix is finalised. The reason for this is to ensure that the best possible representation of the piano's keyboard is found. As mentioned the criteria for choosing the best matrices is through choosing the result that spans the largest portion of the keyboard.

Using the four observations (corners)  $(i_x, j_x)$  where  $x=1..4$ , the coefficients can be determined  $p$  by multiplying both sides by the inverse of the square matrix as depicted in Fig. 4.8.<sup>3</sup>

$$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ i_4 \\ j_4 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 & -i_1 i'_1 & -i_1 j'_1 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 & -j_1 i'_1 & -j_1 j'_1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 & -i_2 i'_2 & -i_2 j'_2 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 & -j_2 i'_2 & -j_2 j'_2 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 & -i_3 i'_3 & -i_3 j'_3 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 & -j_3 i'_3 & -j_3 j'_3 \\ i'_4 & j'_4 & 1 & 0 & 0 & 0 & -i_4 i'_4 & -i_4 j'_4 \\ 0 & 0 & 0 & i'_4 & j'_4 & 1 & -j_4 i'_4 & -j_4 j'_4 \end{bmatrix} \begin{bmatrix} p_{00} \\ p_{01} \\ p_{02} \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{20} \\ p_{21} \end{bmatrix}$$

Figure 4.8: Using the four observations  $p$ , to determine coefficients  $p$ , to then create the perspective transformation matrix.<sup>3</sup>

Now that the perspective transformation matrix has been solved it can be used throughout the rest of the run-time of the application. The next step of the initial setup is finding the keys location and associating the MIDI number values to each of key's locations.

## 4.2 Finding the Keys

After the perspective transformation matrix has been found from Chapter 4.1, the resulting image of the geometric transformation using this matrix will be analysed for individual key locations and key values, i.e., where the keys are in the image and what MIDI number they represent. Likewise, with the perspective matrix due to the specification that the camera is stationary throughout the run-time of the application; once the key's location and value have been found it shall not change within the run-time of the application and therefore only needs to be found once.

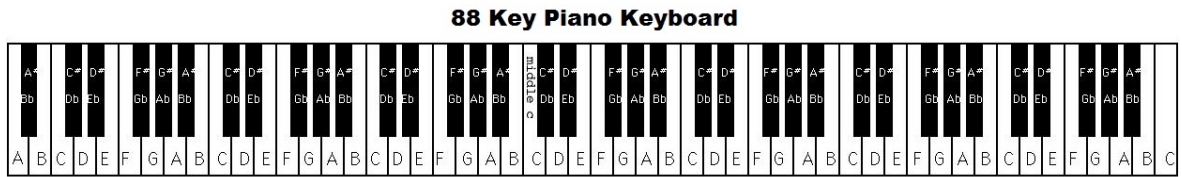


Figure 4.9: Modern piano keyboard with key values.

Fig. 4.11 shows an example of the classification of the keys of an 88-key piano. The standard piano has 88 keys with 52 white keys and 36 black keys. The black keys have a reoccurring pattern throughout the piano; two blacks followed by three blacks. The only exception to this pattern which can be seen in Fig. 4.11 is at the beginning where there is only one single black key. These exceptions only occur at the start of the keyboard. Therefore, the algorithm could be adjusted to handle this. However, to make the application available for non-standard pianos<sup>§</sup>, and some keyboards and synthesizers which typically contain less than 88 keys, I need an algorithm that does not rely on only a standard piano's keyboard. In Chapter 4.1 I located the black keys in an object to confirm that the object found was intend the keyboard. The same algorithm for finding black keys will be used on the geometrically transformed keyboard image, found using the matrix from Chapter 4.1.

### 4.2.1 Locating the Black Keys.

Since the keyboard contains black keys within a white background (white keys), there is a high contrast between them. Therefore, the separation between each black kes is well defined due to this high contrast. For white keys, this is not the case, as there is only a small region

<sup>§</sup>Some upright pianos, particularly from the early 20th-century, have only 85 keys, in this case there would be an exception at the top of the piano

of separation which may not be evident under bright lighting conditions. Therefore, finding the black keys is the easier approach through applying region labelling to the inverse of the binary threshold image. An inverse of the binary threshold image is applied to focus on the black features rather than the white features.

First the inverse threshold of the image is found using an Otsu thresholding algorithm. The inverse binary threshold image focuses on the black keys for feature detection. By performing feature detection for blobs with a similar area, individual black keys are identified. Algorithm 1 is used to return an object of the black keys. Figure 4.12 shows the result of the algorithm 1

---

**Algorithm 1** Find the Black Keys
 

---

```

1: procedure BLACK KEYS(InputImage)
2:   src  $\leftarrow$  Input Image Mat
3:   lowThres, highThres  $\leftarrow$  otzuInvThres(src) int
4:   maxBlackno  $\leftarrow$  0 int
5:   optimalThres  $\leftarrow$  0 int
6:   blackkeys  $\leftarrow$  NULL Object
7:   while highThres > lowThres do
8:     dst  $\leftarrow$  InvThreshold(src, highThres) Mat
9:     blackkeys  $\leftarrow$  findContoursSimilarArea(blackkeys)
10:    if sizeOf(blackkeys) > maxBlackno then
11:      maxBlackno  $\leftarrow$  sizeOf(blackkeys)
12:      optimalThres  $\leftarrow$  highThres
13:    end if
14:    highThres = highThres - 1
15:  end while
16:  dst  $\leftarrow$  InvThreshold(src, optimalThres)
17:  blackkeys  $\leftarrow$  findContoursSimilarArea(blackkeys) return blackkeys
18: end procedure

```

---



Figure 4.10: Black keys in purple 'blobs' depicting the result of algorithm 1.

### 4.2.2 Locating the White Keys.

With the black keys locations found, the location of white keys can be inferred. At this point, an empirical observation is taken from studying the keyboard technique in Section 2: a pianist striking a white key will at least be covering the bottom half of the key. Therefore, this has led me to ignore the bottom half the key entirely.

Examining Figure 4.11, a pattern of the piano's keyboard arises, there is always at least one white key between two black keys. The exceptions across the keyboard are between  $A\sharp/B\flat$  and  $C\sharp/D\flat$ , where there are two white keys B and C, and between  $D\sharp/E\flat$  and  $F\sharp/G\flat$  in each octave (recall Chapter 2.5), where there are two white keys E and F as can be seen in Figure 4.11.

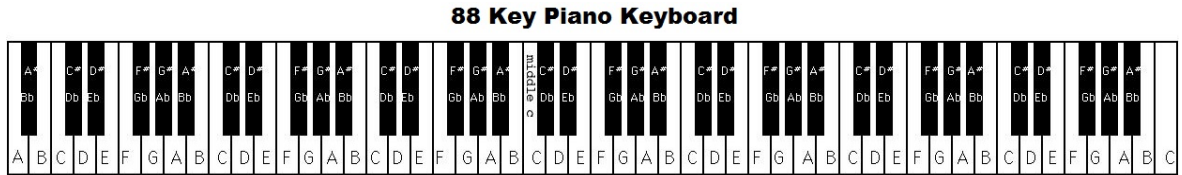


Figure 4.11: Modern piano keyboard with key values.

To find where these exceptions exist I find the median distance between all black keys found. Then iterating from left to right, if the distance between the current black key and the next black key is greater than the median distance, then two white keys are present between these black keys. To find the line that segregates these two white keys, the midpoint between the current black key and the next black key is found using the midpoint formula:

$$midpoint = \left( \frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right) \quad (4.1)$$

A vertical line at the midpoint is used to segregate these two white keys. This process is repeated until the end of the keyboard is reached.

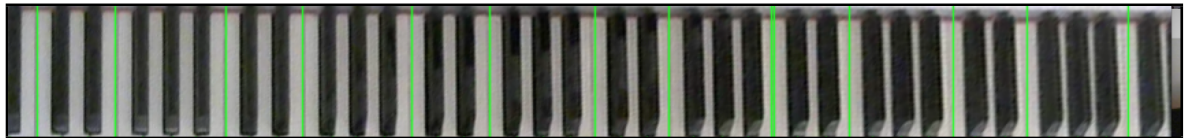


Figure 4.12: Result of using midpoint to infer the line (labelled in green) that segregates two white keys.



### 4.2.3 Associating the Keys to their Respective MIDI Values.

Keyboard	Note name	MIDI number
	C8	108
	B7	107
	A7	106
	G7	104
	F7	103
	E7	102
	D7	101
	C7	100
	B6	99
	A6	98
	G6	97
	F6	96
	E6	95
	D6	94
	C6	93
	B5	92
	A5	91
	G5	90
	F5	89
	E5	88
	D5	87
	C5	86
	B4	85
	A4	84
	G4	83
	F4	82
	E4	81
	D4	80
	C4	79
	B3	78
	A3	77
	G3	76
	F3	75
	E3	74
	D3	73
	C3	72
	B2	71
	A2	70
	G2	69
	F2	68
	E2	67
	D2	66
	C2	65
	B1	64
	A1	63
	G1	62
	F1	61
	E1	60
	D1	59
	C1	58
	B0	57
	A0	56
	G0	55
	F0	54
	E0	53
	D0	52
	C0	51
	B-1	50
	A-1	49
	G-1	48
	F-1	47
	E-1	46
	D-1	45
	C-1	44
	B-2	43
	A-2	42
	G-2	41
	F-2	40
	E-2	39
	D-2	38
	C-2	37
	B-3	36
	A-3	35
	G-3	34
	F-3	33
	E-3	32
	D-3	31
	C-3	30
	B-4	29
	A-4	28
	G-4	27
	F-4	26
	E-4	25
	D-4	24
	C-4	23
	B-5	22
	A-5	21

Figure 4.13: Modern piano keyboard with its MIDI values.

Once all the keys have been located, they are associated with their respective MIDI value. To determine the MIDI value of each key, there needs to be one single MIDI reference that the rest of keys can be worked out from. If my solution was limited to a modern piano I could start from the left of the keyboard with the first MIDI number (see figure 4.13) and increment while moving up to each subsequent key. However, as my solution has not been limited to just a modern keyboard, the following process is applied.

Through the studying of music theory, it was discovered that a pianist usually takes their bearings from middle C, note that middle C is also the note that generally divides the treble and bass staves in written music<sup>¶</sup>. Middle C can be found any keyboard by counting the number of C's and locating the "middle" c. Therefore, to automate this process, I created an

<sup>¶</sup>The subject is significantly more complicated than this, as the dispersion of notes on the staves is also used to differentiate which hand plays what and also to clarify the movement of polyphony in complex part writing; however, in simple written music, notes that are below middle C are written in the bass clef and notes that are above middle C are written in the treble clef; all transcription software automatically divides amongst the staves according to this general principal.

A diagram of a piano keyboard showing the relationship between white and black keys. The white keys are labeled B, C, D, E, F. The black keys are labeled C# and D# above them, and Db and Eb below them. The black keys are also labeled B and B below them.

The number of times this pattern is found is calculated, and the pattern that contains middle C is deduced as follows:

Then location of Middle C is at pattern  $= \frac{k}{2}$

Then location of Middle C is at pattern  $= \lceil \frac{k}{2} \rceil$

(4.2)

Figure 4.15 shows the result of this algorithm on keyboards of various sizes, i.e., to prove this choice of pattern works on all keyboard sizes.

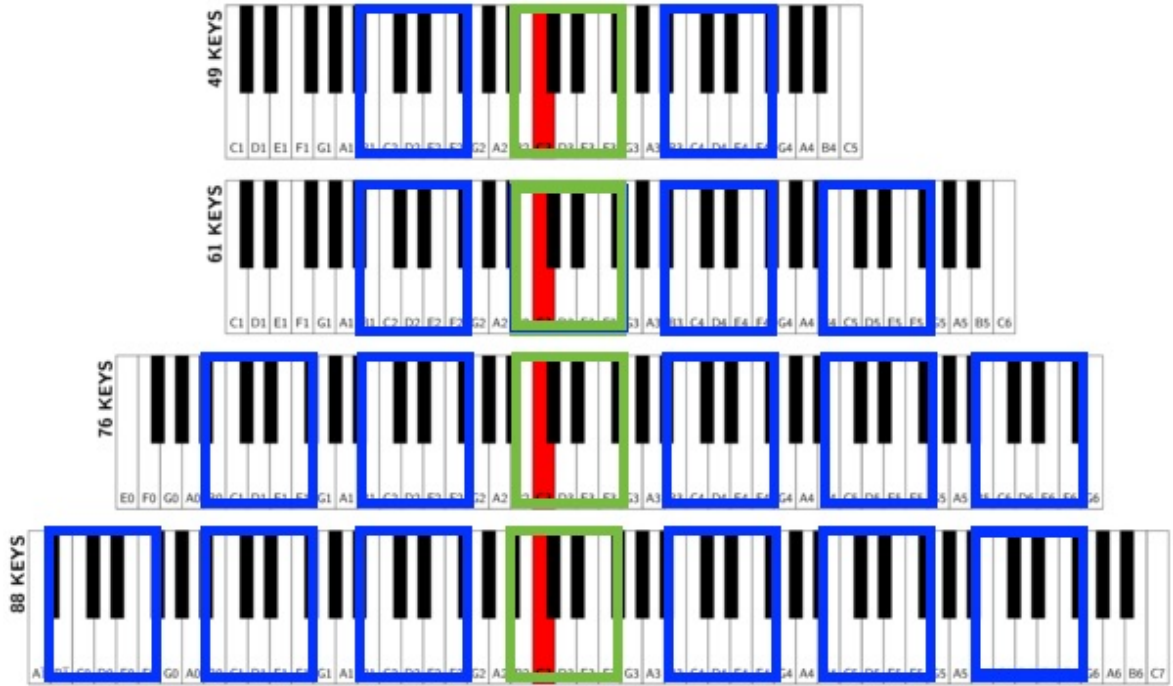


Figure 4.15: Unique pattern to find C applied in blue/green: the location of middle C is in green found using equation 4.2.

The second w within the resulting pattern  $w(w)bwbww$  is the C key labelled in red in Figure 4.15. To find the MIDI numbers that represent each key in the scene, I use the rules of MIDI. In MIDI, middle C is given the value of 60, using middle C as a reference I increment the value of each key that is to the right of middle C, and decrement the value for each key that is to the left of middle C. Fig. 4.13 illustrates the output of the algorithm for a modern keyboard with its associated MIDI numbers.




As with the perspective matrix, once the key's location and value have been found, it shall not change within the run-time of the application and therefore only needs to be found once. A representation of the keyboard which includes the location of each key within the geometrically transformed image and its associated MIDI number will be utilised during converting the video feed to sheet music. However, before the transcription can begin one more initial setup attribute needs to be found, the reference beat. This last and final attribute required in the initial setup is a reference to the beat of the music that the pianist intends for the performance used in Section 5.4 for creating the MIDI file.

### 4.3 Reference Beat of the Music

The last and final attribute required in the initial set up is a reference to the beat of the music that the pianist intends for the performance. As mentioned in 2.3.4, the rhythm of the music is invariant to the tempo the pianist plays at. Remember that in theory, if two pianists play the same performance the resulting sheet music should be equivalent, regardless of whether or not one pianist's performance was at a faster or slower tempo than the other.

As discussed, it requires a reference to the tempo at which the pianist intends to play to be able to interpret and transcribe the performance of a pianist in any meaningful way. Therefore, my application allows the pianist to 'count themselves in'. 'Counting yourself in' indicates that the musician has to strike the middle C key at the same intervals as they would set a metronome<sup>||</sup>. The application can then interpret the beat of the music to be used throughout the performance, by counting the interval of these strikes as a duration of frames. Hence, in the initialisation setup, after locating the keyboard and its keys, the application will instruct the pianist to count themselves in.

The process by which the strikes of middle C are recorded is through using a simplified version of Chapter 5. The same procedure is followed, however, rather than converting the strikes of middle C to a MIDI file, the sum of the intervals is averaged to be used in the following circumstances:

- The result of this calculation is what will be used in the transcription to represent the duration of a single beat. If we recall again from Chapter 4.3 a single beat is represented as a , by having a reference to the number of frames that equal this note, all other note values can easily be inferred from this; e.g., twice the frame count would result in a transcription duration of , half the frame count would result in a transcription duration of , etc., etc.
- The resulting single beat is the period by which the median background model will be updated. I.e. if the reference beat was ten frames then the background model would be updated every tenth frame. The reason for this will be discussed in Chapter 5.1.

---

<sup>||</sup>Modern digital metronomes have a similar tap-tempo feature that allows the user to input a tempo by tapping a button or key four times. This feature is also found on most contemporary digital audio workstations (DAWs such as Logic X or Ableton) and transcription software.

## 4.4 Conclusion

The initialisation of my application is now complete. During the run-time, the user will be notified that they can begin their performance on the piano.

To recap, in this Chapter I have:

- In Section 4.1, presented the algorithm for finding the perspective matrix which is used to transform every consecutive frame within the application thereafter.
- In Section 4.2, located the position and value of the keys within the transformed image, which will be used for identifying what key has changed and what MIDI value it represents.
- In Section 4.3, the reference beat of the music was found, for providing a means of interpreting the tempo of a pianist's performance. This will be used when creating the MIDI file and defines the interval for updating the background model.

Upon the completion of the initialisation step, Chapter 5 describes the algorithms used to convert the video feed to sheet music.

## CHAPTER 5

---

### Converting Video Feed to Sheet Music

---

Upon completion of the initial setup, the transcription begins. Three attributes have been defined at this point: the perspective matrix, the location and value of the keys, and the reference beat of the performance. All of these attributes are vital for the transcription algorithms to work. Every frame that is fed into the application from this point on is geometrically transformed using the perspective matrix. The resulting image of this geometric transformation is used in every component, 'the background model, removal of the hands, and identifying key changes', until a key change is found.

The process by which I identify key changes is through the use of background subtraction (outlined in Section 5.1 of this Chapter). Background subtraction is the process of subtracting a foreground image from a background image, to create a foreground mask of a moving object in the scene. The moving object in my application is the key changes that are to be found. A model is required for this background image so it can adjust to rapid changes of lightening, and the introduction of shadows of hands into the scene by the pianist. The image by which my model is created and periodically updated is the geometrically transformed image from each frame of the video feed. The frequency it is updated is defined by the reference beat, which I have just discussed in Chapter 4.3.

The next stage within transcription is removing the hands from the scene, which will be outlined in Section 5.2 of this Chapter. The hands pose a problem for background subtraction as they will be included in the foreground mask. However, the only moving objects that

should be presented in the foreground mask are the keys. Therefore, the hands are ignored and consequently removed from the difference image.

The next stage within transcription is identifying the key changes which are outlined in Section 5.3 of this Chapter. Using the reference to the location and value of the keys from Chapter 4.2, and with the key change events being the only moving objects in the foreground mask, I can analyse and apply my criteria by which a key can be classified as pressed or released. The results of the key change events are converted to MIDI events using the algorithms in Section 5.4.

The final step of this algorithm is through the use of an open-source library known as Muscore, outlined in Section 5.5 of this Chapter. Through Muscore a PDF version of the sheet music is rendered through a scripting tool that passes the MIDI file created.

By the end of this chapter, all components of the transcription software will be completed. At that stage within the application, the pianist is directed to the sheet music of their performance.

## 5.1 Median Background Model

The aim of the background model is to be used to perform background subtraction for foreground segmentation of the key changes. As this application was designed with a stationary camera, this problem was suitable for solving with a background model based method. This involved initialising a background model over the first few frames using just the background (keyboard with no hands) in the scene. Therefore, the performer is asked to wait approximately 5 seconds before they should begin playing the piano after executing the application.

Many different background models were tested against the ground truth to find the model that yielded the best results. During this testing phase the model that outperformed the rest was the median background. The median background model provided an adequate background model which immediately reflects sudden scene changes,<sup>14</sup> and offers acceptable accuracy while achieving a high frame rate with limited memory requirements.<sup>15</sup> In research, these properties have made the median background extremely popular in background subtraction<sup>16, 17</sup>

The median background model is computed by taking the median pixel value of last  $n$  frames. However, this would be an extremely costly operation with an extensive memory management

requirement by storing the last  $n$  frames. Therefore, the median background can be significantly sped up by keeping a histogram of the sum of last  $n$  frames for each pixel, updating it using an ageing rate.

Problems do arise with the median background, specifically due to its ability to react to sudden scene change.<sup>14</sup> This sudden reaction means that when identifying prolonged key presses, i.e., the key is held for some beats, the pressed key can become part of the background model resulting in a loss of the intended duration of the note. To solve this problem, I periodically update the background model based on the reference beat, i.e., for a reference beat of 10 frames, the model is updated on every 10th frame.

Another problem which is inheritable to all background models is static objects becoming part of the model. This problem can occur when sections of the hands (usually the front of the hand) become relatively static due to the piece performed spending an extended period at a particular section of the keyboard.

While a potential solution to this problem could be restricting skin pixels from the background model, for simplicity, a skin pixel mask of the background model is found to remove it when identifying key changes.



## 5.2 Create a Mask of the Hands

The next stage within transcription is removing the hands from the scene. The hands pose a problem for background subtraction as they will be a moving object that is not used for identifying key changes, besides for narrowing the area on the keyboard where they are possible during that frame. Consequentially, the algorithm for identifying changes completely ignores the section of the keys where the pianist's hands are present. Therefore, every frame from the video feed and the background model passes through the remove hands component. A union of the masks from each image is created so that no skin pixels are present within the foreground mask found in Section 5.3 of this Chapter.

An overview of the algorithm for finding skin pixels:

- A. Identify skin pixels as a mask through explicitly defined skin region rules in RGB colour model.
- B. Improve algorithm using hue and saturation in HSV colour model.
- C. Use morphological transformations to clean up binary image.
- D. Morphological operations to include outline of the hand.



Figure 5.1: The mask of the hand found at each stage of the algorithm overlaid on the frame it is analysing for illustration.

**A) Identify skin pixels as a mask through explicitly defined skin region rules in RGB colour model.** To remove the hands from the scene, I have explicitly defined a skin region in the RGB colour model. The first of the rules is taken from:<sup>18</sup>

$$\begin{aligned}
 (R, G, B) \text{ is classified as skin if:} \\
 R > 95 \text{ and } G > 40 \text{ and } B > 20 \\
 \max\{R, G, B\} - \min\{R, G, B\} > 15 \text{ and} \\
 |R - G| > 15 \text{ and } R > G \text{ and } R > B
 \end{aligned}
 \tag{5.1}$$

**B) Improve algorithm using hue and saturation in HSV colour model.**

After (A) the piano's white keys are incorrectly identified as within this skin region due to their colour. To omit the white keys from being identified the second condition is added so

that before a pixel can give a response in the mask, it must have a low saturation. Saturation is the degree of strength of colour in an image and ranges from 0 to 1. Having a low saturation implies the pixel has no colour and is thus white. Therefore, true skin pixels will have a higher degree of saturation than the white key's pixels. Also, a third condition using hue is added to improve the algorithm further. Hue describes colour in an image and ranges from 0 to 360. Before a pixel can give a response in the mask, it must adhere to the following rule:

$$(H, S, V) \text{ is classified as skin if:} \quad (5.2)$$

$$( Hue < 25 \parallel Hue > 230 ) \ \&\& \ ( Saturation > 0.1 )$$

The hue component is selected as there should be a strong contrast between the skin and the white and black keyboard.<sup>19</sup> The results of these two rules being applied can be seen in Figure 5.1 part (B). Adding these conditions has caused less non-skin pixels to be identified. However, still, these rules alone will not purely solve removing the hands, as some small regions are still identified as skin pixels due to lightening. Therefore, the binary image is 'cleaned up' using a morphological transformation to remove the small region of errors.

### ***C) Use morphological transformations to clean up binary image.***

A morphological operation is a mathematical operation using nonlinear algebra to logically test every possible position within an image against a structuring element. Therefore, it can be thought of as a transformation of a structuring element in every possible location within the image.

The two most basic operations are erosion and dilation. Dilation adds pixels to the boundaries of the image, while erosion removes pixels from the boundaries of the image. Selecting the right structuring element for the task is the most important aspect of erosion and dilation. The structuring element that worked best for this application was a rectangular structuring element of size 11 X 11 by retaining large objects and removing small objects.

To remove the small non-skin pixel errors, a closing followed opening was performed on the binary image using the structuring element above. These are combinations of these operations above, a closing is an erosion followed by a dilation, and an opening is a dilation followed by erosion. Figure 5.1 (C) shows the removal of false positive skin pixels using the morphological transformations above.

However, there is still one final problem that needs to be addressed: the presence of shadows

on the sides of the pianist's fingers.

***D) Morphological operations to include outline of the hand.***

Shadows cause the skin to darken and therefore are not identified by the explicit skin region-based method. To include these pixels, the hands are dilated to include the outline of the hand using the same rectangular structuring element as in step C.

At this point, a mask of the hand has been found. This process is carried out for both the foreground and background image. The two resulting masks are OR'd together to create a single mask that can be used to remove the hands from the difference image in the next stage of the algorithm, identifying key changes.

### 5.3 Finding the Key Changes

Finding the key changes is the most significant component of this transcription software. There has been a substantial amount of pre-processing before this stage of the system could be performed. At this stage, the perspective transformation matrix to find the keyboard has been calculated and applied, the location and value of every key have been classified, the background model created, and a mask of the hands obtained. The final stage before creating the MIDI file of the pianist's performance is identifying the key changes in the video feed as they occur. Given that key change events occur as moving objects in the scene, a well-defined process for identifying moving objects in a scene is applied known as background subtraction. Recall in Chapter 4.2.2, when finding the white keys only the top of the keys were found. This empirical method was applied due to the bottom of the white key being predominantly covered by the finger striking it. Therefore, only the top of the keyboard is examined for key changes illustrated in green in Figure 5.2.

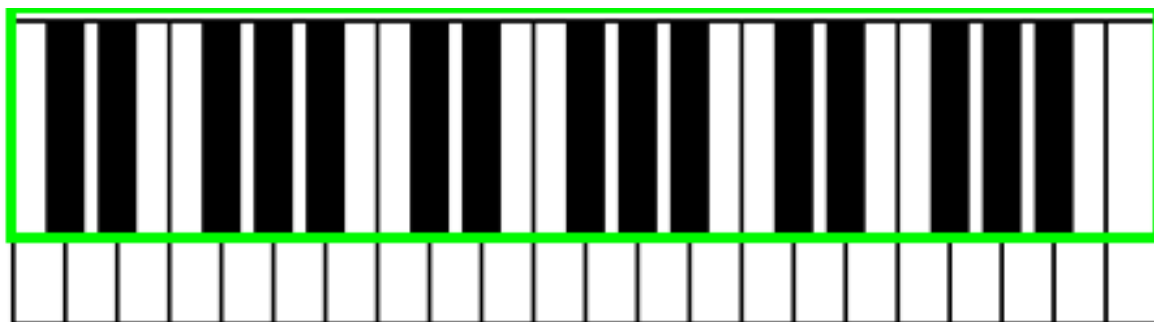


Figure 5.2: Zone of the keyboard where the key changes are classified.

The outline of the background subtraction based method:

- A. Compute difference image from background and foreground image.
- B. Find connected components using a contours based method within the binary image.
- C. Identify key changes using the connected components.
- D. Transfer results of the key change events to the MIDI interface.

**A) Compute difference image from background and foreground image:** To identify moving keys in the scene I perform a background subtraction. Background subtraction is the most widely used method for identifying moving objects in a scene; this basic method is used in numerous research papers, e.g. (Horprasert,1999),<sup>20</sup> and particularly popular in motion

detection in surveillance research used in (Singla, 2014)<sup>21</sup> (Sakaida, 2002).<sup>22</sup> The segmentation process is usually based on greyscale intensity, colour, shape, or texture,<sup>23</sup> in this case it will be dealing with greyscale\* intensities by converting the video feed which is in RGB colour to greyscale. The subtraction is performed on a background image and a foreground image to produce a foreground mask. In Section 5.1 of this Chapter, I described how my background is modelled. This background model will be used for extracting a background image at each frame, to segment a foreground mask by getting the difference image of the background and the foreground (current frame). The current frame in my case is the frame from the video feed at which I am trying to find potential key changes. The difference image produced will contain both the key changes and the hands, therefore, to remove the hands, the hands mask of both foreground, and the background, are found as described in Section 5.2 as a single mask. This mask is exclusive OR'd with the resulting difference image of the background subtraction method to remove the hands. Once the hands have been removed, the positive values and negatives values in the difference image are separated into different binary images. Separating into two different "difference images" based on their sign, allows the analysis of black and white key changes in isolation.<sup>11</sup> For the conversion of the greyscale image to binary, a threshold of 10 is used. Therefore, for the greyscale image with a range 0-255, anything above 10 is given the value one within the binary image. The threshold is set low to lose as little information as possible, while also removing noise. However, as I am splitting the negative and positive values of the difference image, the difference between the two images is calculated pixel by pixel with the negative values going to one image and the positive values to another; the absolute value of the pixel difference must still be greater than 10. The negative and positive values are split to allow each key type to be analysed in isolation. The following outlines which key type is processed in which difference image:

- **Positive Difference Image:** When a white key change occurs, pixels of the neighbouring black key become exposed due to the action of pushing the white key down. Therefore, the greyscale values will change from bright to dark. Hence, there will be a positive difference in the difference image. I.e., only white keys will be examined in this image while black keys are ignored.
- **Negative Difference Image:** When a black key change occurs, pixels of the neighbouring white key become exposed due to the action of pushing the black key down. Therefore, the greyscale values will change from dark to bright. Hence, there will be a negative difference in the difference image. I.e., only black keys will be examined in this image while white keys are ignored.

---

\*Grayscale is a single-channel image that carries only the intensity/illuminance value of an image.

Of course, this operation is not symmetric, therefore, the order of the pixel-wise difference, background frame minus the current frame, is imperative otherwise the rules governed in the positive and negative images would need to be swapped.

By splitting the key change events into separate testing sets, I reduce the occurrences of false positive key changes. These false positive changes occur due to keys exposing pixels of neighbouring keys. Although this is process by which my algorithms identify key changes when they are all contained in one difference image they can overlap among keys. For example, imagine a white key event exposing more black of its neighbouring black key when pressed. Had the key changes not been separated, a black key event could also be observed during this white key's event. But, with the current method, as the black key has gone from dark to dark there will be no information on this event in the negative difference image where the black key events are examined. Only in the black key region of the positive difference image and only white keys are examined in this image.

Figure 5.3 illustrates an overview of process of background subtraction based method that has been discussed.

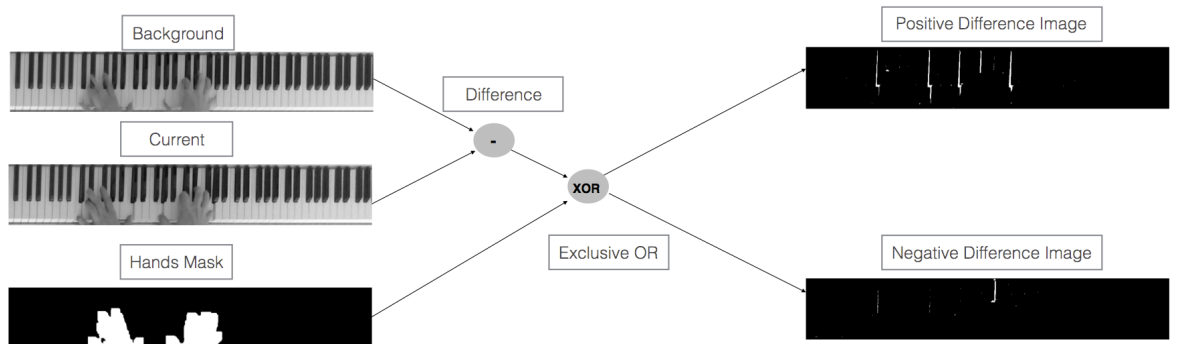


Figure 5.3: Illustration of the background subtraction based method applied on the background and foreground image to create the two difference images.

### ***B) Find contours in binary image and identify key changes:***

The resulting positive and negative binary image contain the moving key objects within the scene. Figure 5.4 illustrates the action of the key within an acoustic piano (similar action is simulated in a digital keyboard). The importance of this image is the orientation of the key. The action of striking this key forces the whole key to move in a vertical direction, but it

remains horizontal throughout. Therefore, when a key is struck by a pianist's finger a change of the height of the key should be found in the difference images. Hence, to test if an object approximately the size of the key has changed within the difference image, the connected components within the binary image are found.

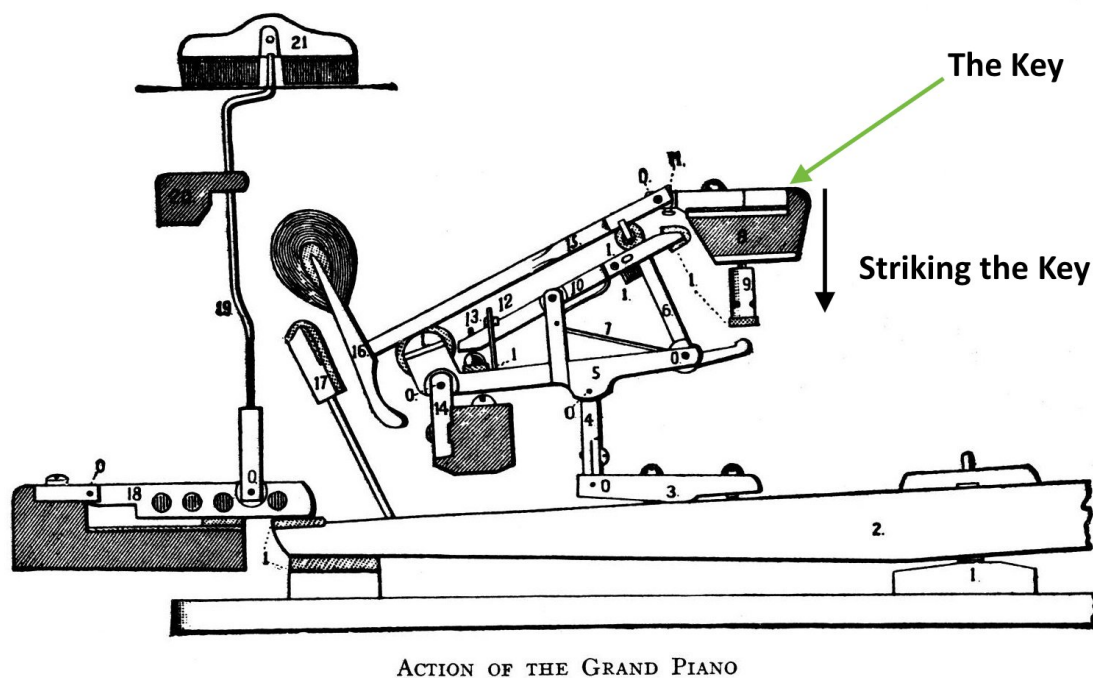


Figure 5.4: Anatomy of a key within a piano used for illustrating the action of a key when it is struck by a finger.<sup>4</sup>

Connected-component labelling involves labelling pixels of the same object with a single reference. In OpenCV, a common function for allowing connected-components labelling is a technique known as contours. In the contours technique, only boundary points of the connected-components are labelled in the binary regions; this is a more efficient approach than also labelling the body points.

Once the connected-components labelling has completed the algorithm searches for keys that are within the proximity of the hands. The proximity of the hands is illustrated in Figure 5.6, where every key within one extra white note (illustrated in blue) on either side of the hand is taken into account, including if a black note exists between that extra white note. This rule is based on studying the technique of the keyboard discussed in Chapter 2.2 due to for



instance when a pianist strikes an octave or chord<sup>†</sup> the finger may only enter a small region of the note; study Figure 5.5. Therefore, if there was any failure in the skin detection due to shadowing or lightening conditions this note could be excluded completely thus I include all key changes within one white note of the each side of hands masks.

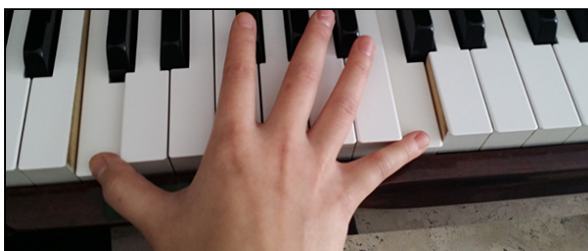


Figure 5.5: Octaves are played normally at the edge of the keys which leaves only a small region of skin that can be found

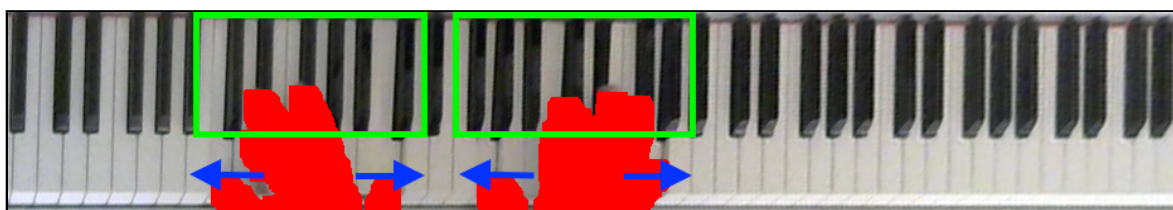


Figure 5.6: The box in green illustrates the zone of the keyboard that is examined for key changes by using the hand's proximity to exclude impossible note change. This zone is found by moving one key(blue) from the hands(red) on each side

Finally, after finding the keys within the proximity of the hands, connected components are searched for that are within these keys; the positive of contours for white keys and negative image of contours for black keys. If a connected component exists that is within the boundaries of a key and is at least half of keys original height, a key change has occurred. During development, half of key's height worked as an optimal criterion for a key change due to the omission of sections of the key changes being viewed due to the hands. Recall in Chapter 4.2.2, that when I looked for white keys only the top of the keys was found. This empirical observation was used as the bottom of the white key is always covered by the finger striking it. Therefore, when looking for key changes, the original height of the white key is the same as the black key. Examining Figure 5.2 at the start of this Section you can see key change zone for white and black keys is the same height.

<sup>†</sup>A chord is several keys played together.

***C) Transfer results to MIDI interface:***

There is two events on a key: the "pressed" and the "released" event. The "pressed" event occurs when the key is pressed down by the pianist, and the "released" event occurs when the same key is released some frames later. The case of this key change is created into an object containing its note value and the frame number it was pressed and released.

The process by which this object is created is a follows. If a key change event is observed, the MIDI number of that key is searched for in a list of key change events in memory. If the MIDI number does not exist in that list of key change events it is added to the list along with the frame event it occurred; This is the pressed event. If the MIDI number does exist in a list of the key change events, then the last observed frame for this key event within the list and the current frame are compared. If the current frame is only greater than the last observed frame by one frame, then the current frame is added to the list with this key as an observed key change within the duration of the pressed event. If the current frame is greater than last observed frame in the list by more than one frame, then a "released" event for the key in the list has occurred, and it is sent to the MIDI file. The current event is then added to the list as a new pressed event and waits for is subsequently released event.

With the results of the key changes ready to be transferred to the MIDI interface. The final step before the rendering of the sheet music can occur to this MIDI interface.

## 5.4 Convert the Key Changes to MIDI

During the final stages of execution of my program, the application parses the results of the key changes found to a MIDI file.

To create and write the MIDI file, I made use of a C++ library called *Midifile*. This provides an interface for importing header information and events into a MIDI file. As mentioned in Chapter 2.8.1 the events stored in a MIDI file identify information on how the song is played through note-on and note-off events attributed by ticks (duration, measured in quarter notes ♩). The events store the pitch of the key as a MIDI number. The relationship between pressing and releasing a key on the piano is associated in MIDI with a note-on and a note-off event respectively. Figure 5.7 gives brief structure of the MIDI file

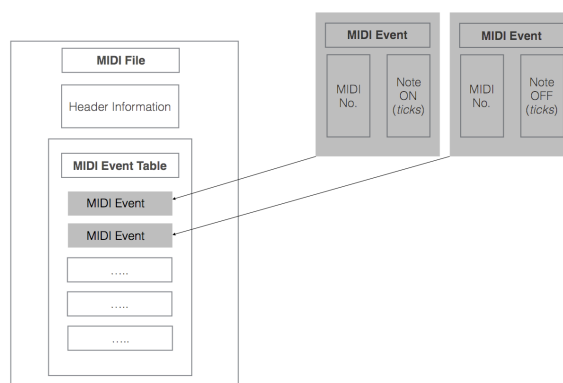


Figure 5.7: Simplified structure of the MIDI file that is created using header information and MIDI events

### 5.4.1 Method

As discussed in Section 5.3 of this Chapter. Key changes are stored as objects in the application using MIDI numbers with their pressed frame and released frame numbers before being transferred to the MIDI file. The pressed frame and released frame numbers are converted to MIDI note-on and note-off events respectively. A note-on event is calculated in ticks using the following equation using a default ticks per quarter of 80 b.p.m (this is the intended

tempo of my testing videos to be disclosed in Chapter 6.2).

$$\text{Number of ticks} = (\text{Pressed frame} / \text{Reference beat}) \times \text{Ticks per quarter} \quad (5.3)$$

The process by which note-off is found is more complex than a note-on. As mention in Chapter 2.8.2 a pianist cannot play metronomically as there is always some discrepancies. In MIDI applications such as Logic X, they use a process known as beat mapping to visualises the beat so that the user can alter their MIDI file to fix their imperfectionst.

In my application, I attempt to automate this process by rounding the number of frames to the nearest duration based on the reference beat found in Section 4.3. The following equation shows how the duration is calculated:

$$\text{Duration} = (\text{Released frame} - \text{Pressed Frame}) / \text{Reference beat} \quad (5.4)$$

Using the result from the above equation, the number of ticks can be found using that table at Figure 5.8. This table illustrates the relationship between rounding and calculating the number of ticks for the MIDI file.






Notes	Duration	Number of Ticks
	$3.5 \leq D < 4$	4 x TPQ
	$1.75 \leq D < 2.25$	2 x TPQ
	$0.75 \leq D < 1.25$	1 x TPQ
	$0.25 \leq D < 0.75$	0.5 x TPQ
	$0.25 < D$	0.25 x TPQ

Figure 5.8: Relationship from musical theory to duration to MIDI ticks

For example, if I have a frame length of 15 frames for a key change and a reference beat of 5 then dividing by the reference beat equals a duration of 3. This duration searches for the

range closest to it, in this case the second category of between  $1.75 \leq D < 2.25$ . Taking the larger number away from the duration allows the remaining duration to be found, therefore,  $3 - 2.25$  leaves  $0.75$ . The remaining  $0.75$  duration closest range is to the third category of between  $0.75 \leq D < 1.25$  with nothing remaining. Therefore, the final result for this key change in ticks would then be  $2 \times TPQ$  for the second category of between  $1.75 \leq D < 2.25$  plus  $1 \times TPQ$  for the third category of between  $0.75 \leq D < 1.25$ . Hence the final result for ticks for the note off event in the MIDI file for this frame length of 15 is  $3 \times TPQ$ . This can be applied for any duration by starting from the top of the table, finding the most suitable range, and always taking the greatest value in the category away from the duration until there is no (positive) remaining duration to be found.

At the current stage, the MIDI file has been created through C++ library *Midifile* with the header information and events parsed from the results of Section 5.3. This marks the end of the transcription, the resulting MIDI file is transferred to an open source application to render the sheet music.

## 5.5 Convert the MIDI to Sheet Music



Figure 5.9: Transferring of the MIDI file to the MuseScore application

After the pianist performance, the MIDI file created is converted to sheet music through an open source application known as MuseScore. The application allows you to create, play, and print sheet music from a MIDI file. At the end of the execution of my application, the MIDI file created is sent as a command line argument to MuseScore. This file is then read and pertinent track information is extracted to render the sheet music as a PDF. The sheet music is then displayed to the user within my application.

## 5.6 Conclusion

The transcription is now complete. The user will have notified the application that they are finished with their performance. The application has been transcribing their performance in real-time. Therefore, upon notifying the application that the performance is over their sheet music will be rendered and available to view/share.

To recap in this Chapter, I have

- In Section 5.1, introduced the median background model.
- In Section 5.2, created a mask of the hands for removal from the difference image in Section 5.3.
- In Section 5.3, classified key changes through background subtraction based-method.
- In Section 5.4, parsed the results of Section 5.3 to a MIDI file.

- In Section 5.5, rendered the sheet music through Musescore using the MIDI created in Section 5.4.

Upon the completion of the development of this piano transcription software begins Chapter 6.1 to discuss the evaluation and results of the system I have created.

---

## Evaluation and Results of the System

---

### 6.1 System Evaluation

In "Experimental Models for Validating Technology", Zelkowitz and Wallace discuss the need for a concise taxonomy of experimentation and validation for technology that is present in other academic fields such as physics and mathematics. In software development, it is too often the case that researchers explaining a new technology perform the experiments themselves to show the effectiveness of their technology. The researchers are both the creator and the user of the new technology and do not use any external data, or known theories to show their technology conforms to some underlying model or requirement.<sup>12</sup> Taking guidance from Zelkowitz and Wallace, I will use a known theory of evaluation in piano studies for evaluating the difficulty of a piece. I will use the grading system in the Associated Board of the Royal Schools of Music (ABRSM). Using this grading system, I can have a performer play a piece randomly from each grade and then evaluate the precision and recall of my solution for each piece. Given that the grades difficulty increases reasonably linearly, I hypothesise that the performance of my application will reduce reasonably linearly with the grades, i.e., as the pieces become more complex the performance of the algorithm will decrease.

To retrieve meaningful results from my test dataset I had to explore a number of different methods of evaluation, each revealing unique issues, before I was able to determine my proposed evaluation.



### 6.1.1 Apparent Evaluation

An apparent evaluation given the goal of my application would be to merely compare the resulting sheet music of my application to the originally scored sheet music e.g., when the performer plays Bach Prelude 1 in C Major the resulting sheet music "should" look the same as an original score of Bach's Prelude. But as discussed in Chapter 2.8.2 due to minor performance discrepancies, some interpretive others unintended, to play a "perfect" performance of the piece's score is impossible, and only a concert pianist could get close. Therefore, evaluating in this way would be ineffective for assessing the performance of this application. Due to the very nature of the application, it can only capture the pianist genuine performance.

### 6.1.2 Intended Evaluation

Given that this application evaluates their actual performance perhaps using a well-defined transcription solution such as a MIDI keyboard as ground truth would give the best possible evaluation. This evaluation could work by asking a performer to play the test dataset on a MIDI keyboard that is connected to a computer device enabled with transcription software. A video would be captured while the pianist is playing on this MIDI keyboard. Therefore, two transcriptions could be found concurrently. Hence they could be compared using the well-defined MIDI results as ground truth. This would be useful as a musician using this application would not only be concerned with whether a note was found or missed but to what degree the application had failed. Often in musical theory, equivalence is not as important as similarities. Although there are exact transformations in musical theory such as transposition. Often repetition of the motif (a dominant or recurring idea in a musical work) is through resemblance in some informal way. In my application, there may be a loss of pitches, durations, dynamics, etc., which such an evaluation solution could explore using the following technique. By comparing two musical piece similarities to some degree, i.e., a similarity metric of the ground truth MIDI recorded from a keyboard and the resulting MIDI file created from the computer vision processing I could produce metrics that evaluate all these attributes. In computer science and related fields, a similarity is often measured by the inverse of some distance formula, Appendix A outlines an overview of how this similarity metric would have been implemented. However, the very nature of a MIDI keyboard is a digital device that transfers data, therefore, it has latency issues. Latency problems occur all over computer science particularly in fields such as networking and distributed system. MIDI devices are no exception to latency, due to

a) Non-deterministic delays from transferring MIDI signals from the keyboard via the MIDI

cable.

b) Non-deterministic delays from the receiving computer due to it using an audio buffer that delays playback and disrupts MIDI timing<sup>24</sup>

Therefore, there is no easy way to calibrate the frame rate of the video feed with the beats per quarter notes of the MIDI file. The procedure required to develop such a complex modelling of the two domains was too time-consuming even to consider and may not have even yielded worthwhile results. Therefore, the intended evaluation had to be replaced with two conventional computer vision metrics; precision and recall, frequently used in pattern recognition as a binary classification.

### 6.1.3 Proposed Evaluation

The proposed method of evaluation involves using precision and recall for the performance of my application. As discussed in Chapter 2.9.2, computer vision technique's for music transcription is in its preliminary stages. Therefore, there are no external video datasets of piano transcription containing ground truth that could be used as a baseline for evaluation. Hence, I had to create my own ground truth from the videos I captured of these selected pieces from the ABRSM grading system. There are intrinsic issues with creating your own ground truth due to the inevitable bias of modelling your test data to your result data. But by being aware of this prejudice, I tried to be as unbiased as possible when creating the ground truth dataset.

#### Interface for Creating Ground Truth

Creating your own ground truth is a time-consuming process. To ease the process, I developed a user-friendly interface. The interface has a comparable initialisation to the transcription application, but determining the reference beat, as shown in the initial setup section within Figure 6.1. After the setup is complete, the user is prompted with an interface for inputting the ground truth on a controlled feed of the testing videos. A short illustration of three different snapshots of this interface occurring over the course of identifying a single key change is shown within Figure 6.1 (Note no key change actually occurs in these snapshots as it is merely for illustration purposes). The user interacts with the interface using a selection of keyboard and mouse events.

- A user can move between frames using a keyboard entry of letter lowercase 'n' for next frame and using a keyboard entry of letter lowercase 'p' for the previous frame.

- A user can indicate a key press by hovering over the key in question and clicking the LEFT mouse button. The key will turn blue when it has been selected, seen within the interface in Figure 6.1. The MIDI number of the key selected and the frame number the event occurred are stored in memory.
- A user can indicate a release of this key in future frames by hovering over the key in question and clicking the RIGHT mouse button. The key will return to its original colour when it has been selected, as seen within the interface in Figure 6.1. The MIDI number of the key selected and the frame number the key release event occurred are matched with the previous key press event within memory and written to a ground truth file.

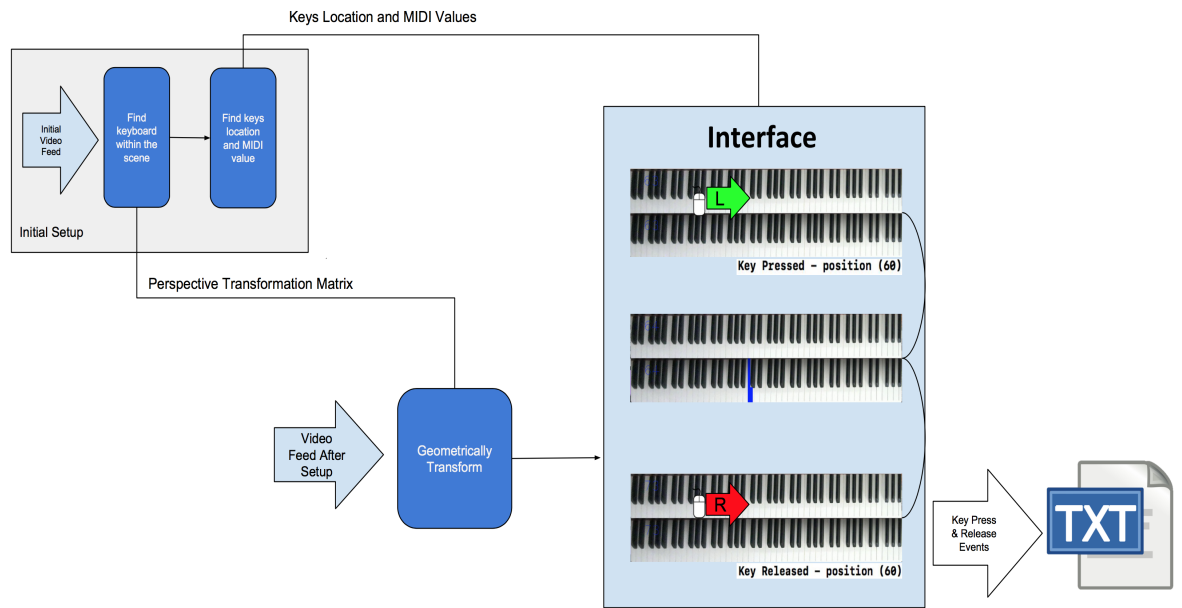


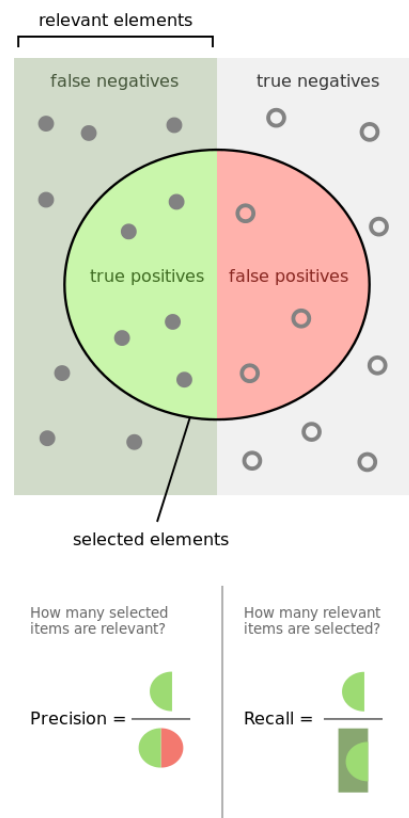
Figure 6.1: Architecture and brief illustration of the interface for creating ground truth from the videos captured.

## 6.2 System Results

As outlined in Section 6.1 the framework that provides a progressive difficulty of pieces for testing of my hypothesis was a known theory of evaluation in piano studies; the Associated Board of the Royal Schools of Music (ABRSM) grading system. To avoid bias selection, an experiment was performed by directing a pianist to choose one piece from each of the first six grades of the ABRSM examinations. Six videos were captured using a Logitech webcam (*HD C270*) of a pianist playing each of these pieces at a steady tempo of 80 beats per minute. For the test dataset, a steady tempo was assisted through the use of a metronome and guaranteed only to the best of the ability of the performing pianist.

Given the proposed evaluation of precision and recall discussed in Section 6.1.3, I had to create my own ground truth of these videos. Utilising the interface presented in Chapter 6.1.3 I created the ground truth and exported it to a text file. Similarly, the results of my transcription were also ported to a text file. An algorithm was developed to determine the true positive, false positive, and false negative values by comparing the ground truth and the result text files. These values were used for calculating the precision and recall metrics.

The following diagram and list explains the parameters and the equations used for calculating the performance metric in relation to properties of this application.

Figure 6.2: Visual explanation of precision and recall<sup>5</sup>

- True positive: Number of key changes correctly identified
- False positive: Number of key changes incorrectly identified
- False negative: Number of non-key changes incorrectly identified
- Precision: How many selected key changes are relevant key changes
- Recall: How many relevant key changes are selected key changes

Table 6.1 depicts the precision and recall of each video in the test dataset. The first three grades were shot on a standard digital keyboard while the last three grades were shot on a concert grand piano for testing diversity.

Table 6.1: Performance Metrics

Grade	Name	Precision	Recall
1	Dance (Elena Malychева)	65.51	86.36
2	Twinkle, Twinkle Little Star, Nursery Rhymes (Mozart)	86.51	97.64
3	Bagatelle No. 25 in A minor, Fur Elise (Beethoven)	83.69	99.48
4	Prelude in C major, BWV 846 (Bach, Johann Sebastian)	84.61	93.07
5	Solfeggio in C minor, H.220 (Bach, Carl Philipp Emanuel)	84.67	90.20
6	Prelude, Op. 28, No. 20 (Chopin)	67.34	94.82
<b>Average</b>		78.72	93.57
<b>Median</b>		84.15	93.95

### 6.2.1 Precision and Recall

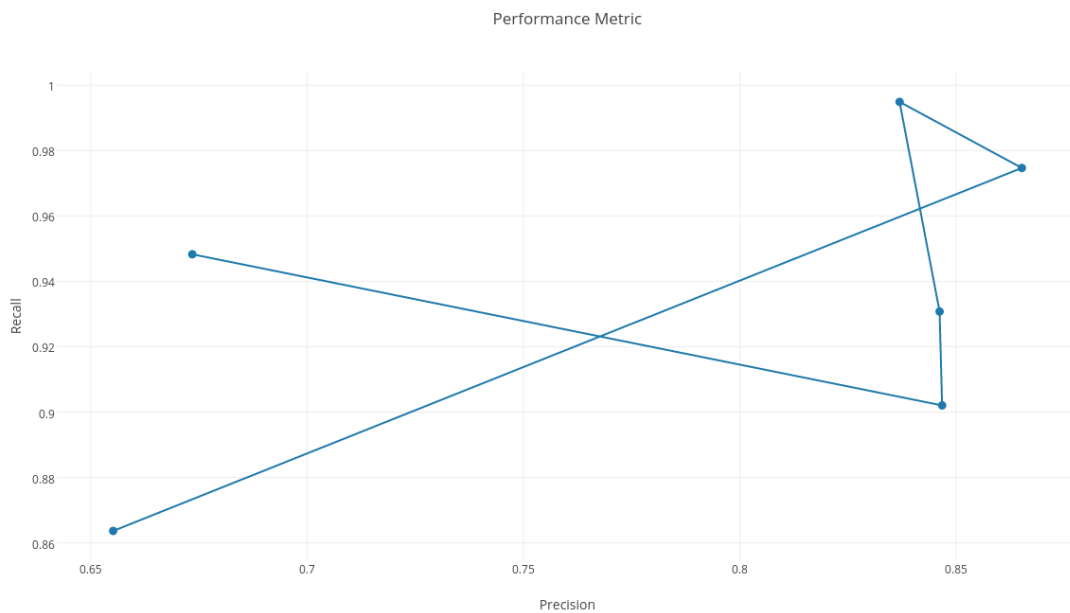


Figure 6.3: Performance metric of precision graphed with recall. Results from the six dataset videos with the sheet music taken from the first six grades of ABRSM

The results of the performance metric do not align with the hypothesised result I proposed at the start of this Chapter as there is no clear relationship between the difficulties of the piece and the precision of my algorithms. I hypothesised that the precision would decline in a relatively linear fashion, but there is no linear relationship to this affect. Instead, graphing

the results of the precision against recall yielded two outliers within the performance of my system. The two outliers occurred when the precision deviated dramatically from the median precision at Grade 1 and at Grade 6 with a deviation from the median of 18.63% and 16.80% percent respectively. The following discussion outlines the reason for these outliers.

## Discussion of the Precision Results

### Grade 1

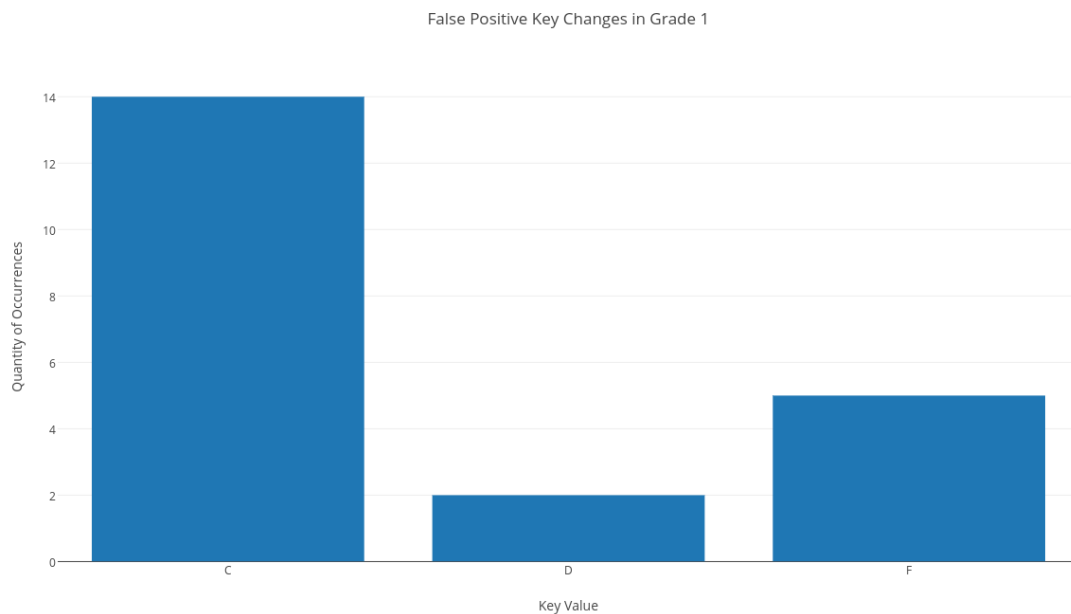


Figure 6.4: Key value and amount of occurrences off the false positive key changes within the grade 1 video

The Grade 1 piece deviated so far from the median due to the keys required to play the piece. The simplicity of the piece meant that it concentrated on a limited and a repetitive set of the keys, C and F\*. These keys can cause particular problems from certain angles, as they have only one neighbouring black key. Given the angle of the camera during this performance, shown in Figure 6.5 it affected the keys due to the following reasons.

---

\*In music theory the piece would be defined as having a limited pitch class.

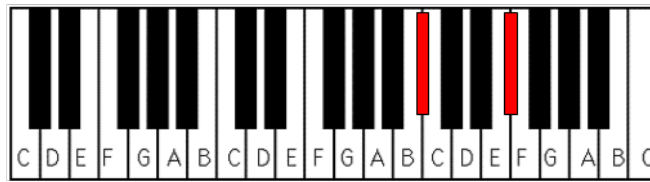


Figure 6.5: Position of the camera relative to the keyboard for the recording of the Grade 1 piece.

Figure 6.5 shows the position of the camera relative to the keyboard within the performance. The labels in red show that for key F and C there is no black key to be exposed when they are pressed. This negatively affects my algorithms, recalling from Section 5.3 that a white key change is identified through the action of it being pressed down revealing the black pixels of its neighbouring black key. However, for F and C at this angle, the only revealed pixels are the slightly darker pixels of the wood at the side of the neighbouring white key which can be seen in Figure 6.6 on the left for an acoustic piano. However, this particular video was shot on an electronic keyboard, where the keys are made of plastic, meaning the neighbouring white keys are also white as shown in Figure 6.6. Therefore, during this video, this resulted in repeated responses due to the algorithm relying only on the contrast between the white key and the revealed plastic of the neighbour white key, which was dependant purely on the difference in lightening. Hence this concludes that my algorithms are best suited for an acoustic piano, where the keys are usually made of wood and not plastic as was not the case in the Grade 1 video.

Note that had the performance been taken from the other side of the piano these keys would not have caused a problem, and instead the B and E keys would have been affected. But B and E were not as prevalent within this piece, and therefore this change would increase the performance of this particular piece. Of course, this still would not be a solution to the problem, and merely would have only solved this case.



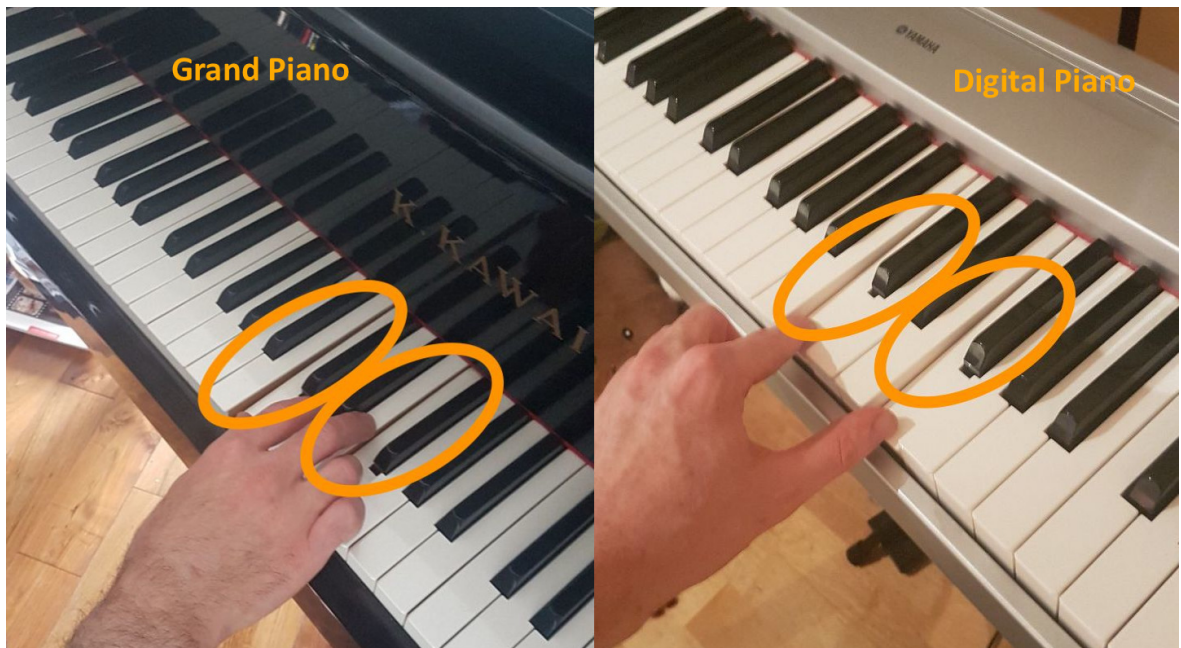


Figure 6.6: Difference of the neighbouring exposed note for a C and F for an acoustic piano (wood keys) and digital piano (plastic).

## Grade 6

After the preliminary grade (Grade 1) the pieces introduce chords and long sustained note values, some known as a fermata (a hold or pause of the note that exceeds its notated duration). These chords or sustained notes can result in false positive key change events due to the foreground key changes becoming a part of the background model. This happens when a long-pressed key exceeds the learning rate of the background model and becomes a part of it. When this happens, it results in (a) an early release event for the key in question (b) false positive key change events at a neighbouring key.

The reason for (a): due to the released key becoming part of the background the foreground key change and model are the same. Therefore, the algorithm believes the key change has stopped as there is no difference noticed.

The reason for (b): due to the algorithm now believing that the key has released another key event is going to occur when the key is actually released. In Section 5.3, I discussed the design of my algorithm for key changes. In that discussion, I outlined that white key changes and black key changes were dealt with in separate difference images; positive and negative.

Recall from Section 5.3:

- **Positive Image:** When a white key change occurs pixels of the neighbouring black key become exposed due to the action of pushing the white key down. Therefore, the greyscale values will change from bright to dark. Hence, there will be a positive difference in the difference image.
- **Negative Image:** When a black key change occurs pixels of the neighbouring white key become exposed due to the action of pushing the black key down. Therefore, the greyscale values will change from dark to bright. Hence, there will be a negative difference in the difference image.

In this scenario, because the released key is a part of the background model when the foreground released event occurs, the difference noticed if the scenario was a white key is now from dark to bright which is the criteria for a black key change. Likewise, the difference noticed if the scenario was a black key is now from bright to dark which is the criteria for a white key change. This does not cause a problem when the separation between a white key and a black key is well defined, i.e., when the keyboard is near the location of the stationary camera. However, when this separation becomes a lot smaller, i.e., the section of the keyboard, that is farthest away from the stationary camera, e.g., Figure 6.7 shows the distinction between black and white notes diminishing as they get further from the camera. Due to the angle and distance of the camera from the end of the keyboard, the separation between a white key and the black key becomes extremely narrow. This causes the scenario described above, resulting in producing false positive key change events not for the key that has become part of the background model but for their neighbouring keys.



Figure 6.7: Purple 'blobs' illustrate the result of finding the black keys using the feature detection algorithm. Notice the distance between black notes decreases as you get farther from the camera (Notice this camera was positioned on the left)

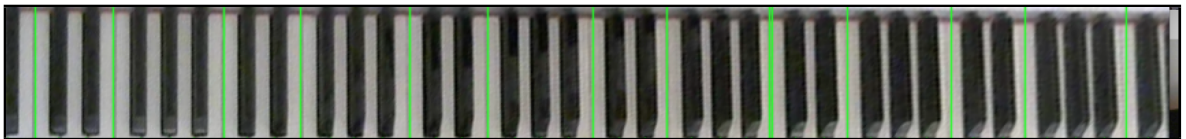


Figure 6.8: Result of inferring the line (in green) that segregates two white notes. Notice the distance between black notes decreases as you get farther from the camera (Notice this camera was positioned on the left of the piano)

The Grade 6 piece consists entirely of advanced long sustained chords progressions at the farthest end of the keyboard, which explains why it also sways so far from the other test datasets. Had time permitted a solution for this could have been individual background models for every individual key. Each keys background model could then not update during its pressed frames, solving foreground-pressed events becoming part of the background model. However, with this new model not updating during key presses, it could result in more problems that the current model solves such as rapid lighting changes and shadows of the hands. Therefore, there would need to be some significant investigation of the merits and demerits such an approach.

### 6.2.2 Recall

The recall is affected when omission occurs in the video feed. Omission is a major problem in computer vision specifically in surveillance. An example of omission is an object that is being tracked disappears in the scene due to another object in the scene blocking it from the sight of the camera. Omission occurs in two cases in my application. Figure 6.9 left represents the omission case 1 and Figure 6.9 right represent omission case 2.

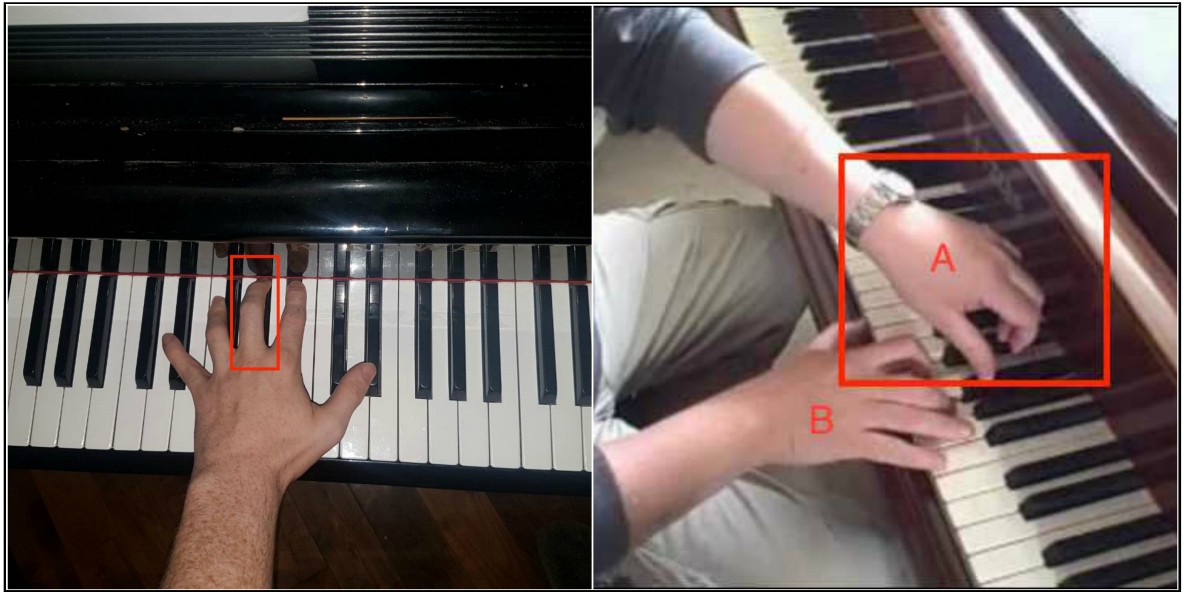


Figure 6.9: Omission cases 1 on the left and case 2 on the right

- Case 1: If the performer presses the key at the top, the finger will cover the key change detection area completely. (This goes against my empirical method that a pianist only strikes the bottom of the key)
- Case 2: If the performer crosses hands, the arm and/or hand could cover the key change detection area completely. Figure 6.9 shows hand A is covering hands B's key changes from being classified

### Discussion of the Recall Results

Unfortunately, I cannot transcribe key changes that the camera cannot see. However, if I cannot see it, maybe I can hear it. As will be discussed in future work for refining and improving my classification of key changes, I could combine the capabilities of audio signal processing. Using the audio that is captured with the video feed I could apply the refinement of the classification of the video processing and potentially find omission cases when they appear.

There is no clear relationship between the difficulties of the pieces and recall in these experiments. Recall is affected purely by the style of the musician, whether they tend towards certain movements throughout the performance of a piece. However, while there is no clear relationship, if the test data-set was to be performed on a more advanced piece such as the

final two grades of ABRSM, Grade 7 and 8. I hypothesis that it would result in the enormous amount of omission in selected pieces due to the need to constantly cross hands and exercise the full breath of the keyboard to accomplish some of the more technological demanding pieces. This would be particularly evident in a piece that is of the level of a concert pianist. For example, Beethoven Symphonies, a set of nine piano piece by Franz Liszt of Ludwig van Beethoven's symphonies 19 which are considered the most technically demanding piano music ever written would result in an enormous amount of omission. If omission could be solved in these pieces of music due to their recognised technological difficulties among the whole classical community, the argument for computer vision solving piano music transcription could not be denied.

### 6.2.3 Success of the Project

The performance metrics revealed some limitations in the background model, but also some specifications that a user should abide by for the best possible performance of this application. Firstly, the user should perform on an acoustic piano due to reason disclosed in precision in Section 6.2.1. The goal of this dissertation was to create a readily available device, but it was also to allow transcription on an acoustic piano which is not possible without expensive infra-red sensors installed on the strings. Therefore, abiding by this specification adheres to the motivation of this application. The application performs best with pieces that have a fast rhythm. As discussed in recall in Section 6.2.1 long sustained note caused a problem by becoming part of the background model. In conclusion, this application performs to a satisfactory performance for videos shot on an acoustic for pieces or compositions with a fast rhythm.

### 6.2.4 Limitations in the Proposed Framework

Due to the time-consuming process of creating ground truth I was limited in the number of videos I could use for my evaluation. As the difficulties of the pieces progress so too did the challenge in creating their ground truth. But had the time granted, for instance, I would have liked to have extended the data-set up as far as Grade 8; the final ABRSM grade.

I also recognised the fact that faster tempos may affect the precision of my computer vision algorithms and would have liked to explore this theory, particularly if the tempo got close to the frame rate. Although, pieces played at such a prestissimo (as fast a possible) tempo would be more common among pieces that go beyond the standard ABRSM examination to their senior certificate and diploma level. Therefore, experimentation would have

had to extend beyond my grading system to explore this theory.

In addition, had time permitted I would have tested the other three major components: locating the keyboard, locating the keys, and identifying skin pixels against some ground truth; which could have yielded more insight into the results of the evaluation. However, recalling that the goal of this dissertation was to design and implement an automated piano transcription software that would transcribe a piece of music played on the piano that was captured on video. Therefore, the results of the final output of the whole application were the most important aspect that needed to be evaluated for this system. Evaluating these other components without any available ground truth would have been a huge workload. Also, similar problems would arise when creating the necessary ground truth by the researcher (me) explaining a new technology also creating the test data to show the effectiveness of their (my) technology. Therefore, to create good ground truth for these components, it would require a series of strict experimentation regulated by an independent third party.

---

## Conclusion and Future Work

---

### 7.1 Conclusion

At the beginning of this dissertation, the goal was to design and implement a prototype automated piano transcription software that could correctly transcribe a piece of music played on the piano to sheet music from only a video feed. The approach for solving this goal was achieved to a degree through the use of computer vision algorithms on the captured video feed. This was a challenging task as at the beginning of the project there was only one piece of published literature available on the problem, due to all publicly available research concentrating on audio signal processing for piano transcription. The lack of research meant no publicly available third-party test data-set existed. Hence the test data-set used had to be recorded and developed from scratch. The testing portion of the application has been achieved through the creation of an interface that enabled a more efficient method of obtaining ground truth results from the recorded videos than visually writing out each key change event by hand.

No clear relationship was identified between the complexity of the piece to the precision of the application. The performance of the system worked best on an acoustic piano, and when the rhythm of the piece was fast. The average precision and recall were 78.72% and 93.57% respectively, with two outliers dramatically affecting precision.

## 7.2 Time Spent

The goal proposed in this dissertation represents a vast system, with a considerable amount of complex components that were required to build a system of this magnitude. Particularly in the need to possess an understanding of the ontology of the music domain to a minimum standard of an ABRSM diploma. Without this knowledge, the conversion algorithms used to create the semantics of this dissertation would not have been possible.

But the single component that took the most considerable amount of time was finding the keyboard in Chapter 4.1. The reason the development of this component exceeded the others was due to the keyboard being in a relatively unpredictable environment. After the keyboard is found and transformed almost everything besides lightening can be anticipated. But for finding the keyboard, the webcam is placed in an entirely new environment each time, with the only specification being that only one piano can be present within the frames. This created a lot of difficulties in fine tuning the algorithm in particular, as its success affected the rest of the application.

Also, a significant amount of geometry was required in the algorithms within this component, particularly when estimating the sides of the keyboard in Section 4.1.1. In C++, the maths libraries available did not provide any of the specific functionality that was required. Therefore, all the geometry functions were developed from the ground up.

## 7.3 Future work

### 7.3.1 Investigate Different Background Model

As discussed in Section 6.2.1, difficulties occur within the background model when a key is sustained for a long period: the key press becomes a part of the model. The result affects the precision of my application by introducing false positive key changes. Had time permitted I could have investigated the following solution: individual background models for every key. Each keys background model could then not update during its pressed frames, solving foreground-pressed events becoming part of the background model. However, with this new model not updating during key presses, it could result in more problems that the current model solves, such as rapid lighting changes and shadows of the hands. Therefore, this approach has been left as further work to increase the precision of this transcription software.



### 7.3.2 Refine Classification with Audio Processing for a Greater Precision

In the early stages of development of this dissertation, there was an intention of using both the video and audio signals captured by the device. By fusing the two domains together, I believed I could significantly increase the precision of transcribing; one domains weakness potentially being the other domains strength. For example, in section 6.2.1 on omission I could not transcribe notes that I could not see. However, if I cannot see it, maybe I can hear it. Using the audio feed, a refinement of my classification of the key changes could be have been found by combining the capabilities of audio signal processing. The audio transcription could be accomplished through multiple  $F_0$  estimations and periodicity transforms. As discussed in Chapter 2 the problem with music transcription using an audio signal is it is inherently difficult. However, as audio signal processing would not be my only means of transcription, it would have been an extra layer of refinement. The hope of the using audio processing for multiple  $F_0$  analysis would be to help cases such as omission and increase the precision of the events I can not detect in the computer vision domain. Using the audio that is captured in the video feed, I could apply the refinement of the classification of the video processing. But due to the large scope of the project, time did not allow for the development of the audio processing domain.

### 7.3.3 Neural Networks to Predict Key Changes

Through machine learning, a neural network could be fed with tonnes of examples of melodies and progressions within music. Over time, the neural network would be able to build a fuzzy relationship between these progressions and build a map of these relationships. Using this map, when a pianist is performing I could feed in the previous melody already found within the performance and use it to predict what keys to focus on in the next frames, thereby predicting where the key changes will occur before they even happen. This process could work particularly well in a genre specific performance. For instance, if a user told the application it will be playing a classical piece, then I could feed the neural networks with only classical melodies and progressions and the many rules of counterpoint and harmony that govern the unfolding of a piece of classical music. Why this would work well is because in classical music there are quite strict musical rules where some progressions which may be evident in other genres like jazz would never be present in a classical piece of a given period. With the application being aware of keys that can not or would be unlikely to be in the next frame, it can narrow down the set of keys it has to check.

If neural networks can learn melodies and progressions, it could also learn entire pieces.

The neural network could learn every classical piece written by each of the most influential composers, e.g., Beethoven, Bach, Mozart, etc. If the user decided to play one of Beethoven's Piano Sonatas with all of Beethoven works already been fed into the neural network the application could recognise the piece after a few bars. With the piece recognised that applications transcription becomes significantly easier as it expects where the next key change will be before it happens.

### 7.3.4 Augmented Reality MIDI-to-Piano Learning Assistant

Augmented reality is becoming more end-user accessible than ever before. Numerous headsets such as Microsoft's HoloLens and the Meta 2 are already shipping to developers and will be available to customers in the near future. These developers are trying to find new and exciting ways to harness the technology and make it useful to end-users.

There are many barriers to learning how to play the piano. One such barrier is learning how to read the sheet music. This requires time and effort, and the ability to mentally translate notes on a page into key presses with your hands. Without being able to read sheet music, it is difficult to learn new songs, and even once you have learned, it can take time to perform the translation that allows you to actually play the song correctly.

Augmented reality could remove this barrier through creating a learning assistant that can take any MIDI file and augment the notes with the correct timing over a piano using an AR headset. MIDI files are available for almost every sheet music that has been produced. Therefore, this would allow students of any level to learn to play almost any song on the piano, in perfect time and without having to be able to read sheet music.

It could achieve this through creating a 3D representation of the song from the MIDI file. Then tracking the piano itself using a computer vision algorithm presented in Section 4.1. Finally, after the piano is found, the 3D representation would be augmented over the piano to give the appearance of the notes lining up with the piano keys.

The algorithms of this dissertation could then be used for identifying if their intended key changes actually occurred. Using this systems algorithm, the application could give feedback to the user through colouring the keys green when keys were pressed correctly and red when they were missed. Thereby completely virtualizing a piano teacher using the algorithms of this dissertation and expanding them into augmented reality.

## 7.4 An Improved Skin Detection Algorithm

Although the skin detection algorithm performance was satisfactory, explicit skin defined regions can be problematic for different races due to varying skin types. For instance, in the test videos used in this dissertation, the skin type was caucasian.

A more appropriate solution to include these skin type would be to learn user's skin colour in the initial setup. This could be achieved in a number of ways for example: asking the user to play their hands in a certain position and calculating their skin pixels, or calculating the pixel values of moving objects in the skin that are not keys, i.e., hands. Using this skin sample space, an algorithm such as back-projection would identify skin pixels like in Section 5.2. Using a technique known as Back-projection, a histogram of the sample space could be created, normalise the sample space so that the maximum number is 1, and back-project the normalised histogram onto each frame. The result is a probability image that indicates the probability for each pixel in the frame belonging to that sample space. However, due to time this approach could not be developed and therefore has been left as future work.

---

## Bibliography

---

- [1] A. P. Klapuri, “Automatic Music Transcription as We Know it Today,” *Journal of New Music Research*, vol. 33, no. 3, pp. 269–282, Sep. 2004. [Online]. Available: <http://www.tandfonline.com/doi/abs/10.1080/0929821042000317840>
- [2] W. A. Sethares, *Rhythm and Transforms*. Springer, 2007.
- [3] K. Dawson-Howe, *A Practical Introduction to Computer vision with OpenCV*. Wiley, 2014.
- [4] “Musical Instrument Interfaces | a peer-reviewed journal about...” [Online]. Available: <http://www.aprja.net/musical-instrument-interfaces/>
- [5] Wikipedia, the free encyclopedia, “Precision recall,” 2013. [Online]. Available: <https://commons.wikimedia.org/wiki/File:Precisionrecall.svg>
- [6] “History of the Piano | Pianonet.” [Online]. Available: <http://pianonet.com/all-about-pianos/history-of-the-piano/>
- [7] E. Taylor, *The AB Guide to Music Theory, Part 1*.
- [8] P. D. Lehrman, *MIDI for the Professional*.
- [9] W. A. Sethares and T. W. Staley, *Periodicity Transforms*, 1999.
- [10] S. P. R. Glenbrook. (2002) Fundamental frequency and harmonics. [Online]. Available: [http://www.eng.buffalo.edu/shaw/student/m3\\_digimusic/02\\_documents/sound\\_wave/ResonanceStandingWaves\\_FundamentalFrequencyHarmonics.htm](http://www.eng.buffalo.edu/shaw/student/m3_digimusic/02_documents/sound_wave/ResonanceStandingWaves_FundamentalFrequencyHarmonics.htm)

- [11] A. Mohammad and C. Howard, “Real-time piano music transcription based on computer vision,” *IEEE Transactions on Multimedia*, vol. 17, no. 12, pp. 2113 – 2121, 2015.
- [12] M. V. Zelkowitz and A. R. D. R. Wallace, “Experimental models for validating technology,” *IEEE*, vol. 31, no. 5, pp. 23–31, 1998.
- [13] R. Descartes, *Discourse on the Method*, 1637.
- [14] M. H. Hung, J. S. Pan, and C. H. Hsieh, “Speed up temporal median filter for background subtraction,” pp. 297–300.
- [15] M. Piccardi, “Background subtraction techniques: a review,” vol. 4, pp. 3099–3104. [Online]. Available: <http://ieeexplore.ieee.org/abstract/document/1400815/>
- [16] E. Komagal, A. Vinodhini, A. Srinivasan, and B. Ekava, “Real time background subtraction techniques for detection of moving objects in video surveillance system,” pp. 1–5.
- [17] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, “Detecting moving objects, ghosts, and shadows in video streams,” vol. 25, no. 10, pp. 1337–1342.
- [18] J. Kovac, P. Peer, and F. Solina, *Human skin color clustering for face detection*. IEEE, 2003, vol. 2. [Online]. Available: <http://ieeexplore.ieee.org/abstract/document/1248169/>
- [19] R. M. Jusoh, N. Hamzah, M. H. Marhaban, and N. M. A. Alias, “Skin detection based on thresholding in RGB and hue component,” pp. 515–517, 2010-10.
- [20] T. Horprasert, D. Harwood, and L. S. Davis, “A statistical approach for real-time robust background subtraction and shadow detection,” vol. 99, pp. 1–19, 1999. [Online]. Available: [https://www.researchgate.net/profile/Larry\\_Davis3/publication/228609042\\_A\\_statistical\\_approach\\_for\\_real-time\\_robust\\_background\\_subtraction\\_and\\_shadow\\_detection\\_Proc\\_Int\\_Conf\\_Computer\\_Vision/links/09e4150c7748b678ff000000.pdf](https://www.researchgate.net/profile/Larry_Davis3/publication/228609042_A_statistical_approach_for_real-time_robust_background_subtraction_and_shadow_detection_Proc_Int_Conf_Computer_Vision/links/09e4150c7748b678ff000000.pdf)
- [21] N. Singla, “Motion detection based on frame difference method,” vol. 4, no. 15, pp. 1559–1565, 2014. [Online]. Available: <https://pdfs.semanticscholar.org/6811/362eeea2e24ccd2834b8110a6a670a7b627f.pdf>
- [22] S. Sakaida, M. Naemura, and Y. Kanatsugu, “Moving object extraction using background difference and region growing with a spatiotemporal watershed

- algorithm,” vol. 33, no. 12, pp. 11–26, 2002. [Online]. Available: <http://onlinelibrary.wiley.com/doi/10.1002/scj.10237/abstract>
- [23] P. W. R. S. A. A. Carlos. (2017) Image segmentation smart surveillance interest group. [Online]. Available: <http://www.ssig.dcc.ufmg.br/image-segmentation/>
- [24] N. L. Sound, “Identifying & Solving PC MIDI & Audio Timing Problems.” [Online]. Available: <http://nolimitsoundproductions.blogspot.com/2014/05/identifying-solving-pc-midi-audio.html>
- [25] K. S. Orpen and D. Huron, “Measurement of similarity in music: A quantitative approach for non-parametric representations,” *Computers in Music Research*, vol. 4, pp. 1–44, 1992.
- [26] R. C. Brill, Eric; Moore, “An improved error model for noisy channel spelling correction,” *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, vol. 38, no. 1, p. 286 293, 2000.

---

## Appendix

---

## APPENDIX A

---

### Metric for similarity in music

---

A musician using this application will not only be concerned whether a note is found or missed, but to what degree the application has failed. There may be a loss of pitches, durations, dynamics, etc. Often in musical theory, equivalence is not as important as similarities. Although there are exact transformations in musical theory such as transposition. Often repetition of the motif (a dominant or recurring idea in a musical work) is through resemblance in some informal way. Therefore, there is a need to compare two musical piece similarities to some degree. In computer science and related fields, a similarity is often measured by the inverse of some distance formula. In 1992, Keith S. Orpen and David Huron wrote a paper on the "Measurements of the Similarities of Music" through a quantitative approach for non-parametric representations. The paper discusses the use of the Damerau-Levenshtein metric otherwise known as the edit distance.

The edit distance algorithm is defined as the minimum edit distance between two strings known as the source and target. The algorithm defines three operations: insertion, deletion, and substitution. A dissimilar substitution is defined as the replacement of a letter in the source that is different from the current source letter. Each operation can be assigned a numeric penalty based on the magnitude of the operation. For example, each insertion could have a penalty of +1, deletion a penalty of +1, and substitution a penalty of +1. However, a similar substitution can be thought of as an insertion and a deletion, if a substitution were assigned a penalty of +2 it would make it redundant.<sup>25</sup>



The following equation (1) is taken from<sup>26</sup> and describes the algorithm. To express the edit distance of string a to b by  $d_{a,b}(i, j)$  I define an index  $i$  of a and an index  $j$  for b such that a recursive function is defined as

$$d_{a,b}(i, j) = \begin{cases} \max(i, j) & *1 \\ \min \begin{cases} d_{a,b}(i-1, j) + 1 \\ d_{a,b}(i, j-1) + 1 \\ d_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \\ d_{a,b}(i-2, j-2) + 1 \end{cases} & *2 \\ \min \begin{cases} d_{a,b}(i-1, j) + 1 \\ d_{a,b}(i, j-1) + 1 \\ d_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & *3 \end{cases} \quad (\text{A.1})$$

\*1 : if  $\min(i, j) = 0$ ,

\*2 : if  $i, j > 1$  and  $a_i = b_{j-1}$  and  $a_{i-1} = b_j$

\*3 : otherwise.

where  $1_{(a_i \neq b_j)}$  is the indicator function equal to 0 when  $a_i = b_j$  and equal to 1 otherwise.

Each recursive call matches one of the cases covered by the Damerau–Levenshtein distance:

- $d_{a,b}(i-1, j) + 1$  corresponds to a deletion (from a to b).
- $d_{a,b}(i, j-1) + 1$  corresponds to an insertion (from a to b).
- $d_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)}$  corresponds to a match or mismatch, depending on whether the respective symbols are the same.
- $d_{a,b}(i-2, j-2) + 1$  corresponds to a transposition between two successive symbols.

The Damerau–Levenshtein distance between is then given by the function value for full strings:  $d_{a,b}(|a|, |b|)$  where  $i = |a|$  denotes the length of string a and  $j = |b|$  is the length of

b. As proved by.<sup>26</sup>

The performance of the algorithm is:

- Time:  $O(ab)$
- Space:  $O(ab)$
- Backtrace:  $O(a + b)$

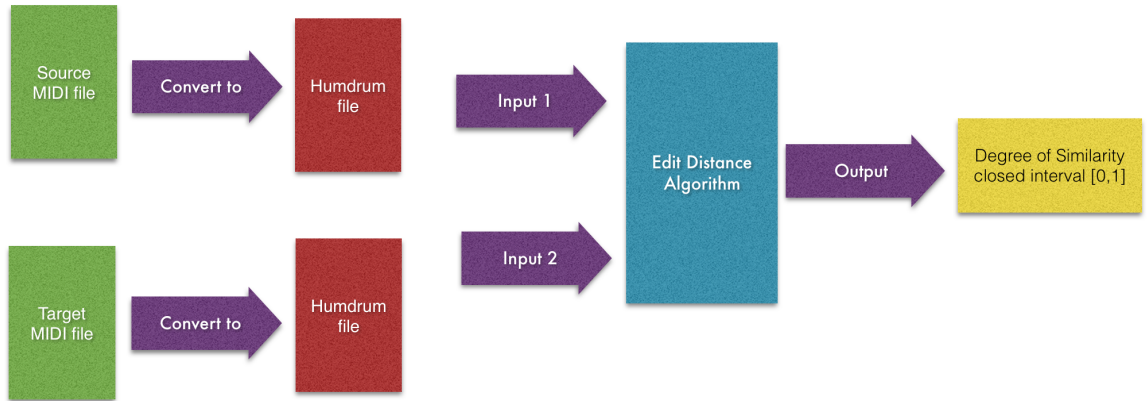


Figure A.1: Input and output of the edit distance algorithm

Using this algorithm, I can compute the similarities of my application's sheet music and the real sheet music. The target file will be a ground truth MIDI file recorded from an electronic piano that supports MIDI transcription. The source which will be the output MIDI file from my application of that same recording. The edit distance will be performed on the source to calculate the minimum distance to transform it into the target. The two inputs for this algorithm will be the source and target MIDI files after they have been converted to humdrum files as shown in Fig. A.1. Depending on the information extracted from the MIDI, Keith S. Orpen and David Huron showed that similarity could be computed for relative pitches, semitone intervals, diatonic intervals, harmonies, dynamics, etc. Using the humdrum application designed by Orpen and Huron, I can convert all note value data tokens into integers. By not limiting yourself to the ASCII table a note represented as C# can correspond to an integer value rather than two separate ASCII values 0x43 and 0x23. In my case I will use the humdrum application to compare a note's value and their duration. For example the

following passage: C#, D, F, D, A, C# can be represented as 1, 2, 3, 2, 4, 1 and can be compared to another passage that uses the same data token conversion to find the similarity based on the edit distance. The result of this edit distance will be a closed interval  $[0, 1]$ . 1 indicating equivalence and 0 indicating no degree of similarity.