Samuel Kerr

Automatic guitar tablature generation from classical guitar performances using Computer Vision

Computer Science Tripos – Part II

Clare College

May 14, 2021

Declaration

I, Samuel Kerr of Clare 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. I am content for my dissertation to be made available to the students and staff of the University.

Signed Samuel Kerr

Date May 14, 2021

Proforma

Candidate Number:	**2408A**
Project Title:	**Automatic Guitar Tablature Generation from Classical Guitar Performances using Computer Vision**
Examination:	**Computer Science Tripos – Part II, July 2021**
Word Count:	**11558**[1]
Lines of Code:	**3376**[2]
Project Originator:	Dr Christopher Town
Supervisor:	Dr Marwa Mahmoud

Original Aims of the Project

The aim of this project was to explore how Computer Vision techniques can be used for the automatic transcription of classical guitar performances, producing guitar tablature. To make this more feasible, this project aims to only transcribe videos that adhere to certain constraints such as requiring the videos to be recorded from a stationary camera and have good lighting. The extension work for this project was to improve the systems performance under relaxed constraints, such as poor lighting.

Work Completed

An automatic transcription system for the classical guitar was implemented. This project generates guitar tablature as a text file, logs any errors that occured during transcription and can also produce an annotated version of the input video showing all features detected. An extension was completed allowing for the transcription of videos under relaxed constraints. A quantitative evaluation was completed by computing the precision, recall and F_1 metrics on videos ranging in musical complexity, background, guitar angle and lighting. A qualitative evaluation was also completed by comparing the output of this system to the output of an audio-based transcription system.

[1] Counted using teXcount online: https://app.uio.no/ifi/texcount/online.php

[2] Counted using `find . -name '*.py' | xargs wc -l`

Special Difficulties

None

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Guitar Tablature . . . 2
- 1.3 Related Work . . . 3

2 Preparation **5**
- 2.1 Starting Point . . . 5
- 2.2 Requirements Analysis . . . 5
  - 2.2.1 Success Criteria . . . 5
  - 2.2.2 Input Constraints . . . 6
  - 2.2.3 Project Deliverables . . . 7
- 2.3 Computer Vision Theory . . . 7
  - 2.3.1 Thresholding . . . 7
  - 2.3.2 Morphological Transformations . . . 8
  - 2.3.3 Image Smoothing . . . 9
  - 2.3.4 Canny Edge Detection . . . 9
  - 2.3.5 Hough transform . . . 10
  - 2.3.6 Hierarchial Clustering . . . 10
- 2.4 Software Engineering Techniques . . . 10
  - 2.4.1 Development Workflow . . . 11
  - 2.4.2 Code Style . . . 11
  - 2.4.3 Testing . . . 11
  - 2.4.4 Languages, Libraries and Tools . . . 11

3 Implementation **12**
- 3.1 Methodology . . . 12
- 3.2 Background Subtraction . . . 15
- 3.3 Soundhole Segmentation . . . 16
  - 3.3.1 Soundhole Detection . . . 16
  - 3.3.2 Soundhole Tracking . . . 16
- 3.4 String Detection . . . 18
  - 3.4.1 Neck borders estimation . . . 18
  - 3.4.2 Perspective Transform . . . 19
  - 3.4.3 Candidate String Detection . . . 20
  - 3.4.4 Filtering down to 6 lines . . . 22
- 3.5 Fret Detection . . . 23

3.5.1 Fretboard isolation and pre-processing . . . 24
3.5.2 Enhancing fret edges: . . . 24
3.5.3 Detect candidate frets: . . . 24
3.5.4 Skin detection: . . . 25
3.5.5 Remove and replace frets: . . . 25
3.6 Fingertip Detection . . . 26
3.7 Plucking Detection . . . 27
3.7.1 Optical Flow . . . 27
3.8 Tab Generation . . . 28
3.8.1 Note timing constraints . . . 28
3.9 Extensions . . . 29
3.10 Repository Overview . . . 30

4 Evaluation **31**
4.1 Evaluation Method . . . 31
4.1.1 Scenes . . . 31
4.1.2 Pieces . . . 32
4.1.3 Quantitative Approach . . . 33
4.1.4 Obtaining ground truth . . . 33
4.2 Quantitative Metrics . . . 34
4.2.1 Precision, Recall and F1 . . . 34
4.2.2 Macro-Averaging . . . 34
4.2.3 Evaluation Results . . . 35
4.3 Processing time . . . 38
4.4 Qualitative Comparison To MelodyScanner . . . 38

5 Conclusions **40**

Bibliography **40**

A Guitar Terminology **44**

B Evaluation Pieces **45**

Chapter 1

Introduction

1.1 Motivation

Transcribing music is a laborious task and in many cases requires the knowledge and experience of a skilled musician in order to be performed well. This gives rise to the need for Automatic Music Transcription (AMT); the automated process of converting music played into musical score, such as staff notation or guitar tablature.

Transcribing classical guitar pieces presents its own unique challenges within the field of AMT. This is because although it is possible to accurately determine the pitch of a single note from audio alone, it is not possible to determine which string-fret combination was used to produce that note on the guitar without knowing the timbre [1] of the guitar being played. This means it is challenging to reliably determine the guitar tablature of a piece being played from raw audio data alone. Moreover, the guitar is a polyphonic [2] instrument. This further complicates the task of transcription because when overlapping notes with varying pitch, loudness and timbre are combined, it becomes very difficult to infer the musical attributes within the resulting signal.

Specialist hardware has also been developed for transcribing the guitar known as a MIDI guitar. These work by using seperate MIDI channels for each string in order to know the pitch of that string in real time, which can then be used to determine the fretting of each note. However, MIDI guitars are expensive and often have usability issues including a variation in the recognition time from one string to another.

To overcome these problems, AMT can also be treated as an inverse graphics problem provided a video of the performance is available. As is common with inverse graphics problems, AMT of classical guitar performances is an ill-posed [3] problem as a unique solution would require computing the depth in the scene, which is ill-posed by itself. To make this problem more feasible, this project aims to only transcribe videos recorded under the constraints listed in Section 2.2.2.

[1] The timbre of an instrument is a quality in its sound that is unique to that specific instrument

[2] Polyphonic instruments are able to produce two or more notes at the same time

[3] Ill-posed refers to not satisfying Hadamard's definition of a well-posed problem: https://en.wikipedia.org/wiki/Wellposed_problem

Guitar tablature is prefered amongst amateur guitarists as it is extremely simple to read, write and play from. Amateur guitarists often find it helpful to follow video tutorials for the songs they are wanting to learn, with many videos now being freely available online [4]. These videos can be filtered by resolution and then manually filtered to ensure they meet the constraints of this project. Large collections of guitar tablatures are available for popular songs online [5] however there may not be tablature available that matches what is played in the tutorial as every song can be played in multiple ways.

This dissertation aims to create a system capable of transcribing suitable videos into guitar tablature. This project looks to overcome the limitations of hardware and audio-based solutions and produce a transcription system that any guitarist could benifit from. In particular, project would assist beginner and amateur guitarists as they learn by producing the guitar tablature that corresponds to the tutorials they use. Furthermore, this project would serve as a useful tool to any classical guitarist as it can either fully automate the process of transcribing videos, or assist them with the transcription process when they use its annotated video output.

1.2 Guitar Tablature

The classical guitar is similar to an acoustic guitar but with the main differences being that the neck does not narrow and it typically uses nylon strings instead of metal. Please see appendix A for a labelled diagram of a classical guitar to clarify any guitar terminology that is unclear in this work.

Guitar tablature, often shortened to "guitar tabs", is a form of musical notation used specifically for the guitar. Unlike staff notation which indicates the pitch of each note played, guitar tabs describes the string and fret used to produce each note by reading off the row for the string and the number for the fret. Guitar tabs can be conveniently stored as ASCII characters in a text file. An example of guitar tabs showing a pentatonic scale[6] pattern is shown in Figure 1.1.

```
e|-------------------------------3--5--
B|-------------------------3--5--------
G|--------------------2--5--------------
D|--------------2--5--------------------
A|--------3--5-------------------------
E|--3--5-------------------------------
```

Figure 1.1: Guitar tablature for the pentatonic scale pattern starting at fret 3

[4] https://www.youtube.com/results?search_query=classical+guitar+tutorial finds thousands of videos
[5] https://www.ultimate-guitar.com/explore has an archive with over 1,100,000 songs
[6] The pentatonic scale is a musical scale that has five notes within the same octave

1.3 Related Work

AMT is an active area of research within the field of signal processing[9] [15]. There are three main catagories of AMT systems, these are: human assisted systems, instrument specific systems, genre specific systems. This project is concerned only with instrument specific systems, in particular, for the classical guitar. This project is focusing on just transcribing the classical guitar and so would be considered a specialist AMT system.

Audio based AMT systems typically involve using onset/offset detection to recognise the occurance of a note as well as multi-pitch detection to find the note pitch. More advanced systems may also aim to estimate the loudness of each note, the length of each note, and rhythm of play. Arndt and Li (2012) [5] developed an end-to-end system for transcribing monophonic guitar audio producing excellent results. Their system was able to achieve over 90% accuracy across a range of monophonic [7] guitar pieces. Lance and Nelson (2012) [4] produced a similar system but allowed polyphonic pieces. On a small dataset of 22 files, they achieved 60% accuracy in onset detection, 70% accuracy in pitch detection and an 85% accuracy in chord labelling. This demonstrates the challenge that polyphonic music presents within AMT. A major limitation of audio based systems is their inability to distinguish which string-fret combination was used to play a note which can result in incorrect tablature even for correctly identified notes. Barbancho et al (2012) [8] proposes a method for handling the ambiguity of fingertip positions from audio by using a Hidden Markov Model and heuristics to decide the most likely fingertip positions given the sequence of notes.

AMT systems have been developed that use visual information to aid the transcription process. Gillet and Richard (2005) [14] used both audio and visual features to transcribe drum sequences, resulting a note recognition rate of 85.8%. This was done to help cope with the polyphonic nature of the music, and because some musical information is easier to see visually. Similarly, McGuinness et al [25] found that vision and audio information were complimentary during the transcription process, for example, using vision showed improvement over audio-only approaches for the detection of tom and cymbal hits.

Akbari and Cheng (2015) [3] used a computer-vision based automatic music transcription system for the piano. This system used a fixed camera above the keyboard and had a high F1-score over 0.95 and a latency of only 7.0 ms. This approach relied heavily on classical computer vision techniques such as the Hough transform. More recently, Li et al (2020) [21] developed a similar system that was robust to changes in illumination. This system used semantic segmentation for keyboard detection and a CNN classifer to detect pressed keys.

Work has been completed on detecting and tracking guitar features using Computer Vision. Many early methods relied on markers present on the guitar. Motokawa and Saito (2006) [26] used markers to track the pose and position of the guitar and simulate playing the guitar in Augmented Reality (AR). Chutisant and Hideo (2007) [20] tracked fingertips with coloured markers by integrating Bayesian classifiers into particle filters. Since then multiple marker-less methods to fingertip detection have been developed. Kerdvibulvech et al (2007) [19] proposed a marker-less method for fingertip detection that worked by first finding the skin region using a Bayesian

[7] Monophonic music has only one note occuring at any point in time. Polyphonic has multiple notes occuring at once.

classifier and then applying a local-linear-mapping (LLM) network to a low-dimensional representation of the image. Burns (2007) [11] developed a system capable of tracking guitarists fingertips as well as guitar strings and frets. The Burns (2007) evalaution shows the relative effectivness of using projective signatures, geometric properties, the circular Hough transform and coloured markers for fingertip detection. Notably, geometric properties performed the best after coloured markers, having high accuracy and precision and its only poor performance occuring in complex backgrounds. Scarr and Green (2010) [28] developed a system involving string and fret detection as well as fingertip detection. The Scarr and Green fingertip detection method scored an accuracy of 69% across all fingertip positions being tested. However, they only used the first three frets and did not detect the onset of any notes whereas this project will use the first 11 frets and detect note onsets automatically.

Chapter 2

Preparation

This chapter discusses the work completed in preperation for this project. Section 2.1 explains the starting point for this project, detailing any relevant knowledge and experience gained prior to starting. Section 2.2 details the core and extension success criteria of this project followed by an outline of the deliverables and risks associated with the project before defining the input constraints required to make this project feasible. Section 2.3 describes the theory that underpins this project. This chapter concludes with Section 2.4 which describes and justifies the software engineering tools and techniques used throughout the completion of this project.

2.1 Starting Point

- At the start of this project I had some experience using Python, NumPy and Matplotlib from the Part 1A numerical computing and Part 1B Data Science courses. I also had experience using Python during a summer internship and from personal projects.
- At the beginning of this project, I had very limited experience using the PyCharm IDE, though I was familiar with the IntelliJ IDEA which is very similar.
- I had no prior taught knowledge of Computer Vision or any experience using the OpenCV and Scikit-learn library.

2.2 Requirements Analysis

This section begins with a discussion of the core and extension success criteria for this project, all of which were met. It then disucusses the risks and deliverables associated with the success criteria and how these risks where mitigated.

2.2.1 Success Criteria

I outlined in my project proposal the core success criteria of the project and a number of possible extensions. Upon completing this project all of the success criteria were met. During the early phase of my project I worked to further research the possible extensions included in

my project proposal and decide which are of them are feasible given the time constraints of the project, how well they suit the aims of the project and how well they lend themselves to a quantitative evaluation.

This project has met all of its success criteria, which are the following:

1. **(Core)** Successfully implement each functional component required for transcribing videos of classical guitar performances using Computer Vision. These functional components are:
   (a) Background Subtraction
   (b) Soundhole Detection/Tracking
   (c) String Detection
   (d) Fret Detection
   (e) Fingertip Detection
   (f) Plucking Detection
   (g) Tablature Generation
2. **(Core)** Be able to transcribe monophonic and polyphonic music into guitar tablature.
3. **(Extension)** Develop a more robust implementation that performs well under relaxed constraints. This includes working under poor lighting conditions.

2.2.2 Input Constraints

In order to make this project more feasible and fall within the scope of a part II project, I have defined input constraints on the videos this system is expected to transcribe well for the in order for core criteria to be satisfied. There criteria are:

1. The videos will be recorded from a stationary camera with the guitar facing the camera.
2. The soundhole and strings will be clearly visible.
3. The scenes will be well lit with no drastic changes in lighting occuring during the recording.
4. The music will not use bar chords[1].
5. Only the first 11 frets and the nut[2] on the fretboard will be used during playing.
6. No peripheral devices such as a capo are used.

[1]Bar chords are a type of guitar chord where a finger holds down multiple strings.
[2]The large white fret at the end of the fretboard (fret 0)

2.2.3 Project Deliverables

This project can be divided into several components. Table 2.1 shows each deliverables as well as its type, priority and difficulty. These are defined as:

1. **Type** (Functional, Non-Functional) - The deliverables labeled Functional are components implimented that provide functionality for the system. The deliverables labelled Non-Functional are components required to measure the performance of the system.

2. **Criteria** (Core, Extension) - The deliverables labeled Core are required in order to satisfy the core success criteria. The deliverables labelled Extension are optional components not required in order satisfy the core success criteria but would improve the technical complexity of the project.

3. **Difficulty** (Low, Medium, High) - This indicates the relative difficulty in achieving the deliverable.

Deliverable	**Criteria**	**Type**	**Difficulty**	**Risk**
Background Subtraction	Core	Functional	Low	Low
Soundhole Tracking	Core	Functional	Low	Medium
String Detection	Core	Functional	High	High
Fret Detection	Core	Functional	High	High
Fingertip Detection	Core	Functional	High	Medium
Plucking Detection	Core	Functional	Medium	Medium
Improve performance under relaxed constraints	Extension	Functional	High	Low
Quantitative Evaluation	Core	Non-Functional	Medium	High
Qualitative Evaluation	Core	Non-Functional	Medium	High

Table 2.1: Project deliverables breakdown

2.3 Computer Vision Theory

In this section I explain the relavant theory that underpins this project. This includes how images are represented and some of . Lastly, discuss the image correspondance problem and its relation to the optical flow problem.

2.3.1 Thresholding

Thresholding is a very simple technique used commonly in Computer Vision for the task of segmenting images. Thresholding works by converting a grayscale into one where only two values are used, usually 255 (for white) and 0 (for black). There are three implimentations of thresholding used in this project:

1. Simple Thresholding

Simple thresholding works by comparing intensity value of each pixel in the image to a **fixed** threshold value and if it is less than or thethreshold value the mask pixel has a value of 0, otherwise it has a value of 255.

2. Adaptive Thresholding

   As simple thresholding is highly susceptible to changes in lighting affecting the intensity of the grayscale pixels. To account for this, it is sometimes better to calculate the threshold value for each pixel as a function of its neighbouring pixel values. An example function commonly used is to calculate the mean for an $N \times N$ window around the pixel and subtract a constant.

3. Otsu's Method Thresholding

   Otsu's method aims to find the optimum global threshold for an image. It assumes there are only two classes in the image, foreground and background, and looks to minimise the weighted within-class variance across the two classes. This is defined as finding t (intensity value) that minimizes:

$$\sigma_w^2(t) = (\sum_{i=0}^{t} P(i))\sigma_1^2(t) + (\sum_{i=t+1}^{N} P(i))\sigma_2^2(t)$$

where $\sigma_c^2(t)$ is the variance of a gaussian fit for class c.

Figure 2.1: Total within class variance for the two-class problem

2.3.2 Morphological Transformations

Morphological transformations are operations to change a mask by convolving it with a binary *structuring element*. The structuring element used decides the nature of the transformation applied.

Erosion and dilation are the two most basic morphological transformations. Eroision works by performing a 2D convolution across the image with the kernel. The pixel in the output image will be 1 when **every** pixel under a kernel value of 1 is also 1. This means that close to an edge, not every pixel will satisfy this and so the thickness of the edge will be decreased giving an eroded effect. Similarly, dilation works by having having an output value of 1 when **any** pixel under a kernel value of 1 is also 1. This results in edges getting thicker and appearing dilated.

Erosion and dilation can be combined to have certain intended affects as well. A morphological *open* refers to erosion followed by dilation and a morphological *close* refers to dilation followed by erosion. Open is useful for removing small white regions within the mask, such as those caused by noise. Close is useful for fills any small patches of black within a white region of the mask.

2.3.3 Image Smoothing

When detecting edges it is often neccessary to first smooth the image. This is because noise in the image often lead to lots of small spurious edge information that is not useful in finding prominent connected edges. Small spurious edge information is caused by high frequency variations in value. Smoothing reduces this high frequency information by altering each pixels' value as a function of its surrounding pixel values. This is done by convolving a smoothing kernel with the image where the kernel approximates a smoothing function. Gaussian blurring is a smoothing method that is highly effective at removing gaussian noise from an image. The filtering kernel is calculated as a discrete approximation to the Gaussian function described in Equation 2.1.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

2.3.4 Canny Edge Detection

Edge detection is a technique commonly used in image processing. Edges are often very informative features of an image as they usually indicate object shape and position. Edges exist in other domains than just luminance, for example colour gradients and edges are highly infomative for semantic segmentation.

The Canny edge detector works by first smoothing an image to reduce the noise within the image. Next a *Sobel* filter is applied to the image, both in the horizontal and vertical directions, producing the horizontal edge image G_x and the vertical edge image G_y. G_x and G_y approximate the horizontal derivative and vertical derivative of I respectively and are calculated as shown in Equation 2.2.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I \tag{2.2}$$

Using G_x and G_y, the edge magnitude and direction can be calculated as shown in Equation .

$$|G| = \sqrt{G_x^2 + G_y^2} \qquad \theta = \tan^{-1}(\frac{G_y}{G_x}) \tag{2.3}$$

Next the edges are thinned by *non-maximum suppression*. Each pixel is compared to its neighbours in the direction of the gradient. If the pixel is not the local maximum of its neighbours it is suppressed by being set to zero. The final stage of the Canny edge detector is *Hysteresis Thresholding*. The edge magnitude of each pixel is checked against a min-value and max-value. Any edges with magnitude over max-value are considered strong edges and any edges with magnitude below the min-val are disregarded. Lastly, any edges with gradient magntides in the middle are only kept if they are connected to strong edge.

2.3.5 Hough transform

The Hough transform[12] is a voting-based feature extraction technique for detecting curves within an image represented in their parameterized form. The Hough transform for straight lines is an extremely useful tool within Computer Vision and is often used in combination with the Canny edge detector to solve the task of line detection. The Hough transform works by the following procedure:

- Each pixel in the edge image votes on the lines in the parameter space (r, θ) that pass through them.
- If a line has enough votes, the accumulator bin for that the line parameters fall into is found and incremented. The accumulator space has the same dimensions of the parameter space.
- By finding local maxima within the accumulator space, the most likely line candidates are found.

2.3.6 Hierarchial Clustering

Hierarchical clustering[27] is a clustering method that aims to build a hierarchy of clusters, often represented by dendrogram. Unlike other clustering algorithms, such as K-means[18] the number of clusters does not need to be known before hand. There are two main types of hierarchial clustering:

1. **Agglomerative:** This is a bottom-up approach to building the dendrogram. Each data point starts as its own cluster and are merged.
2. **Divisive:** This is a top-down approach to building the dendrogram. A single cluster starts by containing all data points which then splits, with each new cluster then undergoing the spliting process.

The result of the hierarchial clustering algorithm heavily depends on the *metric* and the *linkage criteria* used. The metric, such as euclidean distance, manhattan distance or maximum distance will determine how the distance between data-points is measured. The linkage criteria, such maximum-linkage clustering or minimum-linkage clustering determines how the distance between sets of data-points is measured.

2.4 Software Engineering Techniques

As this was a substantial project, it was vital that I undertook it with proper software engineering principles in mind. Following a clear methodology, a consistent code style, testing where appropriate and conforming to appropriate legislation all mean the project is kept on track and produces a robust, well written, extensible product should any other developer wish to work on it.

2.4.1 Development Workflow

Throughout this project I used an agile development methodology where work was divided into two week sprints[23]. During these two week slots I looked to make **visible** progress on a deliverable component. I looked to complete each component as quickly as possible so that I could begin to evaluate the system from end-to-end. I then began iteratively improving each component by focusing on the poorest performing component in each two week sprint.

2.4.2 Code Style

Guido van Rossum, the creator of Python, listed one of it's design principles as 'Readability counts.' [2]. - this is for good reason. I followed the PEP 8 [1] styling guide for Python which was automated by the PyCharm insepections functionality. I also included comments and docstrings in my code where appropriate.

2.4.3 Testing

Due to the nature of this project, most of the functions implimented did not lend well to unit testing as a large number of components are feature detection. Instead, I opted to test the system in an end-to-end manner by running the program and viewing the output of each functional component as the frames are transcribed. This was helped by storing and displaying an annotated version of each frame showing its features detected.

2.4.4 Languages, Libraries and Tools

This project was implimented in Python 3.8, a popular language for Computer Vision projects. A key library chosen for this project was OpenCV (Open Source Computer Vision Library), which is written in C++. OpenCV offers wrappers, along with updated documentation, for both Python and Java. OpenCV was chosen as it is open source and offers highly optimized implimentations of computer vision and machine learning algorithms. The IDE chosen for this project was PyCharm due its integrated support for version control, debugging, refactoring and enforcing the PEP 8 python styling standard.

I used a private github repository to version control of source code for this project, making sure to commit regularly. I used a seperate private github repository to store the latex files for this dissertation. I chose `git` over alternatives such as `hg` (mercurial) because I most familiar and experienced with git; this helps minimises the new tools I am required to learn when completing this project.

Table 2.2 lists each of the third-party libraries used for this project as well as their licences. All of these licences allow for them to be used and redistributed with my software.

Library	Version	Purpose	Licence
matplotlib	3.4.1	Plotting and visualizing evaluation results	PSF Licence
numpy	1.20.2	Impliments array data structures and algorithms	BSD Licence
opencv-contrib-python	4.5.1.48	Impliments computer vision algorithsm	BSD Licence
scikit-learn	0.24.1	Impliments clustering algorithms	BSD Licence
pillow	8.2.0	Saving and loading images	HPND Licence

Table 2.2: Summary of libraries used

Chapter 3

Implementation

Having read the relevant theory and documentation, I began the implementation stage of this project. This chapter begins with an summary of the project methodology. The rest of this chapter describes, in detail, how each stage of the frame transcription algorithm was implimented. This chapter explains how each frames' background is subtracted, it details how the guitar features are detected on each frame, it describes how the fingertips of the guitarist are detected and finally it shows how string plucks are detected. This chapter concludes with an overview of the project repository.

3.1 Methodology

Each frame of the video is processed in order, sequentially, and is transcribed individually. Once all frames have been processed, a text file containing the guitar tablature is generated. Each frame undergoes a sequence of distinct stages which are the following:

1. The frame background is subtracted by comparing the current frame to the first frame of the video. The absalute difference between the images is found and simple thresholding is applied to produce a foreground mask.

2. The first feature to detect is the soundhole of the guitar. Using the foreground image, the guitars' soundhole is detected by using a *Circular Hough Transform* to find candidate circles and the smallest circle is assumed to be the soundhole. Soundhole detection occurs every $^1/_3$rd of a second, while every other frame tracks the previousl frames soundhole using a *CLSR tracker*. As part of the extension work, the option to manually select the soundhole region was also implimented.

3. Next, a decision is made as to whether detect the features of the current image. If the last detected note was recent then the features of current image may be hard to detect reliably due to strings blurring or touching, therefore:

    - If there was a note, the detected strings, frets and fingertip positions from the note Frame are reused again for the current Frame instance.

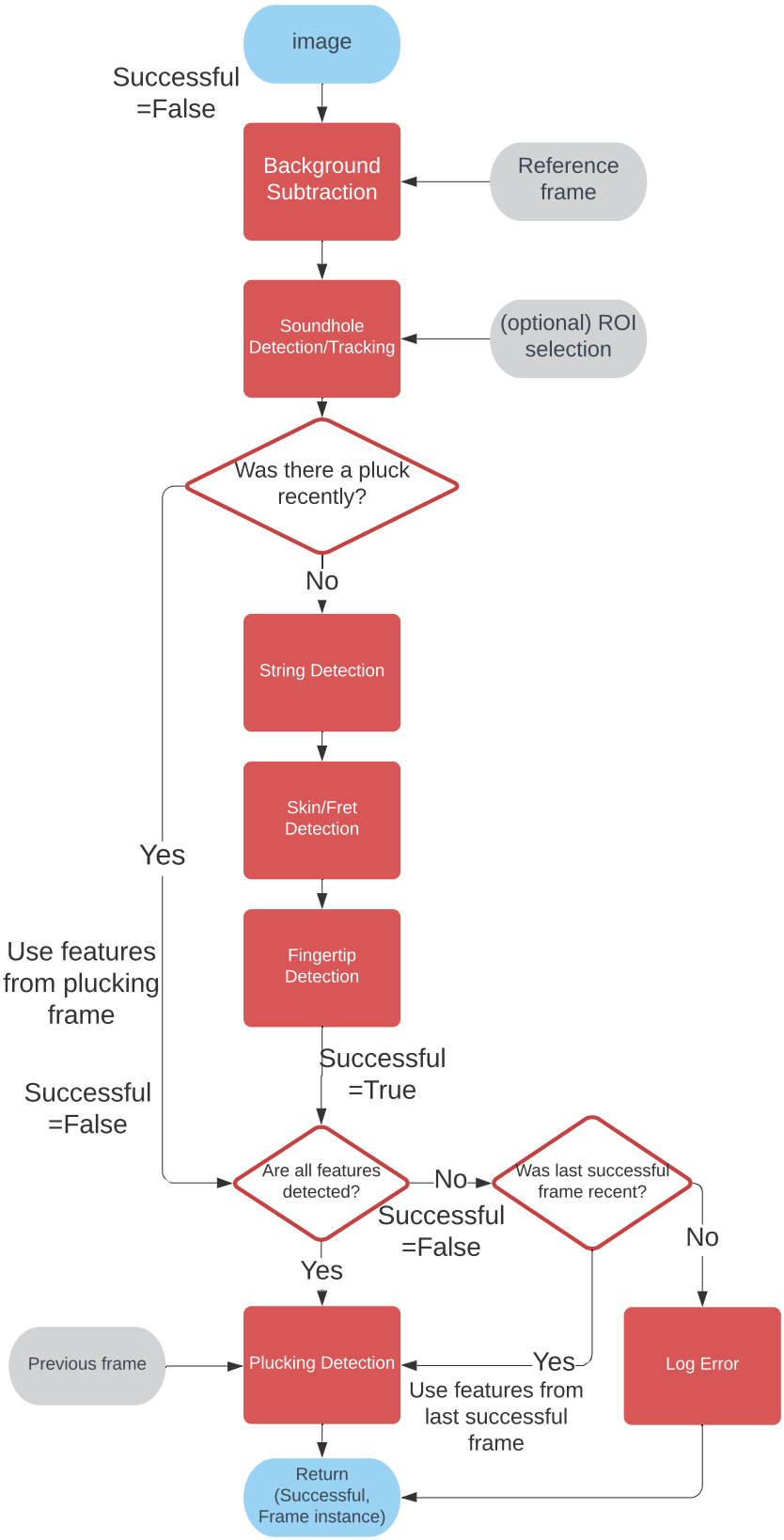
- Otherwise, the strings of the guitar are detected. Skin detection occurs on the fretboard of the guitar which is neccessary for the accurate detection and replacement of frets occluded by the hand. The detected skin contour is used again to find the positions of any fingertips on the fretboard.

4. If any of these features were not detected then an attempt to recover is made by opting to use the features of the most recent frame that did successfully detect it's own features. The stage that failed as well as it's reason for failure is logged to a text file.

5. Using the features found so far, a map from each string number to the fret currently being pressed down for that string is built. However, to detect when a note occurs signs of a string being plucked must also be detected. This is done by tracking the motion of points along the guitar strings on adjacent frames.

6. The string-fret combinations of any plucked strings are then sent to `TabsGenerator` to be included in the generated tabs text file once every frame has been transcribed.

Figure 3.1: Frame transcription algorithm



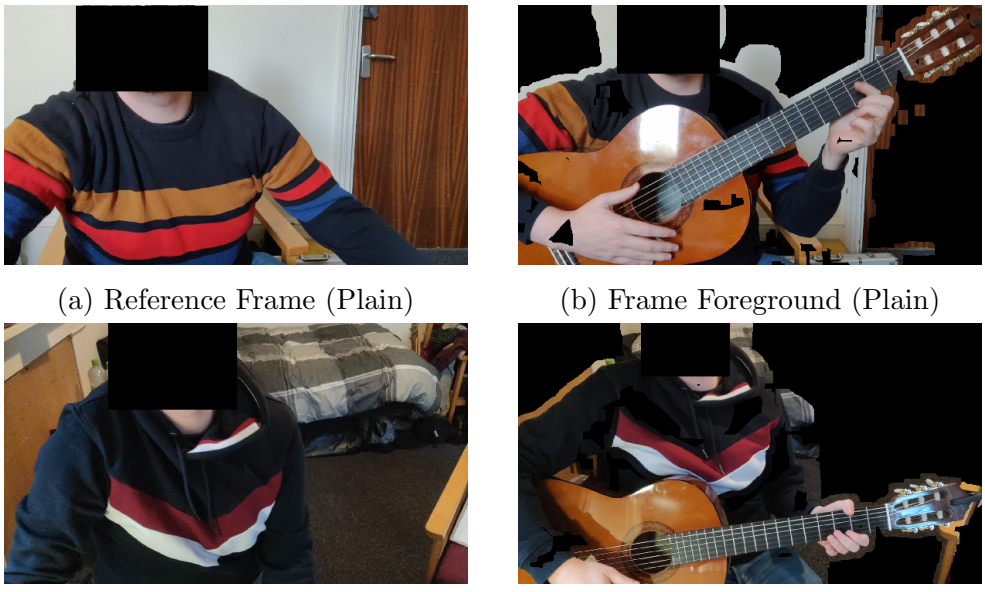
3.2 Background Subtraction

Background subtraction is an approach used to find the foreground in an image and is useful for simplifying processing done later on the image. Background subtraction makes the soundhole detection stage of the frame transcription algorithm more reliable as it removes areas of the image that do not contain the guitar, and could contain circular objects.

OpenCV offers a number of advanced methods for background subtraction, many of which rely heavily on machine learning. For this project, as the camera is static and with the background being almost entirely static, it is both reasonable and more efficient to simply store first frame of the video as a reference frame and use that to calculate the foreground for the current frame.

Two implimentations of background subtraction were implemented. These were the methods `_get_foreground` and `_get_foreground_convex_hull`. Both use the same approach to find the foreground mask but `_get_foreground_convex_hull` then finds the convex hull of the mask to ensure there are no holes are in the foreground; in practice, this was unnecessary.

When pre-processing each frame, a gaussian blur with a kernel size of 21 is applied and the image is grayscaled. This large kernel size was chosen as it produces a coarse-grained foreground. A matrix of the absalute difference between the pre-processed current frame and reference frame is then calculated. Given a static background and camera, as well as no drastic changes in illumination, you would expect pixels in the background to keep roughly the same grayscaled value over time. Therefore I apply threhsolding to the absolute difference matrix with the assumption that a difference over 25 is part of the foreground. The resulting foreground mask is then dilated to fill in any small patches that were incorrectly recognised as background. The foreground mask is converted to RGB and applied to the image by an element-wise AND operation, so all pixels detected as the background are set to [0,0,0] which is black in the RGB format while all foreground pixels keep their value giving just the foreground. Figure 3.2 shows the result.



(a) Reference Frame (Plain)

(b) Frame Foreground (Plain)

(c) Reference Frame (Room)

(d) Frame Foreground (Room)

Figure 3.2: Example frames after background subtraction is applied

3.3 Soundhole Segmentation

Having isolated the foreground of the current frame it is now possible to begin detecting the features of the guitar and guitarist. The first feature to be detected is the guitar soundhole.

Two implimentations of this were written as part of the core and extension work. The core implimentation uses a circular hough transform to detect the soundhole at regular intervals and uses a CSRT tracker to track the last detected soundhole. The extension implimentation allows the user to select an initial region of interest (ROI) for the soundhole which is tracked using a CSRT tracker throughout the video. This decision was taken because I discovered through experimentation that detecting the soundhole does not work accurately in low light conditions.

3.3.1 Soundhole Detection

Given an image of the foreground in the current frame, the image is smoothed and grayscaled before a circle Hough Transform is applied [31]. The circle Hough Transform works very similarly to how the standard Hough Transform for straight lines works (see Section 2.3.5). Each pixel in the edge image (I) votes on the circles in the parameter space of (r, x_0, y_0). This voting system is done by calculating the rate of change of edge intensity as r varies for the candidate circle. This derivative would be expected to be large if a circle was present, and circles with sufficiently many votes (K) are then returned as detected circles. Equation 3.1 demonstrates condition.

I call the OpenCV method `HoughCircles` to detect all candidate circles and I take whichever candidate has the smallest radius as the soundhole. This was chosen because I noticed it is far more common to get false positives from large circles than small ones.

$$\left| \frac{\partial}{\partial r} \oint_{(r,x_0,y_0)} \frac{I(x,y)}{2\pi r} \, \mathrm{d}s \right| \geq K \tag{3.1}$$

Performance considerations

A traditional implimentation of the circular Hough Transform is $\mathcal{O}(n^4)$ for an $n \times n$ sized image with more sophisticated methods giving a reduction in time complexity to $\mathcal{O}(n^3 \log n)$[16]. This can be further reduced to $\mathcal{O}(n^3)$ without using a gradient calculation[7] which does lose some performance. As the videos this system accepts are (1920×1080) resolution, it is neccessary to prune the parameter space being voted on in order to decrease the time taken to detect the soundhole. This was achieved by passing arguments to `HoughCircles` that constrain the radius of circle that can be voted on as well as not allowing two circles to be too close to each other. In the case of two circles near each other, only the circle with the most votes is returned.

3.3.2 Soundhole Tracking

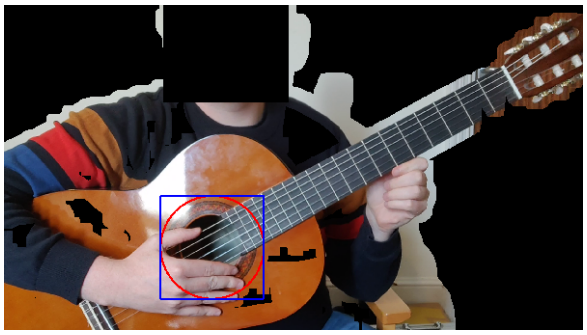
The `SoundholeTracker` class stores a CSRT tracker instance as internal private state as well as a tracking interval. The detection/tracking algorithm is as follows:

The detection and tracking algorithm:

- On each frame number divisible by the tracking interval, a circular hough transform is performed to detect the soundhole.
- If no circle is found, the tracking continues until the next interval.

The OpenCV implimentation of the CSRT tracker is based off Lukežič et al[22]. I chose the CSRT tracker implimentation based on its high accuracy in comparison to the alternative OpenCV tracker implimentations. It does have the trade-off that it has a much higher computational cost than many of the other OpenCV trackers but as real time computation is not a requirement of this project it made sense to choose it anyway[17]. Figure 3.3 shows an example of a successful detection of the soundhole despite partial occlusion and a large change in luminance near the soundhole edge.

(a) Soundhole detected with partial occlusion and luminance changes



(b) Soundhole tracked dispite occlusion from hand

Figure 3.3: Example of soundhole detection and tracking

3.4 String Detection

So far we have seen how the frame foreground is calculated and the guitar soundhole is detected. In this section I discuss how the strings of the guitar are detected. An outline of the string detection algorithm is given in Algorithm 1: For the rest of this section I will describe in detail how each stage of this algorithm works and its purpose.

Algorithm 1: String Detection

Input: Frame instance

Output: All 6 guitar strings parameterized

1. Estimate the borders of the guitar neck
2. Perform a perspective transform on the image using the neck borders and image boundaries
3. Detect candidate strings
4. Filter down to 6 strings

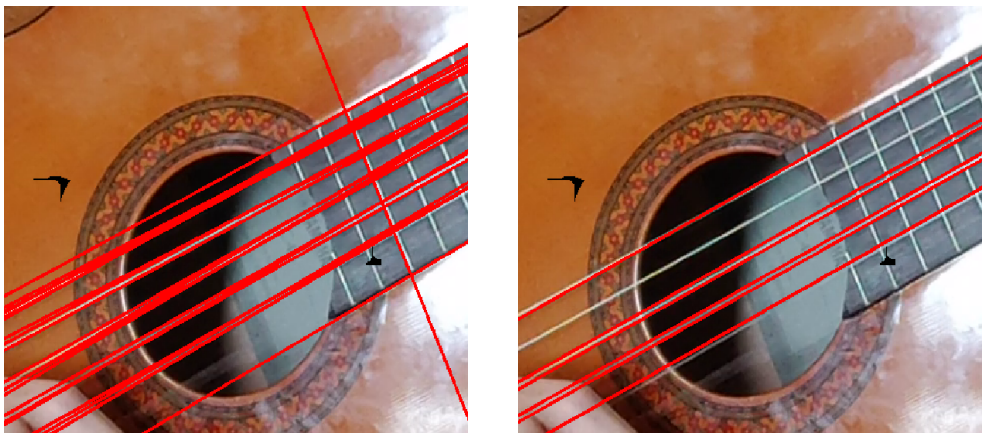
3.4.1 Neck borders estimation

Similar to Scarr and Green (2010)[28], I want to detect the borders of the fretboard as they define the region of the image containing the strings, frets and fingertips. I have chosen to adopt a coarse-to-fine strategy, where I first loosely bound the neck of the guitar to help with detecting strings, then using the top and bottom detected strings I create an accurate bound on the guitar neck (Section 3.5.1). After detecting the frets I can use the nut (fret 0) of the guitar and the location of the soundhole to precisely bound the fretboard of the guitar (see Section 3.5.3).

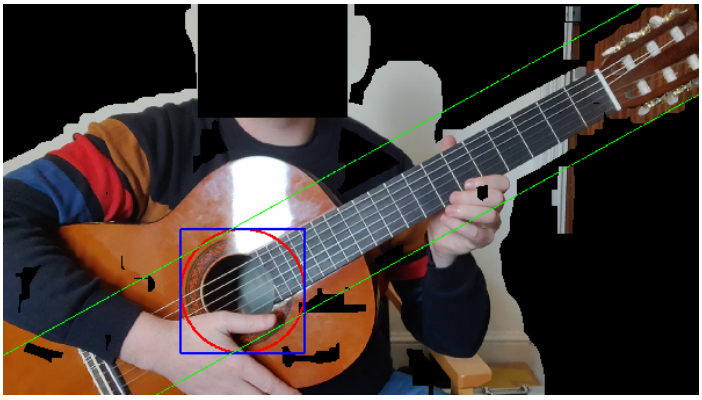
The function `estimate_neck_borders` produces two lines which are a loose bound on the guitar neck. This is done by the angle filtering method outlined as:

- Firstly the cropped soundhole image is smoothed and grayscaled in order to improve the performance of the Canny edge detector and Hough transform.
- A Hough transform produces a set of candidate lines each represented in polar form as a two element array $[r, \theta]$.
- The median angle θ_m is then calculated and all lines are filtered to be within $\theta_m \pm 0.01$. This is done because all of the strings and fretboard borders will have approximately the same θ and there will be many more of them detected than lines from frets or hands.
- From the remaining lines $[(\theta_1, r_1), (\theta_2, r_2), \ldots, (\theta_n, r_n)]$, the resulting θ_m will be approximately the angle of the neck borders.
- The estimate neck borders are returned as $[[\theta_m, r_m - \Delta], [\theta_m, r_m + \Delta]]$ where a large Δ is suitably large constant that was chosen to ensure the entire fretboard is contained within the border estimates.

The effect of filtering and the resulting neck estimates is shown in Figure 3.4.



(a) All detected lines (b) Lines remaining after angle filtering



(c) Resulting neck border estimates

Figure 3.4: Example of angle filtering on lines detected at soundhole

3.4.2 Perspective Transform

The next stage of the string detection algorithm is to perform a perspective transform. This is done to make it easier to dilate the string edges and detect them during the later stages of the string detection algorithm. The function `perspective_transform_into_rect` performs a perspective transform on the current frane using the neck borders and the boundaries of the image. The four intersection points between the neck borders and the image will be perspective corrected to become the corners of a rectangular image. The function `change_perspective` makes calls to the OpenCV method `getPerspectiveTransform` to calculate the (3×3) *homography matrix* M such that Equation 3.2 holds. M maps from the 2D projective image space to a 3D world coordinate system and where w is the scaling factor. M is chosen such that the four rectangular points given are in the same plane after M is applied to them.

$$w \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.2}$$

The width and height of the output image are then found by calculating the euclidean distance between the four sides of the rectangle and choosing the maximum width and height. Finally, the OpenCV method `warpPerspective` produces a perspective corrected image by applying M to the image, and scales the image to produce an output of the correct width and height.

An example of the perspective corrected image is given in Figure 3.5

Later on, it is neccessary to use the *inverse* transform M^{-1} to transform points back to the image coordinate system. M^{-1} is calculated using the `pinv` function in the NumPy linear algebra library which uses the method given in Dokmanić et al (2013) [13].

Figure 3.5: Example output after perspective transform

3.4.3 Candidate String Detection

Using the neck image, I can now try to detect the strings reliably. The function `detect_strings_from_neck` impliments the following algorithm:

Algorithm 2: Detecting candidate strings

Input: Perspective corrected neck image

Pre-Processing:

1. Apply Gaussian blur and grayscale the image
2. Apply Canny edge detection

Enhance String Edges:

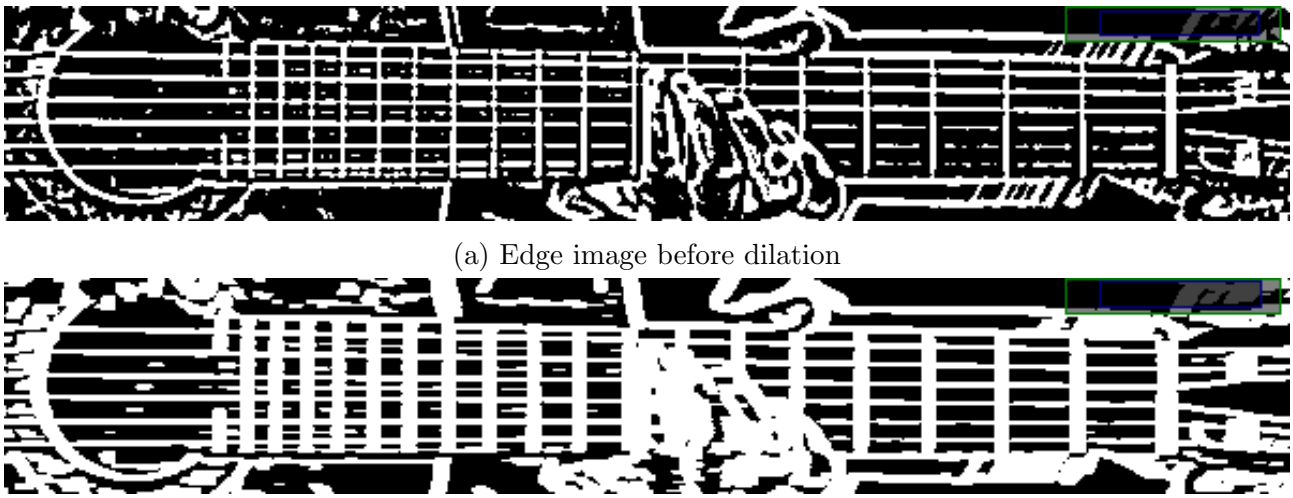
1. Dilate the image using a horizontal structuring element

Detect Candidate Lines:

1. Detect lines using a probabilistic Hough transform
2. filtering the lines by angle ($\theta \pm 0.02$)
3. Cluster the detected lines using agglomerative hierarchial clustering

Enhancing String Edges:

I apply a morphological dilation with a horizontal shaped structuring element (1×12) in order to make the string detection algorithm effective in low-light conditions. This horizontal structuring element was chosen because in the perspective-corrected neck image all of the string lines are almost exactly horizontal. Dilating with a horizontal element therefore has the effect of dilating along the fretboard, connecting the string edges together incase there are gaps. This is important because the bottom three strings are very thin and often have regions of the strings with no edge structure detected. This can be seen clearly on the first few frets of Figure 3.6.



(a) Edge image before dilation

(b) Edge image after dilation

Figure 3.6: Example of dilating the edge image with horizontal structuring element

Detect Candidate Lines:

To detect candidate strings, I used the OpenCV method `HoughLinesP` which impliments a version of the *probabalistic* Hough transform (PHT) described in Matas et al (1999)[24]. There are a number of differences between the PHT and the standard HT. Firstly, the PHT has a lower computation time than the standard HT. Moreover, the PHT returns line segments which are defined by a pair of coordinates as apposed the classical HT which returns the line in polar form with no information about the length or location of the detected line. The PHT therefore allows me to fine tune the lines I am trying to detect further than just using a voting threshold alone. In particular, I am able to only target **long, well connected line segments** that have the required number of votes. This was achieved by passing additional parameters to `HoughLinesP` such as the minimum accepted line length and maximum gap between voting pixels on a line.

Next, I apply angle filtering (as explained in Section 3.4.1) and cluster the lines using the `AgglomerativeClustering` class defined in the SkLearn Cluster library. This class impliments hierarchial clustering from the bottom-up. This clustering algorithm was chosen because it does not require knowledge the data to have a fixed number of clusters. The metric used was the euclidean distance between the lines in parameter space. The linkage used was ward, which aims to minimize the variance of the clusters being merged. Using an algorithm which looks for exactly 6 clusters, such the K-means, would be less reliable because often the borders of the neck are detected as strings which would shift the cluster centers. The effect of this can stage can be seen in Figure .

(a) All candidate lines

(b) Lines after clustering

Figure 3.7: String candidate lines after hierarchical clustered

3.4.4 Filtering down to 6 lines

We have seen how the neck of the guitar is isolated and how a perspective transform can be used to isolate the neck. We have discussed how string enhancment is performed as well how crucial its role is within the algorithm. We then saw how a probabalistic Hough transform can be applied with agglomerative clustering to produce a set of clustered string candidates. I will now explain how I ensure that exactly six strings are detected and the algorithm used to ensure the correct six are picked.

Assuming more than six string candidates have been detected, the filtering process to get just six is the following:

1. **Remove Outliers:**
   The line clusters are given as $[(m_1, c_1), (m_2, c_2) \ldots, (m_n, c_n)]$ in cartesian form. Continue to remove the line candidate with a c value furthest from the median c_m until 7 or less strings remain. Recalculate c_m after each removal.

2. **Evenly Space The Strings:**
   From the 7 strings, assume that the middle 5 are all strings, leaving 2 lines left to choose from as the 6th string. Calculate the average perpendicular distance between the 5 lines assumed to be strings, this is an estimate of the string spacing. From there, choose the 6th string to be whichever line has a perpendicular distance to its neighbour that is closest to the average distance.

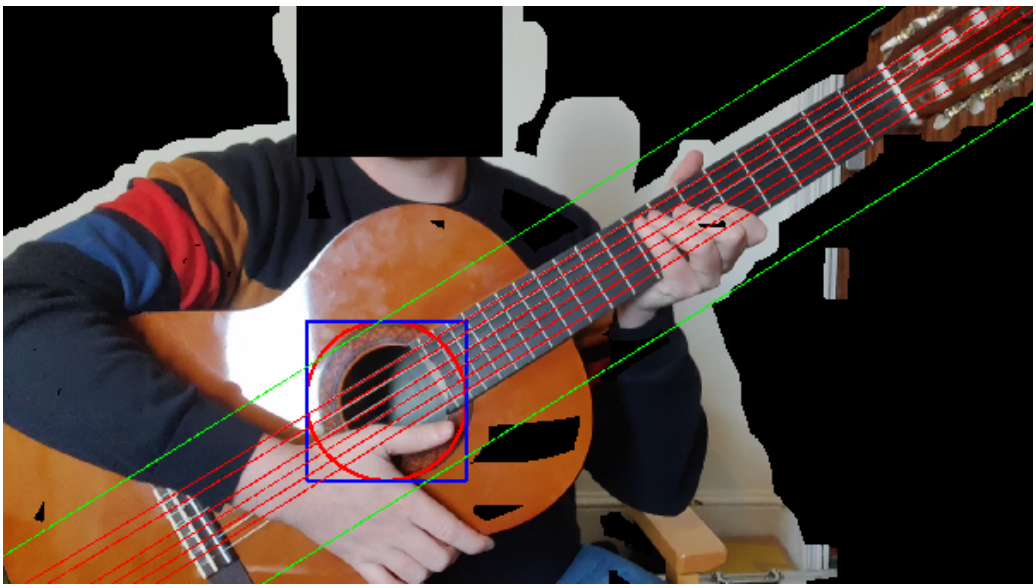


Figure 3.8: String detection algorithm output

3.5 Fret Detection

So far we have seen how the soundhole and strings of the guitar are detected. We are now going to discuss how the frets of the guitar are detected.

An outline of the method used to detect frets is given in Algorithm 3.

Algorithm 3: Fret Detection Algorithm

Isolate the fretboard:

1. Use top and bottom strings to perspective transform to the fretboard
2. Apply a perspective transform the center of the soundhole and remove any parts of the image to the left of the soundhole.

Pre-processsing:

1. Apply Gaussian blur and grayscale
2. Apply adaptive thresholding

Enhance fret edges:

1. Apply a morphological open using a vertical structuring element

Detect candidate frets:

1. Use a scan across the columns of the fretboard image to find all fret candidates
2. Clustering candidates by horizontal distance to identify all candidate frets
3. Assume the fret furthest to the right (largest r value) is the nut of the fretboard

Skin detection:

1. Crop the fretboard to remove columns to the right of the nut.
2. Apply Otsu thresholding
3. Find contour of skin and its extreme points

Remove and replace frets:

1. Remove all fret candidates under the skin region
2. Use the expected spacing of frets to replace missing frets

---

3.5.1 Fretboard isolation and pre-processing

A precise estimate of the neck borders is calculated using the top and bottom string. Another perspective transform is applied to the original frame. The neck image is cropped to remove anything to the left of the soundhole by first mapping the center of the soundhole to the fretboard image using the homography matrix M. The neck image is then smoothed and grayscaled to prepare it for adapative thresholding. Adaptive thresholding was chosen because the strings and the frets are very thin edges and have grayscale values that are drastically different to the monochromatic fretboard. The neck image after adaptive thresholding is shown in Figure 3.9.

Figure 3.9: Guitar neck after pre processing

3.5.2 Enhancing fret edges:

In order to only allow edges corresponding to frets to be present during the detection process a morphological open is applied with a vertical structuring element $(50, 1)$. The act of eroding the image vertically removes the horizontal edges from the string strings because they are thin. The vertical dilation then restores and enhances the vertical lines reduced during erosion. The edge image after fret enhancment is shown in Figure 3.10.

Figure 3.10: Guitar neck after morphological open with vertical structuring element

3.5.3 Detect candidate frets:

To detect candidate frets a scan line algorithm was implimented. Each column of the image is iterated across and if the more than half of the pixels in that column are white then it is considered a fret candidate. The array of fret candidates are then clustered into groups if they are within ϵ horizontal pixels of each other and the median of each cluster is chosen as a fret candidate. The rightmost cluster (largest r value) is assumed to be the nut of the guitar neck and the neck image is further cropped to now only show the fretboard, removing the headstock and background.

Figure 3.11: Candidate frets found

3.5.4 Skin detection:

As the hand on the fretboard could either cover up frets or produce false candidate frets, it is important to remove candidates near the skin region and replace them. To detect the skin, I use Otsu thresholding. This is an appropriate algorithm to use because the skin will appear as foreground on the mostly monochromatic fretboard. To remove strings and frets from the foreground a morphological open operation with a square kernel is used. The contours of the foreground are found using the OpenCV method `findContours` onto the skin mask. This works by implementing Suzuki and Abe (1985) [29]. As there can be multiple skin contours (such as 1 from each finger) a contour is considered skin if it meets all of the following criteria:

1. The contour touches the bottom of the image.
2. The contour area is sufficiently large.
3. The contour is not too wise (wider than the image width/3).
4. The contour is not too narrow. (fixed constant)
5. The contour is not too close to the soundhole (within image width/3). This does not stop fingertip detection at high frets because this project is constrained to only transcribe music that can only the first 11 frets of the fretboard.

Figure 3.12: Detected skin contours annotated on fretboard

3.5.5 Remove and replace frets:

All candidate lines with r values that fall within the left-most and right-most points of the skin contours are removed and replaced. The replacement algorithm works similarly to the one designed by Scarr and Green [28]. The closest pair of fret candidates are assumed to be correctly detected. From there, the algorithm works its way towards the right of the fretboard by considering each pair of frets in turn. Whenever the seperation between `(fret[i], fret[i-1])` is 1.4× larger than the seperation between `(fret[i-1], fret[i-2])` a missing fret is assumed to be at index `i`. Its parameters are calculated using Equation 3.3 and it is inserted in-place. Lastly, as this project only requires the use of the first 11 frets, the first 12 frets are stored as the extra fret is required for plucking detection (see Section 3.7).

$$\begin{aligned} r_i &= \frac{r_{i-1} - r_{i-2}}{0.94387} + r_{i-1} \\ \theta_i &= \text{mean}(\theta_1, \ldots, \theta_n) \end{aligned} \tag{3.3}$$

Figure 3.13: First 12 frets after replacement

3.6 Fingertip Detection

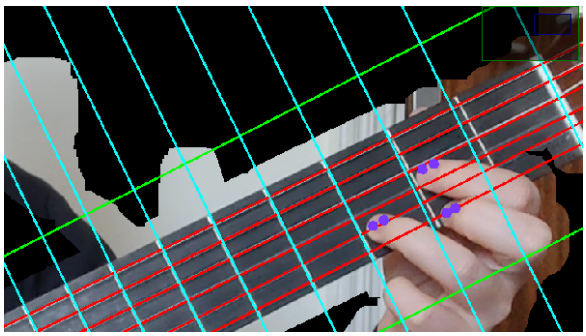
So far, we have seen how the guitar soundhole, strings and frets have been detected. We have also seen how using soundhole, strings and frets we were able to produce a tight bound around the guitar fretboard and a perspective transform had been used to normalize the spacing between the strings and frets. We also saw how by treating skin detection as foreground detection, we could apply Otsu thresholding to the fretboard image and find the contours corresponding the the skin on the guitar. This section discusses how the fingetips of the guitarists left hand are retrieved which will later be used to find the fretting of a note.

The approach used to detect fingertip positions from a skin contour is the following:

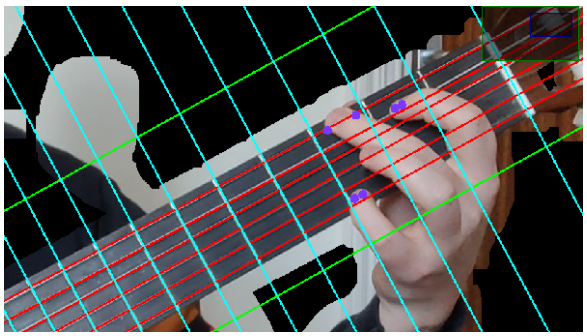
1. Approximate the skin contour to a polygon. Each point on the polygon is represented by the coordinates $[x, y]$ in the fretboard image.

2. Find all local maxima on the skin polygon.

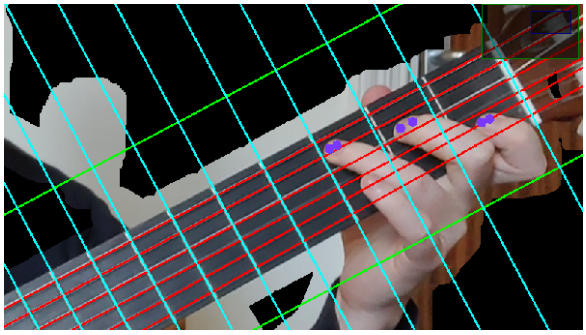
The OpenCV method `approxPolyDP` approximates the skin contour to a polygon. This method impliments the Doughlas-Peucker algorithm[30] for polygon approximation. This returns an array of points on the polygon in clockwise order. In order to find the local maxima of the polygon, we iterate through each point on the polygon and consider ϵ neighbours to its left and ϵ neighbours to its right. A point is considered a local maxima if its y-ordinate is equal to the max of its $2 \times \epsilon$ neighbours' y-ordinates. To account for the fact that the fingertip is slightly higher than where the guitarist applies pressure, we adjust each local maxima by moving it a fixed 10 pixels down in the fretboard image. Example output of the detected fingertips is shown in Figure 3.14.



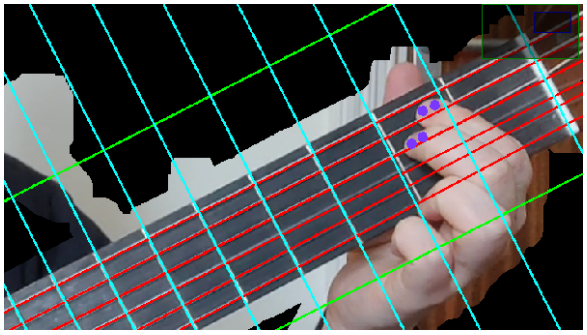
(a) D chord fingertips correctly detected



(b) G chord fingertips correctly detected



(c) C chord fingertips correctly detected



(d) E Minor chord fingertips correctly detected

Figure 3.14: Example images of detected fingertips

3.7 Plucking Detection

So far we have seen how all of the features on the guitar are detected as well as the fingertips on the fretboard. This has allowed up to build a map from each string number to the fret being held down on that string by assuming a fingertip presses the string it is closest to. This section will explain how a note is detected.

When a string is plucked, the act of pressing on the string and releasing the string causes a a very small motion perpendiculat to the strings direction. The strategy chosen and implimented to detect this motion was to track 1 point on each string between frames and calculate each points motion between frames, this is refered to as the optical flow of the point. From there, the component of the motion vector that is perpendicular to the strings direction is considered. Each Frame instance stores plucks as an array of 6 booleans to represent which of the 6 strings are plucked on that frame. The plucked strings will determine which string/pressed fret pairs already detected are included as notes for this frames transcription.

3.7.1 Optical Flow

OpenCV offers implimentations of both sparse optical flow and dense optical flow. However, since I am only interested in the motion of the strings in particular I decided to use sparse optical flow method `calcOpticalFlowPyrLK`, an implimentation of the Lucas-Kanade method [6]. This gives the advantage to requiring far less computation time for each frame than dense optical flow which would calculate the motion of every pixel in the image. For each string the point I track a (15×15) region of pixels at the midpoint of the 11th and 12th fret along the string. This produces a set of motion vectors $\overrightarrow{v_1}, \ldots, \overrightarrow{v_6}$ where $\overrightarrow{v_i}$ is the vector of motion for

string i.

Next, the component of motion that is perpendicular to each strings' direction is calculated. This is found by taking the dot product of v_i with s_i where s_i is a unit vector in a direction perpendicular to string i. This component is used because it is common for the tracked point to move slightly along the string and this would give no indication as to how far the string moved while being plucked.

The resulting scalar components of motion are used to find the onset of a string pluck. This is possible becausse during the onset and offset phases of a pluck, a string moves rapidly perpendicular to its direction for only a few frames. The motion perpendicular motion scalar p_i indicates a pluck if Equation 3.4 holds. By ensuring p_i is greater than 1, the string must atleast be moving a considerable amount to be considered plucked. Moreover, by using the median the system is are more robust to outliers which is very important for so few data points.

$$p_i > \max\left(2 \times \text{median}(p_1, p_2, \ldots, p_6), 1\right) \tag{3.4}$$

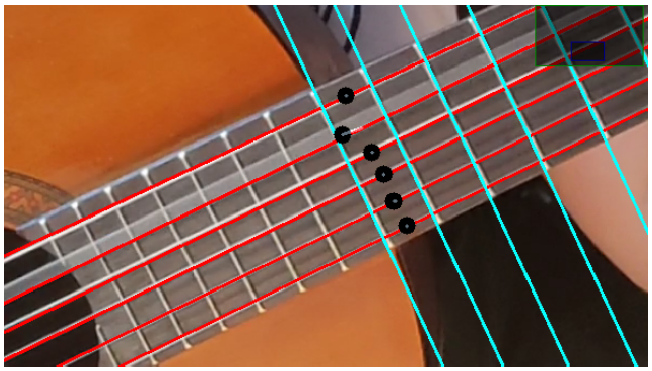


Figure 3.15: Example of motion detected on pluck (second string plucked)

3.8 Tab Generation

So far we have seen how each feature of the guitar and guitarist is detected. This section discusses the final stage of the video transcription process: how the notes detected are converted into guitar tabs.

3.8.1 Note timing constraints

After each frame has its notes generated, the `Engine` class sends (string,fret) pairs to the `TabsGenerator` instance. However, depending on the timing of a note, a note could either be included as a single note, paired with a recent note as a simultaneous pluck or discarded entirely.

Discarded Notes: If a note is recieved on a string that has been plucked within the last $\frac{2}{3}$'s of a second, then it is discarded. This is to cope with the fact strings will typically vibrate and oscillate for a considerable amount of time on each pluck, which will mean multiple notes are detected. This allows for a string to be plucked 90 times per minuete, which is reasonable as that is a fast tempo.

Simultaneous Notes: If a note is a recieved, not discarded and the last note added to the tablature occured within $\frac{1}{2}$ a second of the current note, then the two notes are paired together as if they were detected on the earliest notes frame. This could happen for all six strings simultaneous.

Singular Notes: If a note is not discarded or paired with another note in the tablature, it is included as a singular note. The `_last_pluck_frame` field on the `TabsGenerator` instance is set to the current frame number and is used for the next notes' timing calculations.

3.9 Extensions

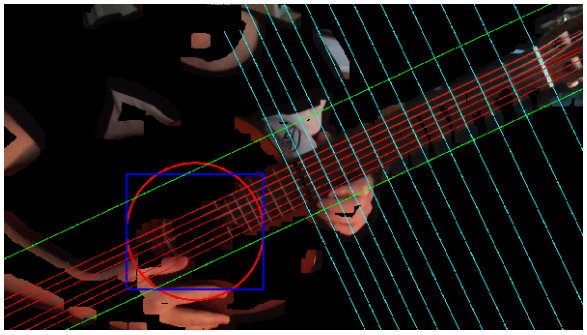
Transcription with relaxed constraints

To allow for the transcription of videos under relaxed constraints I have implimented manual soundhole selection. This works by prompting the user selecting an initial region of interest (ROI) containing the guitar soundhole early in the video. If the soundhole is not visible on the first frame of the video, the user can choose to skip forward by 1/3rd of a second until it is visible. Once an initial ROI is selected, that region is tracked for the remainder of the video as described in Section 3.3.2.

Justificaion for this is given below: In low-light conditions, the `HoughCircles` method is likely to fail at detecting the soundhole. This is because it first applies a Canny edge detector to its input before performing the Hough Transform. The Canny edge detector is unreliable in low-light enviroments unless its parameters have been tuned. parameters are carefully tuned for each image. Without correct edge information, the circular Hough transform will fail to detect the soundhole. Furthermore, even with correct edge information, the `HoughCircles` aims to detect perfect circles and is less reliable at detecting the soundhole when it is at a steep angle to the camera. Therefore, I concluded that a circular Hough transform approach would be unable to work reliably under relaxed constraints.

Figure 3.16 shows an example frame where the user is prompted to draw the soundhole ROI as well as the successful annotation of that frame once it is annotated. This figure also shows how the implimentation of background subtraction removes too much in low light conditions.

(a) Dark video frame before being processed. The user sees this in a window titled 'select soundhole ROI'.



(b) Annotated frame once ROI is drawn. String and frets are detected successfully, with no false fingertips.

Figure 3.16: Example of manual soundhole detection on a dark input video

3.10 Repository Overview

The following is a high level overview of the repository. I have included example video files, audio files as well as their respective ground truth representations in the evaluation_data directory. The README.md explains how to run the program as well as more details on how to perform the evaluation. All of the code was written from scratch.

```
GuitarTranscription/
├── evaluation_data/.....Contains example videos, audio files and ground truth data
├── scripts/.....Contains functions for performing the evaluation
│   ├── collect_ground_truth.py
│   └── evaluate.py
├── src/.....Contains the source code of GuitarTranscription
│   ├── core/.....Contains the Engine class and program entry point
│   │   ├── main.py.....Program entry point
│   │   └── engine.py.....Impliments the Engine class
│   ├── frame_processing/.....Contains code for processing each frame
│   │   ├── frame.py.....Impliments the Frame class
│   │   ├── frame_processor.py.....Impliments the FrameProcessor class
│   │   ├── background_subtraction.py.....Impliments the BackgroundSubtractor and BackgroundException classes
│   │   ├── soundhole_tracker.py...Impliments the SoundholeTracker and SoundholeException classes
│   │   ├── string_detector.py.....Impliments the StringDetector and StringException classes
│   │   ├── fret_detector.py.....Impliments the FretDetector and FretException classes
│   │   ├── plucking_detector.py.....Impliments the PluckingDetector and PluckingException classes
│   │   └── fingertip_detector.py.....Impliments the FingertipDetector and FingertipException classes
│   ├── tabs_generation/.....Contains code to validate and generate guitar tablature
│   │   └── tabs_generator.py.....Impliments the TabsGenerator class
│   └── util/.....Contains utility functions used throughout the program
│       ├── geometry.py.....Contains code for geometry and perspective calculations
│       └── opencv_helper.py.....Contains code for loading, annotating and displaying images
├── tests/.....Contains unit tests for pure functions
└── README.md
```

Chapter 4

Evaluation

4.1 Evaluation Method

As my goal was to evaluate this project in a systematic and thorough manner I have chosen to use a dataset of five scenes and five musical pieces, totally 25 videos. Each scene represents a set of enviromental conditions for the input video, and each piece ranges in musical complexity and the demands it places on the system.

4.1.1 Scenes

The five scenes use two different backgrounds, two different orientations for the guitar and three types of lighting. Table 4.1 gives a description of each scene in terms of these components. By using different backgrounds, we see how well the background subtraction component performs. I chose to vary the orientation of the guitar to see how well the system would cope with different people holding and playing the guitar differently. As systematically approach to varying lighting was taken, with the following three standards of lighting defined:

- **Good lighting:** Lots of natural light entering the room and the bedroom light on.
- **Medium lighting:** Some natural light entering the room and the bedroom light off.
- **Poor lighting:** No natural light, bedroom light off, only a desklamp on.

Scene	**Background**	**Guitar Angle**	**Lighting**	**Run Mode**
Scene 1	Room	Horizontal	Good	Core
Scene 2	Room	Diagonal	Good	Core
Scene 3	Plain	Horizontal	Medium	Core
Scene 4	Plain	Diagonal	Medium	Core
Scene 5	Room	Diagonal	Poor	**Extension**

Table 4.1: Scenes breakdown

4.1.2 Pieces

Appendix B shows the tablature for each of the pieces described below. Five pieces were chosen to evaluate this project, each ranging in overall musical complexity and each focuses on a particular aspect of guitar playing that the system should hope to transcribe. Table 4.2 shows how each piece scores in terms of the complexity for the right hand, called string complexity as well as complexity for the left hand, called fret complexity. These metrics are defined as follows:

String complexity:

- **Low:** The piece is monophonic, with no simultaneous notes.
- **High:** The piece is polyphonic, two or more strings are plucked simultaneously

Fret complexity:

- **Low:** Only one string held down at any one time and uses simple hand shapes.
- **Medium:** Two or more strings held down at one time and hand shapes are complex.
- **High:** Three or more strings held down at one time, hand shapes are complex.

Piece	String complexity	Fret complexity	Av. Duration (s)	Description
Piece 1	High	Low	22.9	Single and double notes, no frets used
Piece 2	Low	High	47.0	Single notes on C,G,Em,D chord hand shapes
Piece 3	Low	Medium	64.6	Pentatonic scale pattern repeated twice
Piece 4	Low	Low	70.6	First six frets used on all strings
Piece 5	High	High	66.3	First 36 notes of Für Elise

Table 4.2: Piece complexity breakdown

4.1.3 Quantitative Approach

As each note in the output tablature is represented by a string-fret pair, I have chosen to evaluate the performance of the string and fret components seperately. This was done to avoid weaknesses in a single component, such as plucking detection or fingertip detection, leading to a poor result overall dispite other components performing well.

There are two main questions I am looking to answer in this evalaution:

"How well does the system detect when a string is plucked?"

This is measured by the **string F1-score metric**. This was converted to binary classification problem by treating each frame as having six samples (one for each string) and two classes, plucked or not plucked. As it is possible for a note to be detected a few frames before or after a given notes' ground truth frame, it is neccessary to use a sliding window technique when calculating the confusion matrix. As described in Section 3.8, it is not possible for a string to be plucked more than once for a given $^2/_3$'s second period. Therefore, a window size of 20 frames was chosen because all videos used were recorded at 30 FPS because it would not be possible for a string to be predicted as plucked more than once within a window. The algorithm for producing the string confusion matrix is as follows:

- Each frames predictions are considered individually and in order
- If frame $i + 10$ has a note in the ground truth then all strings predicted to be plucked in the range $[i, i + 20]$ are considered as if they occured on frame $i + 10$. Update the confusion matrix and update $i = i + 20$.
- Else consider frame i individually, update the confusion matrix and update $i = i + 1$.

"How well does the system track the fingertips?"

This is measure by the **fret F1-score metric**. This was converted to a multi-class classification problem. There are a total of 20 classes due to the 19 frets and one case where no fretting could be predicted on the frame due to an error. Each note in the ground truth is treated as a sample. The ground truth stores the frame on which every note occurs in the video as well as the string-fret pair used for that note. The fret confusion matrix is calculated by looking up the fretting predicted for the ground truth onset frame and strings. To facilitate this, a hashmap from every frame number to the predicted string-fret pairs was also produced when processing each video in the dataset.

4.1.4 Obtaining ground truth

The ground truth for this project is a hashmap from frame number to the notes played on that frame. This was collected manually by first processing all evaluation videos to annotate each frame with the frame number, I then watched each of the videos in VLC media player and stored the hashmaps in Python pickle files.

4.2 Quantitative Metrics

4.2.1 Precision, Recall and F1

To evaluate the performance of the transcription system, I have chosen to measure **precision**, **recall** and **f1-measure** of the string and fret components of each note. This section explains the classical definition of each of these metrics while the next section demonstrates how they were applied to this project.

The precision (Equation 4.1) of a classifier measures the ratio of samples correctly labelled class i to the total number samples labelled class i. This gives an indication of whether the samples labelled class i are likely to be correct, but says nothing about whether all samples that are class i were correctly labelled.

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{4.1}$$

Recall (Equation 4.2) measures the ratio of samples labelled class i to the total number of sampled that were class i. This given an indication of whether a given sample of class i is likely to be labelled class i. However, it says nothing about whether other classes will also be labelled class i.

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{4.2}$$

It is possible to have a high precision and low recall, or high recall with low precision but in the case of a high quality transcription we hope to have both a high recall and precision. F1-measure is the harmonic mean of precision and recall, giving an overall indication of how good a classifier is.

$$f1\text{-}measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{4.3}$$

4.2.2 Macro-Averaging

For binary classification problems, you often choose the precision, recall and f1-measure of the least common class to indicate the performance of the overall classifier. In cases where there are a large number of classes, a common way to calculate the overall precision, recall and f1-measure is macro-averaging. Macro-averaging (Equation 4.4)is simply taking the mean across all classes.

$$precision = \frac{1}{n}\sum_{i=i}^{n} precision_i \tag{4.4}$$

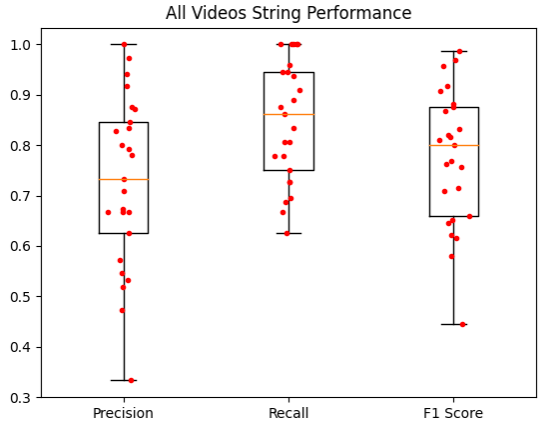
4.2.3 Evaluation Results

Note Component	String			Fret		
Scenes	Av. Precision (%)	Av. Recall (%)	Av. F1 Score (%)	Av. Precision (%)	Av. Recall (%)	Av. F1 Score (%)
Scene 1	84.7 ± 12.9	94.4 ± 7.0	89.2 ± 10.4	78.4 ± 11.0	88.5 ± 8.2	83.0 ± 9.1
Scene 2	78.4 ± 7.3	89.0 ± 11.1	83.1 ± 8.3	85.2 ± 9.5	89.9 ± 6.4	87.2 ± 6.7
Scene 3	69.3 ± 10.6	90.7 ± 6.5	78.1 ± 7.3	75.5 ± 14.8	77.4 ± 17.6	76.2 ± 16.0
Scene 4	55.3 ± 6.5	73.2 ± 5.7	62.4 ± 2.7	69.5 ± 19.9	58.8 ± 22.6	63.2 ± 21.1
Scene 5	75.9 ± 22.4	76.6 ± 9.7	74.5 ± 15.5	78.5 ± 18.6	83.3 ± 17.3	80.7 ± 17.8

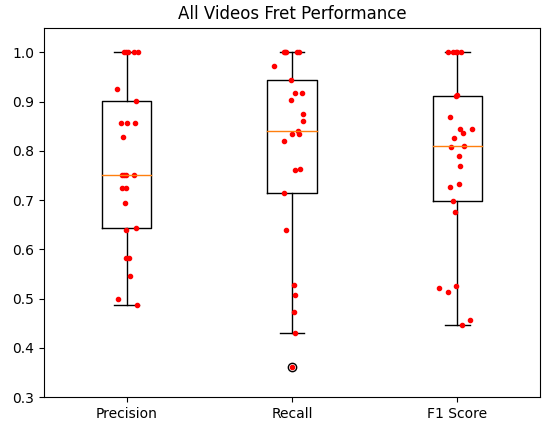
Table 4.3: Performance breakdown by scene. Standard deviation values are shown.

Note Component	String			Fret		
Piece	Av. Precision (%)	Av. Recall (%)	Av. F1 Score (%)	Av. Precision (%)	Av. Recall (%)	Av. F1 Score (%)
Piece 1	81.0 ± 9.8	82.5 ± 14.5	81.6 ± 12.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
Piece 2	75.3 ± 13.7	87.3 ± 12.3	80.4 ± 12.2	63.3 ± 10.0	64.4 ± 24.0	62.9 ± 16.7
Piece 3	57.8 ± 15.9	84.2 ± 12.5	68.1 ± 15.2	66.6 ± 9.6	76.1 ± 17.3	70.8 ± 13.2
Piece 4	82.5 ± 16.3	87.2 ± 11.6	83.6 ± 10.9	83.0 ± 4.2	82.8 ± 10.7	82.4 ± 6.0
Piece 5	67.0 ± 12.3	82.8 ± 4.1	73.6 ± 8.5	74.1 ± 14.9	74.5 ± 13.5	74.2 ± 13.9

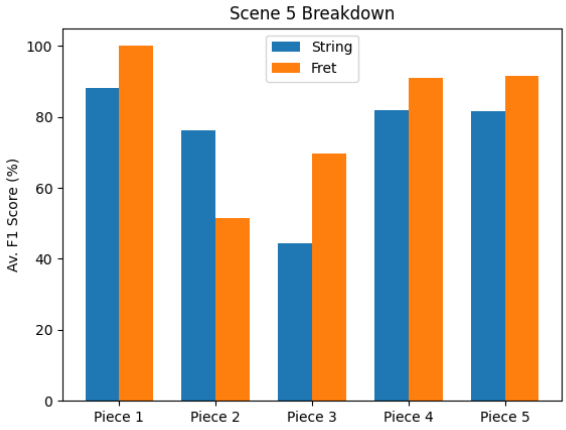
Table 4.4: Performance breakdown by piece. Standard deviation values are shown.



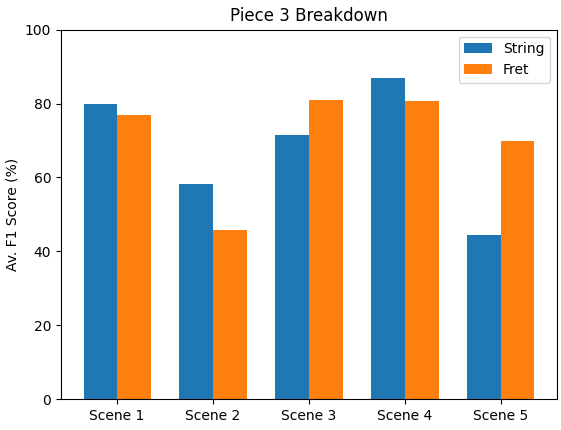
(a) All videos string component boxplot



(b) All videos fret component boxplot



(a) Scene 5 breakdown by piece



(b) Piece 3 breakdown by scene

Evidence of achivements

Table 4.3 shows how the system performs across each of the five scenes.

1. **(Core)** Successfully impliment each functional component required for transcribing videos of classical guitar performances using Computer Vision. (✔)

   Scenes 1 and 2 have all of the constraints required by the core criteria of this project and so high quality transcription in scenes 1 and 2 would prove this criteria met.

   Scenes 1 and 2 have the highest scores across all metrics. This means that the system performs best in good lighting conditions. Scenes 1 and 2 have string F1-scores of 89.2 and 83.1 respectively. A high string F1-score suggests that the system is very likely to detect a note if one is played, and very unlikely to detect a note if no note is played. As the string F1-scores are high, this means the system detects the onset of a note and the string is used for that note reliably. This suggests that the plucking detection module, as well as all of its dependencies, work reliably in videos under the constraints of this project. Scenes 1 and 2 have fret F1-scores of 83.0 and 87.2 respectively. A high fret F1-score suggests that the system is likely to correctly determine the fret of a note on average across all the frets. This proves that the system detects the fretting used for a note reliably, and so the fingertip detection module and all of its dependencies work reliably under the constraints of this project. Collectively, these results show that the intended functionality for this project was implimented properly and performs well under the constraints included in the core criteria .

2. **(Core)** This project should be able to transcribe monophonic and polyphonic music into guitar tablature. (✔)

   Pieces 2,3 and 4 are monophonic while pieces 1 and 5 are polyphonic. As piece 4 is the simplest monophonic piece, and piece 1 is the simplest polyphonic piece, a high quality transcription of pieces 1 and 4 would prove this criteria met.

   All pieces have a relatively high string F1-score between 68.1 and 83.6. The best performing piece is piece 4 which is monophonic; this result is followed by piece 1 which is polyphonic. This suggests that the system copes well with transcribing both simple monophonic and polyphonic pieces of music. Moreover, piece 5 is the most musically complex piece and has a string F1-score of 73.6 and fret F1-score of 74.2, which are both still high. Collectively these results show the system is capable of reliably detecting the onset of notes in both monophonic and polyphonic pieces. Moreover, this system has been shown to reliably detect the onset and string used for complex polyphonic pieces, which is considered a hard task for any AMT system[10].

3. **(Extension)** Develop a more robust implimentation that performs well under relaxed constraints. This includes working under poor lighting conditions. (✔)

   Scene 5 has poor lighting that is so poor the core implimentation is unable to transcribe the videos. A good quality transcription of pieces in scene 5 would show that the extension work completed allowed for videos to be transcribed even under relaxed input constraints.

This system produced guitar tablature for all of the scene 5 videos. The scene 5 F1-score for strings and fretting is 74.5 and 80.7 respectively. These are high scores and suggest that the system is capable of detecting when a note occurs, the string used for that note and the fret used for that note even in very poor lighting conditions. The scene 5 F1-scores are both higher than the scene 4 F1-scores, demonstrating superior performance can be gained by manually selecting the soundhole in poor lighting conditions compared to automatically detecting the soundhole in medium lighting conditions. However, the standard deviations for the string and fret F1-scores are 15.5 and 17.8. These are relatively large and much larger than the deivations seen in scene 1 and 2 which have good lighting. This can be explained in Figure 4.2a where it shows that all pieces except piece 3 were transcribed well. Figure 4.2b shows that scene 5 was the worst at transcribing piece 3, with scene 2 being the second worst. As scene 2 is well-lit, this suggests that the poor performance is more likely due to the piece than the scene illumination. Since the average string and fret F1-scores for scene 5 are still relatively high, with scene 5 outperforming scene 4, this demonstrates that the extension work was successful in producing a more robust system capable of transcibing videos under relaxed constraints.

Further discussion and analysis

- **Illumination** This system performed well across all levels of illumination. The best results came from scene 1 and 2 which is not surprising as they have the best lighting. Good lighting improves the reliability of each module. Scenes 3 and 4 have relatively poor results as they have medium lighting but still use automatic soundhole detection. Scenes 3 and 4 show a significant decrease in string precision and fret precision, resulting in lower f1 scores. This suggests that decreased illumination has a strong affect on the systems ability to reliably detect plucks and fretting. Though scene 3 and 4 also have lower string and fret recall, the spread in scores is much larger. This demonstrates a large **limitation of this evalaution** which is that large standard deviations can occur due to the small size of the dataset. This could not be avoided due to the time constraints of the part II project and the time it takes to manually collect the ground truth for each video. Using more videos in the evalaution would not be feasible.

- **Background** This system worked well across all backgrounds tried, in all levels of illumination. This demonstrates that it is robust to changes in background.

- **Guitar Angle** This system worked well across all guitar orientations tried, in all levels of illumination. This shows that is robust to changes in guitar orientation.

- **Chord Recognition** Piece 2 consists entirely of chords shapes in the following order: C, G, Em, D. Piece 2 has strong results for onset detection due to a string f1 score of 80.4%. However, piece 2 has the lowest fret precision at only 63.3. This is to be expected as chords are much more difficult to identify that simple hand shapes with only one finger being used, this means the system. This result suggests that the system performs well at detecting chords but it certainly one of its weakest areas.

- **Musical Complexity** Piece 5 is overall the most complex piece used in this evalaution. Dispite this, it beats piece 3 in its string F1 score, as well as pieces 2 and 3 for its frets

F1 score. This can be explained by the large number of notes, with many not being very difficuty to transcribe. The easier notes allow for an overall good score that would otherwise be similar to piece 2.

Piece 3 is a clear outlier in terms of performance. Dispite it not being the most complex to play, the hand shapes used in piece 3 lend them selves well to occlusion of fingertips. This means that while the hand moves across the fretboard, the fingertips often naturally cover each other from the cameras perspective and so it is impossible to infer the fretting of the piece from vision alone. This demonstrates a major **limitation of this system** which is that it can not cope with occlusion. A system that combines both audio and visual information would be able to overcome this case.

4.3 Processing time

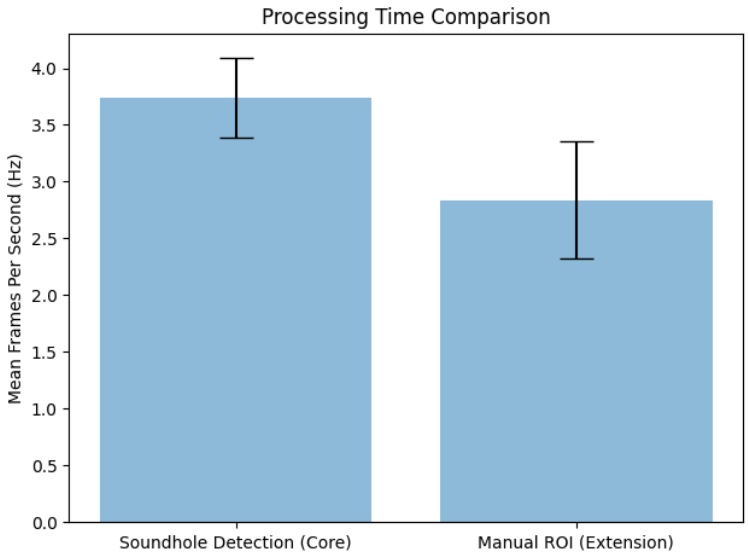


Figure 4.3: Processing time comparison between automatic soundhole detection and manual. Error bars represent the standard deviation.

Figure 4.3 shows the average frquency at which frames are processed when using automatic soundhole detection in comparison to having one ROI be selected and tracked throughout. When the soundhole is detected, this system processes videos at a rate of roughly 3.6 FPS, which is roughly 8.33 times longer than real time processing. When a soundhole ROI is manually selected, the rate of frame processing seems lower but this is because the time taken by the user to select the soundhole is not accounted for.

4.4 Qualitative Comparison To MelodyScanner

Melody Scanner[1] is an online platform that was built and maintained by professional developers. Melody Scanner supports the automatic transcription of various instruments including the piano, violin, flute and guitar. Melody Scanner uses an audio-based deep-learning end-to-end

[1]Melody Scanner is available at: https://melodyscanner.com/

transcription system for the guitar. Melody Scanner works by performing signal processing with a deep learning model trained on a large amount of labelled training data.

```
e|------0---|--0---0-------|--------------|--------------|----------0---|--0---0-----------
B|--------4-|----4---0-3-1-|--------------|--0---------0-|--1---------4-|----4---0-3-1-----
G|----------|--------------|--2---2-----2-|------1---1---|------2-------|---------------2--
D|----------|--------------|----2-----2---|----2---2-----|----2---2-----|------------------
A|----------|--------------|--0-----3-----|--------------|--0-----------|------------------
E|----------|--------------|--------------|--0-----------|--------------|------------------
```

(a) Ground Truth (Fur Elise: high string and fret complexity)

```
e|------0---|--0---0-------|----------------|-----------------|--1-------0-----|-9---9----------
B|--------4-|----3---0-3-1-|----------------|-2-----------0-0-|---------------4-|---4---0-3-1----
G|----------|--------------|--2---0-2-----2-|---------1---1---|------2---------|-------------2--
D|----------|--------------|----2-------3---|-------2---2-----|--0-2---2-------|----------------
A|----------|--------------|--0-------3-----|-----------------|----------------|----------------
E|----------|--------------|----------------|-0-0-0-----------|------------0---|----------------
```

(b) Project Output

```
e|------0---|-0---0---------|---------------|-------------|----------0---|--0---0------------
B|--------4-|---3---0-0-3-1-|---------------|-0---------0-|--1---------4-|----4---0-0-3-1----
G|----------|---------------|-2---2-------2-|-----1---1---|------2-------|-----------------2--
D|----------|---------------|---2-----2-2---|---2---2-----|----2---2-----|--------------------
A|----------|---------------|-0-----3-------|-------------|--0-----------|--------------------
E|----------|---------------|---------------|-0-----------|--------------|--------------------
```

(c) Melody Scanner Output

Figure 4.4: Output comparison on Scene 1 Piece 5 (| inserted to help visual comparison). My project has good results that are comparable to that of Melody Scanner, especially given that Für Elise is very complex.

Figure 4.4 shows that my system produces guitar tablature that is similar to the ground truth, with $^{30}/_{36}$ notes entirely correct with 0 missed onsets, 6 wrong notes and 6 extra notes. By comparison, Melody Scanner produces a better transcription with $^{36}/_{36}$ notes entirely correct, with 0 missed onsets, 0 wrong notes and 2 extra notes. This is still a great overall result for my system as Für Elise is a very complex musical piece.

In terms of run-time, Melody Scanner produced this tablature after just 66.1 seconds of processing, however my system took 563.9 seconds. This means my system was 8.53 times slower. As the video was 72.4 seconds long, this means Melody Scanner produced this transcription in real time while my system took 7.79 times longer than real time processing.

Chapter 5

Conclusions

This project has provably met all of its success criteria. A summary of this projects achivements is given below:

1. An end-to-end automatic transcription system for the classical guitar was implimented using Computer Vision. Each of the functional components described in Section 2.2 was implimented.
2. An extension was completed which allowed the user to manually select the soundhole. This extension was successful in allowing videos under relaxed constraints to also be transcribed.
3. A methodical qualitative evaluation was carried out with a dataset of 25 videos across 5 scenes and 5 pieces. This demonstrated the systems ability to reliably transcribe monophonic and polyphonic guitar pieces, across a variety of enviroments even in low light conditions.
4. A qualitative evaluation was completed by comparing the tablature generated by this system and an alternative system Melody Scanner. The results showed comparable performance on Für Elise, which is a highly complex polyphonic piece of music.

A limiting factor in this projects performance was its inability to deal with fingertip occlusion. This suggests that a multimodal system that combines either traditional signal processing techniques or modern deep learning techniques with computer vision would undoubtably perform even better.

This was the largest software project I have worked on and it was a pleasure to work diligently on it throughout the year. I have seen the importance of taking a disciplined approach to software development, using the proper tools and techniques required to deliver a functional, high quality product. Starting this project with no prior knowledge or experience with Computer Vision, it has been thrilling to read and learn more as I worked; after completing this project I am keen to undertake more projects within this field.

This project has provably met the requirements set for it, and I am satisfied with the performance achieved. I believe this system would serve as a useful tool to any beginner or amateur guitarists.

Bibliography

[1] Python pep 8 styling guide: https://www.python.org/dev/peps/pep-0008/#introduction. Accessed 22 April 2020.

[2] Python's design philosophy: https://python-history.blogspot.com/2009/01/pythons-design-philosophy.html. Accessed 22 April 2020.

[3] Mohammad Akbari and Howard Cheng. Real-time piano music transcription based on computer vision. *IEEE Transactions on Multimedia*, 17:1–1, 12 2015.

[4] Lance Alcabasa and Nelson Marcos. Automatic guitar music transcription. In *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 197–202, 2012.

[5] Carl-Fredrik Arndt and Lewis Li. Automated transcription of guitar music. Accessed 23 April 2021.

[6] S Baker and I Matthews. Lucas-kanade 20 years on: A unifying framework. 2004.

[7] D.H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[8] Ana M. Barbancho, Anssi Klapuri, Lorenzo J. Tardon, and Isabel Barbancho. Automatic transcription of guitar chords and fingering from audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(3):915–921, 2012.

[9] Emmanouil Benetos, Simon Dixon, Zhiyao Duan, and Sebastian Ewert. Automatic music transcription: An overview. *IEEE Signal Processing Magazine*, 36(1):20–30, 2019.

[10] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: Challenges and future directions. *Journal of Intelligent Information Systems*, 41, 12 2013.

[11] Anne-Marie Burns. *Computer Vision Methods for Guitarist Left-Hand Fingering Recognition*. PhD thesis, 01 2007.

[12] L. Chandrasekar and G. Durga. Implementation of hough transform for image processing applications. In *2014 International Conference on Communication and Signal Processing*, pages 843–847, 2014.

[13] Ivan Dokmanić, Mihailo Kolundžija, and Martin Vetterli. Beyond moore-penrose: Sparse pseudoinverse. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6526–6530, 2013.

[14] O. Gillet and Gaël Richard. Automatic transcription of drum sequences using audiovisual features. volume 3, pages iii/205 – iii/208 Vol. 3, 04 2005.

[15] B S Gowrishankar and Nagappa U Bhajantri. An exhaustive review of automatic music transcription techniques: Survey of music transcription techniques. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 140–152, 2016.

[16] Christopher Hollitt. Reduction of computational complexity of hough transforms using a convolution approach. In *2009 24th International Conference Image and Vision Computing New Zealand*, pages 373–378, 2009.

[17] Misbah Ahmad Kaleem Ullah, Imran Ahmed and Iqbal Khan. Comparison of person tracking algorithms using overhead view implemented in opencv. *Center of Excellence in Information Technology, Institute of Management Sciences (IMSciences) Hayatabad, Peshawar*, 2019.

[18] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[19] Chutisant Kerdvibulvech and Hideo Saito. Vision-based detection of guitar players' fingertips without markers. In *Computer Graphics, Imaging and Visualisation (CGIV 2007)*, pages 419–428, 2007.

[20] Chutisant Kerdvibulvech and Hideo Saito. Vision-based guitarist fingering tracking using a bayesian classifier and particle filters. In Domingo Mery and Luis Rueda, editors, *Advances in Image and Video Technology*, pages 625–638, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[21] Jun Li, Wei Xu, Yong Cao, Wei Liu, and Wenqing Cheng. Robust piano music transcription based on computer vision. pages 92–97, 07 2020.

[22] Alan Lukežič, Tom'aš Voj'iř, Luka Čehovin Zajc, Jiř'i Matas, and Matej Kristan. Discriminative correlation filter tracker with channel and spatial reliability. *International Journal of Computer Vision*, 2018.

[23] R. Martin. Clean code: A handbook of agile software craftsmanship. Prentice Hall, 2008.

[24] J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78(1):119–137, 2000.

[25] Kevin McGuinness, Olivier Gillet, Noel E. O'Connor, and Gaël Richard. Visual analysis for drum sequence transcription. In *2007 15th European Signal Processing Conference*,

pages 312–316, 2007.

[26] Yoichi Motokawa and Hideo Saito. Support system for guitar playing using augmented reality display. In *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 243–244, 2006.

[27] Sakshi Patel, Shivani Sihmar, and Aman Jatain. A study of hierarchical clustering algorithms. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 537–541, 2015.

[28] Joseph Scarr and Richard Green. Retrieval of guitarist fingering information using computer vision. 2010.

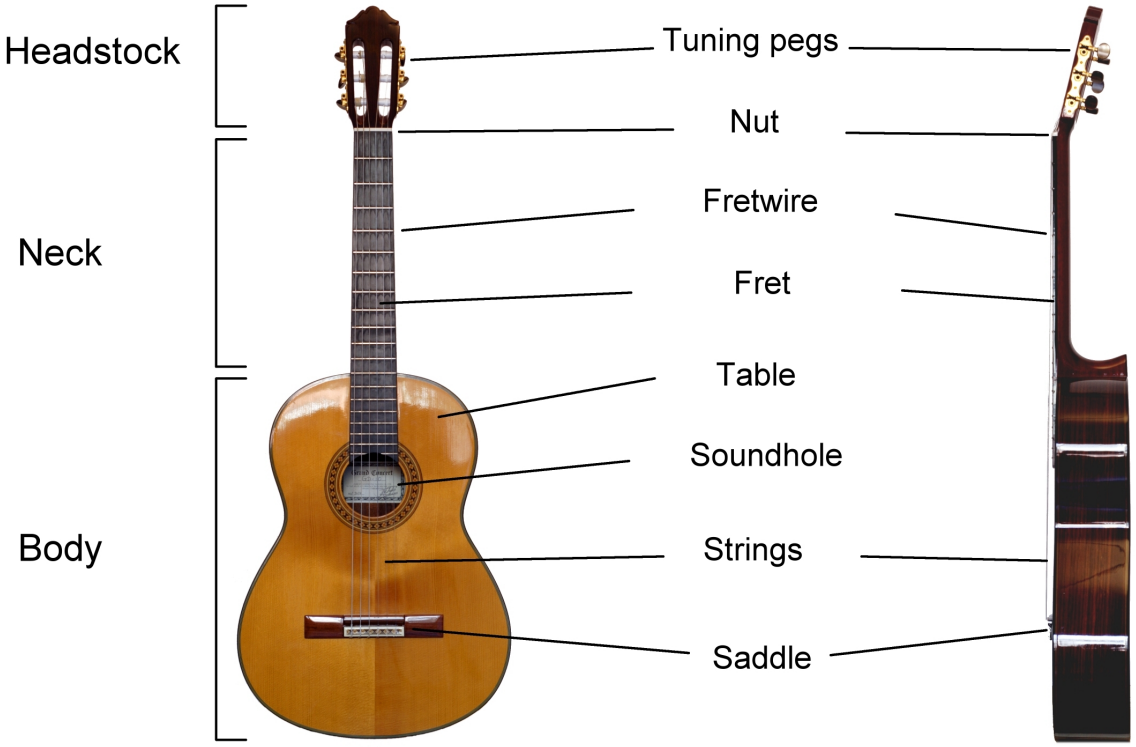
[29] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Comput. Vis. Graph. Image Process.*, 30:32–46, 1985.

[30] Shin-Ting Wu, Adler Silva, and Mercedes Márquez. The douglas-peucker algorithm: Sufficiency conditions for non-self-intersections. *J. Braz. Comp. Soc.*, 9:67–84, 04 2004.

[31] Virendra Yadav, Saumya Batham, Anuja Acharya, and Rahul Paul. Approach to accurate circle detection: Circular hough transform and local maxima concept. pages 1–5, 02 2014.

Appendix A

Guitar Terminology



Appendix B

Evaluation Pieces

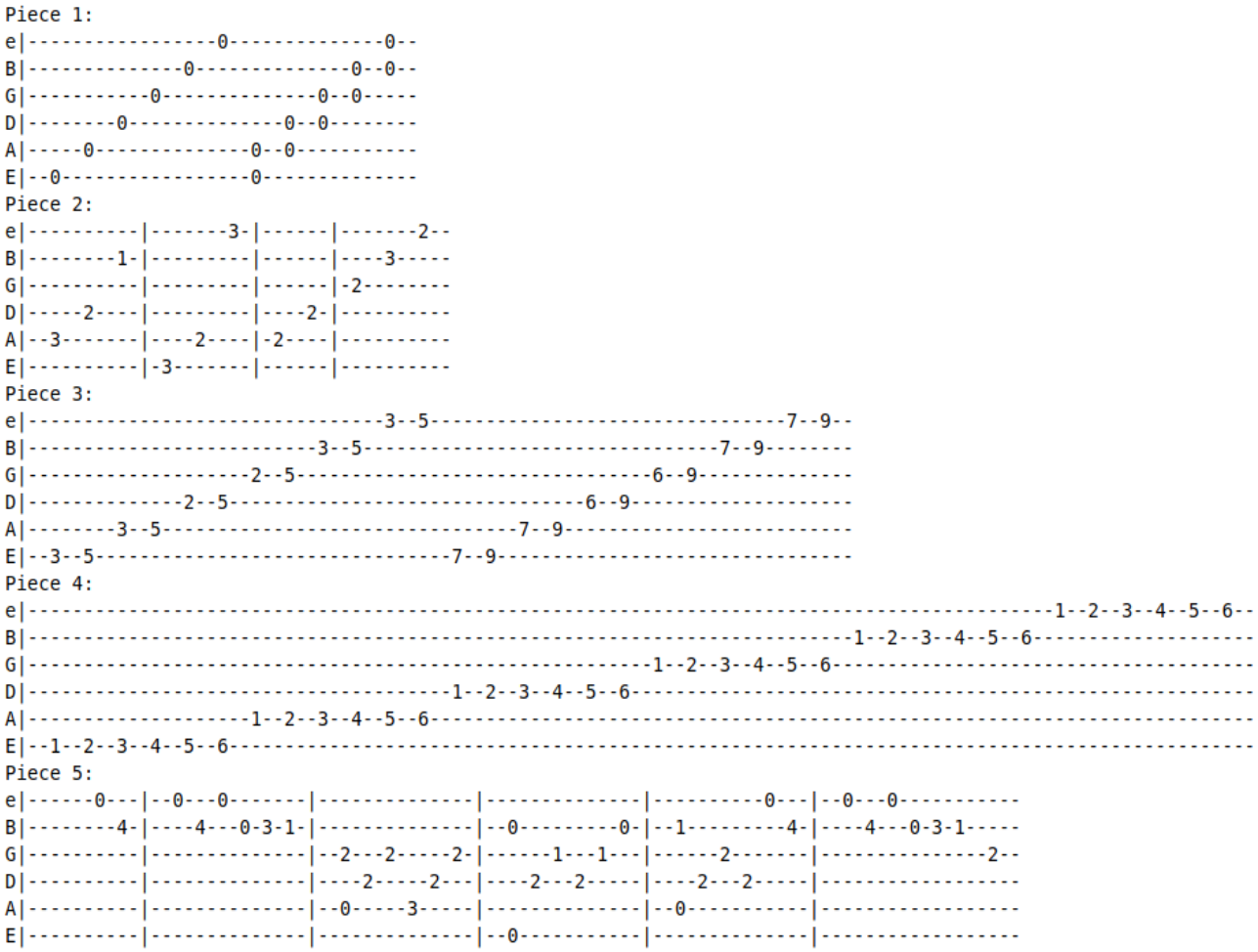


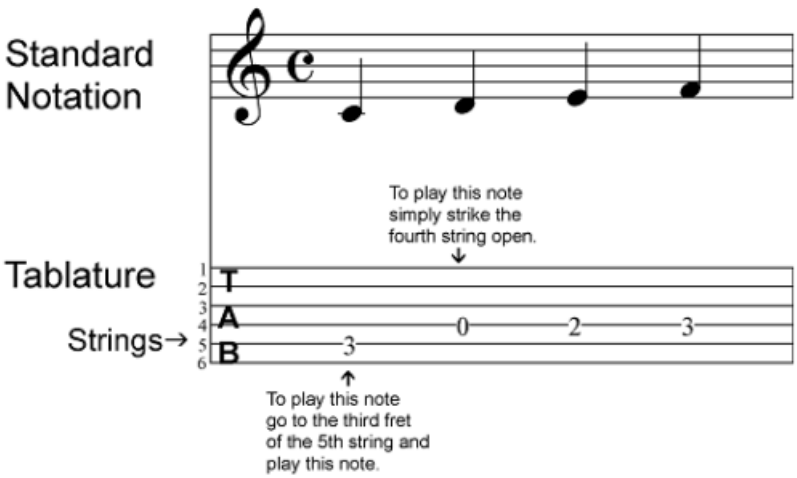
Figure B.1: All pieces tablature (| lines added to improve readability)

Project Proposal

Introduction and Description of the Work

Guitar tablature, often shortened to guitar "tabs", is a way of transcribing music specifically for the guitar and is most popular with amateur guitar players. As apposed to the standard music notation (staff notation) used in sheet music, it describes the fretting used to produce each note rather than simply the pitch of the note itself. A comparison is given below in figure 1. Indicating the string and the fret used is necessary because fretted instruments, such as the guitar, allow for notes of the exact same pitch to be produced from multiple string and fret combinations. For example, 6th string on the 5th fret is the same pitch as 5th string on the 0th fret. This makes staff notation harder to use as a guitarist is required to calculate, on-the-fly, the most suitable fingering to play the given sequence of notes. It is for this reason that many amateur guitarists greatly prefer using guitar tablature as a form of transcription.

Figure 1: Comparison of staff notation and guitar tablature



Although guitar tablature is useful, writing guitar tablature is a boring, tedious task and one that is far too often necessary. With videos and audio recordings of guitar performances freely available on the internet, many guitarists choose to use videos of performances to help them learn. However, guitarists will often find that the corresponding guitar tablature for the video is not available. Even if they find guitar tablature for the same song, it may use a different fingering to that used in the video, which makes the learning process harder. This leads to many amateur guitarists

feeling frustrated as they are not willing to manually write the guitar tablature used in the video.

Thankfully, some solutions exist in both hardware and software, all to varying success. One hardware solution is to use a MIDI guitar; this uses separate MIDI channels assigned to each string in order to know the pitch of that string in real time, which can be used to deduce the fret position. However, MIDI guitars are expensive and often have usability issues including a variation in the recognition time from one string to another. Software solutions based purely on audio also currently exist, but these too have clear problems. Despite being able to often produce a sensible fretting, it is very difficult to reliably reproduce the same fretting unless the timbre (the unique sound) of the instrument being played is known. Another major problem with audio based transcription is the need to isolate the instrument that is being transcribed. This means that there are clear advantages to using a vision based approach, however it is still very difficult to do accurately.

For my dissertation, I will produce a system capable of transcribing video footage of a classical guitar being played into guitar tablature. This system will only need to work on videos featuring performances where certain constraints have been applied, such as having a stationary guitar or a suitable camera angle. In order to produce guitar tablature, I only need to detect the string and fret combinations for each note and the time at which the note is played. I will not need to detect the length of each note, or other information such as percussive effects. To complete this project, I will primarily use Computer Vision techniques. The software will need to: detect the strings and frets on the guitar, detect when strings have been plucked, track the locations of the fingertips on the fretboard and give feedback to the user to indicate where there are likely to be errors in the tablature produced.

Starting Point

Personal Starting Point

My language of choice for this project is Python and I plan to primarily use the libraries Numpy and OpenCV. I have some prior experience with Python and Numpy. This mainly comes from courses in the Computer Science tripos, namely Part 1A Numerical Computing and Part 1B Data Science. In addition, I have used Python

to complete personal projects as well as projects in industry during my last summer internship. I plan to use VSCode (Visual Studio Code) as my code editor for this project, which I have a good amount of experience using.

I have not had any experience with Computer Vision prior to this project, so I will need to quickly learn and understand the relevant theory and documentation during the preparation phase if I am to make a successful implementation. I intend to use OpenCV, an open source Computer Vision library that supports C++ and C, with wrappers for Python and Java. Numpy will be required to handle a lot of the data manipulation outside of OpenCV.

As part of my extensions, I mention the possible use of either a SVM or CNN. I do not have any practical experience training, using or evaluating these tools. I also do not have any experience with any libraries that are typically used for these tools such as Scikit-learn, PyTorch, Keras or Tensorflow.

For these reasons, this project will require a significant amount of work from me in order for it to be successful. It will require a substantial amount of theory to be learned as well as the design and successful implementation of various algorithms, using multiple tools, in order for it to be effective.

There is some overlap between this project and this years Tripos courses, namely:

- The Computer Vision Part II course, which involves the theory behind many of the Computer Vision techniques I will apply over the course of the dissertation. I do plan on taking this course for lectures, supervisions and in the exam but since it is in Lent, it will mainly be used to reaffirm my understanding already gained from implementing this project.
- The Deep Neural Networks unit of assessment, which is Lent course I will be taking. This covers much of the relevant theory required for implementing a classifier based approach to fingertip detection, but will likely be running after I have began work on that extension, if I do manage to work on it at all.
- The Machine Learning and Bayesian Inference course, similar to Deep Neural Networks, this Lent course covers much of the theory related implementing a classifier based approach to fingertip detection but it will not be directly useful as it occurs after I should have began work on that extension. I am not intending on taking supervisions for this course, or choosing this question in the exam, however I will watch the lectures and take notes to reinforce my knowledge.

Background

As mentioned in the introduction, automatic music transcription software does exist using audio alone for the guitar [2] [3]. Audio based musical transcription has also been applied to as well as other instruments including the piano and drums [1] [4]. More recently, computer vision has started to be used for transcriptions of the piano [5]. This, combined with the fact that continued research into different methods involving both computer vision [6] [7] and machine learning [9] for fretting extraction on the guitar mean a vision based approach to automatically transcribing a guitar performance is becoming more possible.

I will be reading these papers, and others, for research but I will be designing and implementing my own solution independently from scratch.

Resources Required

I plan to use my personal laptop for the writing and executing of my code, as well the writing of my dissertation. I also have access to an old laptop which can be used in the case of my personal laptop failing. The specifications of these machines are:

- Personal laptop: A Dell XPS 15 laptop with an Intel Core i7 6700HQ CPU running at 2.6GHz, an Nvidia GTX 960M GPU with 2GB of VRAM, 16 GB of system RAM, 1.5 TB of SSD storage, running the Ubuntu 20.04 operating system but with sufficient space to install Windows 10 if necessary.
- Secondary laptop: An Acer Aspire V15 Nitro with Intel Core i7 7700HQ CPU running at 2.8GHz, an Nvidia GTX 1050 GPU with 4GB of VRAM, 16 GB of system RAM, 256 GB SSD storage and 1 TB of magnetic disk storage, running the Windows 10 operating system with sufficient space to install Linux if necessary. This machine has some screen and keyboard issues, as well as being quite

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

In particular, if either machine fails I can continue working on the other. I also have money saved so that I could repair either machine if either of them failed. If, for

whatever reason, both laptops fail on me then I will use the MCS machines while I work to get a new machine as soon as possible.

To protect myself from data loss I have multiple systems in place:

- Changes to the source code for this project, as well as the latex source code for the dissertation will be pushed to a private git repository hosted on GitHub. For each day I make changes to this repository I will be sure to commit and push at least once.
- I will keep the local copy of this git repository automatically synced and backed up onto my personal Google Drive account. This means any changes to my local files, even un-pushed ones, will be backed up onto Google Drive.

I do not require additional resources for this project. I plan to use my own laptop for all development. I will be using my own guitar, or guitars I can borrow from friends, during the development and evaluation of this project. For recording video, I will use my phone camera which has a 4k resolution and records video at 60fps. If this camera is not sufficient, I will purchase or borrow one that is. Similarly, if my phone breaks, I will purchase another phone that will replace it. All videos will be backed up onto Google Photos, and then downloaded to my laptop for processing.

Project Structure

For the dissertation, I plan to produce a system which is capable of taking a video of a classical guitar being played (under certain constraints) and produce the corresponding guitar tablature to a reasonable level of accuracy. It will give feedback to the user on which parts of the transcription which are likely to be wrong. All processing will occur offline and I do not expect the application to run in real time.

I have divided this project into multiple sub-modules:

- Foreground extraction - This module will extract the guitar and person in the scene from the background. This will be helped by constraining the input such that the background is plain and any clothing visible will not be a similar colour to the guitar or the skin of the player. The guitar, clothes and background will

not be skin coloured. This will likely use the GrabCut algorithm for image segmentation. For this to be effective, it may be necessary for the guitar to be stationary in the video.

- Strings and Frets detection - This module will detect and create a representation of the strings and frets on the guitar. Using Canny Edge Detection I will likely be able to find the lines on the guitar, I may also use a Hough Transform to extract lines from the image. I may be able to create a bounding box around the neck of the guitar and interpolate to find the expected position of the lines. As for frets, these will be found after the strings by looking for lines perpendicular to the strings. I can also exploit the fact that the horizontal spacing between frets increases exponentially as a function of distance along the fretboard.

- Image segmentation - This module will produce two boxed, normalized, segments that will later be used by the plucking detection and fingertip detection modules. The first segment will be around the sound-hole and will be needed to help with detecting a plucked string. This module will find the sound-hole of the guitar and segment it, it will then normalize it such that the strings are horizontal. To do this, I plan to use a Circular Hough Transform to detect the sound hole, then normalize it. The second segment will be around the left hand and will be needed for fingertip detection. To do this, I plan to use skin detection, segment the left hand on the fretboard and finally, normalize it. For high accuracy, it may be necessary to have a plain background and no skin coloured clothing or indeed a skin coloured guitar.

- Plucking detection - This module will take an image of the sound hole to find the occurrence of a string being plucked, and therefore a note being generated. It will have to store the time at which it was generated too. One potential way to detect a pluck is use a temporal algorithm to try find the point in time in which the string is most bent and then starts to oscillate. This may involve trying to find the angle the string is bent during a plucking motion. This is more general than hand segmentation or skin detection methods as it works regardless of whether a pic or finger is used to pluck. This may be very difficult to detect reliably and so a more sophisticated approach or a camera with a higher frame rate could be required.

- Fingertip detection - This modules receives a normalized segmented image of the left hand of the guitar player which should be over the fretboard. This will then look to perform fingertip detection. There are multiple different approaches I am considering for this currently. The first is to perform a Circular Hough Transform to look for the semi-circular shape of a fingertip. This is similar

to what is proposed in [6]. The second is the to perform skin detection to binarize the image and then find the contour using a border following algorithm. With this contour feature, this algorithm will then look for the highest string that intersects with each local maximum in the contour and infer these are the fingertip locations. A different contour based approach was used in [8]. Finally, I am also considering hand segmentation, whereby I will look to segment the hand into fingers. A useful extension for this is to track which fingertip on the hand is pressing which string, this allows for a more detailed form of tablature.

- Sanity check - This module will look at the output produced and highlight any clear issues with it. For example, some fretting is impossible, such as reaching 10 frets apart almost instantly. This feedback will be given to the user to highlight times in the video that there have likely been mistakes. This will be done by analysing the output for fretting that seems impossible to play, or possibly using heuristics to estimate this.

I have mentioned above the algorithms I intend to use. Note that for many of them, some experimentation will be required to find the appropriate thresholds, for example with Hough transforms and Edge detection. I will also need to experiment with different algorithms for skin detection and contour tracing if necessary.

OpenCV primarily uses the matrix data structure to encapsulate the 2d array of pixels representing an image. I will also use the matrix data structure to represent the vectors and lines of the strings and frets. I expect the coordinates on a convex hull or contour would be stored in arrays using the Numpy library.

Input constraints

I will need to introduce constraints onto the input videos. This will make the videos easier to automatically transcribe while also giving the core project more achievable goals from which I can slowly build from in my extensions.

The list of input constraints:

- A suitable camera setup and guitar placement. This means the camera should be looking down slightly onto the fretboard making the fingertips of the player easier to see. The guitar should be as close as possible while still allowing the entire fretboard and sound-hole to remain visible.

- A plain background is used and plain clothing is worn.
- No skin coloured clothing, guitar or background.
- The guitar remains stationary throughout the video.
- The scene remains well lit throughout the video.
- Each finger can only press down on one string at a time on the fretboard. This means no bar chords can be performed. This is because I will only be detecting the fingertips used for fretting, and so I am assuming only one string is being pressed by one fingertip.
- No peripheral devices are used, such as a capo.
- I will also be constraining what the performer can play. No frets above the 12th fret should be used. This is because frets higher than 12 are very narrow which means individual fingertips are likely very hard to see and detect.
- The performer should place their hands in a way that cooperates with the software if possible. This means not covering the sound-hole or hiding their fingertips unnecessarily.

Possible Extensions

There are a large number of possible extensions I can think of for this project. I would like to implement some of them which would either make the system more useful or provide an opportunity to compare techniques:

1. Work to improve the systems performance with less environmental constraints on input videos. This could mean in different backgrounds, using different camera angles or in worse lighting.
2. Improve how the system performs when all frets are used.
3. Individual Finger Tracking - Notate which fingers were used to press on each string. This would build on top of fingertip detection, possibly requiring joint angles to be calculated.

4. Rhythm Detection - This module would analyse the notes' timestamps and possibly keep track of when the strumming hand is above or below the strings to help find strumming patterns.

5. Add a GUI - This will allow for the selection of a video and an output directory. It should also display and save any transcription feedback to the user.

6. Detect the use of a capo - The user may have to indicate which videos use a capo for this to be reliable. Moreover, the capo must be reliably detectable in some way.

7. Detect other features such as string bends or slides. Out of these, probably the most feasible is detecting string bends. This would likely be a temporal algorithm that stores and analyses changes in the gaps between strings.

8. Re-implement the fingertip detection module using a SVM or possibly a CNN. In order to reduce the number of categories, I plan to classify which strings are pressed and which frets are pressed separately. This may use similar features to those I discussed extracting for the core implementation, such as the hand contour, the hand convex hull, or performing hand segmentation. Based on the features I choose to extract, I can classify each finger to one of the six possible strings or none. Separately, I could then try classify how far a finger has moved up or down the fretboard from its base position probably relying on hand segmentation and the angle of each segment.

   I do not know of any training data available for this, so I may have to create a small training set. I could automate this by using coloured fingertip tags to detect the true location of the fingertips and then use each frame of video as a training image.

Criterion for Success

I can measure the success of this project by looking at whether I achieve the following goals:

1. Each sub-module has been implemented and guitar tablature is successfully generated for videos under the constraints I mentioned. This demonstrates that the planned functionality of each sub-module was achieved.

2. The system provides information to the user to let them know where in the transcription an error is likely to have occurred.

3. A quantitative evaluation of the project has been carried out by comparing the outputs of test videos to the actual guitar tablature.

4. Though this system is not required to run in real time, it should finish processing short videos (at most few minuets) in under a few hours.

Though the extensions are not required for the project to be a success, any extensions implemented would mean it is more useful and more technically complex.

I would like to separate the evaluation into six main questions:

1. How accurate is the system at detecting when a string is plucked?

2. How well does the system detect the fretting used in the video?

3. How accurate is the system at various levels of musical complexity?

4. How well does the system work in non-ideal environments?

5. How does the system compare to existing software solutions?

6. How long does the system take to process input?

In order to perform this evaluation, I will need to create a set of songs which vary in musical complexity. For each song, videos will be made with varying by multiple factors such as the person featured, the guitar used, the background and the lighting in order to form a comprehensive evaluation. An ethics review will be necessary as other people will be using this system as part of the evaluation.

Due to covid-19, it is vital I make sure it is safe for other people to test this system. This will likely mean I will explain to them how to set up the camera and other factors properly and they will record the video on their own in their room. I will sanitize the the guitar and other equipment before and after each test. Of the people I currently know who have agreed to test this system, both are in my college and one is in my household.

I may use videos taken from the internet if they are of a suitable quality. If possible, I will use music from the "guitar grades" exams, a form of qualification in classical

guitar playing, which has grades ranging from one to eight. However, in order to obtain performances of these pieces with the setup I require, I will need at least one person of grade eight ability to use the system which may prove difficult. In that case, I may include pieces from lower grades only, or if necessary, I will classify music using alternative metrics. Finally, it would be very useful to see how well my system performs against software that currently exists, either audio or video, however this will require some research.

When analysing a video, I may divide each video into short periods of time and evaluate each of these independently to give more fine grained analysis. To measure the accuracy of different parts of the system, there are multiple possible approaches but they will all likely rely on comparing the true guitar tablature to the output in some way, though the best approach to this is still unclear.

Timetable and Milestones

To help me stay organised and on track, I plan to divide my time into 2 week slots.

Slot O - 9th October - 23rd October

- Continue researching computer vision techniques as well as familiarise myself with OpenCV in Python
- Make a private Github repository to for the project code and dissertation
- Finish my project proposal and submit it with approval

Milestone: Dissertation idea fully decided, supervisor confirmed and background research complete

Deadline: Proposal Deadline - 23rd October

Slot 1 - 24rd October - 6th November

- Create the basic class structure of the application
- Submit the ethics review application

- Find a camera angle and background setup that is suitable for this project and easily replicated
- Record a suitable amount of footage for developing the foreground extraction as well as string/fret detection module
- Begin working on the foreground extraction module

Milestone: Foreground extraction module has begun development but may not be complete as I will be committed to DSP. A suitable amount of footage is recorded for early development. Ethics review application is approved.

Deadline: DSP unit of assessment task (date to be confirmed)

Slot 2 - 7th November- 20th November

- Finish the foreground extraction module
- Begin work on the string/fret detection module

Milestone: The foreground extraction module will be completed.

Deadline: DSP unit of assessment task (date to be confirmed)

Slot 3 - 21st November - 4th December

- Continue working on string and fret detection module if needed
- Begin work on the image segmentation module
- If time allows, I could begin work on plucking detection

Milestone: The string and fret detection and image segmentation modules are both implemented.

Deadline: DSP unit of assessment task (date to be confirmed)

Slot 4 - 5th December - 18th December

Michealmas term has ended and so I will have more time to commit to the project.

- Continue work on plucking detection

- Begin work on fingertip detection

Milestone: As both plucking detection and fingertip detection are implemented, the first full transcription will be produced.

Slot 5 - 19th December - 1st January

- Create module to sanity check the output
- Create a script for comparing two guitar tablature files: this will be used for evaluation

Milestone: A full implimentation is complete and evaluation can begin

Slot 6 - 2nd January - 15th January

Allow time for revision and catch up with anything I was not able to complete up until now.

- Begin considering possible extensions and researching them
- Create the basic dissertation format and begin planning what I expect to be in each chapter.

Milestone: Dissertation has started to be written

Slot 7 - 16th January - 29th January

- Record or obtain videos needed for my evaluation, including with other people and in other backgrounds
- Begin work on Possible Extensions
- Begin writing my progress report

Slot 8 - 30th January - 12th February

- Finish performing evaluations and continue work on extensions
- Finish writing my progress report

- Submit progress report

Deadline: Progress Report Submission Deadline 5th February

Deadline: Progress Report Presentation 11th-16th February

Deadline: DNN unit of assessment work (date to be confirmed)

Slot 9 - 13th February - 26th February

- Finish extensions and begin writing the introduction/preparation chapters of the dissertation
- Give progress report presentation

Milestone: Introduction and preparation chapters of the dissertation are finished

Deadline: DNN unit of assessment work (date to be confirmed)

Slot 10 - 27th February- 12th March

- Begin writing the implementation section of the dissertation
- Prepare the source code for submission

Deadline: DNN unit of assessment work (date to be confirmed)

Milestone: Source code is finished and ready for submission

Slot 11 - 13th March - 26th March

- Finish writing the implementation section

Milestone:

Slot 12 - 27th March - 9th April

- Write the evaluation and conclusion sections
- Refine any previously completed chapters and make sure the bibliography is complete

Milestone: Dissertation first draft is complete

Slot 13 - 10th April - 23rd April

- Begin refining my dissertation based on feedback from my supervisor, director of studies etc

Slot 14 - 24th April - 7th May

- Make any changes necessary to the dissertation, continuing to use feedback

Milestone: Dissertation final draft is complete

Slot 15 - 8th May - 21st May

- Submit dissertation and source code

Deadline: Dissertation Deadline (electronic) 14th May

Deadline: Source Code Deadline (electronic copies) 14th May

References

[1] E. Benetos, S. Dixon, Z. Duan and S. Ewert, "Automatic Music Transcription: An Overview," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 20-30, Jan. 2019, doi: 10.1109/MSP.2018.2869928.

[2] L. Alcabasa and N. Marcos, "Automatic Guitar Music Transcription," 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT), Kuala Lumpur, 2012, pp. 197-202, doi: 10.1109/ACSAT.2012.78.

[3] A. M. Barbancho, A. Klapuri, L. J. Tardon and I. Barbancho, "Automatic Transcription of Guitar Chords and Fingering From Audio," in IEEE Transactions on Audio, Speech, and Language Processing, vol. 20, no. 3, pp. 915-921, March 2012, doi: 10.1109/TASL.2011.2174227.

[4] C. Wu et al., "A Review of Automatic Drum Transcription," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 26, no. 9, pp. 1457-1483, Sept. 2018, doi: 10.1109/TASLP.2018.2830113.

[5] Akbari, M., and Cheng, H. Real-time piano music transcription based on computer vision. IEEE Transactions on Multimedia 17, 12 (2015), 2113–2121.

[6] Burns, Anne-Marie Wanderley, Marcelo. (2011). Computer vision method for guitarist fingering retrieval.

[7] Kerdvibulvech, C., and Saito, H. Vision-based detection of guitar players' fingertips without markers. In Computer Graphics, Imaging and Visualisation, 2007. CGIV'07 (2007), IEEE, pp. 419–428.

[8] J. Scarr and R. Green, "Retrieval of guitarist fingering information using computer vision," 2010 25th International Conference of Image and Vision Computing New Zealand, Queenstown, 2010, pp. 1-7, doi: 10.1109/IVCNZ.2010.6148852.

[9] Kerdvibulvech C., Saito H. (2007) Vision-Based Guitarist Fingering Tracking Using a Bayesian Classifier and Particle Filters. In: Mery D., Rueda L. (eds) Advances in Image and Video Technology. PSIVT 2007. Lecture Notes in Computer Science, vol 4872. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77129-6_54