William J A Crane

Extracting MIDI Data From Video of a Piano Using Computer Vision

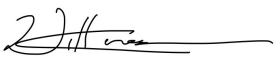
Computer Science Tripos – Part II

King's College

May 7, 2020

Declaration

I, William Crane of King's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: 

Date: 04/05/2020

Proforma

Candidate Number:	**2396C**
Project Title:	**Extracting MIDI Data From Video of a Piano Using Computer Vision**
Examination:	**Computer Science Tripos – Part II, July 2020**
Word Count:	**9,550**[1]
Line Count:	**1,271**[2]
Project Originator:	The Student
Supervisor:	Aamir Mustafa

Original Aims of the Project

To create a system where homophonic piano music can be transcribed to the MIDI file format from video input. To demonstrate the effectiveness of such a system and investigate the limits of automatic music transcription when confined to video data alone.

Work Completed

The system was successfully implemented, achieving an accuracy of 72.0% when applied to a data set of homophonic piano pieces of varying difficulty. System parameters were tuned and a method for comparing MIDI files was implemented. Two different background subtraction models were explored with a Gaussian mixture model method producing the best results in this problem context.

Special Difficulties

The evaluation section was severely compromised by the outbreak of COVID-19. Both the quality and quantity of data suffered to such an extent that my evaluation procedure was forced to change as discussed in Section 4.1

[1]This word count was computed using the Overleaf Word Count feature

[2]This line count includes only the lines present in the final systems. No code from prototypes or figure creation was included in this total

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Automatic music transcription (AMT) is the process by which the pitch[1] and timing of notes in a musical performance can be extracted from data and written down. A successful AMT system would have "clear potential for both economic and societal impact"[3] with uses in education, music indexing and music analysis. However, despite dating back to 1975[13], the accuracies achieved by AMT systems are "still significantly below that of a human expert,"[4] and research in this area "has not converged on a single approach."[3] This dissertation investigates the performance of an AMT system using video as its only input. It explores the advantages, disadvantages and possible uses computer vision techniques may have with respect to the current audio signal processing AMT systems.

I restricted my domain to a piano for three main reasons: First, it is arguably the most popular and available instrument excluding the voice. Secondly, being stationary with distinct black and white keys, the instrument is well suited to the domain of computer vision. Finally, a system built for a piano is likely to be adaptable to other keyboard instruments such as organs or harpsichords. However, the piano may also seem an odd choice. Due to the instrument's popularity and versatility, there exist electric pianos with the ability to transcribe to the MIDI format directly. Why then would you use anything else for this purpose? To answer this we must first understand the mechanics of the piano key.

In a piano, pressing a key triggers a padded hammer head to hit a string or collection of strings as shown in Fig.1.1. This hammer is weighted such that it hits the string in a single stroke whenever the corresponding key is pressed. The pressure and speed of the key press determines the force with which the hammer hits the string. Due to the complexity of this action and the fact that a standard piano has 88 keys, high quality electric pianos are expensive and a relatively new phenomenon[2]. Only the highest quality electric pianos attempt to physically recreate the traditional hammer action and so, from a performer's

[1]The pitch of a note describes its sounding frequency; e.g. the note A4 sounds at 440 Hz.

[2]Although electric pianos without MIDI capabilities date back to the Fender Rhodes in 1946, the MIDI standard itself was not invented until the 1980s

perspective, an electric piano is likely to provide an inferior user experience in comparison to acoustic pianos. Also, almost all electric keyboards attempt to emulate the mechanics of a piano key but other keyboard instruments such as the clavichord and the harpsichord have significantly different key mechanics with electric counterparts either expensive or unavailable. Other attempts to retrofit acoustic pianos with additional transcription hardware have proved to be expensive[3].

With the wide availability of video recording and smart phones, computer vision techniques may lead the way to an inexpensive and friction-less AMT system for keyboard instruments.

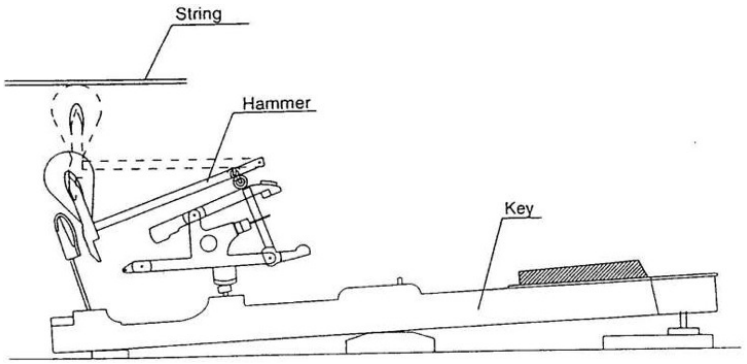


Figure 1.1: The mechanics of a piano key for a grand piano, taken from (The Mechanics of Piano Tuning – Blog, 2015)[7]

1.2 Project Goals

I chose three primary goals for this project.

- To investigate the impressive accuracy claims of existing video AMT systems
- To compare two different methods for background subtraction (discussed further in Section 2.3.8)
- To identify the weaknesses of video AMT systems and explain the dramatic variance in accuracy across varying levels of musical difficulty, as observed in other papers.

[3]The current price for a Silent System™with installation ranges between £1000 and £1500

Chapter 2

Preparation

2.1 Related Work

2.1.1 Audio Signal Processing

A wide variety of different audio signal processing methods have been applied to AMT.

In 2013, Fuentes, Badeau & Richard[2] applied non-negative matrix factorisation to achieve an accuracy[1] of 31.3%. This may seem low but due to their success relative to similar papers, the results was classed as "very promising." In 2015, Su & Yang[17] introduced the combined frequency and periodicity (CFP) approach which achieved an accuracy of 68.67% on the MAPS[8] data set. In 2016 Sigtia, Benetos & Dixon[16] applied an end-to-end neural network which achieved a very impressive accuracy of 74.45% on the same MAPS data set.

Despite the impressive leaps in accuracy, several key challenges remain, especially in the application of more traditional signal processing methods. One such difficulty is found in detecting octaves: When two notes an octave apart are played, the frequencies of the higher note are contained within those already produced by the lower note. Detecting whether one or two notes are playing in this scenario is challenging.

2.1.2 Computer Vision

Compared with audio signal processing methods, computer vision techniques have only recently been considered for AMT, with the earliest papers by Gorodnichy & Yogeswaran[10] in 2006 and Zhang et al.[21] in 2007 both focusing on fingering[2].

In 2015, Akbari & Cheng[1] introduced the first method that demonstrated that computer vision can reliably provide higher accuracies for automatic piano transcription, with their claVision system achieving an accuracy over 95%. Other recent methods have failed to

[1] Accuracy refers to F0 or F1 score within this section. A definition of F1 score can be found in Section 4.3

[2] In music, fingering refers to the choice of finger to play a note

replicate this impressive result with accuracies ranging between 50%[18] and 86%[12]. One of the main limitations of the aforementioned works is the lack of a standardised data set for evaluation; the link to the data set used by Akbari & Cheng has expired and there are no signs that the claVision software has ever been available. This means that comparisons between results from different papers are severely, if not fatally limited.

2.2 Development Strategy

Creating a successful AMT system relies on managing the intrinsic complexity of such a large project. Therefore, it was paramount that I considered high level strategies and took a structured approach to development. The most important concept throughout development was that of decomposition. The system would be broken down into independent stages, with independent processes within them. This allowed each individual method to be unit tested such that bugs were quickly identified and fixed.

The strategy I chose took inspiration from the spiral model. I would initially focus on a fully working prototype, implementing only a single method for background subtraction. Integration testing and full system tests on this prototype would highlight the poorest performing parts of the system and inform the development plan for the next system. The system would then be re-built using this new plan. With this second system, modules would be improved iteratively. This approach would ensure that the time spent on each component of the project would be proportional to their contribution to system performance.

2.2.1 Requirements Analysis

I broke down the project into six modules:

- Keyboard Location
- Piano key detection
- Creating a mask of the pianist's hands
- Detecting key presses
- Converting key press data to the MIDI file format
- Comparing two MIDI files

Each module was further broken down into the computer-vision techniques that might be useful for their implementation. Many techniques would be applicable to multiple modules and require careful consideration of parameters. These techniques are discussed further in Section 2.3.

[2]This first prototype is not discussed in the implementation section. It had no file writing capacities, ran in real-time and the system evaluation was performed by inspection of the system as it ran. The real-time capabilities were dropped when the system architecture was re-worked

2.2.2 Constraints

Achieving invariance to environmental factors such as lighting is often a difficult task for any computer vision application. For example, an image of a white piece of paper under green light may be identical to that of a green piece of paper under white light. Given my lack of familiarity with computer vision techniques I decided to impose certain constraints on the system input:

- The pose of the camera relative to the keyboard must remain constant throughout each video
- The camera must be positioned at the right side of the keyboard
- A single, complete, 88-key keyboard must be within the frame of the video
- The pose of the camera relative to the keyboard must be such that no key fully obscures the top third of any other key
- The lighting conditions for each video must remain as constant as possible throughout the entirety of the video

One common constraint for AMT is that the music be monophonic, i.e. there may only be a single note played at any one time. This subset of AMT is well suited to audio signal processing methods and is widely considered a solved problem, especially for instrument specific systems. Instead, I chose to design for homophonic music where the number of notes present at any time lies within the range [0,88].

2.3 Theory

This section describes the preliminary research required for the implementation of the system modules. The basics of piano technique informs the high-level system design, while the MIDI standard determines the format in which we store and read musical information. A variety of computer vision concepts are then discussed. The HSV colour model provides use to the skin detection method of the hand masking module. Kernel convolution is a fundamental operation and is required for the Canny edge detector. Edge and line detection are both used for keyboard recognition. Finally, background subtraction and contour finding are the primary methods used for key press detection.

2.3.1 Piano Technique

The nuances of piano technique remain a point of discussion amongst high-level pianists, however the fundamentals remain constant. The performer's hands are raised slightly above the keyboard, palms facing down in a position such that the fingers can comfortably press

down keys as shown in Fig. 2.1. The hands can move left and right in order to reach keys with higher or lower note values. Due to the hammer action of the keyboard, the force required to press a given key increases the higher up[3] the key is pressed. This means that the majority of key presses occur lower down the keys, allowing the hands to rest towards the bottom of the keyboard.

Figure 2.1: A standard hand position for piano playing

2.3.2 The MIDI standard

Musical Instrument Digital Interface[20] (MIDI) is a widely adopted standard for sending and storing digital instrument information. Although other forms of musical expression may be captured by MIDI, this project requires only its ability to notate the note values and timings of a performance. MIDI information is packaged into MIDI events and these events are stored with their timestamps into MIDI files. The MIDI file format is incredibly versatile and can be transcribed to sheet music by tools such as MuseScore[15].

2.3.3 The HSV Colour Model

The choice of colour model determines how colour information for each pixel is described. The most common model is the RGB colour model where each pixel has three integers representing that pixel's red, green and blue components. A pixel's colour is the summation of these three components. One of RGB's disadvantages is that it is not an intuitive representation from a human perspective; for example, given RGB pixel values of (74,32,77) and (203,180,205), it's hard to tell that they are both shades of purple.

The HSV colour model attempts to solve this problem by representing colour as a combination of hue, saturation and value. The hue of a pixel corresponds to the perceived true

[3]When referring to the keyboard, the terms 'lower' and 'higher' are described relative to the orientation given in Fig. 2.1

colour of that pixel; saturation and value represent intensity and brightness respectively. In this model the purple pixels would be represented by the HSV triples (296,58,30) and (296,12,80). From this we can conclude that both have the same colour but the second pixel is less saturated and brighter than the first.

2.3.4 Kernel Convolution

Kernel convolution is a matrix operation used for a wide variety of image transformations such as blur, sharpening and line detection. We first define a small matrix of weights to be our kernel matrix. To then convolve this kernel with an image, for each cell in the image matrix[4] we centre our kernel on that cell and take the kernel weighted sum of the surrounding cells. The equation describing this process is given below with I, G and K corresponding to the input, output and kernel matrices:

$$G(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x+i, Y+j)K(i,j) \tag{2.1}$$

A visual example of this operation for a single cell is given in Fig. 2.2.

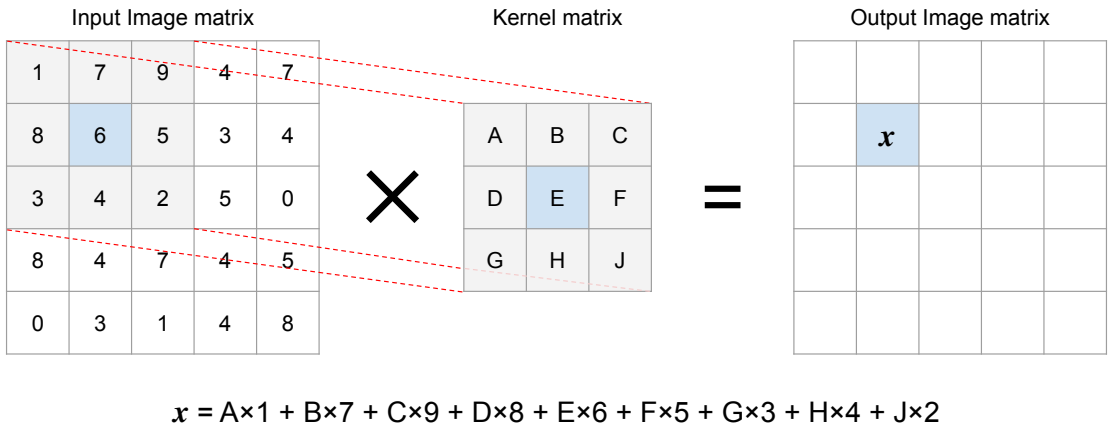


Figure 2.2: Kernel Convolution for a single cell

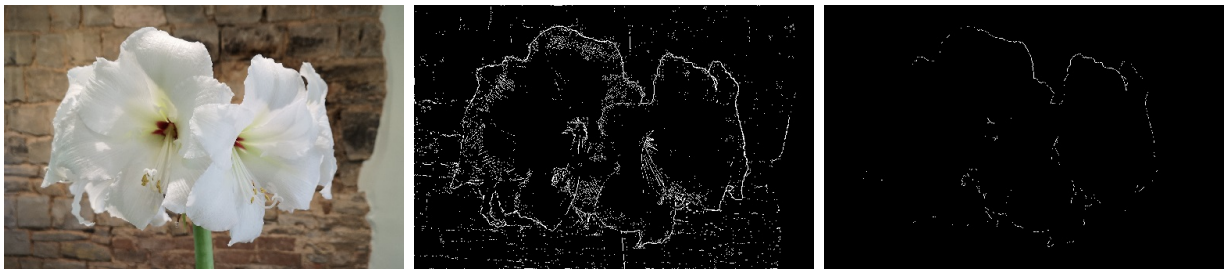
A very useful attribute of kernel convolution is that it is associative. This means we can combine multiple kernels, each with different operations into a single kernel that performs all the individual operations in a single convolution.

2.3.5 Canny Edge Detector

A common problem with simple edge detectors is the abundance of unwanted edges, especially in noisy images. For example, Laplacian edge detection (achieved through kernel

[4]An image can be considered a matrix of pixels: Each matrix cell is a pixel value or triple of values.

convolution) requires a threshold[5] of possible edges to be taken. If set too low, complicated textures or details can generate multiple unwanted edges. If set too high, we lose edges that we would want to be included. This can be seen in Fig.2.3.



(a) Input Image (b) Low threshold (c) High threshold

Figure 2.3: Thresholded Laplacian edge detection

The Canny edge detector mitigates this issue by introducing additional filtering steps for our edges, producing better image segmentations that are far closer to those a human might produce. The Canny edge detector has five stages:

1. Noise reduction: Blur the image using a Gaussian kernel to remove noise
2. Gradient calculation: Find edges by using a Laplacian kernel as in the above method
3. Edge-thinning: Thin the generated edges to a width of one pixel using non-maximum suppression
4. Double threshold: Classify each edge as strong, weak or suppressed using edge intensity data (generated by the Laplacian convolution)
5. Constrain edges: Link together strong edges and remove unconnected weak edges

An example output from the Canny edge detector can be seen in Fig. 2.4.

2.3.6 Hough Line Transform

Hough Space for lines

In Cartesian space we might be used to representing a straight line as $y = a \cdot x + b$. However, this representation is flawed as there is no choice of a and b that can describe vertical lines[6]. Consider instead the following representation for a straight line:

$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

[5]Unless otherwise specified, all mention of thresholding in this paper refers to global thresholding where each pixel is set to 0 or 1 depending on if it is lower or higher than the threshold value

[6]For computational reasons we do not consider the case where a may be unbounded

(a) Input Image (b) Output segmentation

Figure 2.4: Canny Edge Detection

r corresponds to the perpendicular distance from the line to the origin and θ is the angle to the positive x-axis made by this perpendicular line. This representation is superior as it can represent any straight line (in 2D space)

Hough space uses the parameters r and θ as its axes and so every point in Hough space corresponds to a line in Cartesian space. For example, the point (2,0) in Hough space corresponds to the line $x = 2$ in Cartesian space.

Hough Space Line Detection

Given a binary image[7] the Hough Line Transform detects straight lines using two steps. First, we iterate through every positive point (x,y) within the binary image and draw, in Hough Space, the set of all straight lines that might pass through that point. Performing this process in an additive fashion results in every point (r,θ) in Hough Space having a value equal to the number of positive points (x,y) through which the line with parameters (r,θ) may pass. We then take a threshold of the image we have built in Hough Space to return the r and θ values of the most probable lines.

2.3.7 Contours

A contour is a way of describing the bounding region of a shape. It is stored as a sequence of points through which the shape's boundary line passes. To efficiently find the contours within a binary image we use the border following algorithm presented in Suzuki & Be (1985)[19].

2.3.8 Background Subtraction

Background subtraction is a technique which separates the foreground and background in video from a stationary camera. The primary assumption of this method is that the background is static and the foreground is moving. Using this assumption, a model of the video's

[7] A binary image has pixel values of 0 and 1 only

background is maintained, which we subtract from each frame to separate out the new foreground elements. An example use case might be detecting movement in a security feed.

For each type of background subtraction we must define three things:

- Background Model: This stores our current estimate of the background of the video
- Update function: This defines how we update our background model when we receive a new frame of the video
- Subtraction function: Given a new frame and the background model, this produces a difference image

A difference image produced by background subtraction acts like a heat map for foreground objects. The higher the value of a pixel in the difference image, the more likely that pixel is part of the foreground. We threshold this difference image to remove noise and provide a mask of the foreground.

Median Background Model

The Median Background Model is one of the simplest forms of background subtraction. At time t we define the background model as the pixel-by-pixel median image of the frames from time $(t - k)$ to t. We achieve this by storing all frames within this time window k along with the median image of these frames. Our update function therefore takes in a new frame, discards any frame(s)[8] that now fall outside of the time window and recalculates the median background image. Our subtraction function is a literal pixel-wise subtraction of the background model from the current frame.

As we have a simple subtraction, two difference images are produced: a positive difference image which describes foreground objects brighter than the background they cover and a negative difference image which describes foreground objects darker than the background they cover. It is usual to add these into a single difference image where this distinction is not needed. An example of the median background model for a grey-scale camera feed can be seen in Fig.2.5.

Our choice of window size k represents a trade-off between memory and accuracy; If too small, we risk slower moving foreground objects becoming part of our background image. However, the larger the window, the more frames we have to store and average for each new frame. An example of the effect of window size k can be seen in Fig.2.6

Gaussian Mixture Model

The application of Gaussian mixture models for background subtraction was first introduced by Friedman & Russell in 1997[9]. Instead of building an average background image as before, we model each pixel as a distribution described by a mixture of Gaussian functions. Therefore

[8]Multiple frames may be discarded in the case of a variable frame rate

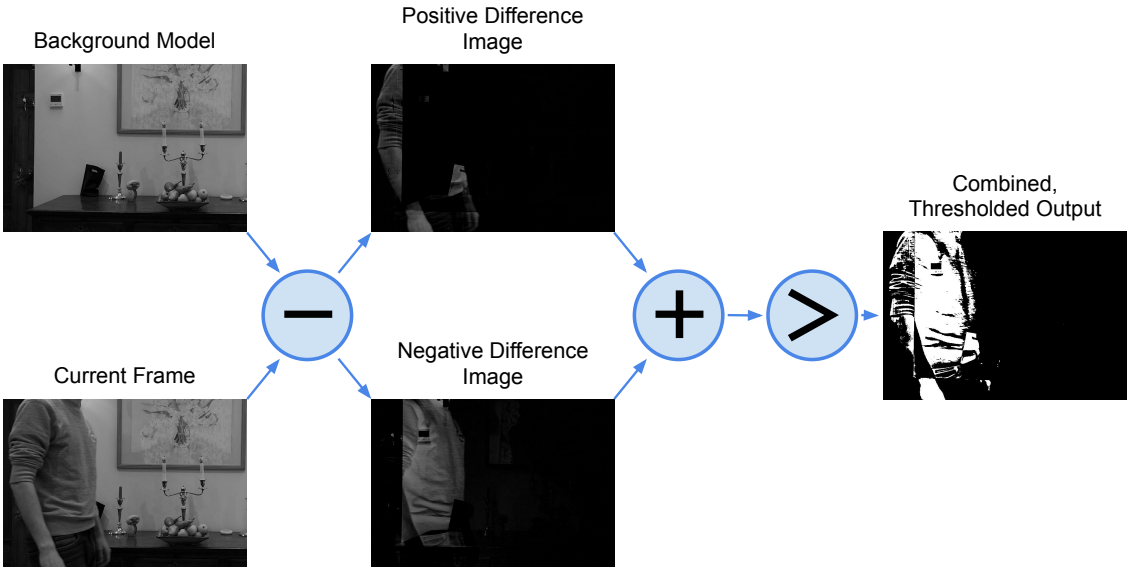
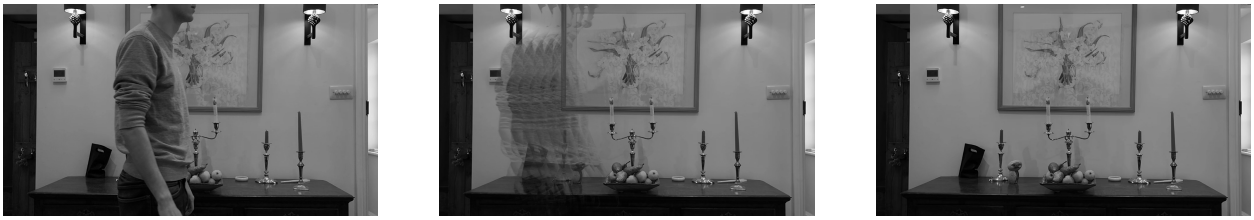


Figure 2.5: Background Subtraction using the Median Background Model



(a) Current Frame

(b) Background Model with small k

(c) Background Model with large k

Figure 2.6: The effect of window size on the Median Background Model

for each pixel, we maintain a mixture of M Gaussians each with mean μ, variance σ^2 and mixing probability[9] π. Each Gaussian represents a clustering of previous pixel values and so corresponds to an object that was at that pixel for a period of time proportional to π. We assume that the largest Gaussian for a given pixel distribution is a result of the background value for that pixel. For example if we have a dark background with a few brighter objects passing in front of it, the mixture should have a single large Gaussian centred around the dark background colour, with a few Gaussians centred around brighter colours.

Therefore the subtraction function is a pixel-wise decision where we find the Gaussian from the mixture that maximises the likelihood of the observed pixel value. If this belongs to the largest Gaussian, it is part of the background, else it is part of the foreground.

An additional problem when using Gaussian Mixture Models is the choice of M (the number of Gaussian functions per pixel). We must choose a value such that each different foreground object is represented by a different Gaussian or risk different foreground objects

[9]The mixing probability of a Gaussian refers to its size/influence on the overall mixture

contributing to the same, overly-large Gaussian. Fortunately, M can be calculated per-pixel and simultaneously with each update using the recursive equations presented in Zivkovic & van der Heijden [22] in such a way that mitigates for this scenario.

The Gaussian nature of this model means it reacts well to noise and the efficiency of this model allows us to increase our window size k without requiring a drastic increase in memory or computation.

2.4 Resources

I chose to use Python and OpenCV[6] for this project due to the wealth of resources available online for this language and library combination. Additionally, OpenCV is a widely used and well documented library with active development and commercial use. OpenCV has a 3-clause BSD license, permitting the use of the software for this project.

2.5 Starting Point

Before this project I had no experience in any areas of computer vision and had not previously used Python outside of the scientific computing course from Part IA. This project was not based on any pre-existing code, however the methods from Akbari & Cheng for claVision provided a rough guide to the implementation of certain modules.

Chapter 3

Implementation

3.1 System Architecture

I divided system execution into three distinct stages as described in Fig.3.1. This decomposition of processes and high level structure allowed for an implementation with unit testable components. The separate initialisation and write stages also facilitated more accurate but expensive methods to be used within them as they are executed only once.

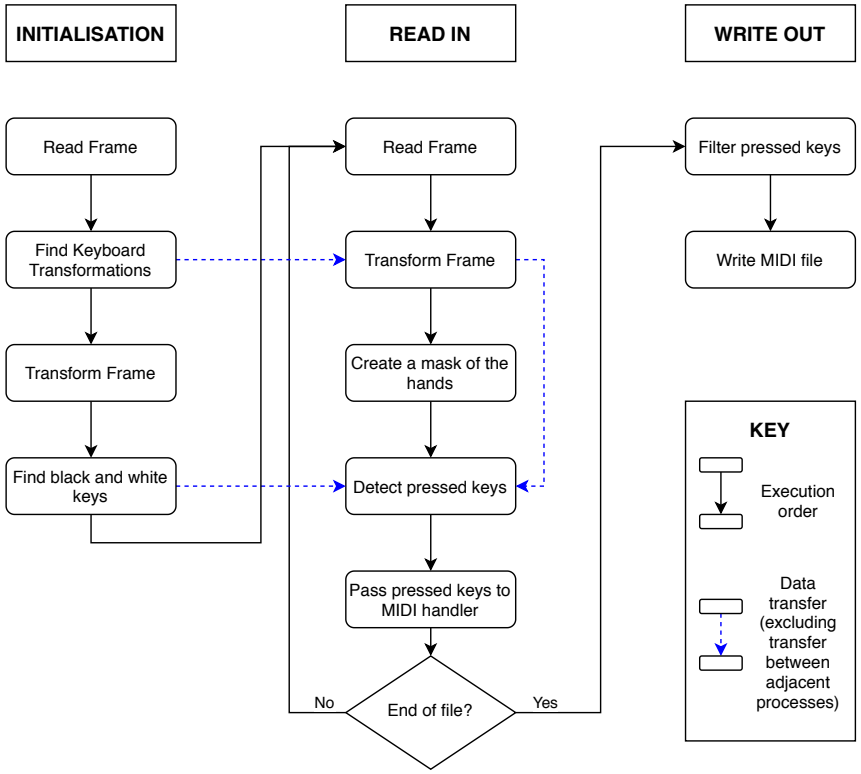


Figure 3.1: System execution flowchart
The structure of this flowchart is reflected in the 'Video_to_MIDI' function

3.2 Locating and Transforming the Keyboard

Problem Statement: Find the keyboard within a given frame of a video. Transform to crop in on the keyboard and correct for camera perspective. We assume that the keyboard is complete and in no way obscured in the given frame.

3.2.1 Bounding the keyboard region

The full-sized piano keyboard is a standardised black and white rectangle as shown in Fig. 3.2. However, due to the perspective of the camera, the keyboard region within the frame will form an irregular quadrilateral. To locate this quadrilateral we observe two structural properties of the keyboard as in Akbari & Cheng. First, that the bottom third of the keyboard will have a greater average brightness than both the upper and middle thirds due to the position of the black keys. Second that the keyboard contains 36 black keys. These two properties will form the basis of our keyboard detection heuristic.

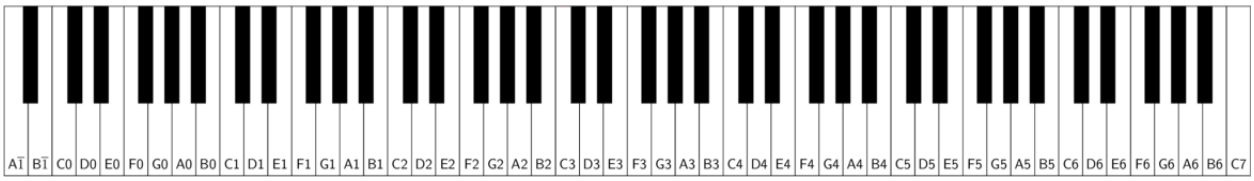


Figure 3.2: The structure of a regular 88-key piano keyboard

We locate the minimum upper and lower bounding lines of the keyboard region using a 5-step method:

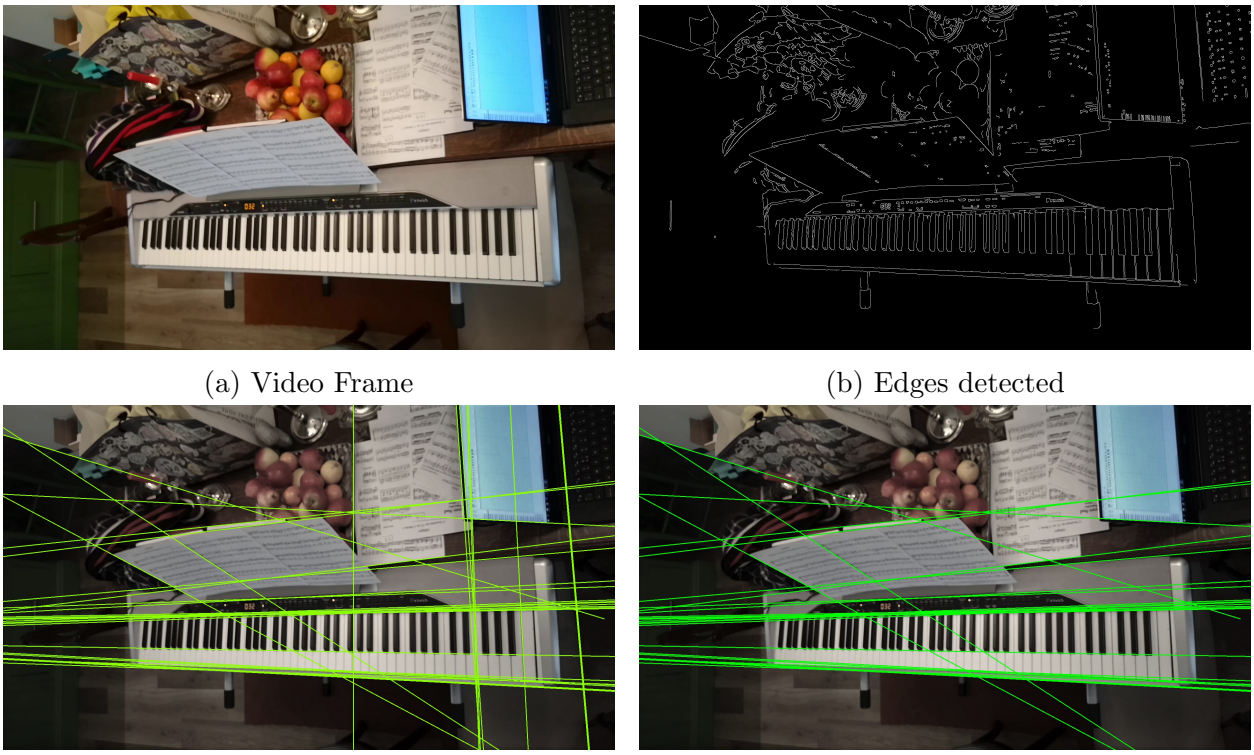
1. Find lines within the frame

The Hough Line transform is used to detect straight lines within the frame as shown in Fig. 3.3c. As Hough line detection requires a black and white edge image, we first apply the Canny edge detector on a grey-scale copy of the frame. OpenCV provides an implementation for both Hough line detection and Canny edge detection, however the threshold for Hough Line detection and the thresholds for edge classification had to be considered.

Varying lighting conditions impacted the number of lines detected and thresholds set too high risked missing the lines we wanted (the top and bottom of the keyboard). As keyboard detection is run only once for each file, it was a worthwhile performance trade off to set lower thresholds that reliably found these lines, despite an increase in unwanted lines.

2. Using our constraints, remove lines which cannot be the top or bottom of the keyboard

As we have constrained the pose of our camera, we are interested only in lines that are near-horizontal; within 45° of the x-axis. We can therefore use θ[1] (the angle to the x-axis made by the perpendicular from the line to the origin) to filter out lines which we need not consider as shown in Fig. 3.3d.



(a) Video Frame (b) Edges detected

(c) Detected lines (d) Filtered lines

Figure 3.3: Hough Line detection and filtering

3. Pair possible keyboard bounding lines

We want to produce pairs of lines that may be along the top and bottom of the keyboard. Therefore, we pair together all lines with similar gradients. This process can be done easily by directly comparing the θ values of each line. These pairings are also filtered to remove lines which are too close together such that a keyboard region could not be contained within them.

[1]Recall that Hough Line detection returns lines in the form (r,θ)

4. Filter pairings using a weak heuristic

For each pair of lines, we want to determine if a valid keyboard region is contained within them. We start with a fast but weak test using one of the previously observed properties of the keyboard, namely that the bottom third is brighter than both the middle and top thirds due to the presence of black keys.

Taking the average pixel brightness of each third is easy to achieve but, due to the perspective of the camera, the keyboard region will be an irregular quadrilateral and the top and bottom of the keyboard will not be parallel. We must therefore transform the frame such that each pair of lines is parallel and the keyboard region to be tested is a trapezium as shown in Fig.3.4b. The calculation of the transformation matrix to achieve such an effect is discussed in Section 3.2.3.

5. Filter pairings using a strong heuristic

If a keyboard region meets the brightness property of the weak heuristic we then further test the region to determine whether the keyboard region contains exactly 36 black keys. Using the transformed frame, we search for black key contours within the region using an iterative threshold method, described in Section 3.3.1. We further constrain the contours, making use of their roughly equal size, aspect ratio and bottom-most points to filter out any contours that may have appeared outside the true keyboard region. If we find 36 black keys between a given pair of lines, we know both that the keyboard is contained within the lines and the position of the black keys within the region. An example of successfully found black keys within the transformed frame can be seen in Fig. 3.4c

3.2.2 Detect the four corners of the keyboard

Given the rough[2] bounding lines of the keyboard along with the black key contours we now search for the keyboard corners such that we can remove the perspective warping of the camera and fully crop in on the keyboard.

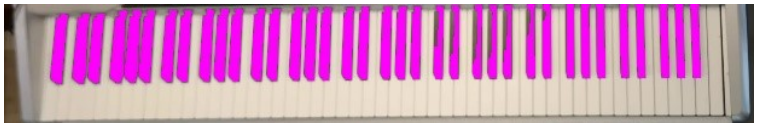
We start by transforming our found black key contours to the coordinate system of the full frame. We next draw a pure white line from the leftmost point of the leftmost key to slightly beyond the centre of the rightmost key such that the outer white keys ($A\bar{1}$ and C7) are connected by the white line as in Fig.3.5a. To ensure that C7 is reached, the distance of the line drawn beyond the rightmost black key is four thirds of the distance between the two rightmost black key centres.

With all the white keys in the frame connected by this line, we iteratively threshold (as described in Section 3.3.1) until we find the contour within the image that corresponds to the connected white key mass as shown in Fig.3.5b. We filter out any other unwanted contours using the fact that the centre of this contour must lie roughly within the centre of our black

[2]The resolution of Hough space for our Hough Line Detection introduces a small inaccuracy for our bounding lines

(a) Bounding lines

(b) Transformed frame



(c) Black Keys detected

Figure 3.4: Testing a keyboard region using the strong heuristic

key contours and that we can define upper and lower limits for it's area based on the sum of black key areas.

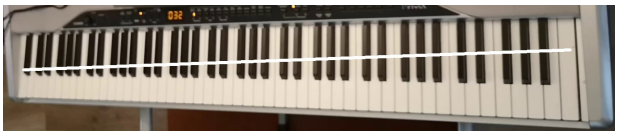
Finally, with the mass of white key contours found we take the points of this contour closest to the contour's bounding box corners as the keyboard corners as shown in Fig.3.5c

3.2.3 Frame Transformation

We want to create a perspective transformation matrix such that the keyboard region is seen in parallel perspective[3]. We consider our camera and image as a vector and plane in 3D space respectively. The perspective transform we require describes a projection from a point such that each of the four corners specified within our plane form a rectangle within the projection. Each of these four corners constrain our solution matrix and so we solve four simultaneous equations to calculate our perspective transformation matrix.

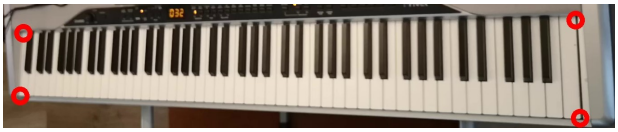
The perspective transformation matrix was calculated using a built in OpenCV function, which takes in four image coordinates and the required rectangle dimensions to produce a projected image in parallel perspective, an example of which can be seen in Fig.3.5d.

[3]Often used in CAD, parallel lines within world space must be parallel within a parallel perspective image



(a) White Line connecting A$\bar{1}$ and C7

(b) Contour for white keys



(c) Keyboard corners

(d) Transformed frame

Figure 3.5: Keyboard corner detection and perspective transformation

3.2.4 Method Limitations

This method successfully found the keyboard for every video in the training set however a slight inaccuracy in the keyboard corners is introduced when the electric piano with silver plastic sides was used (as in Fig3.5). The silver plastic under bright lighting has a similar colour to the white keys and so caused parts of the bottom lip of the piano to be incorporated into the white key contour. This meant that the bottom left corner was sometimes detected too low and so the transformed frame becomes warped towards the left hand side of the keyboard as in Fig.3.5d.

As the application of this system is meant for acoustic[4] pianos whose bodies are usually made of a brown or black finished wood, this specific inaccuracy did not seem worth pursuing. Instead, a simple GUI for manually selecting and storing keyboard corners was created for testing purposes (N.B. it was not used in the production of any evaluation data).

3.3 Key Detection

Problem Statement: Given a parallel perspective image of a full piano keyboard, produce contours for each individual key.

3.3.1 Detecting Black Keys

The primary method used for finding black keys is to take a binary threshold of the image then find contours within this image. Two different forms of thresholding were considered (iterative and Gaussian), but due to the presence of noise and reflections of light on the top of the keys, additional steps were necessary to produce accurate black key contours.

[4]An acoustic instrument has no electric component

Iterative Threshold

Given a threshold value t, the inverse[5] binary threshold B of an 8-bit image I is described:

$$B(x,y) = \left\{ \begin{array}{ll} 255, & \text{for } I(x,y) < t \\ 0, & \text{otherwise} \end{array} \right\}$$

Choosing a constant value for t is imprecise due to the variety in lighting conditions between videos. Therefore to find the optimal t for a given video we define a function to measure the success of a threshold and then iterate through values of t in the range $(0, 255)$ to find that which maximises our success function.

We require the set of black key contours C_t (produced by 3.3.1) as the input to our black key success function:

$$success(C_t) = \left\{ \begin{array}{ll} \sum\limits_{c \in C_t} \text{area}(c), & \text{if } |C_t| = 36 \\ 0, & \text{otherwise} \end{array} \right\}$$

Assuming that our contour filtering removes all contours not covering black keys, this success function ensures that we choose a threshold such that the black key contours produced are maximal and cover as much of the true key space as possible. An example output from this process can be seen in Fig.3.6.

(a) Parallel perspective image

(b) Optimal inverse binary threshold

(c) Black Keys detected

Figure 3.6: Black key detection using iterative thresholding

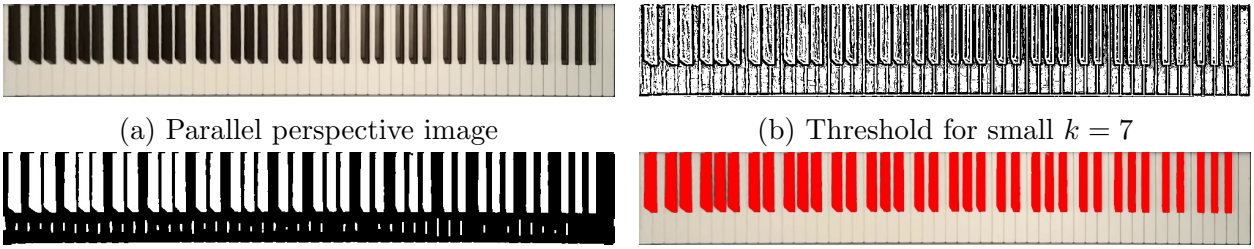
The reflections on the top of black keys can produce valid black key regions that are not covered by the black key contours found as shown by the upper middle keys of Fig.3.6c. This is a limit of global thresholding as higher thresholds cause lower keys to connect, thus causing multiple lower keys to be covered by a single contour which is then filtered out.

[5]As we are searching for black objects, we take the inverse threshold

Gaussian Threshold

Gaussian thresholding is a form of adaptive thresholding that requires no specific threshold value and promises to be able to adapt to varied illumination across an image. A different threshold value is calculated for every pixel using a Gaussian weighted average of the pixels within a k by k bounding box. The threshold values across the image can be generated by convolution of the image with a Gaussian kernel of size k^2.

Choosing a small value of k creates a threshold that is highly sensitive to lighting changes so that only the transitional edges of objects are found, as shown in Fig.3.7b. As we want solid contours of the entire keys, a large value of k was chosen, resulting in the contours of Fig.3.7d. The same contour filtering method is applied as before.



(a) Parallel perspective image

(b) Threshold for small $k = 7$

(c) Threshold for $k = 71$

(d) Black Keys detected

Figure 3.7: Black key detection using Gaussian thresholding[6]

This method is more efficient and more accurate than the iterative thresholding method and was used for key detection. However in Section 3.2 we do not know the size of the keyboard and therefore cannot choose an appropriate k value. Therefore, Gaussian thresholding is used only for these key-finding applications (all other applications use the iterative thresholding method).

Finding and Filtering Contours

Both methods of thresholding produce threshold images with a degree of noise. The Gaussian threshold in Fig.3.7c picks up some of the lines between white keys. We must therefore filter the contours found within these threshold images to ensure we only consider the contours of black keys. Three black key properties are used:

1. The bottom-most part of a black key lies approximately one third of the image height from the bottom of the keyboard

2. The aspect ratio of a black key, accounting for the perspective effects at the left-hand side of the keyboard, must lie between 3:1 and 10:1

3. The black keys are all of approximately equal size

[6] An additional erosion step (discussed in Section 3.4.2) was used on the threshold images to prevent the darker lines between white keys from becoming connected to the black key contours

3.3.2 Detecting White Keys

Because of the faint and thin nature of the lines of separation between white keys, we are unable to simply threshold then filter contours to find our white keys. Instead we must first identify and infer the position of the lines of separation between white keys. Once this is achieved, we can draw both these separating lines and the black key contours in pure black on our image, then threshold and find contours as before.

The simplest way to find the separating lines would be to draw 51 equally spaced vertical lines but this heavily relies on the accuracy of the keyboard recognition method which, as previously discussed, varies slightly from piano to piano. Another method might be to infer the white key positions from the black keys. Unfortunately, when this was implemented, the variety in black key shape caused by parallax towards the left hand side of the keyboard resulted in significant inaccuracies.

Therefore, to ensure both accuracy and generalisability we find the white key separation lines using the following 4 steps:

1. Image Sharpening

To increase the contrast between the grey pixels separating white keys and the white of the keys we sharpen the image by convolution with the following kernel:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

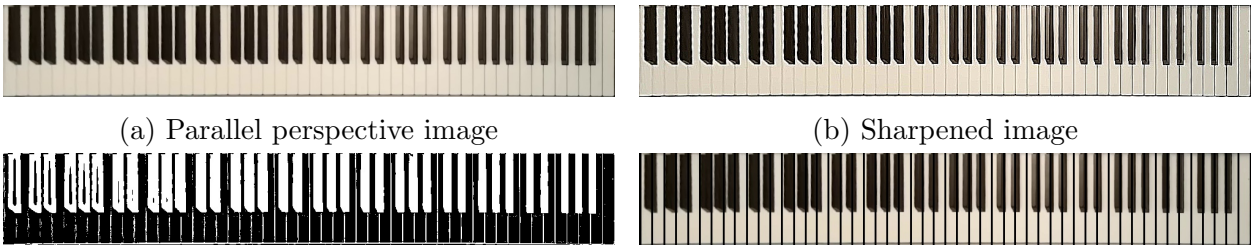
The result of this sharpening is seen in Fig.3.8b.

2. Gaussian Threshold

Fig.3.7c demonstrates that the lines separating white keys can be detected by an adaptive threshold with a large bounding box. For this application, $k = 31$ was used alongside an additional threshold offset of 15 to produce the threshold image in Fig.3.8c.

3. Hough Line Detection

Hough line detection applied to this threshold image produces many unwanted horizontal lines. This is because the threshold of Hough space must be set very low in order to detect the thinner vertical lines. Additionally, the black keys introduce more unwanted vertical lines. Therefore, Hough line detection is applied only to the bottom third of the threshold image and the resolution of θ in Hough space is set to π such that only vertical lines are considered. The resulting lines can be seen in Fig3.8d.



(a) Parallel perspective image

(b) Sharpened image

(c) Gaussian threshold ($k = 31$,offset= 15)

(d) Hough Lines from the bottom third of the threshold image

Figure 3.8: White key separating line detection

4. Filter and Infer separation lines

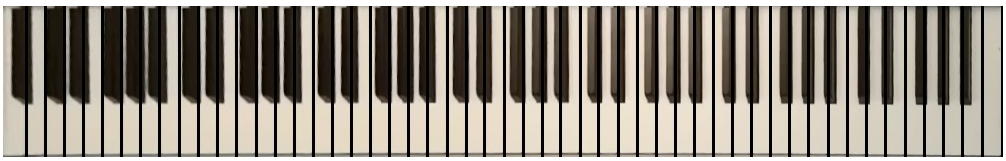
In testing, Hough line detection reliably produced all the lines of separation between white keys. However variety between pianos, lesser resolutions or extreme lighting conditions may cause some lines to be undetected or to be detected multiple times as in Fig.3.9a. To fix this, we calculate the median line separation (avg_dist), sort our input lines ($Lines_{in}$) from left to right and apply the method described in Algorithm 1.

Algorithm 1 White key line processing

$Lines_{out} \leftarrow empty\_list$
$prev\_line \leftarrow (0, 0)$
for i from 0 to len($Lines_{in}$) **do**
    $(\rho_i, \theta_i) \leftarrow Lines_{in}[i]$
    $x_{increase} \leftarrow (\rho_i - prev\_line)$
    if $0.8 \cdot avg\_dist < x_{increase} < 1.2 \cdot avg\_dist$ **then**
        append (ρ_i, θ_i) onto $Lines_{out}$
        $prev\_line \leftarrow (\rho_i, \theta_i)$
    else if $x_{increase} > 1.8 \cdot avg\_dist$ **then**
        append (ρ_i, θ_i) onto $Lines_{out}$
        for new_line from 0 to $\lfloor \frac{x_{increase}}{avg\_dist} \rceil - 1$ **do**
            $\rho_{new\_line} \leftarrow \rho_i - \lfloor avg\_dist \cdot (new\_line + 1) \rceil$
            append $(\rho_{new\_line}, 0)$ onto $Lines_{out}$
            $prev\_line \leftarrow (\rho_{new\_line}, 0)$
        end for
    end if
end for

The result of this processing can be see in in Fig.3.9 where the Gaussian and Hough line thresholds were deliberately set poorly to simulate adverse conditions.

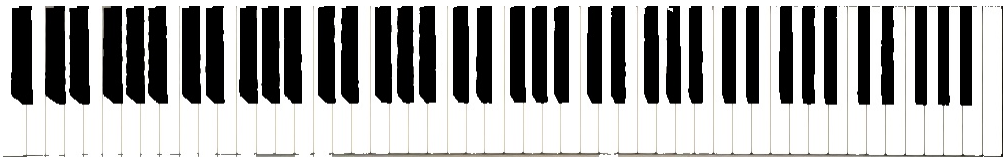
(a) Input



(b) Output

Figure 3.9: White key line processing

(a) Parallel perspective image



(b) Black and White key contours

Figure 3.10: Key detection

3.4 Hand Masking

Problem Statement: For a transformed frame where only the keyboard region is visible, identify all pixels that belong to the hands of the keyboard player.

3.4.1 Skin Detection

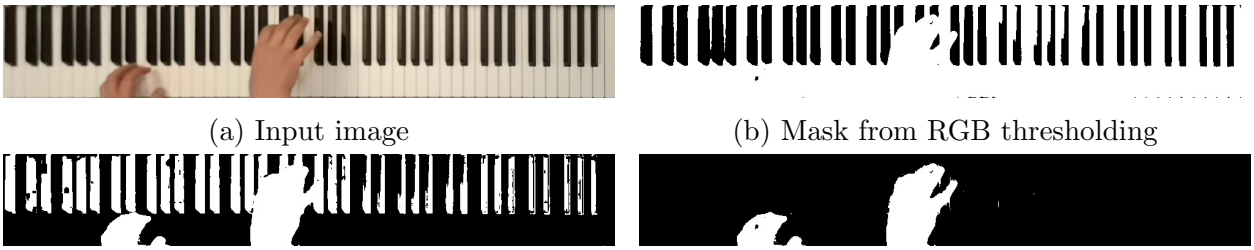
There are many varied approaches for skin detection but given that our method is required to run every frame, efficiency and simplicity were prioritised. Kolkur et al.[11] demonstrate impressive results from a simple, threshold-based algorithm applied across different colour models. Therefore, to detect skin pixels we individually threshold each of the R,G and B components of the image, convert to the HSV colour model and threshold the H and S components. Each threshold is a weak classifier of skin pixels, however their combination (using a pixel-wise logical AND operator) provides a far more accurate classification as shown in Fig.3.11

Given the additional knowledge that the only other parts of the image must be part of the

keyboard, the threshold values chosen are different to those of Kolkur et al. For performance reasons, we also introduce fewer constraints:

$$\begin{cases} \text{RGB} & (R > 100) \wedge (G > 50) \wedge (B > 40) \\ \text{HSV} & (H > 20) \wedge (S > 83) \end{cases}$$

Where R,G,B are 8-bit integers, H lies within $[0, 179]$ and S within $[0, 255]$.



(a) Input image

(b) Mask from RGB thresholding

(c) Mask from HSV thresholding

(d) Combined mask

Figure 3.11: Threshold method for skin detection

3.4.2 Morphological Transformations

Despite its accuracy, some noise remains within the combined mask. To remove this we perform two morphological transformations: an opening followed by a closing. These are composed from two fundamental operations: erosions and dilations.

Erosions and Dilations

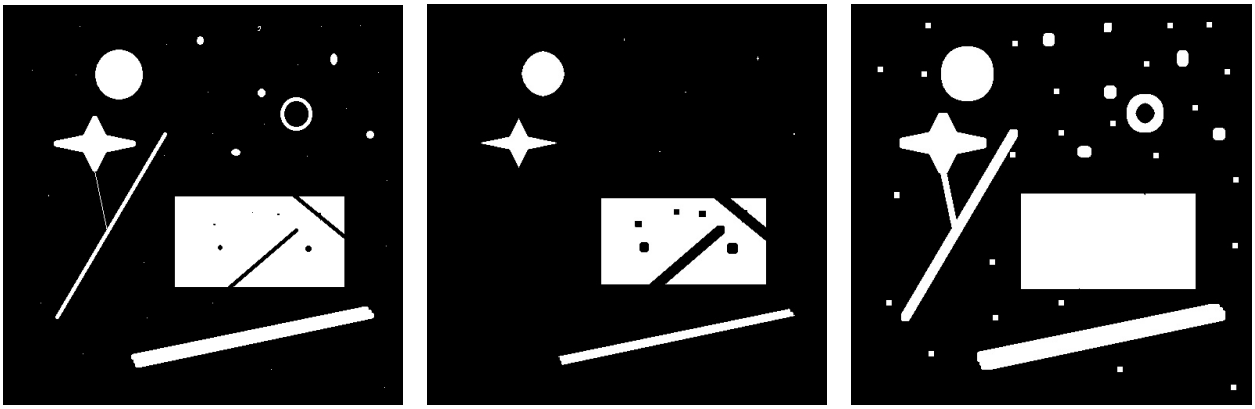
As the name suggests, an erosion eats away at the boundary of objects in a binary image. It can be achieved through a modified kernel convolution where, for a given pixel, we take the logical AND of surrounding pixels instead of a weighted summation.

A dilation is the opposite of an erosion; the boundary of objects within a binary image are expanded. The associated kernel convolution uses a logical OR of the surrounding pixels to achieve this. Fig.3.12 shows both an erosion and a dilation operation.

Openings and Closings

An opening is an erosion followed by a dilation. It can be used to remove small positive objects without changing the size of larger objects.

A closing is a dilation followed by an erosion. It removes small holes within objects without changing their size. Fig.3.13 shows the effect of both transformations and their combination when applied to the hand mask from Fig.3.11d.



(a) Input Image (b) Erosion (c) Dilation

Figure 3.12: An example of an erosion and a dilation operation

(a) Combined mask (b) Opening

(c) Closing (d) Opening followed by a closing

Figure 3.13: Morphological transformations on the combined skin mask

Application of Transformations

After the opening and closing transformations the noise of the mask is significantly reduced, however Fig.3.14a shows that some of the edges of the hands are not masked. This is due to the fact that they are not detected as skin pixels due to the lighting of the scene. Therefore a final dilation to the mask is applied such that these pixels are included, shown in Fig.3.14b.

(a) Masking without dilation (b) Masking with dilation

Figure 3.14: The effect of a final dilation on the hand mask

3.5 Key Press Detection

Problem Statement: Given a transformed frame of the keyboard and a mask of any hands within it, detect the keys that are currently pressed and output their MIDI numbers.

3.5.1 Creating a Difference Image

When a key is pressed, the brightness of the key changes. The cause of this is twofold. The angle of the key changes (causing the light to interact differently with the key) and the side of the adjacent key is no longer obscured. This change in illumination can be seen by using a background subtraction model to calculate a difference image as discussed in Section 2.3.8. Systems using a median background model and a Gaussian mixture model were implemented.

Assuming a perfect background model (i.e. no foreground elements have been incorporated into the model), the hands of the performer will create difference regions within our difference image. We must therefore remove these using the hand mask. Once masked, we find contours within the difference image which we will filter and then match to key presses.

Median Background Model

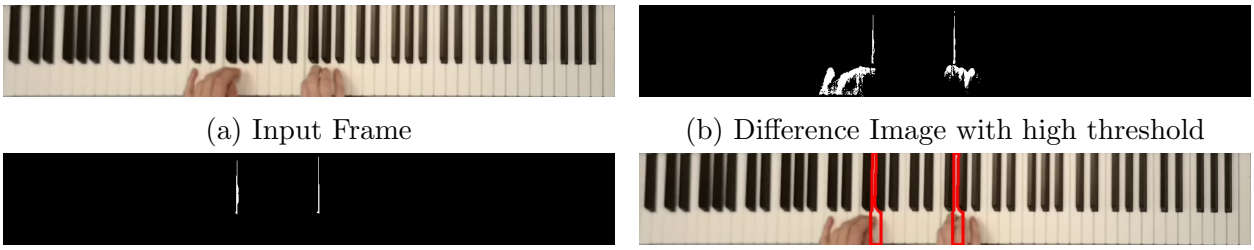
An implementation of the median background model was produced with the addition of a sampling rate q. The presence of long held notes in music means that we want to maximise our time window, such that these held notes do not become a part of the background model. As discussed in Section 2.3.8, this incurs a large memory and computation overhead and so to mitigate this we do not update our model every frame. We instead run our update procedure every q frames, effectively taking a sample of the frames within the time window.

A key advantage of the median background model is that we can distinguish between white key presses and black key presses using the positive and negative difference images created. Black keys become brighter when pressed, white keys become darker, therefore black and white keys can be found in positive and negative difference images respectively.

Gaussian mixture model

OpenCV's MOG2 background subtraction class was used and, to minimise the incorporation of foreground objects into the background model, the history parameter (equivalent to time window size) was set to 10,000. As a single difference image is produced, we risk a scenario where two adjacent keys are pressed to create a single connected region. Section 3.5.3 requires that each connected region within the difference image corresponds to a single key press, therefore the outline of our black and white key contours are drawn onto the difference image in black to segment any such regions.

The Gaussian nature of MOG2 presents another issue for our difference image. As key presses cause small illumination changes, key press regions are prone to noise and may consist of multiple smaller disconnected regions. To connect these, we perform a dilation using a rectangular kernel a single pixel wide in order to avoid increasing the size of, or connecting, regions of noise.



(a) Input Frame

(b) Difference Image with high threshold

(c) Masked Difference Image

(d) Pressed keys detected

Figure 3.15: MOG2 Difference Images

3.5.2 Difference image contour filtering

Both methods of background subtraction produce imperfect difference images. Lighting changes and shadows create noise within the difference image which may cause false key presses. Therefore the contours found within the difference images are filtered by height, area, highest point and x-position. This last filtering step (considering x-position) uses the hand mask to calculate which keys are possible to be pressed given the position of the hands. It achieves this by reducing the hand mask image matrix to a vector along the x-axis using the binary OR operator on each column. This vector is then slightly dilated to account for any small inaccuracies of the hand mask. If the value of the vector is zero at a given x position it means that the hands are not present and therefore the difference image contour is filtered out.

3.5.3 Difference image contour to MIDI number

We assume that each filtered difference image contour corresponds to a single pressed key. To determine which key that is, we take the centre of the difference image contour then perform a point polygon test over all the black and white key contours[7]. The point polygon test determines if a point lies within a contour. Therefore if the point polygon test between the difference image contour centre (x_c, y_c) and key contour q returns positive, we conclude that q is pressed. As we have ordered our key contours from lowest to highest note, matching them to their MIDI values is a simple lookup as the MIDI values are ordered sequentially by pitch. The lowest key of the keyboard has MIDI value 21, the second lowest has value 22 etc.

3.6 Writing to a MIDI file

Problem Statement: Convert the set of pressed key values for each frame into a file in the MIDI file format.

[7]For the median background model based system, we perform the test over either the set of black keys or white keys, depending on if the contours are produced from the positive or negative difference image

3.6.1 Temporal filtering

The inertia of the physical keys of the keyboard restricts the minimum length of a note and the minimum length of a gap within a note. The Guinness world record for most piano key hits in a minute is 824[14] which averages out to roughly 14 notes per second. As the input video files are recorded at 60 frames per second we can safely filter out key press events of a single frame and remove single frame gaps within a note. This filtering removes the real-time capability of the system and so for simplicity we save the set of pressed keys for each frame into a list object and write the MIDI file after the video has been read in its entirety.

3.6.2 Writing MIDI events

As discussed in Section 2.3.2, the MIDI file format stores information as an ordered sequence of timestamped events. The two events of interest for this application are the note-on and note-off events. To convert our data to this format, we iterate through each set of pressed keys and compare it with the set of the previous frame. We create a note-on event for all note values present in the current frame which are not in the previous frame and a note-off event for all values present in the previous frame but not in the current frame. The conversion of events to their corresponding hexadecimal encodings within the MIDI file format is handled by the Mido library[5].

Time management

The MIDI file format does not store absolute time with each event, but instead the number of ticks since the last event; ticks are defined as equal subdivisions of beats. In the header of the MIDI file we determine the number of beats per minute and the number of ticks per beat at 120 and 500 respectively, such that each tick has a 1ms duration. The timestamp in ticks for each event can then be calculated as the change in milliseconds since the last event.

Due to the slight variation in frame-rate across a video, inaccuracy within the time-stamps for each frame is introduced. This is the reason why the resolution was not defined using the camera's frame-rate. Access to a better quality camera with strict fixed frame-rate or time-stamped video output would remove this inaccuracy. Unfortunately access to such equipment was not practical due to the fact that my evaluation started later than intended and was compromised by the outbreak of COVID-19.

3.7 Repository Overview

The code base for this project consists of ten files with 'Video_to_MIDI.py' containing the high level structure for both AMT systems. All other files implement or contribute to a single module[8]. The naming of these files makes their function self evident. Instructions for use of the 'Video_to_MIDI' function can be found in the README.txt document.

[8]Recall the project modules described in Section 2.2.1

Chapter 4

Evaluation

4.1 Intended Approach

Initially, I planned to collect video data from at least ten different piano players. Each would have played eight pieces of music of varying difficulty taken from the ABRSM 2019/20 exam syllabuses. This would have provided a realistic and more challenging data-set than that of Akbari & Cheng's claVision. I would then split this set into testing and training data such that system parameters (the thresholds used in the contour filtering of section 3.5.2) could be tuned while minimising over-fitting. Two hypotheses would be tested. First, that there is significant difference in system accuracy due to the variety in hands and playing styles between players. Second, that there is significant difference in system accuracy between pieces. If this second hypothesis were proved, a further comparison of the effects of piece difficulty and speed would be carried out.

Due to the unforeseen and disproportionate workload of the Part II Robotics unit of assessment, the collection of evaluation data was seriously delayed. Despite this, my plan for data collection remained unchanged as I had access to over ten consenting and highly-skilled piano players in the form of organ and choral scholars who would remain in Cambridge over the Easter revision break. The outbreak of COVID-19 removed this potential. Consideration was given to remote data collection using a video chat however the special hardware required in the form of a MIDI-capable keyboard, cable and interface made this infeasible. Instead, I was forced to compromise my evaluation and make use of the three piano players available to me at home.

4.2 Data Collection

The test set compromised of six pieces of music. To avoid selection bias and provide a range of difficulties, the first listed piece from the 2019/20 ABRSM syllabus was selected for each of grades 1-6. Higher grades were omitted due to their difficulty with respect to available

[0]The list of these pieces can be found on the experiment briefing form which is included in the Appendix

pianists. Each pianist played all 6 pieces on an electric piano with the performances captured simultaneously by both a smartphone camera[1] on a camera stand, and by the MIDI output of the piano. The collected MIDI data forms the ground truth for future evaluation. A sample frame showing the pose of the camera relative to the keyboard can be seen in Fig.4.1.

Figure 4.1: Sample frame of the smartphone camera output

As the amount of time required to learn pieces became a significant factor, a training set was created using pieces of music already known to the pianists. Technical exercises such as scales (a sequence of consecutive notes) and arpeggios (a regular subset of non-consecutive notes within a scale) were also included.

4.3 Evaluation Metrics

The primary method used to evaluate our system treats each note of each frame as an independent event. This has some limitations. The presence of a note is often more important than its precise length and the presence of certain notes within harmonic structures may be more important than others. Techniques such as sequence alignment have been shown to account for such limitations. However, as the pieces in both data sets are homophonic, with any possible combination of 88-keys being pressed in a frame, applying sequence alignment to this domain was impractical. Therefore precision and recall were chosen:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}$$

Precision describes the proportion of predicted events that actually happened. In this application a true positive is (for a given frame) when a note is present in both the ground

[1]The smartphone camera used was the main camera from the Huawei P20 pro, recording at 720p, 60fps

truth and predicted MIDI files. A false positive would be a predicted note that is not present in the ground truth.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

Recall is defined as the proportion of positive events that were correctly detected. A false negative in this application is a note present in the ground truth that was not predicted.

The parameter tuning of the system determines the trade-off between precision and recall. For example, if the parameters are set generously to the point that every note of every frame is predicted to be present, 100% of positive events will have been predicted and recall will be maximised at the expense of an incredibly low precision. Similarly a system that predicts notes only when absolutely certain of their presence will have a very high precision but low recall. Therefore, as we want a balance of both precision and recall we use an F1 score:

$$F1 = 2 \times \frac{Precision \cdot Recall}{Precision + Recall}$$

F1 score is the harmonic mean of precision and recall and provides a single measure for our system's accuracy. Parameter tuning therefore attempts to maximise F1 score.

4.4 Data Alignment

The videos from which we create our predicted MIDI files and the ground truth MIDI files have the problems of being misaligned and of different lengths. The fact that the video and ground truth MIDI files were recorded on separate devices meant that coordinating their starts within the experimental setup was not possible.

Simply aligning both files using the first note-on event of each is likely to result in at least a slight misalignment for any system with either false positives or false negatives. For example, the first note of the predicted file may have been wrongly predicted due to hand masking noise, or instead the true first note of the ground truth MIDI file may not have been detected at all.

To align the files, we assume that the true alignment will result in the highest F1 score. However, testing all possible alignments is computationally expensive, so instead we align our files using the first MIDI event, then test all alignments within 0.5 seconds of this. The cases where this approximation fails to find the true alignment will result in a non-optimal F1 score, however the occurrence of such alignment failures was rare, and only present when system parameters were poorly chosen.

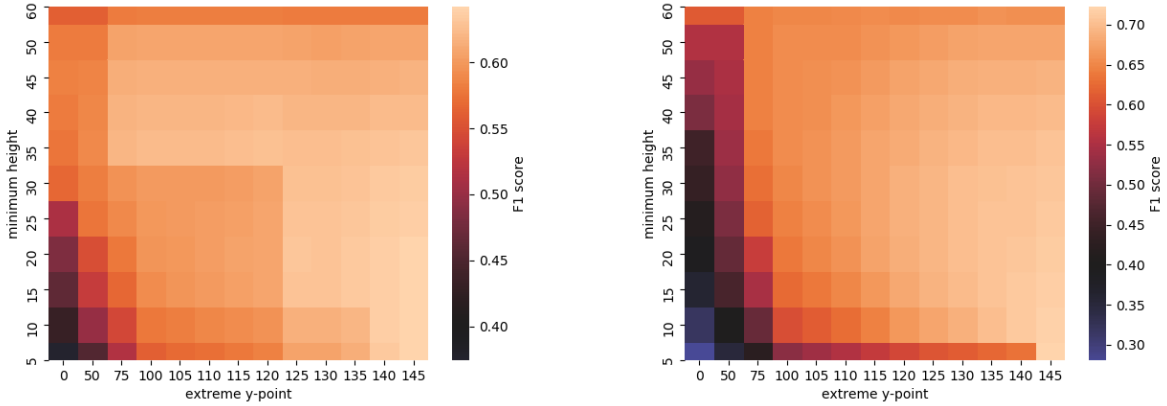
4.5 Parameter Tuning

We have two systems which we want to tune: one using the median background model and another using MOG2 background subtraction. Preliminary experiments with both systems

showed that difference image contour height and extreme y-point were the most significant parameters. These parameters are used to filter noise from the difference image contours. The contour height parameter h removes all contours whose length in the y-axis is less than h. The extreme y-point parameter k removes all contours whose highest point[2] is less than k. Note that the perspective transformation of Section 3.2.3 allows us to set the height and width of our keyboard to 150 and 1000 pixels respectively.

The results of the parameter tuning for the median background and MOG2 systems are shown in Fig. 4.2. In total 143 parameter combinations were tested for each system using the training data set. Unlike the MOG2 system, the median background system did not produce a smooth gradient of F1 score within its heat map. These boundaries represent the thresholds on system parameters required to produce MIDI files that correctly align. This misalignment causes dramatic drops in F1 score as previously discussed.

The optimal pair of contour height and extreme y-point for the median background system was (10,145). The optimal pair for the MOG2 system was (5,145).



(a) Parameter tuning heat map for the Median background system

(b) Parameter tuning heat map for the MOG2 system

Figure 4.2: The results of parameter tuning for the MOG2 and median systems

4.6 Comparison of median background and MOG2 on system accuracy

When applied to the testing set, the overall[3] accuracy for the median background and MOG2 systems was 67.5% and 72.0% respectively. The Median system produced an average pre-

[2]openCV's coordinate system places the origin in the top left of an image. However, for ease of reading this document assumes that the origin is placed in the bottom left corner of an image

[3]Overall F1 score was calculated by summing the total true positives, false positives and false negatives across all files and then calculating precision, recall and F1 with these totals.

cision of 75.2% and an average recall of 61.2%. The MOG2 system produced an average precision and recall of 74.0% and 70.1%. The F1 score distribution for each of the six pieces can be seen in Fig.4.4. Sample MIDI outputs from both systems can be seen in Fig.4.4.

Both systems ran in near real-time. Given a file 126 seconds long, the median system took 188 seconds to process it and the MOG2 system took 138 seconds[4]. However, their running times are dependent on factors such as the number of bounding lines needing to be compared in initialisation and the number of difference image contours per frame so these times will vary depending on the data and the computer running the system.

4.6.1 Analysis

Both systems perform comparably across the data with the exception of the ABRSM grade 3 and 6 pieces. For these pieces, the median system significantly out-performs the MOG2 system. Analysing the grade 3 outputs more carefully reveals that the median system's precision remains high but the recall is very poor in comparison to other grades. The average recall for the median system for grade 3 is 47.3%, a dramatic drop in comparison to similar grades. Inspection of the video log confirmed that many valid key presses were being filtered out, mostly by the extreme y-point parameter. These errors seemed to focus around specific keys, namely C and F. Fig. 4.5 shows the position of C and F keys in blue. These keys are positioned wherever two white keys occur without a black key between them. Because of this, pressing a C or F key does not reveal a black key in its place but instead the side of another white key. The median background model doesn't detect this change as significant enough, (most likely because of the global thresholding applied) and so any resulting contour will not be the same size or height as other contours.

This also explains the poor performance for the grade 6 piece. Both pieces are in the same harmonic key[5] (F major). This harmonic key, combined with the musical style of the pieces, makes C and F key presses prevalent. The proportion of key presses[6] that are C or F keys per piece is compared with an estimate of the median system's error in Fig. 4.6. This figure suggests a relationship between this proportion and the system error, described by $1 - \mathrm{F1}_{avg}$.

Having discovered this issue for the Median system, I wanted to investigate three hypotheses. First, that this issue applies to the MOG2 system. Second, whether this issue applies only to the C and F keys. Finally, if there is a significant difference in key press detection accuracy between the white and black keys of either system. To test each hypothesis, the true positive, false positive and false negative features were calculated for each individual key. Figs. 4.7 & 4.8 show these results for both systems.

[4]Both systems were run on a Dell XPS 15 9560 with an Intel Core i7-7700HQ CPU @2.80GHz with 16.0GB RAM

[5]A harmonic key is a property of the music and not a physical property of an instrument

[6]This metric is calculated frame-wise. The proportion is technically the amount of time a C or F note is played divided by the sum of all note durations

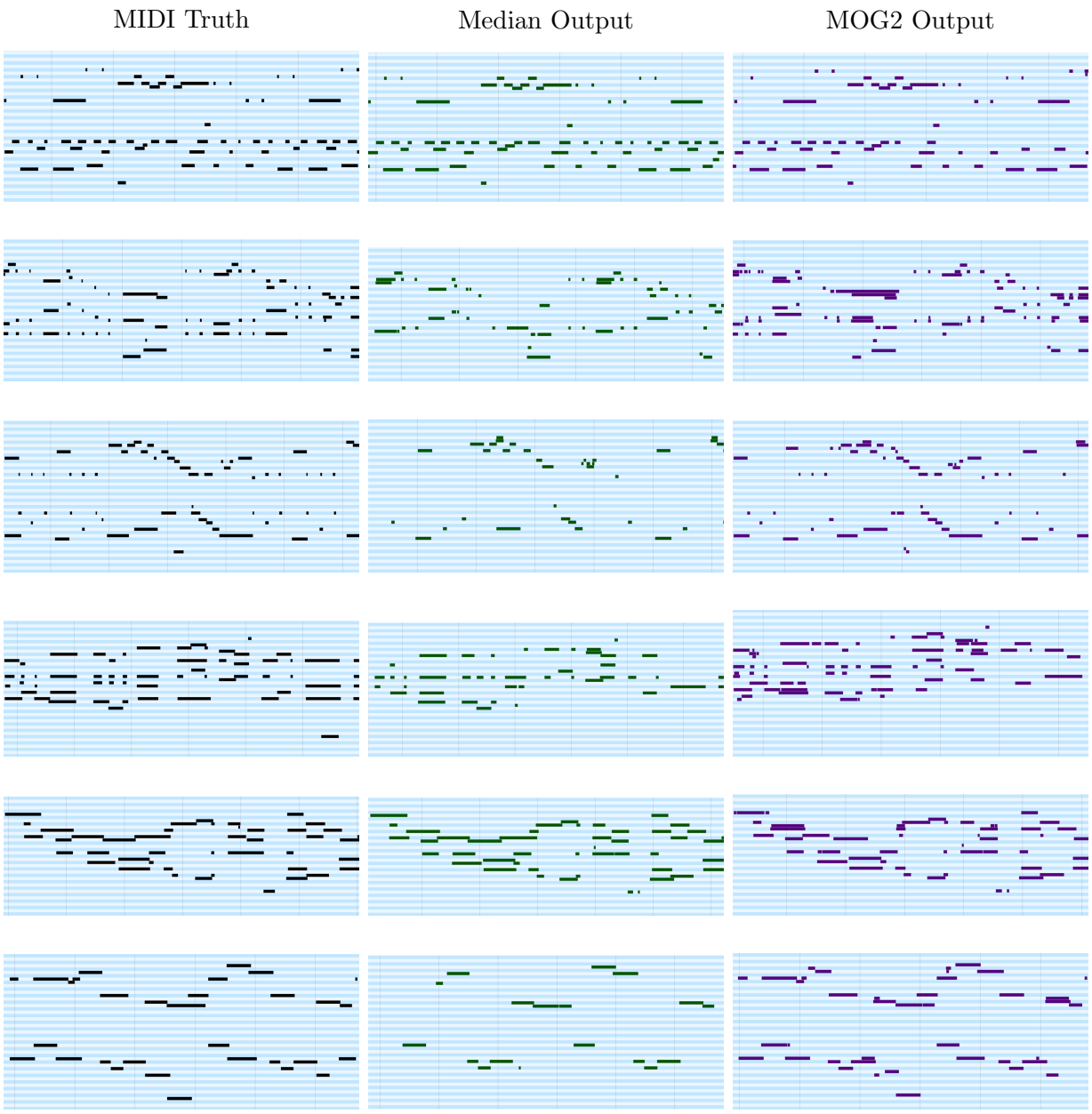


Figure 4.3: Sample MIDI outputs from the Median and MOG2 systems.
Left column: MIDI truth, Middle column: Median output, Right column: MOG2 output.
Rows ordered by grade: First row is output from grade 1, second from grade 2 etc.

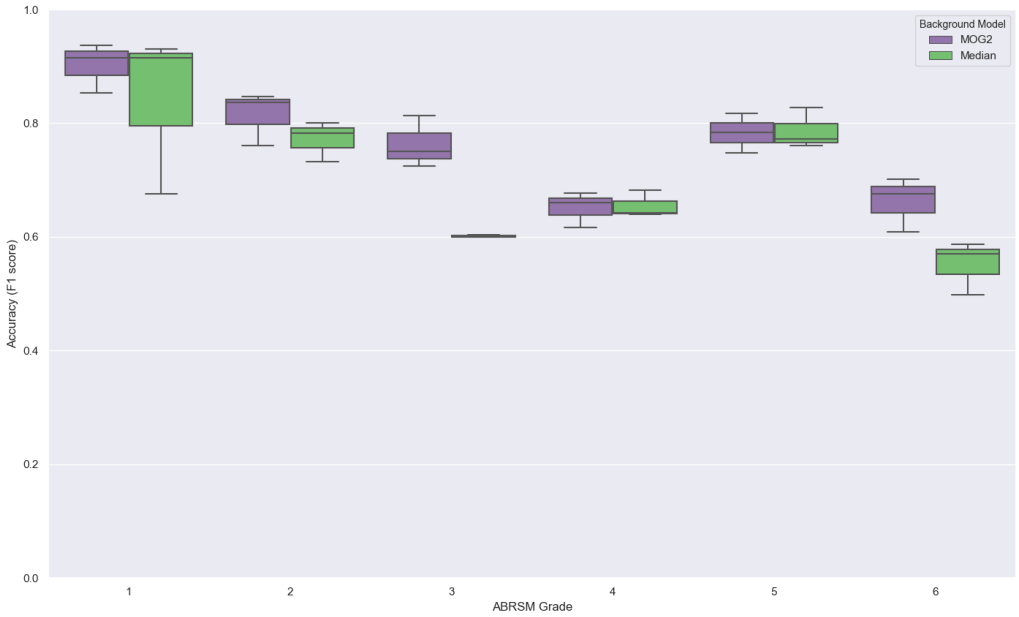


Figure 4.4: Average accuracy across the 6 ABRSM grades for both systems
In general, the MOG2 system out-performs or performs comparably to the median system

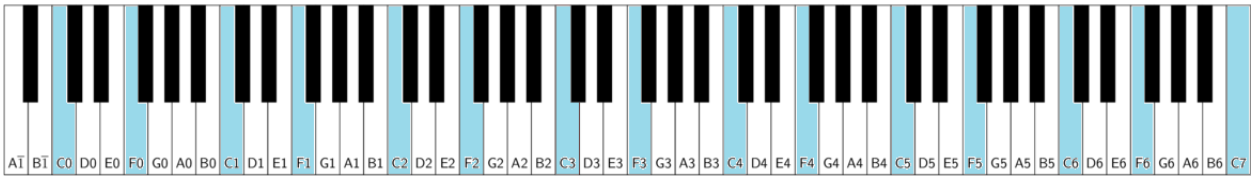


Figure 4.5: The position of C and F keys within the piano keyboard

The C and F key problem can be seen in the Median white key feature graph where fully green bars (false negatives) are spaced in the same manner as C and F keys. This shows the severity of this issue. Practically none of the C and F key presses for the median system are predicted correctly. This pattern of false negatives can be seen in the MOG2 white key feature graph too but to a lesser extent. This demonstrates that the C,F key issue applies to both systems but has a far greater effect on the median system.

The relative similarity of feature distributions between keys of the same colour suggests that there is no equivalent of the C,F key problem for other keys. However, there is a slight difference in the distributions of black and white keys for both systems. The ratios of false positives to false negatives is higher for black keys than white keys. I believe that this is due to the shape of the piano keys themselves, combined with the extreme y-point filtering. At the very top of the keyboard, the black keys are often wider than the white keys

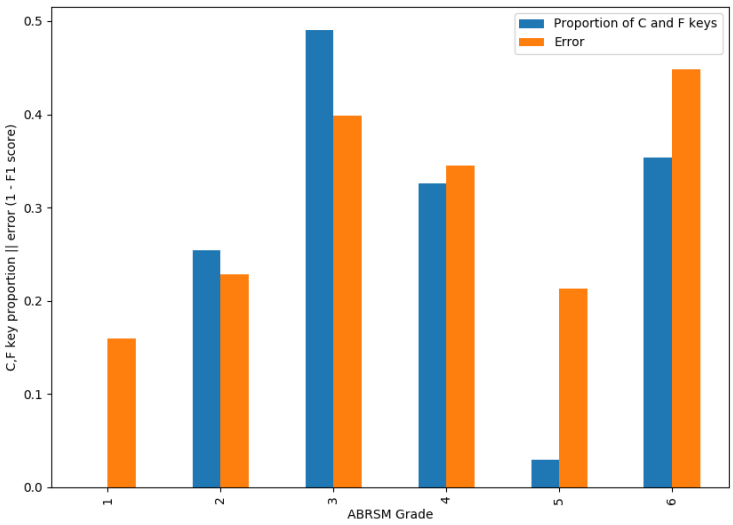


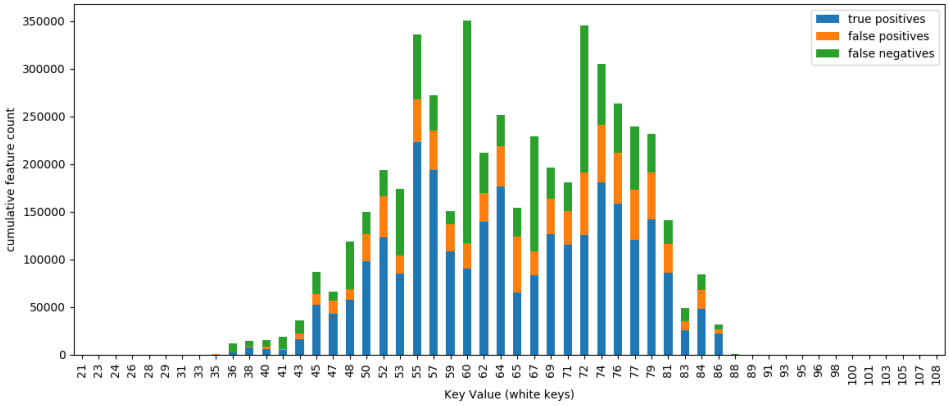
Figure 4.6: The proportion of key press that belong to C or F keys compared to the median system's error ($1 - \mathrm{F1}_{avg}$) for each grade

because of the effect perspective correction has when combined with their slight elevation. This is significant as the extreme y-point parameter is set very high for both systems. This parameter helps filter out noise, but only if that noise is too low. Therefore, noise that occurs towards the top of the keyboard is more likely to be counted as a valid key press. As the proportion of black key to white key area is higher towards the top of the keyboard, noise is more likely to cause a false positive key press for black keys.

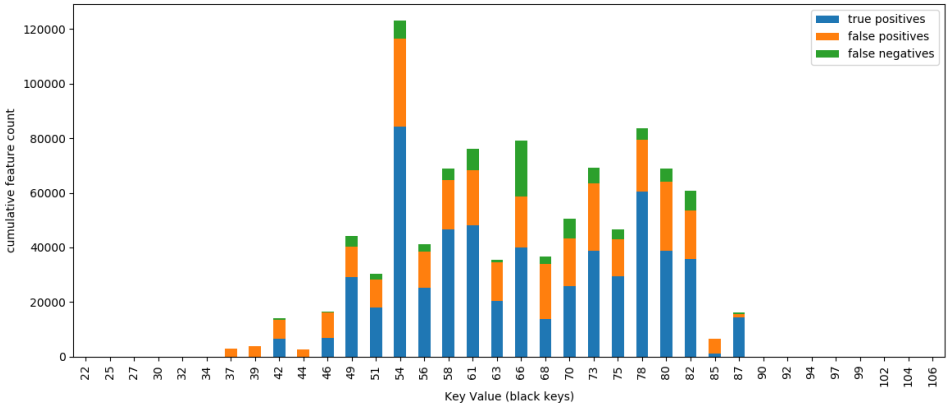
4.6.2 System Limitations

Besides the C,F key issue presented above, the biggest limitation for both systems is the presence of shadows caused by the hands of the player. Shadows cast by the hands can cause illumination changes that are read as key presses because the current contour filtering method fails to filter these contours out. The MOG2 background subtraction method has the capabilities to identify certain changes as shadows however early experimentation proved that this was not accurate enough. Key presses were often identified as shadows and true hand shadows were often identified as foreground objects.

The hands pose another problem to both systems as they can occlude the keys. Although they rest towards the bottom of the keyboard, the hands occasionally move up to occlude keys in their entirety. This is more common when playing black notes as moving up the keyboard can assist playing when certain hand positions are required. In the case of occlusion, both systems will fail to detect any played notes.

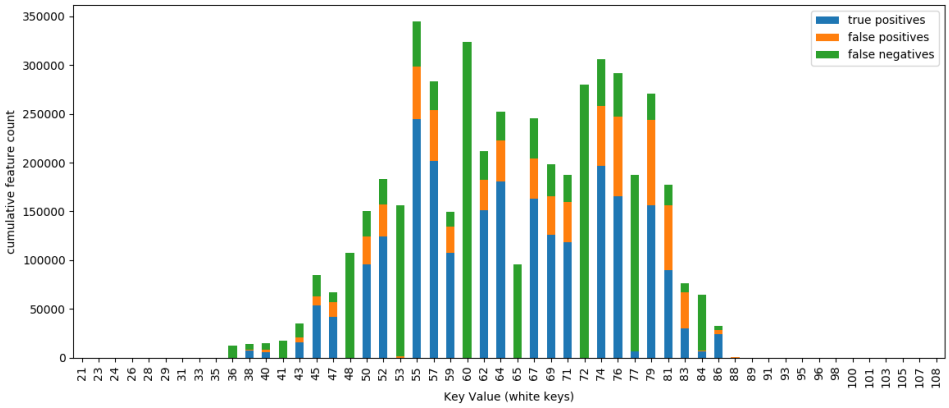


(a) MOG2 white key features

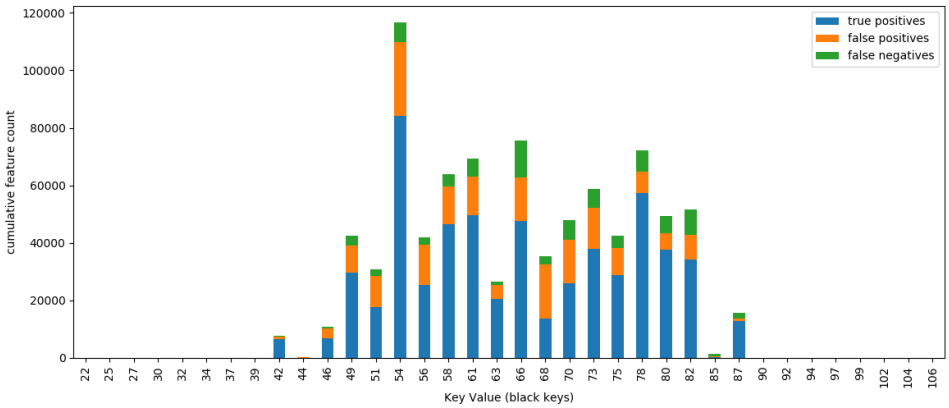


(b) MOG2 black key features

Figure 4.7: Per-key features for the MOG2 system, calculated across all players and grades



(a) Median system's white key features



(b) Median system's black key features

Figure 4.8: Per-key features for the median system, calculated across all players and grades

Chapter 5

Conclusions

This project has been a great success. The success criteria were far surpassed with two working AMT systems implemented, and all three project goals delivered. The flaws in the Median and MOG2 systems have been compared and analysed, with the MOG2 system producing better accuracies on average. The overall F1 score of 72% is an impressive result, especially given the reduction in data set quality caused by the COVID-19 outbreak. The combination of an audio AMT with the MOG2 system presented in this paper is a logical next step towards achieving even higher AMT accuracy.

On a personal note, this project has involved a significant amount of learning. Both the field of computer vision and the Python programming language were new to me at the beginning of this project. If I were to plan this project again, I would suggest a focus on combining video and audio processing methods. To allow time for this, I would remove the automatic keyboard recognition requirement.

5.1 Future Work

The error cases within both Median and MOG2 systems are very different from those found within an audio signal processing AMT system. For example, the octave problem discussed in Section 2.1.1 is not present within the two computer vision systems. However, the double key problem (discussed in Section 4.6.2) present in both computer vision systems is not found within audio signal processing AMT systems. Therefore, I believe the combination of audio signal processing and computer vision techniques could produce AMT systems with far greater accuracies.

There remain opportunities for improvement within the computer vision systems presented.

- Additional parameter tuning which considers model thresholds is likely to lead to higher system accuracies
- Different methods for identifying key presses such as a summation of contours within a key area may be considered

- The C,F key problem might be addressed, potentially with per-key thresholds and parameters
- Invariance to camera pose, movement within the scene and poor lighting conditions might be considered
- Hand tracking may be incorporated to provide a more fine-grain estimation of the set of possible key presses for a given frame

In my opinion, the largest remaining problem is the lack of a standardised data set. The MAPS data set for audio based AMT permits comparison between audio signal processing systems from different papers. No such equivalent exists for AMT from video, which limits the evaluation of this paper. One significant problem in creating such a data set is ensuring appropriate data protection and compensation for the people involved. With regards to this project, all video data used within experiments was destroyed after the completion of this document, as described in the experiment consent form (available in the Appendix).

Bibliography

[1] Mohammad Akbari and Howard Cheng. "ClaVision: Visual Automatic Piano Music Transcription". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. NIME 2015. Baton Rouge, Louisiana, USA: The School of Music, the Center for Computation, and Technology (CCT), Louisiana State University, 2015, pp. 313–314. ISBN: 9780692495476.

[2] G. Richard B. Fuentes R. Badeau. "Harmonic Adaptive Latent Component Analysis of Audio and Application to Music Transcription". In: *IEEE Transactions on Audio, Speech, and Language Processing (2006-2013)* 21.9 (2013), pp. 1854–1866.

[3] E. Benetos et al. "Automatic Music Transcription: An Overview". In: *IEEE Signal Processing Magazine* 36.1 (2019), pp. 20–30.

[4] Emmanouil Benetos et al. "Automatic music transcription: challenges and future directions". In: *Journal of Intelligent Information Systems* 41.3 (2013), pp. 407–434. DOI: 10.1007/s10844-013-0258-3.

[5] Ole Martin Bjørndalen. *Mido*. Version 1.2.9. Sept. 2018. URL: https://github.com/mido/mido.

[6] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[7] dmann200. *The Mechanics of Piano Tuning*. https://www.lapianotuning.com/wordpress/?p=62. Accessed: 23-04-2020. Mar. 2015.

[8] Valentin Emiya, Roland Badeau, and Bertrand David. "Multipitch Estimation of Piano Sounds Using a New Probabilistic Spectral Smoothness Principle". In: *IEEE Transactions on Audio, Speech, and Language Processing* 18.6 (2010), pp. 1643–1654. DOI: 10.1109/tasl.2009.2038819.

[9] Nir Friedman and Stuart J. Russell. "Image Segmentation in Video Sequences: A Probabilistic Approach". In: *CoRR* abs/1302.1539 (2013). arXiv: 1302.1539. URL: http://arxiv.org/abs/1302.1539.

[10] D.o. Gorodnichy and A. Yogeswaran. "Detection and tracking of pianist hands and fingers". In: *The 3rd Canadian Conference on Computer and Robot Vision (CRV06)* (). DOI: 10.1109/crv.2006.26.

[11] S. Kolkur et al. “Human Skin Detection Using RGB, HSV and YCbCr Color Models”. In: *Proceedings of the International Conference on Communication and Signal Processing 2016 (ICCASP 2016)* (2017). DOI: 10.2991/iccasp-16.2017.51.

[12] Robert McCaffrey. “Piano Music Transcription Based on Computer Vision”. PhD thesis. University of Dublin, Trinity College, May 2017.

[13] J. Moorer. “On the Segmentation and Analysis of Sound by Digital Computer”. PhD thesis. Stanford University, 1975.

[14] *Most piano key hits in one minute*. URL: https://www.guinnessworldrecords.com/world-records/most-piano-key-hits-in-one-minute.

[15] Werner Schweer. *Musescore*. URL: https://musescore.org/en.

[16] S. Sigtia, E. Benetos, and S. Dixon. “An End-to-End Neural Network for Polyphonic Piano Music Transcription”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.5 (2016), pp. 927–939.

[17] Li Su and Yi-Hsuan Yang. “Combining Spectral and Temporal Representations for Multipitch Estimation of Polyphonic Music”. In: *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 23.10 (Oct. 2015), pp. 1600–1612. ISSN: 2329-9290. DOI: 10.1109/TASLP.2015.2442411. URL: https://doi.org/10.1109/TASLP.2015.2442411.

[18] P Suteparuk. “Detection of piano keys pressed in video”. PhD thesis. Stanford University, 2014.

[19] Satoshi Suzuki and Keiichi Abe. “Topological structural analysis of digitized binary images by border following”. In: *Computer Vision, Graphics, and Image Processing* 29.3 (1985), p. 396. DOI: 10.1016/0734-189x(85)90136-7.

[20] *The MIDI Association*. URL: https://www.midi.org/.

[21] Bingjun Zhang et al. “Visual analysis of fingering for pedagogical violin transcription”. In: *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA 07* (2007). DOI: 10.1145/1291233.1291361.

[22] Zoran Zivkovic and Ferdinand [van der Heijden]. “Efficient adaptive density estimation per image pixel for the task of background subtraction”. In: *Pattern Recognition Letters* 27.7 (2006), pp. 773–780. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.11.005. URL: http://www.sciencedirect.com/science/article/pii/S0167865505003521.

Appendix

Briefing Form

The purpose of this experiment is to collect video and MIDI data of different performers playing the following pieces from the 2019&20 ABRSM piano syllabus (grades one to six)

Pieces provided:

Attwood: Theme from Theme and Variations, Sonatina No. 4 in D

Diabelli: Lesson in C No. 10 from Die ersten 12 Lektionen, Op. 125

Hook: Allegro 1st movt from Sonatina in F, Op. 12 No. 3

Beethoven: Bagatelle in C WoO 54

J. S. Bach: Aria 4th movt from Partita No. 4 in D, BWV 828

T. A. Arne: Andante 1st movt from Sonata No. 1 in F

As stated in the consent form, there is no evaluation of the accuracy of the playing. You may take a break or opt-out entirely at any time. The session will last anywhere between 15 to 30 minutes.

Data Collection and Retention

Video data of you playing the piano will be recorded along with the MIDI output data from the electric keyboard for each piece. Any results published will be completely anonymised and all video and MIDI data from the experiment will not be stored on the cloud and completely destroyed by 09/05/2020.

Consent Form

Purpose & Procedure

The purpose of this experiment is to collect video data of 6 provided piano pieces being played in order to test the accuracy of MIDI data transcription software I have written. You will be asked to play each of the 6 pieces of music one by one on an electric keyboard with a camera placed above it. The accuracy of your playing will not be assessed, and wrong notes won't in any way effect the experiment.

Data Collection and Retention

Video data of you playing the piano will be recorded along with the MIDI output data from the electric keyboard for each piece. Any results published will be completely anonymised and all video and MIDI data from the experiment will not be stored on the cloud and completely destroyed by 09/05/2020.

Consent

Your signature below confirms that you have read and understood this form and have given your consent to participate. Participation is entirely voluntary, and you may opt-out at any time. Participation is not recommended if you are suffering from any form of hand injury. If you have any questions regarding the experiment or the project, please email REDACTED.

Participant's Signature	Date

Project Proposal

2396C
Kings

Diploma in Computer Science Project Proposal

Extracting MIDI Data From Video of a Piano Using Computer Vision

24/10/19

Project Originator: 2396C

Project Supervisor: Aamir Mustafa

Director of Studies: Dr Timothy G Griffin

Overseers: Dr Larry Paulson and Dr Markus Kuhn

Introduction and Description of the Work

It is often useful to be able to transribe music or create electronic music using a MIDI keyboard but often the quality of many available electric keyboards is lacking; simulating the hammer action of a real piano in an electric keyboard is a tricky and expensive task. This means that for many classical musicians, hand transcription is often the most convenient.

Current audio processing methods of transciption deal with very difficult problems and there is yet to be a system with high enough accuracy and efficiency to be viable. Sensors above the keys or hammers of the piano would get the best accuracy but with 88 keys this solution would be large and inelegant (not to mention the retiscence the owner of said piano might have if you started sticking nearly a hundred sensors and wires inside/on their piano).

On the other hand, computer vision solutions have been shown to be far more viable reaching an average precision and recall of 78.72 % and 93.57 % respectively when tested on a transription task of 6 pieces from the ABRSM piano grades [1]. Computer vision based transription has the potential too for collecting data on fingering (which finger of which hand is used to press which key) which is used, especially by beginners, for learning to play.

This project will involve the implementation of a system that will allow a user to play a piano with a camera sensor above the keyboard and have a MIDI output (not necessarily in real-time) corresponding to the notes played on the piano. MIDI stands for Musical Instrument Digital Interface and is a well documented and widely used standard for describing notes, pitches and other aspects of musical performance. In effect this system will convert the piano into a MIDI keyboard. Using this MIDI data it will then be posible to create a transcription of the music being played using a MIDI to sheet music tool such as *MuseScore* (`https://musescore.org/en`). The evaluation of this project will be based on comparing the MIDI data provided instead of the accuracy of a transciption of this data. This is because different transcription software may output different results when given the same MIDI input.

1. A daily and automatic cloud backup of all the files on my computer with the ability to retrieve deleted files for one month

2. A Git repository with regular pushes of new code

3. A weekly backup of all project files onto a USB stick

Starting Point

I have no prior experience in any areas of computer vision and I have not used Python before beyond the scientific computing course. The project will make use of OpenCV and Python libraries but will not be based off or modifying any pre-existing code. The small collection of articles and papers achieveing similar goals will be an obvious resource for methods and procedure [1] [2] [3] [4].

Substance and Structure of the Project

This project will be written in *Python* (`https://www.python.org`) using *OpenCV* (`http://www.opencv.org`) The process of converting the video into MIDI data can be described in the following steps:

1. Locating the piano position and compensating for orientation. The high contrast and regular pattern of black and white keys makes finding the piano in the images the more achievable part of this task. Once found, a geometric transformation of the keyboard will be required such that the image lines up with the 88 key layout of a piano. To assist with this, a calibration step may be included.

2. Identifying and labelling the keys of the piano. The fact that every black key is spaced apart from each other makes finding them a lot easier than the white keys as the gaps between these keys are minimal. Therefore as explained in the position of the white keys can be inferred once the black keys are found [1].

3. Hand masking; identifying and removing the part of the frame taken up by hands over the piano keys. As the background to the hands we want to remove is black and white, we can use the colour range of skin tones to remove the hands without fear of removing other important parts of the image.

4. Recognising when a key is pressed. To do this we will need to maintain and build up an understanding of what the piano keyboard looks like when no keys are being pressed. By comparing the difference in our current frame to this saved average image we can identify pressed keys.

5. Converting the keys being pressed into appropriate MIDI data. For example, when we believe a key to be pressed we send a "Note On" message.

The evaluation of this project will consider key press accuracy and note length accuracy. A set of scales and pieces of varying difficulty will be played on a MIDI keyboard with the camera above providing our computer vision based MIDI output. With the output from the actual MIDI keyboard taken as ground truth we can then identify any errors in pitch and the average error of note lengths. The trials will be performed by human participants other than myself to avoid bias in the style of playing.

If there is time, these extensions may be considered:

1. Running this system in real-time and considering the impact of latency.

2. Collecting and evaluating velocity data. Velocity describes how fast the key was depressed and so how loud the note should sound. Although this data may not be difficult to collect, I have doubts on the accuracy attainable.

3. Investigating the effect of the camera orientation and position on the accuracy of the MIDI data collected.

4. Using a Kinect or other depth based sensor and compare the accuracy achieved by such a system to one based on an RGB camera.

5. Using multiple camera feeds and combining the probabilities of key presses from them to hopefully improve both coverage and accuracy. This should only be attempted in the case that I am significantly ahead of the given timetable.

Reference

[1] *Piano Music Transcription Based on Computer Vision*, Robert McCaffrey, Dissertation, University of Dublin, Trinity College.
(`https://scss.tcd.ie/publications/theses/diss/2017/TCD-SCSS-DISSERTATION-2017-087.pdf`)

[2] *IEEE Transactions on Multimedia, VOL 17, NO. 12: Real-Time Piano Music Transcription Based on Computer Vision*, Mohammad Akbari & Howard Cheng, Article, University of Lethbridge, Canada.
(`https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7225173`)

[3] *Pianist Motion Capture with the Kinect Depth Camera*, Hadjakos, Aristotelis, Conference Paper , Proceedings of the 9th Sound and Music Computing Conference, SMC 2012, 01/01/2012.
(`https://www.researchgate.net/publication/270818909_Pianist_Motion_Capture_with_the_Kinect_Depth_Camera`)

[4] *Observing Pianist Accuracy and Form with Computer Vision*, Jangwon Lee, Bardia Doosti, Yupeng Gu, David Cartledge, David Crandall & Christopher Raphael, Conference Paper, 2019 IEEE Winter Conference on Applications of Computer Vision, 07/03/2019, ISBN: 978-1-7281-1975-5
(`https://ieeexplore.ieee.org/document/8658842/authors#authors`)

Success Criteria

1. The system should output MIDI data based on the video input. N.B. this sytem is not required to work in real-time to meet this criteria

2. The system must be evaluated to include metrics such as pitch accuracy and note length accuracy.

Timetable & Milestones

To aid in dissertation writing, written notes will be kept throughout the project and various sections of the dissertation will be drafted throughout the Michelmas and Lent terms before the final writing phase.

28th October - 11th November

Setting up and learning to use OpenCV. Begin work on locating the keyboard within the frame.

11th November - 25th November

Complete the keyboard location and test to ensure it is robust enough to account for minor discrepencies in camera position. Begin work on correcting the keyboard image for orientation.

Milestone: Keyboard can be located within frame.

25th November - 9th December

Continue work or complete keyboard orientation correction. *This time frame clashes with one of the larger deadlines for the assessed module I am taking (NLP). Therefore the work load given for this 2 week stretch is deliberately small.*

9th December - 23rd December

Complete keyboard orientation correction and begin work on identifying the black keys.

Milestone: Keyboard has been transformed to evenly lay across a template of a piano keyboard.

23rd December - 6th January

Continue white key identification *As both my 21st and Christmas fall between these two weeks, the work load given is deliberately smaller.*

23rd December - 6th January

Complete white key identification.

Milestone: White and black keys identfied within keyboard region.

6th January - 20th January

Begin work on hand masking.

20th January - 3rd February

Finish work on hand masking and begin work on key press and release recognition.

Milestone: hand objects masked out of keyboard images.

3rd February - 17th February

Finish work on key press and release recognition

Milestone: System can detect when keys are being pressed.

17th February - 2nd March

produce a MIDI output and test the system, implementing any additional required filtering or smoothing of the data.

Milestone: System outputs MIDI data corresponding to key presses.

2nd March - 16th March

Perform trials and evaluate the system as described.

Milestone: Evaluation data collected.

16th March - 13th April

Write up: as I will be writing up the project as a draft as I go and keeping ample notes I have not given a specific timetable for the sections of the write up.

Milestone: A full dissertation draft.

13th April - 6th May

If I have managed to keep to the timetable and have finished the write up, this time can be used for work on extensions.

7th May

Dissertation submission.

Resources Required

I have my own webcam and tripod. I have access to a piano in my room for testing and also pianos within Kings. I also have my own full size MIDI keyboard to use for testing and evaluation. I will be using my own computer and in the case that this fails, I have the option to use another computer I own or move my project across to the MCS.

The computer I will use is a Dell XPS 15 9560 with an Intel Core i7-7700HQ CPU @ 2.80GHz with 16.0GB RAM, an NVIDIA GTX1050 and a 512GB SSD running Windows 10.

I accept full responsibility for all hardware used and I have made contingency plans to protect myself against hardware and/or software failure.

File Backups

As I will be writing both my code and dissertation on my own computer(s) (such that I would not have to bring a large MIDI keyboard into the lab) I will maintain 3 different backups of my work: